# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is Amazon SageMaker?</td>
<td>1</td>
</tr>
<tr>
<td>Are You a First-time User of Amazon SageMaker?</td>
<td>1</td>
</tr>
<tr>
<td>How It Works</td>
<td>2</td>
</tr>
<tr>
<td>Machine Learning with Amazon SageMaker</td>
<td>2</td>
</tr>
<tr>
<td>Explore and Preprocess Data</td>
<td>4</td>
</tr>
<tr>
<td>Model Training</td>
<td>4</td>
</tr>
<tr>
<td>Model Deployment</td>
<td>7</td>
</tr>
<tr>
<td>Hosting Services</td>
<td>7</td>
</tr>
<tr>
<td>Batch Transform</td>
<td>10</td>
</tr>
<tr>
<td>Validating Models</td>
<td>12</td>
</tr>
<tr>
<td>Programming Model</td>
<td>13</td>
</tr>
<tr>
<td>Set Up Amazon SageMaker</td>
<td>15</td>
</tr>
<tr>
<td>Step 1: Create an AWS Account</td>
<td>15</td>
</tr>
<tr>
<td>Step 2: Create an IAM Administrator User and Group</td>
<td>15</td>
</tr>
<tr>
<td>Step 3: Create an Amazon S3 Bucket</td>
<td>16</td>
</tr>
<tr>
<td>Get Started</td>
<td>17</td>
</tr>
<tr>
<td>Step 1: Create an Amazon SageMaker Notebook Instance</td>
<td>17</td>
</tr>
<tr>
<td>Step 1.1: (Optional) Customize a Notebook Instance</td>
<td>20</td>
</tr>
<tr>
<td>Next Step</td>
<td>22</td>
</tr>
<tr>
<td>Step 2: Train and Deploy a Model</td>
<td>22</td>
</tr>
<tr>
<td>Step 2.1: Create a Jupyter Notebook</td>
<td>23</td>
</tr>
<tr>
<td>Step 2.2: Download, Explore, and Transform Data</td>
<td>24</td>
</tr>
<tr>
<td>Step 2.3: Train a Model</td>
<td>26</td>
</tr>
<tr>
<td>Step 2.4: Deploy the Model</td>
<td>29</td>
</tr>
<tr>
<td>Step 2.5: Validate the Model</td>
<td>36</td>
</tr>
<tr>
<td>Step 3: Clean up</td>
<td>40</td>
</tr>
<tr>
<td>Step 4: Additional Considerations</td>
<td>40</td>
</tr>
<tr>
<td>Using Notebook Instances</td>
<td>42</td>
</tr>
<tr>
<td>Create a Notebook Instance</td>
<td>42</td>
</tr>
<tr>
<td>Access Notebook Instances</td>
<td>43</td>
</tr>
<tr>
<td>Limit Access to a Notebook Instance by IP Address</td>
<td>43</td>
</tr>
<tr>
<td>Connect to a Notebook Instance Through a VPC Interface Endpoint</td>
<td>44</td>
</tr>
<tr>
<td>Use Example Notebooks</td>
<td>47</td>
</tr>
<tr>
<td>Set the Notebook Kernel</td>
<td>48</td>
</tr>
<tr>
<td>Install External Libraries and Kernels in Notebook Instances</td>
<td>48</td>
</tr>
<tr>
<td>Associate Git Repositories with Amazon SageMaker Notebook Instances</td>
<td>49</td>
</tr>
<tr>
<td>Add a Git Repository to Your Amazon SageMaker Account</td>
<td>50</td>
</tr>
<tr>
<td>Create a Notebook Instance with an Associated Git Repository</td>
<td>52</td>
</tr>
<tr>
<td>Use Git Repositories in a Notebook Instance</td>
<td>53</td>
</tr>
<tr>
<td>Build a Model</td>
<td>56</td>
</tr>
<tr>
<td>Choose a Machine Learning Algorithm</td>
<td>56</td>
</tr>
<tr>
<td>Use Built-in Algorithms</td>
<td>56</td>
</tr>
<tr>
<td>Common Information</td>
<td>59</td>
</tr>
<tr>
<td>BlazingText</td>
<td>73</td>
</tr>
<tr>
<td>DeepAR Forecasting</td>
<td>82</td>
</tr>
<tr>
<td>Factorization Machines</td>
<td>96</td>
</tr>
<tr>
<td>Image Classification Algorithm</td>
<td>106</td>
</tr>
<tr>
<td>IP Insights</td>
<td>127</td>
</tr>
<tr>
<td>K-Means Algorithm</td>
<td>137</td>
</tr>
</tbody>
</table>
Use OpenAI Gym Interface for Environments in Amazon SageMaker RL ........................................ 397
Use Open Source Environments ........................................................................................................ 397
Use Commercial Environments ........................................................................................................ 397
Distributed Training with Amazon SageMaker RL ........................................................................... 397
Hyperparameter Tuning with Amazon SageMaker RL ...................................................................... 398
Authentication and Access Control ............................................................................................... 399
Authentication ............................................................................................................................... 399
Access Control ............................................................................................................................ 400
Overview of Managing Access ...................................................................................................... 400
Amazon SageMaker Resources and Operations ............................................................................. 401
Understanding Resource Ownership ............................................................................................... 401
Managing Access to Resources ....................................................................................................... 402
Specifying Policy Elements: Resources, Actions, Effects, and Principals ...................................... 403
Specifying Conditions in a Policy ................................................................................................. 404
Using Identity-Based Policies (IAM Policies) ................................................................................ 404
Permissions Required to Use the Amazon SageMaker Console .................................................... 405
Permissions Required to Use the Amazon SageMaker Ground Truth Console ............................... 407
AWS Managed (Predefined) Policies for Amazon SageMaker ....................................................... 408
Control Access to Amazon SageMaker Resources by Using Tags ................................................ 408
Amazon SageMaker API Permissions Reference ........................................................................... 411
Amazon SageMaker Roles .............................................................................................................. 415
CreateNotebookInstance API: Execution Role Permissions ............................................................ 416
CreateHyperParameterTuningJob API: Execution Role Permissions ............................................. 419
CreateTrainingJob API: Execution Role Permissions ...................................................................... 420
CreateModel API: Execution Role Permissions ............................................................................. 422
Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role .......................................................................................................................... 424
Monitoring ................................................................................................................................... 425
Monitoring with CloudWatch ......................................................................................................... 425
Logging with CloudWatch .............................................................................................................. 429
Log Amazon SageMaker API Calls with AWS CloudTrail ............................................................. 430
Amazon SageMaker Information in CloudTrail ............................................................................. 430
Operations Performed by Automatic Model Tuning ..................................................................... 431
Understanding Amazon SageMaker Log File Entries ................................................................... 431
Security ........................................................................................................................................ 433
Notebook Instance Security ......................................................................................................... 433
Notebook Instances Are Internet-Enabled by Default .................................................................. 433
Connect to Amazon SageMaker Through a VPC Interface Endpoint ......................................... 434
Protect Training Jobs by Using an Amazon Virtual Private Cloud .............................................. 435
Configure a Training Job for Amazon VPC Access ....................................................................... 435
Configure Your Private VPC for Amazon SageMaker Training .................................................... 436
Protect Communications Between ML Compute Instances in a Distributed Training Job .......... 437
Using Inter-Container Traffic Encryption ..................................................................................... 438
Protect Endpoints by Using an Amazon Virtual Private Cloud .................................................... 438
Configure a Model for Amazon VPC Access .................................................................................. 438
Configure Your Private VPC for Amazon SageMaker Hosting ....................................................... 439
Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud ...................... 441
Configure a Batch Transform Job for Amazon VPC Access .......................................................... 441
Configure Your Private VPC for Amazon SageMaker Batch Transform ........................................ 441
Training and Inference Containers Run in Internet-Free Mode ...................................................... 443
Amazon SageMaker Ground Truth Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities ........................................................... 444
Amazon SageMaker Ground Truth .............................................................................................. 445
Are You a First-time User of Ground Truth? ................................................................................ 445
Getting started .............................................................................................................................. 446
Step 1: Before You Begin .............................................................................................................. 446
Step 2: Create a Labeling Job ....................................................................................................... 447
What Is Amazon SageMaker?

Amazon SageMaker is a fully managed machine learning service. With Amazon SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, Amazon SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a single click from the Amazon SageMaker console. Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

This is a HIPAA Eligible Service. For more information about AWS, U.S. Health Insurance Portability and Accountability Act of 1996 (HIPAA), and using AWS services to process, store, and transmit protected health information (PHI), see HIPAA Overview.

Are You a First-time User of Amazon SageMaker?

If you are a first-time user of Amazon SageMaker, we recommend that you do the following:

1. Read How Amazon SageMaker Works (p. 2) – This section provides an overview of Amazon SageMaker, explains key concepts, and describes the core components involved in building AI solutions with Amazon SageMaker. We recommend that you read this topic in the order presented.

2. Read Get Started (p. 17) – This section explains how to set up your account and create your first Amazon SageMaker notebook instance.

3. Try a model training exercise – This exercise walks you through training your first model. You use training algorithms provided by Amazon SageMaker. For more information, see Step 2: Train a Model with a Built-in Algorithm and Deploy It (p. 22).

4. Explore other topics – Depending on your needs, do the following:
   • Submit Python code to train with deep learning frameworks – In Amazon SageMaker, you can use your own training scripts to train models. For information, see Use Machine Learning Frameworks with Amazon SageMaker (p. 381).
   • Use Amazon SageMaker directly from Apache Spark – For information, see Use Apache Spark with Amazon SageMaker (p. 381).
   • Use Amazon AI to train and/or deploy your own custom algorithms – Package your custom algorithms with Docker so you can train and/or deploy them in Amazon SageMaker. See Use Your Own Algorithms with Amazon SageMaker (p. 255) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.

5. See the API Reference (p. 506) – This section describes the Amazon SageMaker API operations.
How Amazon SageMaker Works

Amazon SageMaker is a fully managed service that enables you to quickly and easily integrate machine learning-based models into your applications. This section provides an overview of machine learning and explains how Amazon SageMaker works. If you are a first-time user of Amazon SageMaker, we recommend that you read the following sections in order:

Topics
- Machine Learning with Amazon SageMaker (p. 2)
- Explore and Preprocess Data (p. 4)
- Train a Model with Amazon SageMaker (p. 4)
- Deploy a Model in Amazon SageMaker (p. 7)
- Validate a Machine Learning Model (p. 12)
- The Amazon SageMaker Programming Model (p. 13)

How It Works: Next Topic

Machine Learning with Amazon SageMaker (p. 2)

Machine Learning with Amazon SageMaker

This section describes a typical machine learning workflow and summarizes how you accomplish those tasks with Amazon SageMaker.

In machine learning, you “teach” a computer to make predictions, or inferences. First, you use an algorithm and example data to train a model. Then you integrate your model into your application to generate inferences in real time and at scale. In a production environment, a model typically learns from millions of example data items and produces inferences in hundreds to less than 20 milliseconds.

The following diagram illustrates the typical workflow for creating a machine learning model:
As the diagram illustrates, you typically perform the following activities:

1. **Generate example data**—To train a model, you need example data. The type of data that you need depends on the business problem that you want the model to solve (the inferences that you want the model to generate). For example, suppose that you want to create a model to predict a number given an input image of a handwritten digit. To train such a model, you need example images of handwritten numbers.

   Data scientists often spend a lot of time exploring and preprocessing, or "wrangling," example data before using it for model training. To preprocess data, you typically do the following:

   a. **Fetch the data**—You might have in-house example data repositories, or you might use datasets that are publicly available. Typically, you pull the dataset or datasets into a single repository.

   b. **Clean the data**—To improve model training, inspect the data and clean it up as needed. For example, if your data has a *country name* attribute with values *United States* and *US*, you might want to edit the data to be consistent.

   c. **Prepare or transform the data**—To improve performance, you might perform additional data transformations. For example, you might choose to combine attributes. If your model predicts the conditions that require de-icing an aircraft instead of using temperature and humidity attributes separately, you might combine those attributes into a new attribute to get a better model.

   In Amazon SageMaker, you preprocess example data in a Jupyter notebook on your notebook instance. You use your notebook to fetch your dataset, explore it and prepare it for model training. For more information, see Explore and Preprocess Data (p. 4). For more information about preparing data in AWS Marketplace, see data preparation.

2. **Train a model**—Model training includes both training and evaluating the model, as follows:

   a. **Training the model**—To train a model, you need an algorithm. The algorithm you choose depends on a number of factors. For a quick, out-of-the-box solution, you might be able to use one of the algorithms that Amazon SageMaker provides. For a list of algorithms provided by Amazon SageMaker and related considerations, see Use Amazon SageMaker Built-in Algorithms (p. 56).

   You also need compute resources for training. Depending on the size of your training dataset and how quickly you need the results, you can use resources ranging from a single, small general-purpose instance to a distributed cluster of GPU instances. For more information, see Train a Model with Amazon SageMaker (p. 4).

   b. **Evaluating the model**—After you've trained your model, you evaluate it to determine whether the accuracy of the inferences is acceptable. In Amazon SageMaker, you use either the AWS SDK for Python (Boto) or the high-level Python library that Amazon SageMaker provides to send requests to the model for inferences.

   You use a Jupyter notebook in your Amazon SageMaker notebook instance to train and evaluate your model.

3. **Deploy the model**—You traditionally re-engineer a model before you integrate it with your application and deploy it. With Amazon SageMaker hosting services, you can deploy your model independently, decoupling it from your application code. For more information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 7).

   Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, then collect "ground truth," and evaluate the model to identify drift. You then increase the accuracy of your inferences by updating your training data to include the newly collected ground truth, by retraining the model with the new dataset. As more and more example data becomes available, you continue retraining your model to increase accuracy.
Explore and Preprocess Data

Before using a dataset to train a model, data scientists typically explore and preprocess it. For example, in one of the exercises in this guide, you use the MNIST dataset, a commonly available dataset of handwritten numbers, for model training. Before you begin training, you transform the data into a format that is more efficient for training. For more information, see Step 2.2.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 25).

To preprocess data use one of the following methods:

- Use a Jupyter notebook on an Amazon SageMaker notebook instance. You can also use the notebook instance to write code to create model training jobs, deploy models to Amazon SageMaker hosting, and test or validate your models. For more information, see Use Notebook Instances (p. 42)
- You can use a model to transform data by using Amazon SageMaker batch transform. For more information, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).

Train a Model with Amazon SageMaker

The following diagram shows how you train and deploy a model with Amazon SageMaker:
The area labeled Amazon SageMaker highlights the two components of Amazon SageMaker: model training and model deployment.

To train a model in Amazon SageMaker, you create a training job. The training job includes the following information:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 59).

You have the following options for a training algorithm:

- **Use an algorithm provided by Amazon SageMaker**—Amazon SageMaker provides training algorithms. If one of these meets your needs, it's a great out-of-the-box solution for quick model
training. For a list of algorithms provided by Amazon SageMaker, see Use Amazon SageMaker Built-in Algorithms (p. 56). To try an exercise that uses an algorithm provided by Amazon SageMaker, see Get Started (p. 17).

- **Use Apache Spark with Amazon SageMaker**—Amazon SageMaker provides a library that you can use in Apache Spark to train models with Amazon SageMaker. Using the library provided by Amazon SageMaker is similar to using Apache Spark MLLib. For more information, see Use Apache Spark with Amazon SageMaker (p. 381).

- **Submit custom code to train with deep learning frameworks**—You can submit custom Python code that uses TensorFlow or Apache MXNet for model training. For more information, see Use TensorFlow with Amazon SageMaker (p. 390) and Use Apache MXNet with Amazon SageMaker (p. 390).

- **Use your own custom algorithms**—Put your code together as a Docker image and specify the registry path of the image in an Amazon SageMaker CreateTrainingJob API call. For more information, see Use Your Own Algorithms with Amazon SageMaker (p. 255).

- **Use an algorithm that you subscribe to from AWS Marketplace**—For information, see Find and Subscribe to Algorithms and Model Packages on AWS Marketplace (p. 374).

After you create the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

You can create a training job with the Amazon SageMaker console or the API. For information about creating a training job with the API, see the CreateTrainingJob (p. 556) API.

When you create a training job with the API, Amazon SageMaker replicates the entire dataset on ML compute instances by default. To make Amazon SageMaker replicate a subset of the data on each ML compute instance, you must set the S3DataDistributionType field to ShardedByS3Key. You can set this field using the low-level SDK. For more information, see S3DataDistributionType in S3DataSource (p. 869).

**Important**

To prevent your algorithm container from contending for memory, you should reserve some memory for Amazon SageMaker critical system processes on your ML compute instances. If the algorithm container is allowed to use memory needed for system processes, it can trigger a system failure.
How It Works: Next Topic

Deploy a Model in Amazon SageMaker (p. 7)

Deploy a Model in Amazon SageMaker

After you train your model, you can deploy it to get predictions in one of two ways:

• To set up a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
• To get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics
• Deploy a Model on Amazon SageMaker Hosting Services (p. 7)
• Get Inferences for an Entire Dataset with Batch Transform (p. 10)

Deploy a Model on Amazon SageMaker Hosting Services

Amazon SageMaker also provides model hosting services for model deployment, as shown in the following diagram. Amazon SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences.
Deploying a model using Amazon SageMaker hosting services is a three-step process:

1. **Create a model in Amazon SageMaker**—By creating a model, you tell Amazon SageMaker where it can find the model components. This includes the S3 path where the model artifacts are stored and the Docker registry path for the image that contains the inference code. In subsequent deployment steps, you specify the model by name. For more information, see the `CreateModel` (p. 538) API.

2. **Create an endpoint configuration for an HTTPS endpoint**—You specify the name of one or more models in production variants and the ML compute instances that you want Amazon SageMaker to launch to host them.

   When hosting models in production, you can configure the endpoint to elastically scale the deployed ML compute instances. For each production variant, you specify the number of ML compute instances that you want to deploy. When you specify two or more instances, Amazon SageMaker launches them in multiple Availability Zones. This ensures continuous availability. Amazon SageMaker manages deploying the instances. For more information, see the `CreateEndpointConfig` (p. 525) API.

3. **Create an HTTPS endpoint**—Provide the endpoint configuration to Amazon SageMaker. The service launches the ML compute instances and deploys the model or models as specified in the configuration. For more information, see the `CreateEndpoint` (p. 522) API. To get inferences from
the model, client applications send requests to the Amazon SageMakerRuntime HTTPS endpoint. For more information about the API, see the InvokeEndpoint (p. 735) API.

**Note**
When you create an endpoint, Amazon SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. For a list of instance types that Amazon SageMaker hosting service supports, see AWS Service Limits. For a list of the sizes of the storage volumes that Amazon SageMaker attaches to each instance, see Amazon SageMaker ML Instance Types.

To increase a model's accuracy, you might choose to save the user's input data and ground truth, if available, as part of the training data. You can then retrain the model periodically with a larger, improved training dataset.

**Best Practices for Deploying Models on Amazon SageMaker Hosting Services**

When hosting models using Amazon SageMaker hosting services, consider the following:

- Typically, a client application sends requests to the Amazon SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.

- You can deploy a model trained with Amazon SageMaker to your own deployment target. To do that, you need to know the algorithm-specific format of the model artifacts that were generated by model training. For more information about output formats, see the section corresponding to the algorithm you are using in Training Data Formats (p. 64).

- You can deploy multiple variants of a model to the same Amazon SageMaker HTTPS endpoint. This is useful for testing variations of a model in production. For example, suppose that you've deployed a model into production. You want to test a variation of the model by directing a small amount of traffic, say 5%, to the new model. To do this, create an endpoint configuration that describes both variants of the model. You specify the ProductionVariant in your request to the CreateEndPointConfig. For more information, see ProductionVariant (p. 856).

- You can configure a ProductionVariant to use Application Auto Scaling. For information about configuring automatic scaling, see Automatically Scale Amazon SageMaker Models (p. 356).

- You can modify an endpoint without taking models that are already deployed into production out of service. For example, you can add new model variants, update the ML Compute instance configurations of existing model variants, or change the distribution of traffic among model variants. To modify an endpoint, you provide a new endpoint configuration. Amazon SageMaker implements the changes without any downtime. For more information see, UpdateEndpoint (p. 722) and UpdateEndpointWeightsAndCapacities (p. 724).

- Changing or deleting model artifacts or changing inference code after deploying a model produces unpredictable results. If you need to change or delete model artifacts or change inference code, modify the endpoint by providing a new endpoint configuration. Once you provide the new endpoint configuration, you can change or delete the model artifacts corresponding to the old endpoint configuration.
If you want to get inferences on entire datasets, consider using batch transform as an alternative to hosting services. For information see Get Inferences for an Entire Dataset with Batch Transform (p. 10)

How It Works: Next Topic

Validate a Machine Learning Model (p. 12)

Get Inferences for an Entire Dataset with Batch Transform

Get inferences for an entire dataset by using Amazon SageMaker batch transform. Batch transform uses a trained model to get inferences on a dataset that is stored in Amazon S3, and saves the inferences in an S3 bucket that you specify when you create a batch transform job. Batch transform manages all compute resources necessary to get inferences. This includes launching instances and deleting them after the transform job completes.

To perform a batch transform, create a transform job, which includes the following information:

- The path to the S3 bucket where you’ve stored the data to transform.
- The compute resources that you want Amazon SageMaker to use for the transform job. Compute resources are ML compute instances that are managed by Amazon SageMaker.
- The path to the S3 bucket where you want to store the output of the job.
- The name of the model that you want to use in the transform job.

For examples on how to use batch transform, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).

Batch transform is ideal for situations where:

- You want to get inferences for an entire dataset and store them online.
- You don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences.
- You don't need the sub-second latency that Amazon SageMaker hosted endpoints provide.
- You want to preprocess your data before using the data to train a new model or generate inferences.

The following diagram illustrates the workflow of a batch transform job:
Best Practices for Using Batch Transform

- You can create a transform job by using the Amazon SageMaker console or the API. For more information, see CreateTransformJob (p. 562) API.

- After you create a transform job, Amazon SageMaker launches the ML compute instances and uses the model you specify to transform the input data. It stores the results and other output in the S3 bucket that you specified.

- Amazon SageMaker uses Multipart Upload API to upload output data results from a transform job to S3. Typically all multipart uploads run to completion. In case of an error, the multipart upload is stopped and then removed from S3. However there are cases such as a network outage where a multipart upload remains incomplete in S3. To avoid extra storage charges in these cases, we recommend that you add the S3 bucket policy to remove incomplete multipart uploads that are stored in the output S3 buckets.

- For testing model variants, create separate transform jobs for each variant using a validation data set. You can then analyze the results using metrics. Make sure you specify a different ModelName and a unique S3OutputPath location for each transform job.

- For large datasets or data of indeterminate size, such as a video stream, you can create an infinite stream. An infinite stream is when a transform Job is set to continuously input and transform data. The transform job completes either when all the data is transformed or when it is instructed to stop. This approach works only with models from supported algorithms. To create an infinite stream using the API, in the CreateTransformJob request, set SplitType to None and set MaxPayloadInMB to 0. To create an infinite stream using the console, choose None for Split type and set Max payload size (MB) to 0. Amazon SageMaker interprets a MaxPayloadInMB of 0 as no limit on the payload size.
How It Works: Next Topic

Validate a Machine Learning Model (p. 12)

Validate a Machine Learning Model

After training a model, evaluate it to determine whether its performance and accuracy allow you to achieve your business goals. You might generate multiple models using different methods and evaluate each. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You might consider whether your model needs to be more sensitive than specific (or vice versa).

You can evaluate your model using historical data (offline) or live data:

- **Offline testing**—Use historical, not live, data to send requests to the model for inferences.

Deploy your trained model to an alpha endpoint, and use historical data to send inference requests to it. To send the requests, use a Jupyter notebook in your Amazon SageMaker notebook instance and either the AWS SDK for Python (Boto) or the high-level Python library provided by Amazon SageMaker.

- **Online testing with live data**—Amazon SageMaker supports deploying multiple models (called production variants) to a single Amazon SageMaker endpoint. You configure the production variants so that a small portion of the live traffic goes to the model that you want to validate. For example, you might choose to send 10% of the traffic to a model variant for evaluation. After you are satisfied with the model's performance, you can route 100% traffic to the updated model.

For more information, see articles and books about how to evaluate models, for example, Evaluating Machine Learning Models.

Options for offline model evaluation include:

- **Validating using a "holdout set"**—Machine learning practitioners often set aside a part of the data as a "holdout set." They don't use this data for model training.

  With this approach, you evaluate how well your model provides inferences on the holdout set. You then assess how effectively the model generalizes what it learned in the initial training, as opposed to using model "memory." This approach to validation gives you an idea of how often the model is able to infer the correct answer.

  In some ways, this approach is similar to teaching elementary school students. First, you provide them with a set of examples to learn, and then test their ability to generalize from their learning. With homework and tests, you pose problems that were not included in the initial learning and determine whether they are able to generalize effectively. Students with perfect memories could memorize the problems, instead of learning the rules.

  Typically, the holdout dataset is of 20-30% of the training data.
• **k-fold validation**—In this validation approach, you split the example dataset into \( k \) parts. You treat each of these parts as a holdout set for \( k \) training runs, and use the other \( k-1 \) parts as the training set for that run. You produce \( k \) models using a similar process, and aggregate the models to generate your final model. The value \( k \) is typically in the range of 5-10.

**How It Works: Next Topic**

*The Amazon SageMaker Programming Model* (p. 13)

**The Amazon SageMaker Programming Model**

Amazon SageMaker provides APIs that you can use to create and manage notebook instances and train and deploy models. For more information, see *API Reference* (p. 506).

Making API calls directly from code is cumbersome, and requires you to write code to authenticate your requests. Amazon SageMaker provides the following alternatives:

- **Use the Amazon SageMaker console**—With the console, you don’t write any code. You use the console UI to start model training or deploy a model. The console works well for simple jobs, where you use a built-in training algorithm and you don’t need to preprocess training data.

- **Modify the example Jupyter notebooks**—Amazon SageMaker provides several Jupyter notebooks that train and deploy models using specific algorithms and datasets. Start with a notebook that has a suitable algorithm and modify it to accommodate your data source and specific needs.

- **Write model training and inference code from scratch**—Amazon SageMaker provides both an AWS SDK and a high-level Python library that you can use in your code to start model training jobs and deploy the resulting models.

- **The high-level Python library**—The Python library simplifies model training and deployment. In addition to authenticating your requests, the library abstracts platform specifics by providing simple methods and default parameters. For example:

  - To deploy your model, you call only the `deploy()` method. The method creates an Amazon SageMaker model, an endpoint configuration, and an endpoint.

  - If you use a custom framework script for model training, you call the `fit()` method. The method creates a .gzip file of your script, uploads it to an Amazon S3 location, and then runs it for model training, and other tasks. For more information, see *Use Machine Learning Frameworks with Amazon SageMaker* (p. 381).

  - **The AWS SDK**—The SDKs provide methods that correspond to the Amazon SageMaker API (see *Actions* (p. 506)). Use the SDKs to programmatically start a model training job and host the model in Amazon SageMaker. SDK clients authenticate your requests by using your access keys, so you don’t need to write authentication code. They are available in multiple languages and platforms. For more information, see *SDKs*. 

13
In Get Started (p. 17), you train and deploy a model using an algorithm provided by Amazon SageMaker. That exercise shows how to use both of these libraries. For more information, see Get Started (p. 17).

- **Integrate Amazon SageMaker into your Apache Spark workflow**—Amazon SageMaker provides a library for calling its APIs from Apache Spark. With it, you can use Amazon SageMaker-based estimators in an Apache Spark pipeline. For more information, see Use Apache Spark with Amazon SageMaker (p. 381).

How It Works: Next Topic

Get Started (p. 17)
Set Up Amazon SageMaker

In this section, you sign up for an AWS account and then create an IAM user, a security group, and create an Amazon S3 bucket.

If you’re new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 2).

Topics
• Step 1: Create an AWS Account (p. 15)
• Step 2: Create an IAM Administrator User and Group (p. 15)
• Step 3: Create an Amazon S3 Bucket (p. 16)

Step 1: Create an AWS Account

In this section, you sign up for an AWS account. If you already have an AWS account, skip this step.

When you sign up for Amazon Web Services (AWS), your AWS account is automatically signed up for all AWS services, including Amazon SageMaker. You are charged only for the services that you use.

To create an AWS account

1. Open https://aws.amazon.com/, and then choose Create an AWS Account.

   Note
   If you previously signed in to the AWS Management Console using AWS account root user credentials, choose Sign in to a different account. If you previously signed in to the console using IAM credentials, choose Sign-in using root account credentials. Then choose Create a new AWS account.

2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code using the phone keypad.

   Write down your AWS account ID because you’ll need it for the next task.

Step 2: Create an IAM Administrator User and Group

When you create an AWS account, you get a single sign-in identity that has complete access to all of the AWS services and resources in the account. This identity is called the AWS account root user. Signing in to the AWS console using the email address and password that you used to create the account gives you complete access to all of the AWS resources in your account.

We strongly recommend that you not use the root user for everyday tasks, even the administrative ones. Instead, adhere to the Create Individual IAM Users, an AWS Identity and Access Management (IAM) administrator user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
To create an administrator user and sign in to the console

1. Create an administrator user in your AWS account. For instructions, see Creating Your First IAM User and Administrators Group in the IAM User Guide.
   
   **Note**
   We assume that you use administrator user credentials for the exercises and procedures in this guide. If you choose to create and use another IAM user, grant that user minimum permissions. For more information, see Authentication and Access Control for Amazon SageMaker (p. 399).

2. Sign in to the AWS Management Console.
   
   To sign in to the AWS console as a IAM user, you must use a special URL. For more information, see How Users Sign In to Your Account in the IAM User Guide.

Next Step

Step 3: Create an Amazon S3 Bucket (p. 16)

Step 3: Create an Amazon S3 Bucket

In this section, you create an Amazon S3 bucket. You use this bucket to store training data and the results of model training, called model artifacts.

- The model training data
- Model artifacts, which Amazon SageMaker generates during model training

You can store the training data and artifacts in a single bucket or in two separate buckets. For exercises in this guide, one bucket is sufficient. You can use existing buckets or create new ones.

Follow the instructions in Create a Bucket in the Amazon Simple Storage Service Console User Guide. Include sagemaker in the bucket name; for example, sagemaker-datetime.

**Note**
Amazon SageMaker needs permission to access this bucket. You grant permission with an IAM role, which you create in the next step (as part of creating an Amazon SageMaker notebook instance). This IAM role automatically gets permissions to access any bucket with sagemaker in the name through the AmazonSageMakerFullAccess policy that Amazon SageMaker attaches to the role.

Now you are ready to begin the Step 1: Create an Amazon SageMaker Notebook Instance (p. 17) topic that walks you through procedures for creating a notebook, training and deploying a machine learning Amazon SageMaker model.
Get Started

In this section, you create your first Amazon SageMaker notebook instance, and train a model. You train the model using an algorithm provided by Amazon SageMaker, deploy it, and validate it by sending inference requests to the model's endpoint.

You use this notebook instance for all of the exercises in this guide.

If you're new to Amazon SageMaker, we recommend that you read How Amazon SageMaker Works (p. 2) before creating your notebook instance. For general information about notebook instances, see Explore and Preprocess Data (p. 4).

Topics
• Step 1: Create an Amazon SageMaker Notebook Instance (p. 17)
• Step 2: Train a Model with a Built-in Algorithm and Deploy It (p. 22)
• Step 3: Clean up (p. 40)
• Step 4: Integrating Amazon SageMaker Endpoints into Internet-facing Applications (p. 40)

Step 1: Create an Amazon SageMaker Notebook Instance

An Amazon SageMaker notebook instance is a fully managed machine learning (ML) EC2 compute instance running the Jupyter Notebook App. For more information, see Explore and Preprocess Data (p. 4).

Note
If necessary, you can change the notebook instance settings, including the ML compute instance type, later.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebook instances, then choose Create notebook instance.
3. On the Create notebook instance page, provide the following information:
   a. For Notebook instance name, type ExampleNotebookInstance.
   b. For Instance type, choose ml.t2.medium.
   c. For IAM role, create an IAM role.
      i. Choose Create a new role.
ii. (Optional) If you want to use S3 buckets other than the one you created in Step 1 of this tutorial to store your input data and output, choose them.

In Step 1 of this tutorial, you created an S3 bucket with `sagemaker` in its name. This IAM role automatically has permissions to use that bucket. The `AmazonSageMakerFullAccess` policy, which Amazon SageMaker attaches to the role, gives the role those permissions.

The bucket that you created in Step 1 is sufficient for the model training exercise in Getting Started. However, as you explore Amazon SageMaker, you might want to access other S3 buckets from your notebook instance. Give Amazon SageMaker permissions to access those buckets.

**To access more S3 buckets from your Amazon SageMaker notebook instance**

A. If you're not concerned about users in your AWS account accessing your data, choose **Any S3 bucket**.

B. If your account has sensitive data (such as Human Resources information), restrict access by choosing **Specific S3 buckets**. You can update the permissions policy attached to the role you are creating later.

To explicitly control access, Restrict access by choosing **None**, use bucket and object names and tags as supported by the `AmazonSageMakerFullAccess` policy. For more information, see Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role (p. 424).

iii. Choose **Create role**.

Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDHHmmSS`. For example, `AmazonSageMaker-ExecutionRole-20171125T090800`.

To see the policies that are attached to the role, use the IAM console.
Open the IAM console at https://console.aws.amazon.com/iam/.
The following policies are attached to the role:

- A trust policy that allows Amazon SageMaker to assume the role.
- The AmazonSageMakerFullAccess AWS managed policy.
- If you specified access to additional S3 bucket(s) when creating this role, the customer managed policy attached to the role. The name of the customer managed policy is AmazonSageMaker-ExecutionPolicy-YYYYMMDDHHmmSS.

For more information about creating your own IAM role, see Amazon SageMaker Roles (p. 415).

d. (Optional) Choose to access resources in a Virtual Private Cloud (VPC).

To access resources in your VPC from the notebook instance

i. Choose the VPC and a SubnetId.

ii. For Security Group, choose your VPCs default security group. For the exercises in this guide, the inbound and outbound rules of the default security group are sufficient.

iii. To enable connecting to a resource in your VPC, ensure that the resource resolves to a private IP address in your VPC. For example, to ensure that an Amazon Redshift DNS name resolves to a private IP address, do one of the following:
   - Ensure that the Amazon Redshift cluster is not publicly accessible.
   - If the Amazon Redshift cluster is publicly accessible, set the DNS resolution and DNS hostnames VPC parameters to true. For more information, see Managing Clusters in an Amazon Virtual Private Cloud (VPC)

e. If you chose to access resources from your VPC, enable direct internet access. For Direct internet access, choose Enable. Otherwise, this notebook instance won't have internet access. Without internet access, you can't train or host models from notebooks on this notebook instance unless your VPC has a NAT gateway and your security group allows outbound connections. For more information, see Notebook Instances Are Internet-Enabled by Default (p. 433).

f. (Optional) To use shell scripts that run when you create or start the instance, specify a lifecycle configuration. For information, see Step 1.1: (Optional) Customize a Notebook Instance (p. 20)

g. (Optional) If you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.

h. Specify the size, in GB, of the ML storage volume that is attached to the notebook instance. You can choose a size between 5 GB and 16384 GB, in 1 GB increments.

i. (Optional) To associate git repositories with the notebook instance, choose a default repository and up to 3 additional repositories. For more information, see Associate Git Repositories with Amazon SageMaker Notebook Instances (p. 49).

j. Choose Create notebook instance.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the CreateNotebookInstance (p. 546) API.

4. When the status of the notebook instance is InService, choose Open next to its name to open the Jupyter dashboard.
Step 1.1: (Optional) Customize a Notebook Instance

The dashboard provides access to:

- A tab that contains sample notebooks. To use a sample notebook, on the SageMaker Examples tab, choose the sample notebook you want to use, and choose use. For information about the sample notebooks, see the Amazon SageMaker GitHub repository.
- The kernels for Jupyter, including those that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To choose a kernel for your notebook instance, use the New menu.

For more information, see The Jupyter notebook.

To install packages or sample notebooks on your notebook instance, configure networking and security for it, or otherwise use a shell script to customize it, use a lifecycle configuration. A lifecycle configuration provides shell scripts that run only when you create the notebook instance or whenever you start one. When you create a notebook instance, you can create a new lifecycle configuration and the scripts it uses or apply one that you already have.

Note
Each script has a limit of 16384 characters.
The value of the $PATH environment variable that is available to both scripts is /sbin:/bin:/usr/sbin:/usr/bin.
View CloudWatch Logs for notebook instance lifecycle configurations in log group /aws/sagemaker/NotebookInstances in log stream [notebook-instance-name]/[LifecycleConfigHook].
Scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

To create a lifecycle configuration
1. For Lifecycle configuration - Optional, choose Create a new lifecycle configuration.
2. For Name, type a name.
3. (Optional) To create a script that runs when you create the notebook and every time you start it, choose Start notebook.
4. In the Start notebook editor, type the script.
5. (Optional) To create a script that runs only once, when you create the notebook, choose Create notebook.
6. In the Create notebook editor, type the script configure networking.
7. Choose Create configuration.

You can see a list of notebook instance lifecycle configurations you previously created by choosing Lifecycle configuration in the Amazon SageMaker console. From there, you can view, edit, delete existing lifecycle configurations. You can create a new notebook instance lifecycle configuration by

20
choosing Create configuration. These notebook instance lifecycle configurations are available when you create a new notebook instance.

**Lifecycle Configuration Best Practices**

The following are best practices for using lifecycle configurations:

- Lifecycle configurations run as the root user. If your script makes any changes within the /home/ec2-user/SageMaker directory, for example, installing a package with pip, use the command `sudo -u ec2-user` command to run as the `ec2-user` user. This is the same user that Amazon SageMaker runs as.

- Amazon SageMaker notebook instances use conda environments to implement different kernels for Jupyter notebooks. If you want to install packages that are available to one or more notebook kernels, enclose the commands to install the packages with conda environment commands that activate the conda environment that contains the kernel where you want to install the packages.

For example, if you want to install a package only in the python3 environment, use the following code:

```bash
sudo -u ec2-user -i <<'EOF'
# This will affect only the Jupyter kernel called "conda_python3".
source activate python3

# Replace myPackage with the name of the package you want to install.
# You can also perform "conda install" here as well.

source deactivate
EOF
```

If you want to install a package in all conda environments in the notebook instance, use the following code:

```bash
sudo -u ec2-user -i <<'EOF'

# Note that "base" is special environment name, include it there as well.
for env in base /home/ec2-user/anaconda3/envs/*; do
    source /home/ec2-user/anaconda3/bin/activate $(basename "$env")

    # Installing packages in the Jupyter system environment can affect stability of your
    # Notebook Instance. You can remove this check if you'd like to install Jupyter
    # extensions, etc.
    if [ $env != 'JupyterSystemEnv'] ; then
        continue
    fi

    # Replace myPackage with the name of the package you want to install.
    # You can also perform "conda install" here as well.

    source /home/ec2-user/anaconda3/bin/deactivate
done
EOF
```
You are now ready to train your first model. For step-by-step instructions, see Step 2: Train a Model with a Built-in Algorithm and Deploy It (p. 22).

Step 2: Train a Model with a Built-in Algorithm and Deploy It

Now train and deploy your first machine learning model with Amazon SageMaker. For model training, you use the following:

- The MNIST dataset of images of handwritten, single digit numbers—This dataset provides 60,000 example images of handwritten single-digit numbers and a test dataset of 10,000 images. You provide this dataset to the k-means algorithm for model training. For more information, see MNIST Dataset.
- A built-in algorithm—You use the k-means algorithm provided by Amazon SageMaker. K-means is a clustering algorithm. During model training, the algorithm groups the example data of handwritten numbers into 10 clusters (one for each number, 0 through 9). For more information about the algorithm, see K-Means Algorithm (p. 137).

In this exercise, you do the following:

1. Download the MNIST dataset to your Amazon SageMaker notebook instance, then review the data and preprocess it. For efficient training, you convert the dataset from the `numpy.array` format to the `RecordIO protobuf` format. A `numpy.array` is an n-dimensional array object that the NumPy scientific computing library uses, and `RecordIO protobuf` is a binary data format that the Amazon SageMaker K-Means algorithm expects as input.
2. Start an Amazon SageMaker training job.
3. Deploy the model in Amazon SageMaker.
4. Validate the model by sending inference requests to the model's endpoint. You send images of handwritten, single-digit numbers. The model returns the number of the cluster (0 through 9) that the images belong to.

Important

For model training, deployment, and validation, you can use either of the following:

- The high-level Python library provided by Amazon SageMaker
- The AWS SDK for Python (Boto)

The high-level library abstracts several implementation details, and is easy to use. This exercise provides separate code examples using both libraries. If you're a first-time Amazon SageMaker user, we recommend that you use the high-level Python library. For more information, see The Amazon SageMaker Programming Model (p. 13).

There are two ways to use this exercise:
Follow the steps to create, deploy, and validate the model. You create a Jupyter notebook in your Amazon SageMaker notebook instance, and copy code, paste it into the notebook, and run it.

If you're familiar with using sample notebooks, open and run the following example notebooks that Amazon SageMaker provides in the SageMaker Python SDK section of the SageMaker Examples tab of your notebook instance:

- kmeans_mnist.ipynb
- kmeans_mnist_lowlevel.ipynb

**Topics**
- Step 2.1: Create a Jupyter Notebook and Initialize Variables (p. 23)
- Step 2.2: Download, Explore, and Transform the Training Data (p. 24)
- Step 2.3: Train a Model (p. 26)
- Step 2.4: Deploy the Model to Amazon SageMaker (p. 29)
- Step 2.5: Validate the Model (p. 36)

**Step 2.1: Create a Jupyter Notebook and Initialize Variables**

In this section, you create a Jupyter notebook in your Amazon SageMaker notebook instance and initialize variables.

To create a Jupyter notebook:

1. Create the notebook.
   b. Open the notebook instance, by choosing Open next to its name. The Jupyter notebook server page appears:

   ![Jupyter Notebook Server Page](image)

   c. To create a notebook, in the Files tab, choose New, and conda_python3. This pre-installed environment includes the default Anaconda installation and Python 3.
   d. In the Jupyter notebook, under File, choose Save as, and name the notebook.

2. Copy the following Python code and paste it into your notebook. Add the name of the S3 bucket that you created in Set Up Amazon SageMaker (p. 15), and run the code. The get_execution_role function retrieves the IAM role you created at the time of creating your notebook instance.
Step 2.2: Download, Explore, and Transform Data

Now download the MNIST dataset to your notebook instance. Then review the data, transform it, and upload it to your S3 bucket.

You transform the data by changing its format from `numpy.array` to `RecordIO`. The `RecordIO` format is more efficient for the algorithms provided by Amazon SageMaker. For information about the `RecordIO` format, see Data Format.

Topics
- Step 2.2.1: Download the MNIST Dataset (p. 24)
- Step 2.2.2: Explore the Training Dataset (p. 25)
- Step 2.2.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 25)

Step 2.2.1: Download the MNIST Dataset

To download the MNIST dataset, copy and paste the following code into the notebook and run it:

```
%time
import pickle, gzip, numpy, urllib.request, json

# Load the dataset
urllib.request.urlretrieve("http://deeplearning.net/data/mnist/mnist.pkl.gz", "mnist.pkl.gz")
with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
```

The code does the following:

1. Downloads the MNIST dataset (`mnist.pkl.gz`) from the deeplearning.net website to your Amazon SageMaker notebook instance.
2. Unzips the file and reads the following three datasets into the notebook's memory:
   - `train_set`—You use these images of handwritten numbers to train a model.
   - `valid_set`—After you train the model, you validate it using the images in this dataset.
   - `test_set`—You don't use this dataset in this exercise.
Step 2.2.2: Explore the Training Dataset

Typically, you explore training data to determine what you need to clean up and which transformations to apply to improve model training. For this exercise, you don't need to clean up the MNIST dataset. Simply display one of the images in the train_set dataset.

```python
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (2,10)

def show_digit(img, caption='', subplot=None):
    if subplot == None:
        _, (subplot) = plt.subplots(1,1)
    imgr = img.reshape((28,28))
    subplot.axis('off')
    subplot.imshow(imgr, cmap='gray')
    plt.title(caption)
show_digit(train_set[0][30], 'This is a {}'.format(train_set[1][30]))
```

train_set contains the following data:

- train_set[0] contains images.

The code uses the matplotlib library to get and display the 31st image from the training dataset.

Next Step

Step 2.2.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 25)

Step 2.2.3: Transform the Training Dataset and Upload It to Amazon S3

For efficient model training, transform the dataset from the numpy.array format to the RecordIO protobuf format. The RecordIO protobuf format is more efficient for all of the algorithms provided by Amazon SageMaker.

**Important**

For this and subsequent steps, you can choose to use the high-level Python library provided by Amazon SageMaker or the low-level AWS SDK for Python (Boto). If you're a first-time user of Amazon SageMaker, we recommend that you follow the code examples for the high-level Python library.

To transform the dataset, choose one of the following options

- Use the high-level Python library provided by Amazon SageMaker
If you are using the high-level Python library, you skip this step and go to the next step. In the next section, you use the `fit` method for model training, which performs the necessary transformation and upload to S3 before starting a model training job.

- **Use the SDK for Python**

The following code first uses the high-level Python library function, `write_numpy_to_dense_tensor`, to convert the training data into the protobuf format, which is efficient for model training. Then the code uses the SDK for Python low-level API to upload data to S3.

```python
%%time
from sagemaker.amazon.common import write_numpy_to_dense_tensor
import io
import boto3

bucket = 'bucket-name' # Use the name of your s3 bucket here
data_key = 'kmeans_lowlevel_example/data'
data_location = 's3://{}/{}'.format(bucket, data_key)
print('training data will be uploaded to: {}'.format(data_location))

# Convert the training data into the format required by the SageMaker KMeans algorithm
buf = io.BytesIO()
write_numpy_to_dense_tensor(buf, train_set[0], train_set[1])
buf.seek(0)
boto3.resource('s3').Bucket(bucket).Object(data_key).upload_fileobj(buf)
```

**Next Step**

Step 2.3: Train a Model (p. 26)

**Step 2.3: Train a Model**

To start model training, you send a request to the CreateTrainingJob (p. 556) API. In the request, you specify the Amazon Elastic Container Registry path to the training image, the location of the S3 bucket containing your training data, and the resources to use (the type and number of ML compute instances to launch).

**Topics**
- Step 2.3.1: Choose the Training Algorithm (p. 26)
- Step 2.3.2: Create a Training Job (p. 26)

**Step 2.3.1: Choose the Training Algorithm**

To choose the right algorithm for your model, you typically follow an evaluation process. For this exercise, you use the k-means algorithm provided by Amazon SageMaker, so no evaluation is required. For information about choosing algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56).

**Next Step**

Step 2.3.2: Create a Training Job (p. 26)

**Step 2.3.2: Create a Training Job**

To train a model, Amazon SageMaker provides the CreateTrainingJob (p. 556) API. You provide the following information when making this API call:
• The training algorithm—Specify the registry path of the Docker image that contains the training code. For the registry paths for the algorithms provided by Amazon SageMaker, see Common Parameters for Built-In Algorithms (p. 59). In the following examples, when using the high-level Python library, you don't need to explicitly specify this path. The sagemaker.amazon.kmeans.KMeans object knows the path.

• Algorithm-specific hyperparameters—Specify algorithm-specific hyperparameters to influence the final quality of the model. For information, see K-Means Hyperparameters (p. 141).

• The input and output configuration—Provide the S3 bucket where training data is stored and where Amazon SageMaker saves the results of model training (the model artifacts).

The low-level AWS SDK for Python provides the corresponding create_training_job method and the high-level Python library provide the fit method.

To train the model, choose one of the following options.

• Use the high-level Python library provided by Amazon SageMaker.

This Python library provides the KMeans estimator, which is a class in the sagemaker.amazon.kmeans.KMeans module. To start model training, call the fit method.

1. Create an instance of the sagemaker.amazon.kmeans.KMeans class.

   ```python
   from sagemaker import KMeans
   
   data_location = 's3://{}/kmeans_highlevel_example/data'.format(bucket)
   output_location = 's3://{}/kmeans_highlevel_example/output'.format(bucket)
   
   print('training data will be uploaded to: {}'.format(data_location))
   print('training artifacts will be uploaded to: {}'.format(output_location))

   kmeans = KMeans(role=role,
                   train_instance_count=2,
                   train_instance_type='ml.c4.8xlarge',
                   output_path=output_location,
                   k=10,
                   data_location=data_location)
   
   In the constructor, you specify the following parameters:
   • role—The IAM role that Amazon SageMaker can assume to perform tasks on your behalf (for example, reading training results, called model artifacts, from the S3 bucket and writing training results to Amazon S3).
   • output_path—The S3 location where Amazon SageMaker stores the training results.
   • train_instance_count and train_instance_type—The type and number of ML compute instances to use for model training.
   • k—The number of clusters to create. For more information, see K-Means Hyperparameters (p. 141).
   • data_location—The S3 location where the high-level library uploads the transformed training data.

2. To start model training, call the KMeans estimator’s fit method.

   ```python
   %time
   kmeans.fit(kmeans.record_set(train_set[0]))
   ```
This is a synchronous operation. The method displays progress logs and waits until training completes before returning. For more information about model training, see Train a Model with Amazon SageMaker (p. 4).

The model training in this example takes about 15 minutes.

- **Use the SDK for Python.**

This low-level SDK for Python provides the `create_training_job` method, which maps to the `CreateTrainingJob` (p. 556) Amazon SageMaker API.

```python
%time
import boto3
from time import gmtime, strftime

job_name = 'kmeans-lowlevel-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Training job", job_name)

from sagemaker.amazon.amazon_estimator import get_image_uri
image = get_image_uri(boto3.Session().region_name, 'kmeans')
output_location = 's3://{}/kmeans_lowlevel_example/output'.format(bucket)
print('training artifacts will be uploaded to: {}'.format(output_location))

create_training_params = {
    "AlgorithmSpecification": {
        "TrainingImage": image,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": output_location
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.8xlarge",
        "VolumeSizeInGB": 50
    },
    "TrainingJobName": job_name,
    "HyperParameters": {
        "k": "10",
        "feature_dim": "784",
        "mini_batch_size": "500"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 60 * 60
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": data_location,
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "CompressionType": "None",
            "RecordWrapperType": "None"
        }
    ]
}
```
The code uses a *Waiter* to wait until training is complete before returning.

You now have trained a model. The resulting artifacts are stored in your S3 bucket.

**Next Step**

**Step 2.4: Deploy the Model to Amazon SageMaker**

To deploy a model to get predictions, you can choose one of two ways:

- Set up a persistent endpoint to get one prediction at a time, using Amazon SageMaker hosting services.
- Get predictions for an entire dataset, using Amazon SageMaker batch transform.

**Topics**

- [Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services](p. 29)
- [Step 2.4.2: Deploy the Model to Batch Transform](p. 31)

**Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services**

Deploying a model in Amazon SageMaker is a 3-step process:

1. Create a model in Amazon SageMaker— Send a `CreateModel` request to provide information such as the location of the S3 bucket that contains your model artifacts and the registry path of the image that contains inference code. In the next step, you provide the model when you create an endpoint configuration.
2. Create an endpoint configuration— Send a `CreateEndpointConfig` request to provide the resource configuration for hosting. This includes the type and number of ML compute instances to launch for deploying the model. In the next step, you create an endpoint with the `CreateEndpoint` API using this endpoint configuration.
3. Create an endpoint— Send a `CreateEndpoint` request to create an endpoint. Amazon SageMaker launches the ML compute instances and deploys the model. In the response, Amazon
SageMaker returns an endpoint. Applications can send requests to this endpoint to get inferences from the model.

The low-level AWS SDK for Python provides corresponding methods. However, the high-level Python library provides the deploy method that does all these tasks for you.

To deploy the model, choose one of the following options.

- **Use the high-level Python library provided by Amazon SageMaker.**

  The `sagemaker.amazon.kmeans.KMeans` class provides the `deploy` method for deploying a model. It performs all three steps of the model deployment process.

  ```python
  @time
  kmeans_predictor = kmeans.deploy(initial_instance_count=1,
                                   instance_type='ml.m4.xlarge')
  ```

  The `sagemaker.amazon.kmeans.KMeans` instance knows the registry path of the image that contains the k-means inference code, so you don't need to provide it.

  This is a synchronous operation. The method waits until the deployment completes before returning. It returns a `kmeans_predictor`.

- **Use the SDK for Python.**

  The low-level SDK for Python provides methods that map to the underlying Amazon SageMaker API. To deploy the model, you make three calls.

1. Create an Amazon SageMaker model by identifying the location of model artifacts and the Docker image that contains inference code.

   ```python
   @time
   import boto3
   from time import gmtime, strftime

   model_name = job_name
   print(model_name)
   info = sagemaker.describe_training_job(TrainingJobName=job_name)
   model_data = info['ModelArtifacts']['S3ModelArtifacts']
   primary_container = {
       'Image': image,
       'ModelDataUrl': model_data
   }

   create_model_response = sagemaker.create_model(
       ModelName = model_name,
       ExecutionRoleArn = role,
       PrimaryContainer = primary_container)

   print(create_model_response['ModelArn'])
   ```

2. Create an Amazon SageMaker endpoint configuration by specifying the ML compute instances that you want to deploy your model to.

   ```python
   from time import gmtime, strftime
   ```
Step 2.4: Deploy the Model

```python
endpoint_config_name = 'KMeansEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_config_name)
create_endpoint_config_response = sagemaker.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[
        {'InstanceType':'ml.m4.xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])
print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])

3. Create an Amazon SageMaker endpoint. This code uses a Waiter to wait until the deployment is complete before returning.

```%
time
```%time
import time
endpoint_name = 'KMeansEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_name)
create_endpoint_response = sagemaker.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])
resp = sagemaker.describe_endpoint(EndpointName=endpoint_name)
status = resp['EndpointStatus']
print("Status: " + status)
try:
    sagemaker.get_waiter('endpoint_in_service').wait(EndpointName=endpoint_name)
finally:
    resp = sagemaker.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']
    print("Arn: " + resp['EndpointArn'])
    print("Create endpoint ended with status: " + status)
    if status != 'InService':
        message = sagemaker.describe_endpoint(EndpointName=endpoint_name)["FailureReason"]
        print('Create endpoint failed with the following error: {}'.format(message))
        raise Exception('Endpoint creation did not succeed')
```

Next Step

Step 2.5: Validate the Model (p. 36)

Step 2.4.2: Deploy the Model to Batch Transform

Amazon SageMaker offers the ability to use batch transform either through the console or by using the API. For more information, see Get Inferences for an Entire Dataset with Batch Transform (p. 10). The following examples assume that you have already run a training job to create a model (see Step 2.3: Train a Model (p. 26)) and created a Model based on the training output (see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29)).

For an example that uses batch transform, see the batch transform sample notebook.

Topics
- Use Batch Transform (Console) (p. 32)
- Using Batch Transform (API) (p. 33)
Use Batch Transform (Console)

To generate inferences, create transform jobs by using the console.

Create a Batch Transform Job (Console)

To create a batch transform job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker
2. In the navigation pane, choose Inference, then choose Batch transform jobs.
3. Choose Create batch transform job.
4. For Batch transform job configuration, provide the following:
   a. For Job name, type a name for the new batch transform job.
   b. For Model name, choose Choose model, a list of available models appears.
   c. Choose the model that you want to use from the list of available models and choose Choose model.
   d. For Instance type, choose the instance type that you want to use for the batch transform job. To use models created by built-in algorithms to transform moderately-sized datasets, ml.m4.xlarge or ml.m5.large should suffice.
   e. For Instance count, type the number of compute instances to use for the transform job. For most use cases, 1 should suffice. By setting a higher value, you use more resources to reduce the time it takes to complete the transform job.
   f. (Optional) For Max concurrent transforms, type the maximum number of parallel requests to use for each instance node. For most use cases, 1 should suffice. To allow Amazon SageMaker to determine the appropriate number of concurrent transforms, set the value to 0 or leave it blank.
   g. (Optional) For Batch strategy, if you want to use only one record per batch, select SingleRecord. If you want to use the maximum number of records per batch, select MultiRecord.
   h. (Optional) For Max payload size (MB), type the maximum payload size, in MB. You can approximate an appropriate value by: dividing the size of your dataset by the number of records and multiply by the number of records you want per batch. The value you enter should be proportional to the number of records you want per batch. It is recommended to enter a slightly higher value to ensure the records will fit within the maximum payload size. The default value is 6 MB. For infinite stream mode, where a transform job is set to continuously input and transform data, set the value to 0.
   i. (Optional) For Environment variables, specify a key and value for the internal Docker container to use. To add more entries to input more variables, choose Add environment variables. To remove an entry choose Remove next to the respective entry.
5. For Input data configuration, provide the following information:
   a. For S3 location, type the path to the location where the input data is stored in Amazon S3.
   b. For S3 data type, choose the S3 data type. The default value is ManifestFile. If the object is a manifest file that contains a list of objects to transform, choose ManifestFile. To use all objects with a specified key name prefix, choose S3Prefix.
   c. For Compression type, you specify the compression used on the input data. The default value is None. If the input data is uncompressed select None. If the input data is compressed using Gzip, select Gzip.
   d. (Optional) For Split type, choose how to split the data into smaller batches for the transform job. The default is None. If you don't want to split the input data, choose None. If you want to split the data using newline characters, choose Line. If you want to split the data according to the RecordIO format, choose RecordIO. If you want to split the data according to the TFRecord format, choose TFRecord. For infinite stream mode, where a transform job is set to continuously input and transform data, choose None.
e. (Optional) For **Content type**, specified the MIME type that you want to use.

6. For **Output data configuration**, provide the following information:
   a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
   b. (Optional) For **Encryption key**, you can add your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see [KMS-Managed Encryption Keys](#).
   c. (Optional) For **Accept**, specify the MIME type you want to be used for the output data.

7. (Optional) For **Tags**, add tags to the transform job. A tag is a label you assign to help manage your training jobs. A tag consists of a key and a value both of which you define.

8. Choose **Create job**.

**View Details for a Batch Transform Job (Console)**

To see details about a batch transform job, such as job name or status, choose a batch transform job from the batch transform jobs table.

To view the details of a model used in a batch transform job, on the batch transform job details page, choose the link under model name.

**Stop a Batch Transform Job (Console)**

You can stop a batch transform job from the **Batch transform job** page or from the **Batch detail page**.

- **To stop a batch transform job using the Actions menu**
  1. Choose the checkbox near the batch transform job’s name.
  2. In the **Actions** menu, choose **Stop**.

- **To stop a batch transform job from the Batch detail page**
  1. Choose the transformation job name to view its details.
  2. Choose **Stop**.

**Using Batch Transform (API)**

There are several ways to interact with transform jobs. For more information, see [CreateTransformJob](#). You can use the high-level Python library or the low-level SDK (boto3).

**Use Batch Transform (High-level Python Library)**

The following shows how to create a transform job using the high-level Python library:

```python
import boto3
import sagemaker
import json

input_key = 'kmeans_batch_example/input/valid-data.csv'
input_location = 's3://{}/{}'.format(bucket, input_key)
output_location = 's3://{}/kmeans_batch_example/output'.format(bucket)

### Convert the validation set numpy array to a csv file and upload to s3
numpy.savetxt('valid-data.csv', valid_set[0], delimiter=',', fmt='%g')
s3_client = boto3.client('s3')
s3_client.upload_file('valid-data.csv', bucket, input_key)
```
Step 2.4: Deploy the Model

# Initialize the transformer object
transformer = sagemaker.transformer.Transformer(
    base_transform_job_name='Batch-Transform',
    model_name=model_name,
    instance_count=1,
    instance_type='ml.c4.xlarge',

# To start a transform job:
transformer.transform(input_location, content_type='text/csv', split_type='Line')
# Then wait until transform job is completed
transformer.wait()

# To fetch validation result
s3_client.download_file(bucket, 'kmeans_batch_example/output/valid-data.csv.out', 'valid-result')
with open('valid-result') as f:
    results = f.readlines()
    print("Sample transform result: {}{}.format(results[0])")

Use Batch Transform (Low-level SDK (Boto 3))

The following examples demonstrate how to interact with transform jobs using The Amazon SageMaker low-level SDK:

Use the CreateTransformJob Method

To start using batch transform, you need to create a transform job. The low-level AWS SDK for Python provides the corresponding create_transform_job method. First, import the needed libraries (boto3) and classes (gtime and strftime from the time library). Second, set the needed variables (job_name, bucket, prefix, sm, and model_name). The sm variable is especially important as it allows us to use the Amazon SageMaker API. Third you write the request. For information, see CreateTransformJob (p. 562).

To help prevent errors and improve readability, use the variables you created in the previous step in creating the request. Finally, use the request with create_transform_job to start a batch transform job. The following code shows how to use the create_transform_job operation:

import boto3
import sagemaker
import json
from urllib.parse import urlparse
from time import gmtime, strftime

batch_job_name = 'Batch-Transform-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
input_location = 's3://{}/kmeans_batch_example/input'.format(bucket)
output_location = 's3://{}/kmeans_batch_example/output'.format(bucket)

### Convert the validation set numpy array to a csv file and upload to s3
numpy.savetxt('valid-data.csv', valid_set[0], delimiter=',', fmt='%.g')
s3_client = boto3.client('s3')
input_key = '{}/valid-data.csv'.format(urlparse(input_location).path.lstrip('/'))
s3_client.upload_file('valid-data.csv', bucket, input_key)

### Create a transform job
sm = boto3.client('sagemaker')
request = 

```python
sm.create_transform_job(**request)
print("Created Transform job with name: ", batch_job_name)

### Wait until job completion
while(True):
    response = sm.describe_transform_job(TransformJobName=batch_job_name)
    status = response['TransformJobStatus']
    if status == 'Completed':
        print("Transform job ended with status: ", status)
        break
    if status == 'Failed':
        message = response['FailureReason']
        print('Transform failed with the following error: {}.format(message))
        raise Exception('Transform job failed')
        print("Transform job is still in status: ", status)
        time.sleep(30)

### Fetch transform output
output_key = "/valid_data.csv.out".format(urlparse(output_location).path.lstrip('/'))
s3_client.download_file(bucket, output_key, 'valid-result')
with open('valid-result') as f:
    results = f.readlines()
print("Sample transform result: ").format(results[0])
```

**Use the DescribeTransformJob Method**

To see the details of a specific transform job, specify the name in a describe transform jobs request. The low-level AWS SDK for Python provides the corresponding `describe_transform_jobs` method. The following code shows how to use the `describe_transform_job` method:

```python
response = sm.describe_transform_job(TransformJobName=job_name)
```
Use the ListTransformJobs Method

To list previous transform jobs, specify criteria to list and to sort the list. The low-level AWS SDK for Python provides the corresponding list_transform_jobs method. The following code shows how to use the list_transform_jobs method:

```python
import boto3
from time import gmtime, strftime

request = {
    "StatusEquals": "Completed",
    "SortBy": "CreationTime",
    "SortOrder": "Descending",
    "MaxResults": 20,
}

response = sm.list_transform_jobs(**request)
```

**Tip**
When using the list_transform_jobs method, you can specify any or all of the members in the request structure. For more information, see ListTransformJobs (p. 693).

Next Step

**Step 2.5: Validate the Model (p. 36)**

**Step 2.5: Validate the Model**

You now have a model that is deployed in Amazon SageMaker. To validate the model, send sample requests and get inferences. To send requests to an Amazon SageMaker endpoint, use the InvokeEndpoint (p. 735) API.

To validate your model, choose one of the following options.

- **Validate the model (High-level Python Library Provided by Amazon SageMaker)**
  
  The kmeans_predictor returned by the deploy call in the preceding step provides the predict method. To get inferences from the model, call this method.

  1. Get an inference for the 30th image of a handwritten number in the valid_set dataset.

```python
result = kmeans_predictor.predict(valid_set[0][30:31])
print(result)
```

Example response:

```json
[{
    "key": "closest_cluster",
    "value": {
        "float32_tensor": {
            "values": 3.0
        }
    }
},
{
    "key": "distance_to_cluster",
    "value": {
        "float32_tensor": {
            "values": 7.221197605133057
        }
    }
}]
```
The response shows that the input image belongs to cluster 3. It also shows the mean squared distance for that cluster.

**Note**
In the k-means implementation, the cluster numbers and digit they represent don't align. For example, the algorithm might group images of the handwritten number 3 in cluster 0, and images of the number 4 in cluster 9.

2. Get inferences for the first 100 images.

```python
%%time
result = kmeans_predictor.predict(valid_set[0][0:100])
clusters = [r.label['closest_cluster'].float32_tensor.values[0] for r in result]
```

Visualize the results.

```python
for cluster in range(10):
    print('


Cluster {}:'.format(int(cluster)))
digits = [ img for l, img in zip(clusters, valid_set[0]) if int(l) == cluster ]
height = ((len(digits)-1)//5) + 1
width = 5
plt.rcParams["figure.figsize"] = (width,height)
_, subplots = plt.subplots(height, width)
subplots = numpy.ndarray.flatten(subplots)
for subplot, image in zip(subplots, digits):
    show_digit(image, subplot=subplot)
for subplot in subplots[len(digits):]
    subplot.axis('off')
plt.show()
```

This code takes the first 100 images of handwritten numbers from the `valid_set` dataset and generates inferences for them. The result is a set of clusters that group similar images. The following visualization shows four of the clusters that the model returned:
• Validate the Model (SDK for Python)

To send requests to the endpoint, use the `invoke_endpoint` method.

1. Send a request that sends the 30th image in the `train_set` as input. Each image is a 28x28 (total of 784) pixel image. The request sends all 784 pixels in the image as comma-separated values.

```python
import json

# Simple function to create a csv from our numpy array
def np2csv(arr):
    csv = io.BytesIO()
    numpy.savetxt(csv, arr, delimiter=',', fmt='%g')
    return csv.getvalue().decode().rstrip()

runtime = boto3.Session().client('sagemaker-runtime')
payload = np2csv(train_set[0][30:31])
response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                   ContentType='text/csv',
                                   Body=payload)
result = json.loads(response['Body'].read().decode())
print(result)

In the following example response, the inference classifies the image as belonging to cluster 7 (labels identifies the cluster). The inference also shows the mean squared distance for that cluster.

```json```
{'predictions': 
    [{'distance_to_cluster': 7.2033820152282715, 'closest_cluster': 7.0}]
```
2. Run another test. The following code takes the first 100 images from the valid_set validation set and generates inferences for them. This test identifies the cluster that the input images belong to and provides a visual representation of the result.

```python
%%time
payload = np2csv(valid_set[0][0:100])
response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                   ContentType='text/csv',
                                   Body=payload)
result = json.loads(response['Body'].read().decode())
clusters = [p['closest_cluster'] for p in result['predictions']]

for cluster in range(10):
    digits = [ img for l, img in zip(clusters, valid_set[0]) if int(l) == cluster ]
    height = ((len(digits)-1)//5) + 1
    width = 5
    plt.rcParams['figure.figsize'] = (width,height)
    _, subplots = plt.subplots(height, width)
    subplots = numpy.ndarray.flatten(subplots)
    for subplot, image in zip(subplots, digits):
        show_digit(image, subplot=subplot)
    for subplot in subplots[len(digits):]:
        subplot.axis('off')

plt.show()
```

The result is a set of clusters that group similar images. The following visualization shows four of the clusters that the model returned:
3. To get an idea of how accurate the model is, review the clusters and the numbers in them to see how well the model clustered similar looking digits. To improve the model, you might make the following changes to the training job:
   • Change the model training parameters—For example, increase the number of epochs or tweak hyperparameters, such as extra_center_factor. For more information, see K-Means Hyperparameters (p. 141).
   • Consider switching the algorithm—The images in the MNIST dataset include information that identifies the digits, called labels. Similarly, you might be able to label your training data for other problems. You might then use the label information and a supervised algorithm, such as the linear learner algorithm provided by Amazon SageMaker. For more information, see Linear Learner Algorithm (p. 159).
   • Try a more specialized algorithm—Try a specialized algorithm, such as the image classification algorithm provided by Amazon SageMaker instead of the linear learner algorithm. For more information, see Image Classification Algorithm (p. 106).
   • Use a custom algorithm—Consider using a custom neural network algorithm built on Apache MXNet or TensorFlow. For more information, see Use Apache MXNet with Amazon SageMaker (p. 390) and Use TensorFlow with Amazon SageMaker (p. 390).

Next Step
Step 3: Clean up (p. 40)

Step 3: Clean up

To avoid incurring unnecessary charges, use the AWS Management Console to delete the resources that you created for this exercise.

Note
If you plan to explore other exercises in this guide, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/ and delete the following resources:
   • The endpoint. This also deletes the ML compute instance or instances.
   • The notebook instance. You will need to stop the instance before deleting it.
2. Open the Amazon S3 console at https://console.aws.amazon.com/s3/ and delete the bucket that you created for storing model artifacts and the training dataset.
3. Open the IAM console at https://console.aws.amazon.com/iam/ and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at https://console.aws.amazon.com/cloudwatch/ and delete all of the log groups that have names starting with /aws/sagemaker/.

Step 4: Integrating Amazon SageMaker Endpoints into Internet-facing Applications

In a production environment, you might have an internet-facing application sending requests to the endpoint for inference. The following high-level example shows how to integrate your model endpoint into your application.
1. Create an IAM role that the AWS Lambda service principal can assume. Give the role permissions to call the Amazon SageMaker InvokeEndpoint API.
2. Create a Lambda function that calls the Amazon SageMaker InvokeEndpoint API.
3. Call the Lambda function from a mobile application. For an example of how to call a Lambda function from a mobile application using Amazon Cognito for credentials, see Tutorial: Using AWS Lambda as Mobile Application Backend.
Use Notebook Instances

An Amazon SageMaker notebook instance is a fully managed ML compute instance running the Jupyter Notebook App. Amazon SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to Amazon SageMaker hosting, and test or validate your models.

Topics
- Create a Notebook Instance (p. 42)
- Access Notebook Instances (p. 43)
- Use Example Notebooks (p. 47)
- Set the Notebook Kernel (p. 48)
- Install External Libraries and Kernels in Notebook Instances (p. 48)
- Associate Git Repositories with Amazon SageMaker Notebook Instances (p. 49)

Create a Notebook Instance

To create a notebook instance, use either the Amazon SageMaker console or the CreateNotebookInstance (p. 546) API. For an example of using the Amazon SageMaker console to create a notebook instance, see Step 1: Create an Amazon SageMaker Notebook Instance (p. 17).

After receiving the request, Amazon SageMaker does the following:

- **Creates a network interface**—If you choose the optional VPC configuration, it creates the network interface in your VPC. It uses the subnet ID that you provide in the request to determine which Availability Zone to create the subnet in. Amazon SageMaker associates the security group that you provide in the request with the subnet. For more information, see Notebook Instance Security (p. 433).

- **Launches an ML compute instance**—Amazon SageMaker launches an ML compute instance in an Amazon SageMaker VPC. Amazon SageMaker performs the configuration tasks that allow it to manage your notebook instance, and if you specified your VPC, it enables traffic between your VPC and the notebook instance.

- **Installs Anaconda packages and libraries for common deep learning platforms**—Amazon SageMaker installs all of the Anaconda packages that are included in the installer. For more information, see Anaconda package list. In addition, Amazon SageMaker installs the TensorFlow and Apache MXNet deep learning libraries.

- **Attaches an ML storage volume**—Amazon SageMaker attaches an ML storage volume to the ML compute instance. You can use the volume to clean up the training dataset or to temporarily store other data to work with. Choose any size between 5 GB and 16384 GB, in 1 GB increments, for the volume. The default is 5 GB.

**Important**
Only files and data saved within the /home/ec2-user/SageMaker folder persist between notebook instance sessions. Files and data that are saved outside this directory are overwritten when the notebook instance stops and restarts.

**Note**
Each notebook instance's /tmp directory provides a minimum of 10 GB of storage in an instant store. An instance store is temporary, block-level storage that isn't persistent. When the instance is stopped or restarted, Amazon SageMaker deletes the directory's contents. For more information, see Amazon EC2 Instance Store in the Amazon Elastic Compute Cloud User Guide.
Copies example Jupyter notebooks—These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

Access Notebook Instances

To access your Amazon SageMaker notebook instances, choose one of the following options:

- Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose Notebook instances. The console displays a list of notebook instances in your account. To open a notebook instance with a standard Jupyter interface, choose Open Jupyter for that instance. To open a notebook instance with a JupyterLab interface, choose Open JupyterLab for that instance.

The console uses your sign-in credentials to send a CreatePresignedNotebookInstanceUrl (p. 554) API request to Amazon SageMaker. Amazon SageMaker returns the URL for your notebook instance, and the console opens the URL in another browser tab and displays the Jupyter notebook dashboard.

- Use the API.

To get the URL for the notebook instance, call the CreatePresignedNotebookInstanceUrl (p. 554) API and use the URL that the API returns to open the notebook instance.

Use the Jupyter notebook dashboard to create and manage notebooks and to write code. For more information about Jupyter notebooks, see http://jupyter.org/documentation.html.

Limit Access to a Notebook Instance by IP Address

To allow access to a notebook instance only from IP addresses in a list that you specify, attach an IAM policy that denies access to CreatePresignedNotebookInstanceUrl (p. 554) unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access the notebook instance. For information about creating IAM policies, see Creating IAM Policies in the AWS Identity and Access Management User Guide. To specify the list of IP addresses that you want to have access to the notebook instance, use the NotIpAddress condition operator and the aws:SourceIP condition context key. For information about IAM condition operators, see IAM JSON Policy Elements: Condition Operators in the AWS Identity and Access Management User Guide. For information about IAM condition context keys, see AWS Global Condition Context Keys.

For example, the following policy allows access to a notebook instance only from IP addresses in the ranges 192.0.2.0-192.0.2.255 and 203.0.113.0-203.0.113.255:

```json
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
    "Resource": "*",
```
The policy restricts access to both the call to CreatePresignedNotebookInstanceUrl and to the URL that the call returns. The policy also restricts access to opening a notebook instance in the console.

**Note**  
Using this method to filter by IP address is incompatible when connecting to Amazon SageMaker through a VPC interface endpoint. For information about restricting access to a notebook instance when connecting through a VPC interface endpoint, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 44).

### Connect to a Notebook Instance Through a VPC Interface Endpoint

You can connect to your notebook instance from your VPC through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the notebook instance is conducted entirely and securely within the AWS network.

Amazon SageMaker notebook instances support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

**Note**  
Before you create an interface VPC endpoint to connect to a notebook instance, create an interface VPC endpoint to connect to the Amazon SageMaker API. That way, when users call CreatePresignedNotebookInstanceUrl (p. 554) to get the URL to connect to the notebook instance, that call also goes through the interface VPC endpoint. For information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 434).

You can create an interface endpoint to connect to your notebook instance with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see Creating an Interface Endpoint.

When you create the interface endpoint, specify `aws.sagemaker.region.notebook` as the service name. We recommend that you enable private DNS hostnames for your VPC endpoint. If you don't enable private DNS hostnames, users that connect to the notebook instance through the console will not connect through the interface endpoint. In other words, the console will attempt to connect over the internet.
After you have created a VPC endpoint, users can use it to connect to your notebook instance from within your VPC. If you enable private DNS hostnames for your VPC endpoint, users do not need to specify the VPC endpoint when connecting to the notebook instance. Anyone using the Amazon SageMaker API, the AWS CLI, or the console to connect to the notebook instance from within the VPC will connect to the VPC endpoint.

If you do not enable private DNS hostnames for your VPC endpoint, users have to specify the VPC endpoint name when connecting to the notebook instance.

Amazon SageMaker notebook instances support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available.

Connect Your Private Network to Your VPC

To connect to your notebook instance through your VPC, you either have to connect from an instance that is inside the VPC, or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or AWS Direct Connect. For information about Amazon VPN, see VPN Connections in the Amazon Virtual Private Cloud User Guide. For information about AWS Direct Connect, see Creating a Connection in the AWS Direct Connect User Guide.

Restrict Access to Connections from Within Your VPC

Even if you set up an interface endpoint in your VPC, individuals outside the VPC can connect to the notebook instance over the internet.

Important
If you apply an IAM policy similar to one of the following, users can't access the specified Amazon SageMaker APIs or the notebook instance through the console.

To restrict access to only connections made from within your VPC, create an AWS Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every AWS Identity and Access Management user, group, or role used to access the notebook instance.

```json
{
  "Id": "notebook-example-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable Notebook Access",
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl",
        "sagemaker:DescribeNotebookInstance"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-111bbaaa"
        }
      }
    }
  ]
}
```

If you want to restrict access to the notebook instance to only connections made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```json
{
  "Id": "notebook-example-1",
```
Both of these policy examples assume that you have also created an interface endpoint for the Amazon SageMaker API. For more information, see Connect to Amazon SageMaker Through a VPC Interface Endpoint (p. 434). In the second example, one of the values for aws:SourceVpce is the ID of the interface endpoint for the notebook instance. The other is the ID of the interface endpoint for the Amazon SageMaker API.

The policy examples here include DescribeNotebookInstance (p. 619) because typically you would call DescribeNotebookInstance to make sure that the NotebookInstanceStatus is InService before you try to connect to it. For example:

```bash
aws sagemaker describe-notebook-instance
    --notebook-instance-name myNotebookInstance
```

```json
{
    "NotebookInstanceArn":
    "NotebookInstanceName": "myNotebookInstance",
    "NotebookInstanceStatus": "InService",
    "Url": "mynotebookinstance.notebook.us-west-2.sagemaker.aws",
    "InstanceType": "ml.m4.xlarge",
    "RoleArn":
        "arn:aws:iam::1234567890ab:role/service-role/AmazonSageMaker-ExecutionRole-12345678T123456",
    "LastModifiedTime": 1540334777.501,
    "CreationTime": 1523050674.078,
    "DirectInternetAccess": "Disabled"
}
```

```bash
aws sagemaker create-presigned-notebook-instance-url
    --notebook-instance-name myNotebookInstance
```

```json
{
}
```

For both of these calls, if you did not enable private DNS hostnames for your VPC endpoint, or if you are using a version of the AWS SDK that was released before August 13, 2018, you must specify the endpoint URL in the call. For example, the call to create-presigned-notebook-instance-url would be:
Use Example Notebooks

Your notebook instance contains example notebooks provided by Amazon SageMaker. The example notebooks contain code that shows how to apply machine learning solutions by using Amazon SageMaker. Notebook instances use the nbexamples Jupyter extension. This enables you to view a read-only version of an example notebook or create a copy of it so that you can modify and run it. For more information about the nbexamples extension, see https://github.com/danielballan/nbexamples. To view or use the example notebooks, choose the SageMaker Examples tab.

To view a read-only version of an example notebook, on the SageMaker Examples tab, choose Preview for that notebook. To create a copy of an example notebook in the home directory of your notebook instance, choose Use. In the dialog box, you can change the notebook's name before saving it.

For more information about the example notebooks, see the Amazon SageMaker examples GitHub repository.

Note
Example notebooks typically download datasets from the internet. If you disable Amazon SageMaker-provided internet access when you create you notebook instance, example notebooks might not work. For more information, see Notebook Instances Are Internet-Enabled by Default (p. 433).
Set the Notebook Kernel

Amazon SageMaker provides several kernels for Jupyter that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To set a kernel for a new notebook in the Jupyter notebook dashboard, choose New, and then choose the kernel from the list.

Install External Libraries and Kernels in Notebook Instances

Amazon SageMaker notebook instances come with multiple environments already installed. These environments contain Jupyter kernels and Python packages including: scikit, Pandas, NumPy, TensorFlow, and MXNet. These environments, along with all files in the sample-notebooks folder, are refreshed when you stop and start a notebook instance. You can also install your own environments that contain your choice of packages and kernels. This is typically done using conda install or pip install.

To install the R kernel

1. Open a notebook instance.
2. In the Jupyter dashboard, choose New, and then choose Terminal.
3. In the terminal, type the following command:

   ```bash
   conda install --yes --name JupyterSystemEnv --channel r r-essentials=1.6.0
   ```

4. When the command completes, exit from the terminal, and refresh the Jupyter dashboard page.
5. In the Jupyter dashboard page, choose New. R now appears as a Notebook option.

Notebook instances come with Git installed. You can use Git to install projects from GitHub.

To install the Scala kernel

1. Open a notebook instance.
2. In the Jupyter dashboard, choose New, and then choose Terminal.
3. In the terminal, type the following commands:

   ```bash
   git clone https://github.com/jupyter-scala/jupyter-scala.git
cd jupyter-scala
./jupyter-scala
   ```

4. When the command completes, exit from the terminal, and refresh the Jupyter dashboard page.
5. In the Jupyter dashboard page, choose New. Scala now appears as a Notebook option.
The different Jupyter kernels in Amazon SageMaker notebook instances are separate conda environments. For information about conda environments, see Managing environments in the Conda documentation. If you want to use an external library in a specific kernel, install the library in the environment for that kernel. You can do this either in the terminal or in a notebook cell. The following procedures show how to install Theano so that you can use it in a notebook with a conda_mxnet_p36 kernel.

**To install Theano from a terminal**

1. Open a notebook instance.
2. In the Jupyter dashboard, choose **New**, and then choose **Terminal**.
3. In the terminal, type the following commands:

```bash
conda install -n mxnet_p36 -c conda-forge theano
python
import theano
```

**To install Theano from a Jupyter notebook cell**

1. Open a notebook instance.
2. In the Jupyter dashboard, choose **New**, and then choose **conda_mxnet_p36**.
3. In a cell in the new notebook, type the following command:

```bash
!pip install theano
```

## Associate Git Repositories with Amazon SageMaker Notebook Instances

Associate Git repositories with your notebook instance to save your notebooks in a source control environment that persists even if you stop or delete your notebook instance. You can associate one default repository and up to three additional repositories with a notebook instance. The repositories can be hosted in AWS CodeCommit, GitHub or on any other Git server. Associating Git repositories with your notebook instance can be useful for:

- **Persistence** - Notebooks in a notebook instance are stored on durable Amazon EBS volumes, but they do not persist beyond the life of your notebook instance. Storing notebooks in a Git repository enables you to store and use notebooks even if you stop or delete your notebook instance.
- **Collaboration** - Peers on a team often work on machine learning projects together. Storing your notebooks in Git repositories allows peers working in different notebook instances to share notebooks and collaborate on them in a source-control environment.
- **Learning** - Many Jupyter notebooks that demonstrate machine learning techniques are available in publicly hosted Git repositories, such as on GitHub. You can associate your notebook instance with a repository to easily load Jupyter notebooks contained in that repository.

There are two ways to associate a Git repository with a notebook instance:

- **Add a Git repository as a resource in your Amazon SageMaker account.** Then, to access the repository, you can specify an AWS Secrets Manager secret that contains credentials. That way, you can access repositories that require authentication.
• Associate a public Git repository that is not a resource in your account. If you do this, you cannot specify credentials to access the repository.

Topics
• Add a Git Repository to Your Amazon SageMaker Account (p. 50)
• Create a Notebook Instance with an Associated Git Repository (p. 52)
• Use Git Repositories in a Notebook Instance (p. 53)

Add a Git Repository to Your Amazon SageMaker Account
To manage your GitHub repositories, easily associate them with your notebook instances, and associate credentials for repositories that require authentication, add the repositories as resources in your Amazon SageMaker account. You can view a list of repositories that are stored in your account and details about each repository in the Amazon SageMaker console and by using the API.

You can add Git repositories to your Amazon SageMaker account in the Amazon SageMaker console or by using the AWS CLI.

Note
You can use the Amazon SageMaker API CreateCodeRepository (p. 517) to add Git repositories to your Amazon SageMaker account, but step-by-step instructions are not provided here.

Add a Git Repository to Your Amazon SageMaker Account (Console)

To add a Git repository as a resource in your Amazon SageMaker account
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Git repositories, then choose Add repository.
3. To add an AWS CodeCommit repository, choose AWS CodeCommit.

   a. To use an existing AWS CodeCommit repository:
      i. Choose Use existing repository.
      ii. For Repository, choose a repository from the list.
      iii. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      iv. Choose Add repository.
   b. To create a new AWS CodeCommit repository:
      i. Choose Create new repository.
      ii. Enter a name for the repository that you can use in both AWS CodeCommit and Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
      i. Choose Create repository.
4. To add a Git repository hosted somewhere other than AWS CodeCommit:
   a. Choose GitHub/Other Git-based repo.
   b. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   c. Enter the URL for the repository.
d. For **Git credentials**, choose the credentials to use to authenticate to the repository. This is necessary only if the Git repository is private.

i. To use an existing AWS Secrets Manager secret, choose **Use existing secret**, and then choose a secret from the list. For information about creating and storing a secret, see [Creating a Basic Secret](https://docs.aws.amazon.com/secretsmanager/latest/userguide/) in the *AWS Secrets Manager User Guide*.

   **Note**
   The secret must have a staging label of `AWSCURRENT` and must be in the following format:
   ```json
   { "username": "UserName", "password": "Password" }
   ```
   For GitHub repositories, we recommend using a personal access token instead of your account password. For information, see [Creating a Personal Access Token](https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/).

ii. To create a new AWS Secrets Manager secret, choose **Create secret**, enter a name for the secret, and then enter the username and password to use to authenticate to the repository.

   **Note**
   The IAM role you use to create the secret must have the `secretsmanager:GetSecretValue` permission in its IAM policy.
   The secret must have a staging label of `AWSCURRENT` and must be in the following format:
   ```json
   { "username": "UserName", "password": "Password" }
   ```
   For GitHub repositories, we recommend using a personal access token instead of your account password.

iii. To not use any credentials, choose **No secret**.

e. Choose **Create secret**.

---

## Add a Git Repository to Your Amazon SageMaker Account (CLI)

Use the `create-code-repository` AWS CLI command. Specify a name for the repository as the value of the `code-repository-name` argument. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). Also specify the following:

- The default branch
- The URL of the Git repository
- The Amazon Resource Name (ARN) of an AWS Secrets Manager secret that contains the credentials to use to authenticate the repository as the value of the `git-config` argument

For information about creating and storing a secret, see [Creating a Basic Secret](https://docs.aws.amazon.com/secretsmanager/latest/userguide/) in the *AWS Secrets Manager User Guide*. The following command creates a new repository named `MyRepository` in your Amazon SageMaker account that points to a Git repository hosted at `https://github.com/myprofile/my-repo`.

```
aws sagemaker create-code-repository \
   --code-repository-name "MyRepository" \
   --git-config '{"Branch" : "master", \
   "RepositoryUrl" : "https://github.com/myprofile/my-repo", \
```

**Note**
The secret must have a staging label of `AWSCURRENT` and must be in the following format:
```json
{ "username": "UserName", "password": "Password" }
```
For GitHub repositories, we recommend using a personal access token instead of your account password.

Create a Notebook Instance with an Associated Git Repository

You can associate Git repositories with a notebook instance when you create the notebook instance by using the AWS Management Console, or the AWS CLI.

Note
You can use the Amazon SageMaker API CreateNotebookInstance (p. 546) to associate Git repositories with a notebook instance, but step-by-step instructions are not provided here.

Topics
- Create a Notebook Instance with an Associated Git Repository (Console) (p. 52)
- Create a Notebook Instance with an Associated Git Repository (CLI) (p. 53)

Create a Notebook Instance with an Associated Git Repository (Console)

To create a notebook instance and associate Git repositories in the AWS Management Console

1. Follow the instructions at Step 1: Create an Amazon SageMaker Notebook Instance (p. 17).
2. For Git repositories, choose Git repositories to associate with the notebook instance.
   a. For Default repository, choose a repository that you want to use as your default repository. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at /home/ec2-user/SageMaker. When you open your notebook instance, it opens in this repository. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose Add a repository to Amazon SageMaker (opens the Add repository flow in a new window) and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 52). To clone a public repository that is not stored in your account, choose Clone a public Git repository to this notebook instance only, and then specify the URL for that repository.
   b. For Additional repository 1, choose a repository that you want to add as an additional directory. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at /home/ec2-user/SageMaker. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose Add a repository to Amazon SageMaker (opens the Add repository flow in a new window) and then follow the instructions at Create a Notebook Instance with an Associated Git Repository (Console) (p. 52). To clone a repository that is not stored in your account, choose Clone a public Git repository to this notebook instance only, and then specify the URL for that repository.

Repeat this step up to three times to add up to three additional repositories to your notebook instance.
Create a Notebook Instance with an Associated Git Repository (CLI)

To create a notebook instance and associate Git repositories by using the AWS CLI, use the `create-notebook-instance` command as follows:

- Specify the repository that you want to use as your default repository as the value of the `default-code-repository` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To use a repository that is stored as a resource in your Amazon SageMaker account, specify the name of the repository as the value of the `default-code-repository` argument. To use a repository that is not stored in your account, specify the URL of the repository as the value of the `default-code-repository` argument.

- Specify up to three additional repositories as the value of the `additional-code-repositories` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`, and the repository is excluded from the default repository by adding it to the `.git/info/exclude` directory of the default repository. To use repositories that are stored as resources in your Amazon SageMaker account, specify the names of the repositories as the value of the `additional-code-repositories` argument. To use repositories that are not stored in your account, specify the URLs of the repositories as the value of the `additional-code-repositories` argument.

For example, the following command creates a notebook instance that has a repository named `MyGitRepo`, that is stored as a resource in your Amazon SageMaker account, as a default repository, and an additional repository that is hosted on GitHub:

```bash
aws sagemaker create-notebook-instance \
   --notebook-instance-name "MyNotebookInstance" \
   --instance-type "ml.t2.medium" \
   --role-arn "arn:aws:iam::012345678901:role/service-role/AmazonSageMaker-ExecutionRole-20181129T121390" \
   --default-code-repository "MyGitRepo" \n   --additional-code-repositories "https://github.com/myprofile/my-other-repo"
```

**Note**
If you use an AWS CodeCommit repository that does not contain "SageMaker" in its name, add the `codecommit:GitPull` and `codecommit:GitPush` permissions to the role that you pass as the `role-arn` argument to the `create-notebook-instance` command. For information about how to add permissions to a role, see Adding and Removing IAM Policies in the AWS Identity and Access Management User Guide.

Use Git Repositories in a Notebook Instance

When you open a notebook instance that has Git repositories associated with it, it opens in the default repository, which is installed in your notebook instance directly under `/home/ec2-user/SageMaker`. You can open and create notebooks, and you can manually run Git commands in a notebook cell. For example:

```bash
!git pull origin master
```

To open any of the additional repositories, navigate up one folder. The additional repositories are also installed as directories under `/home/ec2-user/SageMaker`. 

53
If you open the notebook instance with a JupyterLab interface, the jupyter-git extension is installed and available to use. For information about the jupyter-git extension for JupyterLab, see https://github.com/jupyterlab/jupyterlab-git.

When you open a notebook instance in JupyterLab, you see the git repositories associated with it on the left menu:

You can use the jupyter-git extension to manage git visually, instead of using the command line:
Build a Model

To build a machine learning model that can be used for making accurate inferences, three steps are required:

• **Select the mode**: You need to choose machine learning model that is appropriate for the type and quantity of data you have, or can collect, and for the problem that you are solving. You can use a built-in algorithm provided by Amazon SageMaker, your own custom algorithm, one of the deep learning frameworks supported by Amazon SageMaker, or an algorithm that you subscribe to from AWS Marketplace. Options are outlined in Choose a Machine Learning Algorithm (p. 56).

• **Train the model**: This can be done in a variety of ways that depend on the type and size of the training data and on the algorithms to be used. Options are outlined in the Train a Model (p. 279) section.

• **Validate the model**: The accuracy of the trained model must be assessed against an additional validation dataset. Some guidance on approaching this step is provided in the Validate a Machine Learning Model (p. 12) topic.

Topics

• Choose a Machine Learning Algorithm (p. 56)
• Use Amazon SageMaker Built-in Algorithms (p. 56)
• Use Your Own Algorithms with Amazon SageMaker (p. 255)

Choose a Machine Learning Algorithm

This section focuses on the first step in building your model: selecting the appropriate machine learning model. This depends on the type of predictions that you want to make and on the type and amount of data available to you.

• For guidance on using a built-in algorithm provided by Amazon SageMaker, see Use Amazon SageMaker Built-in Algorithms (p. 56).

• For guidance on using your own custom algorithm, see Use Your Own Algorithms with Amazon SageMaker (p. 255).

• For guidance on using one of the deep learning frameworks supported by Amazon SageMaker, see Use Machine Learning Frameworks with Amazon SageMaker (p. 381).

• For guidance on using an algorithm that you subscribe to from AWS Marketplace, see Amazon SageMaker Resources in AWS Marketplace (p. 371).

Use Amazon SageMaker Built-in Algorithms

A machine learning algorithm uses example data to create a generalized solution (a model) that addresses the business question you are trying to answer. After you create a model using example data, you can use it to answer the same business question for a new set of data. This is also referred to as obtaining inferences.

Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types.

Because you create a model to address a business question, your first step is to understand the problem that you want to solve. Specifically, the format of the answer that you are looking for influences the
algorithm that you choose. For example, suppose that you are a bank marketing manager, and that you want to conduct a direct mail campaign to attract new customers. Consider the potential types of answers that you're looking for:

- **Answers that fit into discrete categories**—For example, answers to these questions:

  - "Based on past customer responses, should I mail this particular customer?" Answers to this question fall into two categories, "yes" or "no." In this case, you use the answer to narrow the recipients of the mail campaign.

  - "Based on past customer segmentation, which segment does this customer fall into?" Answers might fall into categories such as "empty nester," "suburban family," or "urban professional." You could use these segments to decide who should receive the mailing.

For this type of discrete classification problem, Amazon SageMaker provides two algorithms: **Linear Learner Algorithm (p. 159)** and the **XGBoost Algorithm (p. 246)**. You set the following hyperparameters to direct these algorithms to produce discrete results:

- For the Linear Learner algorithm, set the **predictor_type** hyperparameter to **binary_classifier**.

- For the XGBoost algorithm, set the **objective** hyperparameter to **reg:logistic**.

- **Answers that are quantitative**—Consider this question: "Based on the return on investment (ROI) from past mailings, what is the ROI for mailing this customer?" In this case, you use the ROI to target customers for the mail campaign. For these quantitative analysis problems, you can also use the **Linear Learner Algorithm (p. 159)** or the **XGBoost Algorithm (p. 246)** algorithms. You set the following hyperparameters to direct these algorithms to produce quantitative results:

  - For the Linear Learner algorithm, set the **predictor_type** hyperparameter to **regressor**.

  - For the XGBoost algorithm, set the **objective** hyperparameter to **reg:linear**.

- **Answers in the form of discrete recommendations**—Consider this question: "Based on past responses to mailings, what is the recommended content for each customer?" In this case, you are looking for a recommendation on what to mail, not whether to mail, the customer. For this problem, Amazon SageMaker provides the **Factorization Machines Algorithm (p. 96)** algorithm.

All of the questions in the preceding examples rely on having example data that includes answers. There are times that you don't need, or can't get, example data with answers. This is true for problems whose answers identify groups. For example:

- "I want to group current and prospective customers into 10 groups based on their attributes. How should I group them?" You might choose to send the mailing to customers in the group that has the
highest percentage of current customers. That is, prospective customers that most resemble current customers based on the same set of attributes. For this type of question, Amazon SageMaker provides the K-Means Algorithm (p. 137).

- "What are the attributes that differentiate these customers, and what are the values for each customer along those dimensions." You use these answers to simplify the view of current and prospective customers, and, maybe, to better understand these customer attributes. For this type of question, Amazon SageMaker provides the Principal Component Analysis (PCA) Algorithm (p. 213) algorithm.

In addition to these general-purpose algorithms, Amazon SageMaker provides algorithms that are tailored to specific use cases. These include:

- **Image Classification Algorithm (p. 106)**—Use this algorithm to classify images. It uses example data with answers (referred to as *supervised algorithm*).

- **Sequence-to-Sequence Algorithm (p. 233)**—This supervised algorithm is commonly used for neural machine translation.

- **Latent Dirichlet Allocation (LDA) Algorithm (p. 154)**—This algorithm is suitable for determining topics in a set of documents. It is an *unsupervised algorithm*, which means that it doesn't use example data with answers during training.

- **Neural Topic Model (NTM) Algorithm (p. 173)**—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

**Topics**
- Common Elements of Built-in Algorithms (p. 59)
- BlazingText Algorithm (p. 73)
- DeepAR Forecasting Algorithm (p. 82)
- Factorization Machines Algorithm (p. 96)
- Image Classification Algorithm (p. 106)
- IP Insights Algorithm (p. 127)
- K-Means Algorithm (p. 137)
- K-Nearest Neighbors (k-NN) Algorithm (p. 145)
- Latent Dirichlet Allocation (LDA) Algorithm (p. 154)
- Linear Learner Algorithm (p. 159)
- Neural Topic Model (NTM) Algorithm (p. 173)
- Object2Vec Algorithm (p. 179)
- Object Detection Algorithm (p. 192)
- Principal Component Analysis (PCA) Algorithm (p. 213)
- Random Cut Forest (RCF) Algorithm (p. 218)
- Semantic Segmentation Algorithm (p. 225)
- Sequence-to-Sequence Algorithm (p. 233)
- XGBoost Algorithm (p. 246)
## Common Elements of Built-in Algorithms

The following topics provide information common to all of the algorithms provided by Amazon SageMaker.

**Topics**
- Common Parameters for Built-In Algorithms (p. 59)
- Common Data Formats for Built-in Algorithms (p. 64)
- Instance Types for Built-In Algorithms (p. 71)
- Logs for Built-In Algorithms (p. 72)

## Common Parameters for Built-In Algorithms

The following table lists parameters for each of the algorithms provided by Amazon SageMaker.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>Channel Name</th>
<th>Training Image and Inference</th>
<th>Training Input Mode</th>
<th>File Type</th>
<th>Instance Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlazingText</td>
<td>train</td>
<td>&lt;ecr_path&gt;/blazingtext:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>Text file (one sentence per line with space-separated tokens)</td>
<td>GPU (single instance only) or CPU</td>
</tr>
<tr>
<td>DeepAR Forecasting</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/forecasting-deepar:&lt;tag&gt;</td>
<td>File</td>
<td>JSON Lines or Parquet</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>Factorization Machines</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/factorization-machines:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf</td>
<td>CPU (GPU for dense data)</td>
</tr>
<tr>
<td>Image Classification</td>
<td>train and validation, (optionally) train_lst, validation_lst, and model</td>
<td>&lt;ecr_path&gt;/image-classification:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO or image files (jpg or .png)</td>
<td>GPU</td>
</tr>
<tr>
<td>IP Insights</td>
<td>train and (optionally) validation</td>
<td>&lt;ecr_path&gt;/ipinsights:&lt;tag&gt;</td>
<td>File</td>
<td>CSV</td>
<td>GPU or GPU</td>
</tr>
<tr>
<td>k-means</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/kmeans:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPUCommon (single GPU device on one or more instances)</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>Channel Name</td>
<td>Training Image and Inference Image Registry Path</td>
<td>Training Input Mode</td>
<td>File Type</td>
<td>Instance Class</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>---------------------</td>
<td>-------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>k-nearest-neighbor (k-NN)</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/knn:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU (single GPU device on one or more instances)</td>
</tr>
<tr>
<td>LDA</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/lda:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU (single instance only)</td>
</tr>
<tr>
<td>Linear Learner</td>
<td>train and (optionally) validation, test, or both</td>
<td><code>&lt;ecr_path&gt;/linear-learner:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU</td>
</tr>
<tr>
<td>Neural Topic Model</td>
<td>train and (optionally) validation, test, or both</td>
<td><code>&lt;ecr_path&gt;/ntm:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>Object2Vec</td>
<td>train and (optionally) validation, test, or both</td>
<td><code>&lt;ecr_path&gt;/object2vec:&lt;tag&gt;</code></td>
<td>File</td>
<td>JSON Lines</td>
<td>GPU or CPU (single instance only)</td>
</tr>
<tr>
<td>Object Detection</td>
<td>train and validation, (optionally) train_annotation, validation_annotation, and model</td>
<td><code>&lt;ecr_path&gt;/object-detection:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO or image files (.jpg or .png)</td>
<td>GPU</td>
</tr>
<tr>
<td>PCA</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/pca:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>Random Cut Forest</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/randomcutforest:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>Channel Name</td>
<td>Training Image and Inference Image Registry Path</td>
<td>Training Input Mode</td>
<td>File Type</td>
<td>Instance Class</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Semantic Segmentation</td>
<td>train and validation, train annotation, validation annotation, and (optionally) label_map and model</td>
<td><code>&lt;ecr_path&gt;/semantic-segmentation:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>image files</td>
<td>GPU (single instance only)</td>
</tr>
<tr>
<td>Seq2Seq Modeling</td>
<td>train, validation, and vocab</td>
<td><code>&lt;ecr_path&gt;/seq2seq:&lt;tag&gt;</code></td>
<td>File</td>
<td>recordIO-protobuf</td>
<td>GPU (single instance only)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>train and (optionally) validation</td>
<td><code>&lt;ecr_path&gt;/xgboost:&lt;tag&gt;</code></td>
<td>File</td>
<td>CSV or LibSVM</td>
<td>CPU</td>
</tr>
</tbody>
</table>

For the **Training Image and Inference Image Registry Path** column, use the :1 version tag to ensure that you are using a stable version of the algorithm. You can reliably host a model trained using an image with the :1 tag on an inference image that has the :1 tag. Using the :latest tag in the registry path provides you with the most up-to-date version of the algorithm, but might cause problems with backward compatibility. Avoid using the :latest tag for production purposes.

For the **Training Image and Inference Image Registry Path** column, depending on algorithm and region use one of the following values for `<ecr_path>`.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>AWS Region</th>
<th>Training Image and Inference Image Registry Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factorization Machines, IP Insights, k-means, k-nearest-neighbor, Linear Learner, Object2Vec, Neural Topic Model, PCA, and Random Cut Forest</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>174872318107.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>382416733822.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>404615174143.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>351501993468.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>835164637446.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-2</td>
<td>712309505854.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>AWS Region</td>
<td>Training Image and Inference Image Registry Path</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>LDA</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>266724342769.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>766337827248.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>999911452149.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>258307448986.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>29318134795.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-2</td>
<td>297031611018.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-central-1</td>
<td>353608530281.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-1</td>
<td>999678624901.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td>BlazingText, Image Classification, Object Detection, Semantic Segmentation, Seq2Seq, and XGBoost</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>433757028032.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>811284229777.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>825641698319.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>501404015308.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>306986355934.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td>Algorithm Name</td>
<td>AWS Region</td>
<td>Training Image and Inference Image Registry Path</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>--------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ap-southeast-1</td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td>ap-southeast-2</td>
<td>ap-southeast-2</td>
<td>544295431143.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td>ca-central-1</td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-central-1</td>
<td>eu-central-1</td>
<td>813361260812.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-1</td>
<td>eu-west-1</td>
<td>685385470294.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td>eu-west-2</td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
<tr>
<td>DeepAR Forecasting</td>
<td>us-west-1</td>
<td>632365934929.dkr.ecr.us-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-west-2</td>
<td>156387875391.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>522234722520.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>566113047672.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-gov-west-1</td>
<td>226302683700.dkr.ecr.us-gov-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>633353088612.dkr.ecr.ap-northeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-2</td>
<td>204372634319.dkr.ecr.ap-northeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-south-1</td>
<td>991648021394.dkr.ecr.ap-south-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-1</td>
<td>475088953585.dkr.ecr.ap-southeast-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-southeast-2</td>
<td>514117268639.dkr.ecr.ap-southeast-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ca-central-1</td>
<td>469771592824.dkr.ecr.ca-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-central-1</td>
<td>495149712605.dkr.ecr.eu-central-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-1</td>
<td>224300973850.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-2</td>
<td>644912444149.dkr.ecr.eu-west-2.amazonaws.com</td>
</tr>
</tbody>
</table>

Use the paths and training input mode as follows:

- To create a training job (with a request to the `CreateTrainingJob (p. 556)` API), specify the Docker Registry path and the training input mode for the training image. You create a training job to train a model using a specific dataset.

- To create a model (with a `CreateModel (p. 538)` request), specify the Docker Registry path for the inference image. Amazon SageMaker launches machine learning compute instances that are based on the endpoint configuration and deploys the model, which includes the artifacts (the result of model training).
Common Data Formats for Built-in Algorithms

The following topics explain the data formats for the algorithms provided by Amazon SageMaker.

Topics
- Common Data Formats for Training (p. 64)
- Common Data Formats for Inference (p. 67)

Common Data Formats for Training

To prepare for training, you can preprocess your data using a variety of AWS services, including AWS Glue, Amazon EMR, Amazon Redshift, Amazon Relational Database Service, and Amazon Athena. After preprocessing, publish the data to an Amazon S3 bucket. For training, the data need to go through a series of conversions and transformations, including:

- Training data serialization (handled by you)
- Training data deserialization (handled by the algorithm)
- Training model serialization (handled by the algorithm)
- Trained model deserialization (optional, handled by you)

When using Amazon SageMaker in the training portion of the algorithm, make sure to upload all data at once. If more data is added to that location, a new training call would need to be made to construct a brand new model.

Training Data Formats

Many Amazon SageMaker algorithms support training with data in CSV format. To use data in CSV format for training, in the input data channel specification, specify `text/csv` as the `ContentType`. Amazon SageMaker requires that a CSV file doesn't have a header record and that the target variable is in the first column. To run unsupervised learning algorithms that don't have a target, specify the number of label columns in the content type. For example, in this case `text/csv;label_size=0`.

Most Amazon SageMaker algorithms work best when you use the optimized protobuf recordIO format for the training data. Using this format allows you to take advantage of Pipe mode when training the algorithms that support it. File mode loads all of your data from Amazon Simple Storage Service (Amazon S3) to the training instance volumes. In Pipe mode, your training job streams data directly from Amazon S3. Streaming can provide faster start times for training jobs and better throughput. With Pipe mode, you also reduce the size of the Amazon Elastic Block Store volumes for your training instances. Pipe mode needs only enough disk space to store your final model artifacts. File mode needs disk space to store both your final model artifacts and your full training dataset. See the AlgorithmSpecification (p. 744) for additional details on the training input mode. For a summary of the data formats supported by each algorithm, see the documentation for the individual algorithms or this table.

Note

For an example that shows how to convert the commonly used numPy array into the protobuf recordIO format, see Step 2.2.3: Transform the Training Dataset and Upload It to Amazon S3 (p. 25).

In the protobuf recordIO format, Amazon SageMaker converts each observation in the dataset into a binary representation as a set of 4-byte floats and is then loads it to the protobuf values field. If you are using Python for your data preparation, we strongly recommend that you use these existing transformations. However, if you are using another language, the protobuf definition file below provides the schema that you use to convert your data into Amazon SageMaker protobuf format.

```plaintext
syntax = "proto2";
```
package aialgs.data;

option java_package = "com.amazonaws.aialgorithms.proto";
option java_outer_classname = "RecordProtos";

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float32Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated float values = 1 [packed = true];

    // If key is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [10, 20], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensional tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float64Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated double values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [10, 20], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensional tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as 32-bit ints (int32).
message Int32Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated int32 values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [10, 20], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensional tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// Support for storing binary data for parsing in other ways (such as JPEG/etc).
// This is an example of another type of value and may not immediately be supported.
message Bytes {
    repeated bytes value = 1;

    // If the content type of the data is known, stores it.
// This allows for the possibility of using decoders for common formats
// in the future.
optional string content_type = 2;
}
message Value {
  oneof value {
    // The numbering assumes the possible use of:
    // - float16, float128
    // - int8, int16, int32
    Float32Tensor float32_tensor = 2;
    Float64Tensor float64_tensor = 3;
    Int32Tensor int32_tensor = 7;
    Bytes bytes = 9;
  }
}
message Record {
  // Map from the name of the feature to the value.
  // For vectors and libsvm-like datasets,
  // a single feature with the name `values`
  // should be specified.
  map<string, Value> features = 1;

  // An optional set of labels for this record.
  // Similar to the features field above, the key used for
  // generic scalar / vector labels should ve `values`.
  map<string, Value> label = 2;

  // A unique identifier for this record in the dataset.
  // Whilst not necessary, this allows better
  // debugging where there are data issues.
  // This is not used by the algorithm directly.
  optional string uid = 3;

  // Textual metadata describing the record.
  // This may include JSON-serialized information
  // about the source of the record.
  // This is not used by the algorithm directly.
  optional string metadata = 4;

  // An optional serialized JSON object that allows per-record
  // hyper-parameters/configuration/other information to be set.
  // The meaning/interpretation of this field is defined by
  // the algorithm author and may not be supported.
  // This is used to pass additional inference configuration
  // when batch inference is used (e.g. types of scores to return).
  optional string configuration = 5;
}

After creating the protocol buffer, store it in an Amazon S3 location that Amazon SageMaker can access and that can be passed as part of InputDataConfig in create_training_job.

**Note**
For all Amazon SageMaker algorithms, the ChannelName in InputDataConfig must be set to train. Some algorithms also support a validation or test input channels. These are typically used to evaluate the model's performance by using a hold-out dataset. Hold-out datasets are not used in the initial training but can be used to further tune the model.
Trained Model Deserialization

Amazon SageMaker models are stored as model.tar.gz in the S3 bucket specified in `OutputDataConfig.S3OutputPath` parameter of the `create_training_job` call. You can specify most of these model artifacts when creating a hosting model. You can also open and review them in your notebook instance. When `model.tar.gz` is untarred, it contains `model_algo-1`, which is a serialized Apache MXNet object. For example, you use the following to load the k-means model into memory and view it:

```python
import mxnet as mx
print(mx.ndarray.load('model_algo-1'))
```

Common Data Formats for Inference

Amazon SageMaker algorithms accept and produce several different MIME types for the http payloads used in retrieving online and mini-batch predictions. You can use various AWS services to transform or preprocess records prior to running inference. At a minimum, you need to convert the data for the following:

- Inference request serialization (handled by you)
- Inference request deserialization (handled by the algorithm)
- Inference response serialization (handled by the algorithm)
- Inference response deserialization (handled by you)

Convert Data for Inference Request Serialization

Content type options for Amazon SageMaker algorithm inference requests include: `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Algorithms that don't support these types, such as XGBoost, which is incompatible, support other types, such as `text/x-libsvm`.

For `text/csv` the value for the `Body` argument to `invoke_endpoint` should be a string with commas separating the values for each feature. For example, a record for a model with four features might look like: `1.5,16.0,14,23.0`. Any transformations performed on the training data should also be performed on the data before obtaining inference. The order of the features matters, and must remain unchanged.

For `application/json` is significantly more flexible and provides multiple possible formats for developers to use in their applications. At a high level, in JavaScript, the payload might look like:

```javascript
let request = {
    // Instances might contain multiple rows that predictions are sought for.
    "instances": [
    {
        // Request and algorithm specific inference parameters.
        "configuration": {},
        // Data in the specific format required by the algorithm.
        "data": {
            "<field name>": dataElement
        }
    }
    ]
}
```

You have the following options for specifying the `dataElement`:

- Protocol buffers equivalent:

  ```javascript
  // Has the same format as the protocol buffers implementation described for training.
  let dataElement = {
  ```
Simple numeric vector:

```
// An array containing numeric values is treated as an instance containing a
// single dense vector.
let dataElement = [1.5, 16.0, 14.0, 23.0]

// It will be converted to the following representation by the SDK.
let converted = {
    "features": {
        "values": dataElement
    }
}
```

And, for multiple records:

```
let request = {
    "instances": [
        // First instance.
        {
            "features": [ 1.5, 16.0, 14.0, 23.0 ]
        },
        // Second instance.
        {
            "features": [ -2.0, 100.2, 15.2, 9.2 ]
        }
    ]
}
```

Convert Data for Inference Response Deserialization

Amazon SageMaker algorithms return JSON in several layouts. At a high level, the structure is:

```
let response = {
    "predictions": [{
        // Fields in the response object are defined on a per algorithm-basis.
    }]
}
```

The fields that are included in predictions differ across algorithms. The following are examples of output for the k-means algorithm.

Single-record inference:

```
let response = {
    "predictions": [{
        "closest_cluster": 5,
        "distance_to_cluster": 36.5
    }]
}
```

Multi-record inference:

```
let response = {
    "predictions": [
        // First instance prediction.
    ]
}
```
Multi-record inference with protobuf input:

```
{
  "features": [],
  "label": {
    "closest_cluster": {
      "values": [ 5.0 ] // e.g. the closest centroid/cluster was 1.0
    },
    "distance_to_cluster": {
      "values": [ 36.5 ]
    }
  },
  "uid": "abc123",
  "metadata": "{ "created_at": '2017-06-03' }"
}
```

Amazon SageMaker algorithms also support jsonlines format, where the per-record response content is same as that in JSON format. The multi-record structure is a concatenation of per-record response objects separated by newline characters. The response content for the built-in KMeans algorithm for 2 input data points is:

```
{"distance_to_cluster": 23.40593910217285, "closest_cluster": 0.0}
{"distance_to_cluster": 27.250282287597656, "closest_cluster": 0.0}
```

While running batch transform, it is recommended to use jsonlines response type by setting the Accept field in the CreateTransformJobRequest to application/jsonlines.

**Common Request Formats for All Algorithms**

Most algorithms use several of the following inference request formats.

**JSON Request Format**

Content-type: application/json

**Dense Format**

```javascript
let request = {
    "instances": [
        {
            "features": [1.5, 16.0, 14.0, 23.0]
        }
    ]
};
```

```javascript
let request = {
    "instances": [
        {
            "data": {
```
Sparse Format

```json
{
    "instances": [ 
        {"data": {"features": { 
            "keys": [26, 182, 232, 243, 431], 
            "shape": [2000], 
            "values": [1, 1, 1, 4, 1] 
        } } 
    ], 
    "data": {"features": { 
            "keys": [0, 182, 232, 243, 431], 
            "shape": [2000], 
            "values": [13, 1, 1, 4, 1] 
        } } 
}
```

**JSONLINES Request Format**

Content-type: application/jsonlines

**Dense Format**

A single record in dense format can be represented as either:

```json
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

or:

```json
{ "data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } }
```

**Sparse Format**

A single record in sparse format is represented as:

```json
{"data": {"features": { "keys": [26, 182, 232, 243, 431], "shape": [2000], "values": [1, 1, 1, 4, 1] } } }
```

Multiple records are represented as a concatenation of the above single-record representations, separated by newline characters:

```json
{"data": {"features": { "keys": [26, 182, 232, 243, 431], "shape": [2000], "values": [1, 1, 1, 4, 1] } } }
{"data": { "features": { "keys": [0, 1, 3], "shape": [4], "values": [1, 4, 1] } } }
{"data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } } }
```

**CSV Request Format**

Content-type: text/csv;label_size=0
Note
CSV support is not available for factorization machines.

RECORDIO Request Format

Content-type: application/x-recordio-protobuf

Use Batch Transform with Build-in Algorithms

While running batch transform, it's recommended to use jsonlines response type instead of JSON, if supported by the algorithm. This is accomplished by setting the Accept field in the CreateTransformJobRequest to application/jsonlines.

When you create a transform job, the SplitType must be set according to the ContentType of the input data. Similarly, depending on the Accept field in the CreateTransformJobRequest, AssembleWith must be set accordingly. Please use the following table to help appropriately set these fields:

<table>
<thead>
<tr>
<th>ContentType</th>
<th>Recommended SplitType</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-recordio-protobuf</td>
<td>RecordIO</td>
</tr>
<tr>
<td>text/csv</td>
<td>Line</td>
</tr>
<tr>
<td>application/jsonlines</td>
<td>Line</td>
</tr>
<tr>
<td>application/json</td>
<td>None</td>
</tr>
<tr>
<td>application/x-image</td>
<td>None</td>
</tr>
<tr>
<td>image/*</td>
<td>None</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accept</th>
<th>Recommended AssembleWith</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/x-recordio-protobuf</td>
<td>None</td>
</tr>
<tr>
<td>application/json</td>
<td>None</td>
</tr>
<tr>
<td>application/jsonlines</td>
<td>Line</td>
</tr>
</tbody>
</table>

For more information on response formats for specific algorithms, see the following:

- PCA Response Formats (p. 217)
- Linear Learner Response Formats (p. 172)
- NTM Response Formats (p. 178)
- K-Means Response Formats (p. 144)
- Factorization Machine Response Formats (p. 105)

Instance Types for Built-in Algorithms

For training and hosting Amazon SageMaker algorithms, we recommend using the following EC2 instance types:

- ml.m4.xlarge, ml.m4.4xlarge, and ml.m4.10xlarge
- ml.c4.xlarge, ml.c4.2xlarge, and ml.c4.8xlarge
- ml.p2.xlarge, ml.p2.8xlarge, and ml.p2.16xlarge
Most Amazon SageMaker algorithms have been engineered to take advantage of GPU computing for training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective. Exceptions, such as XGBoost, are noted in this guide. (XGBoost implements an open-source algorithm that has been optimized for CPU computation.)

The size and type of data can have a great effect on which hardware configuration is most effective. When the same model is trained on a recurring basis, initial testing across a spectrum of instance types can discover configurations that are more cost effective in the long run. Additionally, algorithms that train most efficiently on GPUs might not require GPUs for efficient inference. Experiment to determine the most cost effectiveness solution.

For more information on Amazon SageMaker hardware specifications, see Amazon SageMaker ML Instance Types.

Logs for Built-In Algorithms

Amazon SageMaker algorithms produce Amazon CloudWatch logs, which provide detailed information on the training process. To see the logs, in the AWS management console, choose CloudWatch, choose Logs, and then choose the /aws/sagemaker/TrainingJobs log group. Each training job has one log stream per node that it was trained on. The log stream’s name begins with the value specified in the TrainingJobName parameter when the job was created.

Note
If a job fails and logs do not appear in CloudWatch, it’s likely that an error occurred before the start of training. Reasons include specifying the wrong training image or S3 location.

The contents of logs vary by algorithms. However, you can typically find the following information:

- Confirmation of arguments provided at the beginning of the log
- Errors that occurred during training
- Measurement of an algorithms accuracy or numerical performance
- Timings for the algorithm, and any major stages within the algorithm

Common Errors

If a training job fails, some details about the failure are provided by the FailureReason return value in the training job description, as follows:

```python
sage = boto3.client('sagemaker')
sage.describe_training_job(TrainingJobName=job_name)['FailureReason']
```

Others are reported only in the CloudWatch logs. Common errors include the following:

1. Misspecifying a hyperparameter or specifying a hyperparameter that is invalid for the algorithm.

   From the CloudWatch Log:

   ```
   [10/16/2017 23:45:17 ERROR 139623806805824 train.py:48]
   Additional properties are not allowed (u'mini_batch_siz' was unexpected)
   ```

2. Specifying an invalid value for a hyperparameter.

   FailureReason:

   ```
   AlgorithmError: u'abc' is not valid under any of the given
schem

Failed validating u'oneOf' in
schema[u'properties'][u'feature_dim']:
    {u'oneOf':
        [{u'pattern': u'^([1-9]\d*)$', u'type': u'string'},
         {u'minimum': 1, u'type': u'integer'}]
}

FailureReason:
[10/16/2017 23:57:17 ERROR 140373086025536 train.py:48] u'abc'
is not valid under any of the given schemas

3. Inaccurate protobuf file format.

From the CloudWatch log:

[10/17/2017 18:01:04 ERROR 140234860816192 train.py:48] cannot
copy sequence with size 785 to array axis with dimension 784

BlazingText Algorithm

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2Vec and text classification algorithms. The Word2Vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc. Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.

The Word2Vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is called a word embedding. Words that are semantically similar correspond to vectors that are close together. That way, word embeddings capture the semantic relationships between words.

Many natural language processing (NLP) applications learn word embeddings by training on large collections of documents. These pretrained vector representations provide information about semantics and word distributions that typically improves the generalizability of other models that are later trained on a more limited amount of data. Most implementations of the Word2Vec algorithm are optimized for multi-core CPU architectures. This makes it difficult to scale to large datasets.

With the BlazingText algorithm, you can scale to large datasets easily. Similar to Word2Vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures. BlazingText's implementation of the supervised multi-class, multi-label text classification algorithm extends the fastText text classifier to use GPU acceleration with custom CUDA kernels. You can train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. And, you achieve performance on par with the state-of-the-art deep learning text classification algorithms.

The Amazon SageMaker BlazingText algorithms provides the following features:

- Accelerated training of the fastText text classifier on multi-core CPUs or a GPU and Word2Vec on GPUs using highly optimized CUDA kernels. For more information, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.

- Enriched Word Vectors with Subword Information by learning vector representations for character n-grams. This approach enables BlazingText to generate meaningful vectors for out-of-vocabulary (OOV) words by representing their vectors as the sum of the character n-gram (subword) vectors.

- A batch_skipgram mode for the Word2Vec algorithm that allows faster training and distributed computation across multiple CPU nodes. The batch_skipgram mode does mini-batching using the Negative Sample Sharing strategy to convert level-1 BLAS operations into level-3 BLAS operations. This efficiently leverages the multiply-add instructions of modern architectures. For more information, see Parallelizing Word2Vec in Shared and Distributed Memory.
To summarize, the following modes are supported by BlazingText on different types instances:

<table>
<thead>
<tr>
<th>Modes</th>
<th>Word2Vec (Unsupervised Learning)</th>
<th>Text Classification (Supervised Learning)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single CPU instance</td>
<td>cbow</td>
<td>supervised</td>
</tr>
<tr>
<td></td>
<td>Skip-gram</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Batch Skip-gram</td>
<td></td>
</tr>
<tr>
<td>Single GPU instance (with 1 or more GPUs)</td>
<td>cbow</td>
<td>supervised with one GPU</td>
</tr>
<tr>
<td></td>
<td>Skip-gram</td>
<td></td>
</tr>
<tr>
<td>Multiple CPU instances</td>
<td>Batch Skip-gram</td>
<td>None</td>
</tr>
</tbody>
</table>

For more information about the mathematics behind BlazingText, see [BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs](#).

**Topics**
- Input/Output Interface for the BlazingText Algorithm (p. 74)
- EC2 Instance Recommendation for the BlazingText Algorithm (p. 76)
- BlazingText Sample Notebooks (p. 77)
- BlazingText Hyperparameters (p. 77)
- Tune a BlazingText Model (p. 81)

### Input/Output Interface for the BlazingText Algorithm

The BlazingText algorithm expects a single preprocessed text file with space-separated tokens. Each line in the file should contain a single sentence. If you need to train on multiple text files, concatenate them into one file and upload the file in the respective channel.

#### Training and Validation Data Format

**Training and Validation Data Format for the Word2Vec Algorithm**

For Word2Vec training, upload the file under the `train` channel. No other channels are supported. The file should contain a training sentence per line.

**Training and Validation Data Format for the Text Classification Algorithm**

For supervised mode, you can train with file mode or with the augmented manifest text format.

#### Train with File Mode

For supervised mode, the training/validation file should contain a training sentence per line along with the labels. Labels are words that are prefixed by the string `__label__`. Here is an example of a training/validation file:

```
__label__4  linux ready for prime time , intel says , despite all the linux hype , the open-source movement has yet to make a huge splash in the desktop market . that may be about to change , thanks to chipmaking giant intel corp .
```
Note
The order of labels within the sentence doesn't matter.

Upload the training file under the train channel, and optionally upload the validation file under the validation channel.

Train with Augmented Manifest Text Format

The supervised mode also supports the augmented manifest format, which enables you to do training in pipe mode without needing to create RecordIO files. While using the format, an S3 manifest file needs to be generated that contains the list of sentences and their corresponding labels. The manifest file format should be in JSON Lines format in which each line represents one sample. The sentences are specified using the source tag and the label can be specified using the label tag. Both source and label tags should be provided under the AttributeName parameter value as specified in the request.

```json
{"source":"linux ready for prime time , intel says , despite all the linux hype", "label":1}
{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly" , "label":2}
```

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

Model Artifacts and Inference

Model Artifacts for the Word2Vec Algorithm

For Word2Vec training, the model artifacts consist of vectors.txt, which contains words-to-vectors mapping, and vectors.bin, a binary used by BlazingText for hosting, inference, or both. vectors.txt stores the vectors in a format that is compatible with other tools like Gensim and Spacy. For example, a Gensim user can run the following commands to load the vectors.txt file:

```python
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('vectors.txt', binary=False)
word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
word_vectors.doesnt_match("breakfast cereal dinner lunch".split())
```

If the evaluation parameter is set to True, an additional file, eval.json, is created. This file contains the similarity evaluation results (using Spearman's rank correlation coefficients) on WS-353 dataset. The number of words from the WS-353 dataset that aren't there in the training corpus are reported.

For inference requests, the model accepts a JSON file containing a list of strings and returns a list of vectors. If the word is not found in vocabulary, inference returns a vector of zeros. If subwords is set to True during training, the model is able to generate vectors for out-of-vocabulary (OOV) words.

Sample JSON Request

Mime-type: application/json

```json
{
  "instances": ["word1", "word2", "word3"]
}
```

Model Artifacts for the Text Classification Algorithm

Training with supervised outputs creates a model.bin file that can be consumed by BlazingText hosting. For inference, the BlazingText model accepts a JSON file containing a list of sentences and returns a list of labels.
of corresponding predicted labels and probability scores. Each sentence is expected to be a string with space-separated tokens, words, or both.

**Sample JSON Request**

Mime-type: application/json

```json
{
  "instances": ["the movie was excellent", "i did not like the plot ."]
}
```

By default, the server returns only one prediction, the one with the highest probability. For retrieving the top \( k \) predictions, you can set \( k \) in the configuration, as follows:

```json
{
  "instances": ["the movie was excellent", "i did not like the plot ."],
  "configuration": {"k": 2}
}
```

For BlazingText, the content-type and accept parameters must be equal. For batch transform, they both need to be application/jsonlines. If they differ, the Accept field is ignored. The format for input follows:

- **content-type: application/jsonlines**

  ```json
  {"source": "source_0"}
  {"source": "source_1"}
  ```

  if you need to pass the value of \( k \) for top-\( k \), then you can do it in the following way:

  ```json
  {"source": "source_0", "k": 2}
  {"source": "source_1", "k": 3}
  ```

The format for output follows:

- **accept: application/jsonlines**

  ```json
  {"prob": [prob_1], "label": ["__label__1"]}
  {"prob": [prob_1], "label": ["__label__1"]}
  ```

  If you have passed the value of \( k \) to be more than 1, then response will be in this format:

  ```json
  {"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
  {"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
  ```

For both supervised (text classification) and unsupervised (Word2Vec) modes, the binaries (*.bin) produced by BlazingText can be cross-consumed by fastText and vice versa. You can use binaries produced by BlazingText by fastText. Likewise, you can host the model binaries created with fastText using BlazingText.

For more details on dataset formats and model hosting, see the example notebooks Text Classification with the BlazingText Algorithm, FastText Models, and Generating Subword Embeddings with the Word2Vec Algorithm.

**EC2 Instance Recommendation for the BlazingText Algorithm**

For cbow and skipgram modes, BlazingText supports single CPU and single GPU instances. Both of these modes support learning of subwords embeddings. To achieve the highest speed without compromising accuracy, we recommend that you use an ml.p3.2xlarge instance.
For batch_skipgram mode, BlazingText supports single or multiple CPU instances. When training on multiple instances, set the value of the S3DataDistributionType field of the S3DataSource object that you pass to CreateTrainingJob to FullyReplicated. BlazingText takes care of distributing data across machines.

For the supervised text classification mode, a C5 instance is recommended if the training dataset is less than 2 GB. For larger datasets, use an instance with a single GPU (ml.p2.xlarge or ml.p3.2xlarge).

**BlazingText Sample Notebooks**

For a sample notebook that uses the Amazon SageMaker BlazingText algorithm to train and deploy supervised binary and multiclass classification models, see Blazing Text classification on the DBPedia dataset. For instructions for creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances. After creating and opening a notebook instance, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, choose its Use tab, then choose Create copy.

**BlazingText Hyperparameters**

When you start a training job with a CreateTrainingJob request, you specify a training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The hyperparameters for the BlazingText algorithm depend on which mode you use: Word2Vec (unsupervised) and Text Classification (supervised).

**Word2Vec Hyperparameters**

The following table lists the hyperparameters for the BlazingText Word2Vec training algorithm provided by Amazon SageMaker.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>The Word2vec architecture used for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: batch_skipgram, skipgram, or cbow</td>
</tr>
<tr>
<td>batch_size</td>
<td>The size of each batch when mode is set to batch_skipgram. Set to a number between 10 and 20.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 11</td>
</tr>
<tr>
<td>buckets</td>
<td>The number of hash buckets to use for subwords.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2000000</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of complete passes through the training data.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>evaluation</strong></td>
<td>Whether the trained model is evaluated using the WordSimilarity-353 Test.</td>
</tr>
<tr>
<td><strong>learning_rate</strong></td>
<td>The step size used for parameter updates.</td>
</tr>
<tr>
<td><strong>min_char</strong></td>
<td>The minimum number of characters to use for subwords/character n-grams.</td>
</tr>
<tr>
<td><strong>min_count</strong></td>
<td>Words that appear less than min_count times are discarded.</td>
</tr>
<tr>
<td><strong>max_char</strong></td>
<td>The maximum number of characters to use for subwords/character n-grams</td>
</tr>
<tr>
<td><strong>negative_samples</strong></td>
<td>The number of negative samples for the negative sample sharing strategy.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sampling_threshold</td>
<td>The threshold for the occurrence of words. Words that appear with higher frequency in the training data are randomly down-sampled.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive fraction. The recommended range is (0, 1e-3]</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
<tr>
<td>subwords</td>
<td>Whether to learn subword embeddings on not.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: (Boolean) True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>vector_dim</td>
<td>The dimension of the word vectors that the algorithm learns.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>window_size</td>
<td>The size of the context window. The context window is the number of words surrounding the target word used for training.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
</tbody>
</table>

**Text Classification Hyperparameters**

The following table lists the hyperparameters for the Text Classification training algorithm provided by Amazon SageMaker.

**Note**

Although some of the parameters are common between the Text Classification and Word2Vec modes, they might have different meanings depending on the context.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>The training mode.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: supervised</td>
</tr>
<tr>
<td>buckets</td>
<td>The number of hash buckets to use for word n-grams.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2000000</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>early_stopping</td>
<td>Whether to stop training if validation accuracy doesn't improve after a patience number of epochs.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: (Boolean) True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>epochs</td>
<td>The maximum number of complete passes through the training data.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The step size used for parameter updates.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.05</td>
</tr>
<tr>
<td>min_count</td>
<td>Words that appear less than min_count times are discarded.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>min_epochs</td>
<td>The minimum number of epochs to train before early stopping logic is invoked.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>patience</td>
<td>The number of epochs to wait before applying early stopping when no progress is made on the validation set. Used only when early_stopping is True.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 4</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
vector_dim | The dimension of the embedding layer.
  | Optional
  | Valid values: Positive integer
  | Default value: 100
word_ngrams | The number of word n-gram features to use.
  | Optional
  | Valid values: Positive integer
  | Default value: 2

---

**Tune a BlazingText Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Automatic Model Tuning](p. 290).

**Metrics Computed by the BlazingText Algorithm**

The BlazingText Word2Vec algorithm (*skipgram*, *cbow*, and *batch_skipgram* modes) reports on a single metric during training: `train:mean_rho`. This metric is computed on *WS-353 word similarity datasets*. When tuning the hyperparameter values for the Word2Vec algorithm, use this metric as the objective.

The BlazingText Text Classification algorithm (*supervised* mode), also reports on a single metric during training: the `validation:accuracy`. When tuning the hyperparameter values for the text classification algorithm, use these metrics as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>train:mean_rho</td>
<td>The mean rho (Spearman's rank correlation coefficient) on <em>WS-353 word similarity datasets</em></td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:accuracy</td>
<td>The classification accuracy on the user-specified validation dataset</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

**Tunable BlazingText Hyperparameters**

**Tunable Hyperparameters for the Word2Vec Algorithm**

Tune an Amazon SageMaker BlazingText Word2Vec model with the following hyperparameters. The hyperparameters that have the greatest impact on Word2Vec objective metrics are: *mode*, *learning_rate*, *window_size*, *vector_dim*, and *negative_samples*.
DeepAR Forecasting Algorithm

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

In many applications, however, you have many similar time series across a set of cross-sectional units. For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly...
over all of the time series. DeepAR takes this approach. When your dataset contains hundreds of related time series, DeepAR outperforms the standard ARIMA and ETS methods. You can also use the trained model to generate forecasts for new time series that are similar to the ones it has been trained on.

The training dataset consists of one or, preferably, more time series (target). You can optionally associate target time series with a vector of categorical features (with the cat hyperparameter) and a vector of feature time series (with the dynamic_feat hyperparameter). Amazon SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the context_length hyperparameter. To control how far in the future predictions can be made, use the prediction_length hyperparameter. For more information, see How the DeepAR Algorithm Works (p. 86).

### Topics
- Input/Output Interface for the DeepAR Algorithm (p. 83)
- Best Practices for Using the DeepAR Algorithm (p. 85)
- EC2 Instance Recommendations for the DeepAR Algorithm (p. 86)
- DeepAR Sample Notebooks (p. 86)
- How the DeepAR Algorithm Works (p. 86)
- DeepAR Hyperparameters (p. 88)
- Tune a DeepAR Model (p. 93)
- DeepAR Inference Formats (p. 94)

### Input/Output Interface for the DeepAR Algorithm

DeepAR supports two data channels. The required train channel describes the training dataset. The optional test channel describes a dataset that the algorithm uses to evaluate model accuracy after training. You can provide training and test datasets in JSON Lines format. Files can be in gzip or Parquet file format.

When specifying the paths for the training and test data, you can specify a single file or a directory that contains multiple files, which can be stored in subdirectories. If you specify a directory, DeepAR uses all files in the directory as inputs for the corresponding channel, except those that start with a period (.) and those named _SUCCESS. This ensures that you can directly use output folders produced by Spark jobs as input channels for your DeepAR training jobs.

By default, the DeepAR model determines the input format from the file extension (.json, .json.gz, or .parquet) in the specified input path. If the path does not end in one of these extensions, you must explicitly specify the format in the SDK for Python. Use the content_type parameter of the s3_input class.

The records in your input files should contain the following fields:

- **start**—A string with the format YYYY-MM-DD HH:MM:SS. The start timestamp can't contain time zone information.
- **target**—An array of floating-point values or integers that represent the time series. You can encode missing values as null literals, or as "NaN" strings in JSON, or as nan floating-point values in Parquet.
- **dynamic_feat** (optional)—An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must have the same length as the associated target value. Missing values are not supported in the features.
- **cat** (optional)—An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive
integers. For example, the categorical domain \{R, G, B\} can be encoded as \{0, 1, 2\}. All values from each categorical domain must be represented in the training dataset. That's because the DeepAR algorithm can forecast only for categories that have been observed during training. And, each categorical feature is embedded in a low-dimensional space whose dimensionality is controlled by the `embedding_dimension` hyperparameter. For more information, see DeepAR Hyperparameters (p. 88).

If you use a JSON file, it must be in JSON Lines format. For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],
"dynamic_feat": [[1.1, 1.2, 0.5, ..]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":
[[1.1, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":
[[1.3, 0.4]]}
```

In this example, each time series has two associated categorical features and one time series features.

For Parquet, you use the same three fields as columns. In addition, "start" can be the `datetime` type. You can compress Parquet files using gzip (``gzip``) or the Snappy compression library (``snappy``).

The following guidelines apply to training data:

- The start time and length of the time series can differ, but all series must have the same frequency, number of categorical features, and number of dynamic features.
- Shuffle the training file with respect to the position of the time series in the file. In other words, the time series should occur in random order in the file.
- Make sure to set the `start` field correctly. The algorithm uses the `start` timestamp to derive the internal features.
- If you use categorical features (``cat``), all time series must have the same number of categorical features. If the dataset contains the `cat` field, the algorithm uses it and extracts the cardinality of the groups from the dataset. By default, cardinality is "auto". If the dataset contains the `cat` field, but you don't want to use it, you can disable it by setting `cardinality` to "". If a model was trained using a `cat` feature, you must include it for inference.
- If your dataset contains the `dynamic_feat` field, the algorithm uses it automatically. All time series have to have the same number of feature time series. The time points in each of the feature time series correspond one-to-one to the time points in the target. In addition, the entry in the `dynamic_feat` field should have the same length as the `target`. If the dataset contains the `dynamic_feat` field, but you don't want to use it, disable it by setting(`num_dynamic_feat` to ""). If the model was trained with the `dynamic_feat` field, you must provide this field for inference. In addition, each of the features has to have the length of the provided target plus the `prediction_length`. In other words, you must provide the feature value in the future.

If you specify optional test channel data, the DeepAR algorithm evaluates the trained model with different accuracy metrics. The algorithm calculates the root mean square error (RMSE) over the test data as follows:

\[
RMSE = \sqrt{\frac{1}{nT} \sum_{i,t} (\hat{y}_{it} - y_{it})^2}
\]

\(y_{it}\) is the true value of time series \(i\) at the time \(t\). \(\hat{y}_{it}\) is the mean prediction. The sum is over all \(n\) time series in the test set and over the last \(T\) time points for each time series, where \(T\) corresponds to the forecast horizon. You specify the length of the forecast horizon by setting the `prediction_length` hyperparameter. For more information, see DeepAR Hyperparameters (p. 88).
In addition, the algorithm evaluates the accuracy of the forecast distribution using weighted quantile loss. For a quantile in the range \([0, 1]\), the weighted quantile loss is defined as follows:

\[
\text{wQuantileLoss}[\tau] = 2 \sum_{i,t} \frac{Q_{i,t}^{(\tau)}}{|y_{i,t}|}, \quad \text{with} \quad Q_{i,t}^{(\tau)} = \begin{cases} 
(1 - \tau)|q_{i,t}^{(\tau)} - y_{i,t}| & \text{if } q_{i,t}^{(\tau)} > y_{i,t} \\
\tau|q_{i,t}^{(\tau)} - y_{i,t}| & \text{otherwise}
\end{cases}
\]

\(q_{i,t}^{(\tau)}\) is the \(\tau\)-quantile of the distribution that the model predicts. To specify which quantiles to calculate loss for, set the \text{test_quantiles} hyperparameter. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see DeepAR Hyperparameters (p. 88).

For inference, DeepAR accepts JSON format and the following fields:

- "instances", which includes one or more time series in JSON Lines format
- A name of "configuration", which includes parameters for generating the forecast

For more information, see DeepAR Inference Formats (p. 94).

**Best Practices for Using the DeepAR Algorithm**

When preparing your time series data, follow these best practices to achieve the best results:

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference. Regardless of how you set \text{context_length}, don't break up the time series or provide only a part of it. The model uses data points further back than the value set in \text{context_length} for the lagged values feature.

- When tuning a DeepAR model, you can split the dataset to create a training dataset and a test dataset. In a typical evaluation, you would test the model on the same time series used for training, but on the future \text{prediction_length} time points that follow immediately after the last time point visible during training. You can create training and test datasets that satisfy this criteria by using the entire dataset (the full length of all time series that are available) as a test set and removing the last \text{prediction_length} points from each time series for training. During training, the model doesn't see the target values for time points on which it is evaluated during testing. During testing, the algorithm withholding the last \text{prediction_length} points of each time series in the test set and generates a prediction. Then it compares the forecast with the withheld values. You can create more complex evaluations by repeating time series multiple times in the test set, but cutting them at different endpoints. With this approach, accuracy metrics are averaged over multiple forecasts from different time points. For more information, see Tune a DeepAR Model (p. 93).

- Avoid using very large values (>400) for the \text{prediction_length} because it makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating your data at a higher frequency. For example, use 5min instead of 1min.

- Because lags are used, a model can look further back in the time series than the value specified for \text{context_length}. Therefore, you don't need to set this parameter to a large value. We recommend starting with the value that you used for \text{prediction_length}.

- We recommend training a DeepAR model on as many time series as are available. Although a DeepAR model trained on a single time series might work well, standard forecasting algorithms, such as ARIMA or ETS, might provide more accurate results. The DeepAR algorithm starts to outperform the standard methods when your dataset contains hundreds of related time series. Currently, DeepAR requires that the total number of observations available across all training time series is at least 300.
EC2 Instance Recommendations for the DeepAR Algorithm

You can train DeepAR on both GPU and CPU instances and in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, ml.c4.xlarge or ml.c4.2xlarge), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves throughput only for larger models (with many cells per layer and many layers) and for large mini-batch sizes (for example, greater than 512).

For inference, DeepAR supports only CPU instances.

Specifying large values for context_length, prediction_length, num_cells, num_layers, or mini_batch_size can create models that are too large for small instances. In this case, use a larger instance type or reduce the values for these parameters. This problem also frequently occurs when running hyperparameter tuning jobs. In that case, use an instance type large enough for the model tuning job and consider limiting the upper values of the critical parameters to avoid job failures.

DeepAR Sample Notebooks

For a sample notebook that shows how to prepare a time series dataset for training the Amazon SageMaker DeepAR algorithm and how to deploy the trained model for performing inferences, see Time series forecasting with DeepAR - Synthetic data as well as DeepAR demo on electricity dataset, which illustrates the advanced features of DeepAR on a real world dataset. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). After creating and opening a notebook instance, choose the SageMaker Examples tab to see a list of all of the Amazon SageMaker examples. To open a notebook, choose its Use tab, and choose Create copy.

How the DeepAR Algorithm Works

During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series. You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector of feature time series and a vector of categorical features. For more information, see Input/Output Interface for the DeepAR Algorithm (p. 83).

For example, the following is an element of a training set indexed by $i$ which consists of a target time series, $Z_{i,t}$, and two associated feature time series, $X_{i,1,t}$ and $X_{i,2,t}$.
The target time series might contain missing values, which are represented by line breaks in the time series. DeepAR supports only feature time series that are known in the future. This allows you to run "what if?" scenarios. What happens, for example, if I change the price of a product in some way?

Each target time series can also be associated with a number of categorical features. You can use these features to encode which groupings a time series belongs to. Categorical features allow the model to learn typical behavior for groups, which it can use to increase model accuracy. DeepAR implements this by learning an embedding vector for each group that captures the common properties of all time series in the group.

How Feature Time Series Work in the DeepAR Algorithm

To facilitate learning time-dependent patterns, such as spikes during weekends, DeepAR automatically creates feature time series based on the frequency of the target time series. For example, DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency. It uses these derived feature time series with the custom feature time series that you provide during training and inference. The following figure shows two of these derived time series features: $u_{i,1,t}$ represents the hour of the day and $u_{i,2,t}$ the day of the week.

The DeepAR algorithm automatically generates these feature time series. The following table lists the derived features for the supported basic time frequencies.

<table>
<thead>
<tr>
<th>Frequency of the Time Series</th>
<th>Derived Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minute</td>
<td>minute-of-hour, hour-of-day, day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Hour</td>
<td>hour-of-day, day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Day</td>
<td>day-of-week, day-of-month, day-of-year</td>
</tr>
<tr>
<td>Week</td>
<td>day-of-month, week-of-year</td>
</tr>
<tr>
<td>Month</td>
<td>month-of-year</td>
</tr>
</tbody>
</table>

DeepAR trains a model by randomly sampling several training examples from each of the time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. The context_length hyperparameter controls how far in the past the network can see, and the prediction_length hyperparameter controls how far in the future predictions can be made. During training, the algorithm ignores training set elements containing time series that are shorter than a specified prediction length. The following figure represents five samples...
with context lengths of 12 hours and prediction lengths of 6 hours drawn from element \( i \). For brevity, we've omitted the feature time series \( x_{i,t} \) and \( u_{i,2,t} \).

To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. In the example with hourly frequency, for each time index, \( t = T \), the model exposes the \( z_{i,t} \) values, which occurred approximately one, two, and three days in the past.

For inference, the trained model takes as input target time series, which might or might not have been used during training, and forecasts a probability distribution for the next prediction_length values. Because DeepAR is trained on the entire dataset, the forecast takes into account patterns learned from similar time series.

For information on the mathematics behind DeepAR, see DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks.

### DeepAR Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context_length</td>
<td>The number of time-points that the model gets to see before making the prediction. The value for this parameter should be about the same as the prediction_length. The model also receives lagged inputs from the target, so context_length can be much smaller than typical seasonalities. For example, a daily time series can have yearly seasonality. The model automatically includes a lag of one year, so the context length can be shorter than a year.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>epochs</td>
<td>The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also early_stopping_patience. Typical values range from 10 to 1000.</td>
</tr>
<tr>
<td>prediction_length</td>
<td>The number of time-steps that the model is trained to predict, also called the forecast horizon. The trained model always generates forecasts with this length. It can't generate longer forecasts. The prediction_length is fixed when a model is trained and it cannot be changed later.</td>
</tr>
<tr>
<td>time_freq</td>
<td>The granularity of the time series in the dataset. Use time_freq to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, 5min specifies a frequency of 5 minutes.</td>
</tr>
</tbody>
</table>

- **M**: monthly
- **W**: weekly
- **D**: daily
- **H**: hourly
- **min**: every minute

**Required**

Valid values: Positive integer

---

The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year.

**Required**

Valid values: Positive integer

---

The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| cardinality         | When using the categorical features (cat), cardinality is an array specifying the number of categories (groups) per categorical feature. Set this to auto to infer the cardinality from the data. The auto mode also works when no categorical features are used in the dataset. This is the recommended setting for the parameter.  

Set cardinality to ignore to force DeepAR to not use categorical features, even if they are present in the data.  

To perform additional data validation, it is possible to explicitly set this parameter to the actual value. For example, if two categorical features are provided where the first has 2 and the other has 3 possible values, set this to [2, 3].  

For more information on how to use categorical feature, see the data-section on the main documentation page of DeepAR.  

Optional  

Valid values: auto, ignore, array of positive integers, empty string, or  

Default value: auto |
| dropout_rate        | The dropout rate to use during training. The model uses zoneout regularization. For each iteration, a random subset of hidden neurons are not updated. Typical values are less than 0.2.  

Optional  

Valid values: float  

Default value: 0.1 |
| early_stopping_patience | If this parameter is set, training stops when no progress is made within the specified number of epochs. The model that has the lowest loss is returned as the final model.  

Optional  

Valid values: integer |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedding_dimension</td>
<td>Size of embedding vector learned per categorical feature (same value is used for all categorical features). The DeepAR model can learn group-level time series patterns when a categorical grouping feature is provided. To do this, the model learns an embedding vector of size embedding_dimension for each group, capturing the common properties of all time series in the group. A larger embedding_dimension allows the model to capture more complex patterns. However, because increasing the embedding_dimension increases the number of parameters in the model, more training data is required to accurately learn these parameters. Typical values for this parameter are between 10-100.</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate used in training. Typical values range from 1e-4 to 1e-1.</td>
</tr>
</tbody>
</table>
| likelihood           | The model generates a probabilistic forecast, and can provide quantiles of the distribution and return samples. Depending on your data, select an appropriate likelihood (noise model) that is used for uncertainty estimates. The following likelihoods can be selected:  
  - *gaussian*: Use for real-valued data.  
  - *beta*: Use for real-valued targets between 0 and 1 inclusive.  
  - *negative-binomial*: Use for count data (non-negative integers).  
  - *student-T*: An alternative for real-valued data that works well for bursty data.  
  - *deterministic-L1*: A loss function that does not estimate uncertainty and only learns a point forecast.  |

**Optional**

- Valid values: positive integer
- Default value: 10

- Valid values: float
- Default value: 1e-3

- Valid values: One of *gaussian*, *beta*, *negative-binomial*, *student-T*, or *deterministic-L1*.
- Default value: *student-T*
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Optional</th>
<th>Valid values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mini_batch_size</td>
<td>The size of mini-batches used during training. Typical values range from 32 to 512.</td>
<td>Optional</td>
<td>Positive integer</td>
<td>128</td>
</tr>
<tr>
<td>num_cells</td>
<td>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</td>
<td>Optional</td>
<td>Positive integer</td>
<td>40</td>
</tr>
<tr>
<td>num_dynamic_feat</td>
<td>The number of dynamic features provided in the data. Set this to auto to infer the number of dynamic features from the data. The auto mode also works when no dynamic features are used in the dataset. This is the recommended setting for the parameter. To force DeepAR to not use dynamic features, even if they are present in the data, set num_dynamic_feat to ignore. To perform additional data validation, it is possible to explicitly set this parameter to the actual integer value. For example, if two dynamic features are provided, set this to 2. Optional</td>
<td>Positive integer, auto, ignore, or empty string</td>
<td>auto</td>
<td></td>
</tr>
<tr>
<td>num_eval_samples</td>
<td>The number of samples that are used per time-series when calculating test accuracy metrics. This parameter does not have any influence on the training or the final model. In particular, the model can be queried with a different number of samples. This parameter only affects the reported accuracy scores on the test channel after training. Smaller values result in faster evaluation, but then the evaluation scores are typically worse and more uncertain. When evaluating with higher quantiles, for example 0.95, it may be important to increase the number of evaluation samples. Optional</td>
<td>Integer</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
**Parameter Name** | **Description**
--- | ---
num_layers | The number of hidden layers in the RNN. Typical values range from 1 to 4.  
Optional  
Valid values: positive integer  
Default value: 2

test_quantiles | Quantiles for which to calculate quantile loss on the test channel.  
Optional  
Valid values: array of floats  
Default value: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

---

**Tune a DeepAR Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see *Automatic Model Tuning (p. 290).*

**Metrics Computed by the DeepAR Algorithm**

The DeepAR algorithm reports three metrics, which are computed during training. When tuning a model, choose one of these as the objective. For the objective, use either the forecast accuracy on a provided test channel (recommended) or the training loss. For recommendations for the training/test split for the DeepAR algorithm, see *Best Practices for Using the DeepAR Algorithm (p. 85).*

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:RMSE</td>
<td>The root mean square error between the forecast and the actual target computed on the test set.</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:mean_wQuantileLoss</td>
<td>The average overall quantile losses computed on the test set. To control which quantiles are used, set the test_quantiles hyperparameter.</td>
<td>Minimize</td>
</tr>
<tr>
<td>train:final_loss</td>
<td>The training negative log-likelihood loss averaged over the last training epoch for the model.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

**Tunable Hyperparameters for the DeepAR Algorithm**

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact, listed in order from the most to least impactful, on DeepAR objective metrics are: epochs, context_length, mini_batch_size, learning_rate, and num_cells.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 32, MaxValue: 1028</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 1000</td>
</tr>
<tr>
<td>context_length</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 200</td>
</tr>
<tr>
<td>num_cells</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 30, MaxValue: 200</td>
</tr>
<tr>
<td>num_layers</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 8</td>
</tr>
<tr>
<td>dropout_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.00, MaxValue: 0.2</td>
</tr>
<tr>
<td>embedding_dimension</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 50</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-5, MaxValue: 1e-1</td>
</tr>
</tbody>
</table>

**DeepAR Inference Formats**

**DeepAR JSON Request Formats**

Query a trained model by using the model’s endpoint. The endpoint takes the following JSON request format.

In the request, the `instances` field corresponds to the time series that should be forecast by the model.

If the model was trained with categories, you must provide a `cat` for each instance. If the model was trained without the `cat` field, it should be omitted.

If the model was trained with a custom feature time series (`dynamic_feat`), you have to provide the same number of `dynamic_feat` values for each instance. Each of them should have a length given by `length(target) + prediction_length`, where the last `prediction_length` values correspond to the time points in the future that will be predicted. If the model was trained without custom feature time series, the field should not be included in the request.

```json
{
   "instances": [
      {
         "start": "2009-11-01 00:00:00",
         "target": [4.0, 10.0, "NaN", 100.0, 113.0],
         "cat": [0, 1],
         "dynamic_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]
      },
      {
         "start": "2012-01-30",
         "target": [1.0],
         "cat": [2, 1],
         "dynamic_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]
      },
      {
         "start": "1999-01-30",
         "dynamic_feat": [[3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, ...]]
      }
   ]
}
```
The configuration field is optional. `configuration.num_samples` sets the number of sample paths that the model generates to estimate the mean and quantiles. `configuration.output_types` describes the information that will be returned in the request. Valid values are "mean", "quantiles", and "samples". If you specify "quantiles", each of the quantile values in `configuration.quantiles` is returned as a time series. If you specify "samples", the model also returns the raw samples used to calculate the other outputs.

**DeepAR JSON Response Formats**

The following is the format of a response, where `[...]` are arrays of numbers:

```json
{
  "predictions": [
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]  
      },
      "samples": [...],
      "mean": [...]  
    },
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]  
      },
      "samples": [...],
      "mean": [...]  
    },
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]  
      },
      "samples": [...],
      "mean": [...]  
    }
  ]
}
```

DeepAR has a response timeout of 60 seconds. When passing multiple time series in a single request, the forecasts are generated sequentially. Because the forecast for each time series typically takes about 300 to 1000 milliseconds or longer, depending on the model size, passing too many time series in a single request can cause time-outs. It's better to send fewer time series per request and send more requests. Because the DeepAR algorithm uses multiple workers per instance, you can achieve much higher throughput by sending multiple requests in parallel.

By default, DeepAR uses one worker per CPU for inference, if there is sufficient memory per CPU. If the model is large and there isn't enough memory to run a model on each CPU, the number of workers is reduced. The number of workers used for inference can be overwritten using the environment variable
MODEL_SERVER_WORKERS For example, by setting MODEL_SERVER_WORKERS=1) when calling the Amazon SageMaker CreateModel (p. 538) API.

Batch Transform with the DeepAR Algorithm

DeepAR forecasting supports getting inferences by using batch transform using the JSON Lines format. In this format, each record is represented on a single line as a JSON object, and lines are separated by newline characters. The format is identical to the JSON Lines format used for model training. For information, see Input/Output Interface for the DeepAR Algorithm (p. 83). For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1], "dynamic_feat": [[1.1, 1.2, 0.5, ...]]}
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat": [[1.1, 2.05, ...]]}
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat": [[1.3, 0.4]]}
```

Similar to the hosted endpoint inference request format, the `cat` and the `dynamic_feat` fields for each instance are required if both of the following are true:

- The model is trained on a dataset that contained both the `cat` and the `dynamic_feat` fields.
- The corresponding `cardinality` and `num_dynamic_feat` values used in the training job are not set to "".

Unlike hosted endpoint inference, the configuration field is set once for the entire batch inference job using an environment variable named DEEPAR_INFERENCE_CONFIG. The value of DEEPAR_INFERENCE_CONFIG can be passed when the model is created by calling CreateTransformJob (p. 562) API. If DEEPAR_INFERENCE_CONFIG is missing in the container environment, the inference container uses the following default:

```
{
  "num_samples": 100,
  "output_types": ["mean", "quantiles"],
  "quantiles": ["0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"]
}
```

The output is also in JSON Lines format, with one line per prediction, in an order identical to the instance order in the corresponding input file. Predictions are encoded as objects identical to the ones returned by responses in online inference mode. For example:

```
{ "quantiles": { "0.1": [...], "0.2": [...] }, "samples": [...], "mean": [...] }
```

Factorization Machines Algorithm

A factorization machine is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically. For example, in a click prediction system, the factorization machine model can capture click rate patterns observed when ads from a certain ad-category are placed on pages from a certain page-category. Factorization machines are a good choice for tasks dealing with high dimensional sparse datasets, such as click prediction and item recommendation.

**Note**
The Amazon SageMaker implementation of factorization machines considers only pair-wise (2nd order) interactions between features.
Input/Output Interface for the Factorization Machines Algorithm

The factorization machine algorithm can be run in either in binary classification mode or regression mode. In each mode, a dataset can be provided to the test channel along with the train channel dataset. In regression mode, the testing dataset is scored using Root Mean Square Error (RMSE). In binary classification mode, the test dataset is scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold =0.5).

The factorization machines algorithm currently supports training only on the recordIO-protobuf format with Float32 tensors. Because their use case is predominantly on sparse data, CSV is not a good candidate. Both File and Pipe mode training are supported for recordIO-wrapped protobuf.

For inference, factorization machines support the application/json and x-recordio-protobuf formats. For binary classification models, both the score and the predicted label are returned. For regression, just the score is returned.

Please see Factorization Machines Sample Notebooks (p. 97) for more details on training and inference file formats.

EC2 Instance Recommendation for the Factorization Machines Algorithm

The Amazon SageMaker Factorization Machines algorithm is highly scalable and can train across distributed instances. We recommend training and inference with CPU instances for both sparse and dense datasets. In some circumstances, training with one or more GPUs on dense data might provide some benefit. Training with GPUs is available only on dense data. Use CPU instances for sparse data.

Factorization Machines Sample Notebooks

For a sample notebook that uses the Amazon SageMaker factorization machine learning algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Factorization Machines with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

How Factorization Machines Work

The prediction task for a factorization machine model is to estimate a function ŷ from a feature set xᵢ to a target domain. This domain is real-valued for regression and binary for classification. The factorization machine model is supervised and so has a training dataset (xᵢ,yᵢ) available. The advantages this model presents lie in the way it uses a factorized parametrization to capture the pairwise feature interactions. It can be represented mathematically as follows:
\[ \hat{y} = w_0 + \sum_i w_i x_i + \sum_i \sum_{j \neq i} <v_i, v_j> x_i x_j \]

The three terms in this equation correspond respectively to the three components of the model:

- The \( w_0 \) term represents the global bias.
- The \( w_i \) linear terms model the strength of the \( i^{th} \) variable.
- The \( <v_i, v_j> \) factorization terms model the pairwise interaction between the \( i^{th} \) and \( j^{th} \) variable.

The global bias and linear terms are the same as in a linear model. The pairwise feature interactions are modeled in the third term as the inner product of the corresponding factors learned for each feature. Learned factors can also be considered as embedding vectors for each feature. For example, in a classification task, if a pair of features tends to co-occur more often in positive labeled samples, then the inner product of their factors would be large. In other words, their embedding vectors would be close to each other in cosine similarity. For more information about the factorization machine model, see Factorization Machines.

For regression tasks, the model is trained by minimizing the squared error between the model prediction \( \hat{y}_n \) and the target value \( y_n \). This is known as the square loss:

\[ L = \frac{1}{N} \sum_n (y_n - \hat{y}_n)^2 \]

For a classification task, the model is trained by minimizing the cross entropy loss, also known as the log loss:

\[ L = \frac{1}{N} \sum_n [y_n \log \hat{p}_n + (1 - y_n) \log (1 - \hat{p}_n)] \]

where:

\[ \hat{p}_n = \frac{1}{1 + e^{-\hat{y}_n}} \]

For more information about loss functions for classification, see Loss functions for classification.

**Factorization Machines Hyperparameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The dimension of the input feature space. This could be very high with sparse input.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer. Suggested value range: [100000, 100000000]</td>
</tr>
<tr>
<td>num_factors</td>
<td>The dimensionality of factorization.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer. Suggested value range: [2, 1000], 64 is usually optimal.</td>
</tr>
<tr>
<td>predictor_type</td>
<td>The type of predictor.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>binary_classifier</strong></td>
<td>For binary classification tasks.</td>
</tr>
<tr>
<td><strong>regressor</strong></td>
<td>For regression tasks.</td>
</tr>
</tbody>
</table>

**Required**

Valid values: String: binary_classifier or regressor

<table>
<thead>
<tr>
<th><strong>bias_init_method</strong></th>
<th>The initialization method for the bias term:</th>
</tr>
</thead>
<tbody>
<tr>
<td>• normal</td>
<td>Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <strong>bias_init_sigma</strong>.</td>
</tr>
<tr>
<td>• uniform</td>
<td>Initializes weights with random values uniformly sampled from a range specified by [-<strong>bias_init_scale</strong>, +<strong>bias_init_scale</strong>].</td>
</tr>
<tr>
<td>• constant</td>
<td>Initializes the weights to a scalar value specified by <strong>bias_init_value</strong>.</td>
</tr>
</tbody>
</table>

**Optional**

Valid values: uniform, normal, or constant

Default value: normal

<table>
<thead>
<tr>
<th><strong>bias_init_scale</strong></th>
<th>Range for initialization of the bias term. Takes effect if <strong>bias_init_method</strong> is set to uniform.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
</tbody>
</table>

Default value: None

<table>
<thead>
<tr>
<th><strong>bias_init_sigma</strong></th>
<th>The standard deviation for initialization of the bias term. Takes effect if <strong>bias_init_method</strong> is set to normal.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
</tbody>
</table>

Default value: 0.01

<table>
<thead>
<tr>
<th><strong>bias_init_value</strong></th>
<th>The initial value of the bias term. Takes effect if <strong>bias_init_method</strong> is set to constant.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Optional</strong></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512].</td>
</tr>
</tbody>
</table>

Default value: None
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| bias_lr        | The learning rate for the bias term.  
**Optional**  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.1 |
| bias_wd        | The weight decay for the bias term.  
**Optional**  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.01 |
| clip_gradient  | Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval \([-\text{clip}\_\text{gradient}, +\text{clip}\_\text{gradient}]\).  
**Optional**  
Valid values: Float  
Default value: None |
| epochs         | The number of training epochs to run.  
**Optional**  
Valid values: Positive integer  
Default value: 1 |
| eps            | Epsilon parameter to avoid division by 0.  
**Optional**  
Valid values: Float. Suggested value: small.  
Default value: None |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>factors_init_method</td>
<td>The initialization method for factorization terms:&lt;br&gt;  - normal Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by factors_init_sigma.&lt;br&gt;  - uniform: Initializes weights with random values uniformly sampled from a range specified by [-factors_init_scale, +factors_init_scale].&lt;br&gt;  - constant: Initializes the weights to a scalar value specified by factors_init_value.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform, normal, or constant.</td>
</tr>
<tr>
<td></td>
<td>Default value: normal</td>
</tr>
<tr>
<td>factors_init_scale</td>
<td>The range for initialization of factorization terms. Takes effect if factors_init_method is set to uniform.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>factors_init_sigma</td>
<td>The standard deviation for initialization of factorization terms. Takes effect if factors_init_method is set to normal.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>factors_init_value</td>
<td>The initial value of factorization terms. Takes effect if factors_init_method is set to constant.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>factors_lr</td>
<td>The learning rate for factorization terms.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| factors_wd       | The weight decay for factorization terms.  
  **Optional**  
  Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
  Default value: 0.00001 |
| linear_lr        | The learning rate for linear terms.  
  **Optional**  
  Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
  Default value: 0.001 |
| linear_init_method | The initialization method for linear terms:  
  • normal Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by `linear_init_sigma`.  
  • uniform Initializes weights with random values uniformly sampled from a range specified by [-`linear_init_scale`, +`linear_init_scale`].  
  • constant Initializes the weights to a scalar value specified by `linear_init_value`.  
  **Optional**  
  Valid values: uniform, normal, or constant.  
  Default value: normal |
| linear_init_scale | Range for initialization of linear terms. Takes effect if `linear_init_method` is set to uniform.  
  **Optional**  
  Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
  Default value: None |
| linear_init_sigma | The standard deviation for initialization of linear terms. Takes effect if `linear_init_method` is set to normal.  
  **Optional**  
  Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
  Default value: 0.01 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_init_value</td>
<td>The initial value of linear terms. Takes effect if linear_init_method is set to constant.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
<tr>
<td>linear_wd</td>
<td>The weight decay for linear terms.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The size of mini-batch used for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</td>
</tr>
<tr>
<td>rescale_grad</td>
<td>Gradient rescaling optimizer parameter. If set, multiplies the gradient with rescale_grad before updating. Often choose to be 1.0/batch_size.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float</td>
</tr>
<tr>
<td></td>
<td>Default value: None</td>
</tr>
</tbody>
</table>

### Tune a Factorization Machines Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 290).

### Metrics Computed by the Factorization Machines Algorithm

The factorization machines algorithm has both binary classification and regression predictor types. The predictor type determines which metric you can use for automatic model tuning. The algorithm reports a test:rmse regressor metric, which is computed during training. When tuning the model for regression tasks, choose this metric as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:rmse</td>
<td>Root Mean Square Error</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
The factorization machines algorithm reports three binary classification metrics, which are computed during training. When tuning the model for binary classification tasks, choose one of these as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:binary_classification_accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:binary_classification_cross_entropy</td>
<td>Cross Entropy</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:binary_f_beta</td>
<td>Beta</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable Factorization Machines Hyperparameters

You can tune the following hyperparameters for the factorization machines algorithm. The initialization parameters that contain the terms bias, linear, and factorization depend on their initialization method. There are three initialization methods: uniform, normal, and constant. These initialization methods are not themselves tunable. The parameters that are tunable are dependent on this choice of the initialization method. For example, if the initialization method is uniform, then only the scale parameters are tunable. Specifically, if bias_init_method==uniform, then bias_init_scale, linear_init_scale, and factors_init_scale are tunable. Similarly, if the initialization method is normal, then only sigma parameters are tunable. If the initialization method is constant, then only value parameters are tunable. These dependencies are listed in the following table.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>bias_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>bias_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>bias_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>bias_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>epoch</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 1000</td>
<td>None</td>
</tr>
<tr>
<td>factors_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>factors_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>factors_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>factors_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
<td>Recommended Ranges</td>
<td>Dependency</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------</td>
<td>-------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>factors_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>linear_init_scale</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==uniform</td>
</tr>
<tr>
<td>linear_init_sigma</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==normal</td>
</tr>
<tr>
<td>linear_init_value</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>bias_init_method==constant</td>
</tr>
<tr>
<td>linear_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>linear_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRange</td>
<td>MinValue: 100, MaxValue: 10000</td>
<td>None</td>
</tr>
</tbody>
</table>

**Factorization Machine Response Formats**

**JSON Response Format**

Binary classification

```javascript
let response = {
  "predictions": [
    {
      "score": 0.4,
      "predicted_label": 0
    }
  ]
}
```

Regression

```javascript
let response = {
  "predictions": [
    {
      "score": 0.4
    }
  ]
}
```

**JSONLINES Response Format**

Binary classification

```javascript
{"score": 0.4, "predicted_label": 0}
```

Regression
Image Classification Algorithm

The Amazon SageMaker image classification algorithm is a supervised learning algorithm that takes an image as input and classifies it into one of multiple output categories. It uses a convolutional neural network (ResNet) that can be trained from scratch, or trained using transfer learning when a large number of training images are not available.

The recommended input format for the Amazon SageMaker image classification algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format.

Note
To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

For more information on convolutional networks, see:

- Deep residual learning for image recognition Kaiming He, et al., 2016 IEEE Conference on Computer Vision and Pattern Recognition
- ImageNet image database
- Image classification in MXNet

Topics
Input/Output Interface for the Image Classification Algorithm

The Amazon SageMaker Image Classification algorithm supports both RecordIO (application/x-recordio) and image (image/png, image/jpeg, and application/x-image) content types for training in file mode and supports RecordIO (application/x-recordio) content type for training in pipe mode. However, you can also train in pipe mode using the image files (image/png, image/jpeg, and application/x-image), without creating RecordIO files, by using the augmented manifest format. The algorithm supports image/png, image/jpeg, and application/x-image for inference.

Train with RecordIO Format

If you use the RecordIO format for training, specify both train and validation channels as values for the InputDataConfig parameter of the CreateTrainingJob (p. 556) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to application/x-recordio.

Train with Image Format

If you use the Image format for training, specify train, validation, train_lst, and validation_lst channels as values for the InputDataConfig parameter of the CreateTrainingJob (p. 556) request. Specify the individual image data (.jpg or .png files) for the train and validation channels. Specify one .lst file in each of the train_lst and validation_lst channels. Set the content type for all four channels to application/x-image.

A .lst file is a tab-separated file with three columns that contains a list of image files. The first column specifies the image index, the second column specifies the class label index for the image, and the third column specifies the relative path of the image file. The image index in the first column must be unique across all of the images. The set of class label indices are numbered successively and the numbering should start with 0. For example, 0 for the cat class, 1 for the dog class, and so on for additional classes.

The following is an example of a .lst file:

```
5     1   your_image_directory/train_img_dog1.jpg
1000  0   your_image_directory/train_img_cat1.jpg
22    1   your_image_directory/train_img_dog2.jpg
```

For example, if your training images are stored in s3://<your_bucket>/train/class_dog, s3://<your_bucket>/train/class_cat, and so on, specify the path for your train channel as s3://<your_bucket>/train, which is the top-level directory for your data. In the .lst file, specify the relative path for an individual file named train_image_dog1.jpg in the class_dog class directory as class_dog/train_image_dog1.jpg. You can also store all your image files under one subdirectory inside the train directory. In that case, use that subdirectory for the relative path. For example, s3://<your_bucket>/train/<your_image_directory>.

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the InputDataConfig parameter of the
Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a deep learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.

- **HyperParameters** - Specify these algorithm-specific parameters to influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

- **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.

- **OutputDataConfig** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results of model training.

- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.

- **StoppingCondition** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see How It Works.

**Request Syntax**

```json
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableNetworkIsolation": boolean,
    "HyperParameters": {
        "string": "string"
    }
}
```
<table>
<thead>
<tr>
<th>Request Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For information about the parameters that are common to all actions, see Common Parameters (p. 917).</strong></td>
</tr>
<tr>
<td><strong>The request accepts the following data in JSON format.</strong></td>
</tr>
<tr>
<td><strong>AlgorithmSpecification (p. 556)</strong></td>
</tr>
<tr>
<td>The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.</td>
</tr>
<tr>
<td><strong>Type:</strong> AlgorithmSpecification (p. 744) object</td>
</tr>
<tr>
<td><strong>Required:</strong> Yes</td>
</tr>
<tr>
<td><strong>EnableInterContainerTrafficEncryption (p. 556)</strong></td>
</tr>
<tr>
<td>To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.</td>
</tr>
<tr>
<td><strong>Type:</strong> Boolean</td>
</tr>
</tbody>
</table>

```json
"S3Uri": "string"
}

"InputMode": "string",

"RecordWrapperType": "string",

"ShuffleConfig": {

  "Seed": number

}

"OutputDataConfig": {

  "KmsKeyId": "string",

  "S3OutputPath": "string"

},

"ResourceConfig": {

  "InstanceCount": number,

  "InstanceType": "string",

  "VolumeKmsKeyId": "string",

  "VolumeSizeInGB": number

},

"RoleArn": "string",

"StoppingCondition": {

  "MaxRuntimeInSeconds": number

},

"Tags": [

  {

    "Key": "string",

    "Value": "string"

  }

],

"TrainingJobName": "string",

"VpcConfig": {

}
Required: No

EnableNetworkIsolation (p. 556)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note
The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean
Required: No

HyperParameters (p. 556)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

InputDataConfig (p. 556)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3 location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

OutputDataConfig (p. 556)

Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object
Required: Yes

ResourceConfig (p. 556)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 866) object

Required: Yes

RoleArn (p. 556)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-z-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@-_]+\$`

Required: Yes

StoppingCondition (p. 556)

Sets a duration for training. Use this parameter to cap model training costs. To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the CreateModel API.

Type: StoppingCondition (p. 879) object

Required: Yes

Tags (p. 556)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**TrainingJobName (p. 556)**

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: `^[a-zA-Z0-9](\-*[a-zA-Z0-9])*`  

Required: Yes

**VpcConfig (p. 556)**

A VpcConfig (p. 913) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

**Response Syntax**

```json
{
    "TrainingJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrainingJobArn (p. 560)**

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400
ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(p. ) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the 'source-ref' tag that points to the S3 location of the image. The annotations are provided under the "AttributeName" parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a deep learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.
- **HyperParameters** - Specify these algorithm-specific parameters to influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results of model training.
- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
• **StoppingCondition** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see How It Works.

**Request Syntax**

```json
{
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
    },
    "EnableInterContainerTrafficEncryption": boolean,
    "EnableNetworkIsolation": boolean,
    "HyperParameters": {
        "string": "string"
    },
    "InputDataConfig": [
        {
            "ChannelName": "string",
            "CompressionType": "string",
            "ContentType": "string",
            "DataSource": {
                "S3DataSource": {
                    "AttributeNames": [ "string" ],
                    "S3DataDistributionType": "string",
                    "S3DataType": "string",
                    "S3Uri": "string"
                }
            }
        }
    ],
    "OutputDataConfig": {
        "KmsKeyId": "string",
        "S3OutputPath": "string"
    },
    "ResourceConfig": {
        "InstanceCount": number,
        "InstanceType": "string",
        "VolumeKmsKeyId": "string",
        "VolumeSizeInGB": number
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number
    },
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```
"TrainingJobName": "string",

"VpcConfig": {
  "SecurityGroupIds": ["string"],
  "Subnets": ["string"]
}
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 556)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 744) object

Required: Yes

EnableInterContainerTrafficEncryption (p. 556)

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 556)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note

The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 556)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 10 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map
Key Length Constraints: Maximum length of 256.
Value Length Constraints: Maximum length of 256.

Required: No

**InputDataConfig (p. 556)**

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3 location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

**OutputDataConfig (p. 556)**

Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object

Required: Yes

**ResourceConfig (p. 556)**

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 866) object

Required: Yes

**RoleArn (p. 556)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.
Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]++$

Required: Yes

StoppingCondition (p. 556)

Sets a duration for training. Use this parameter to cap model training costs. To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the CreateModel API.

Type: StoppingCondition (p. 879) object

Required: Yes

Tags (p. 556)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 556)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9-]+(-*[a-zA-Z0-9-]+)*$

Required: Yes

VpcConfig (p. 556)

A VpcConfig (p. 913) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

Response Syntax

```
{
    "TrainingJobArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 560)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:training-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(request) It can also contain additional metadata under the metadata tag, but these are ignored by the algorithm. In the following example, the “AttributeName” is “class” and the corresponding label value is “0” for the first image and “1” for the second image:

```json
{"source-ref":"s3://image/filename1.jpg", "class":"0"}
{"source-ref":"s3://image/filename2.jpg", "class":"1", "class-metadata": {"class-name": "cat", "type": "groundtruth/image-classification"}}
```
For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. Amazon SageMaker image classification models can be seeded only with another build-in image classification model trained in Amazon SageMaker.

To use a pretrained model, in the CreateTrainingJob (p. 556) request, specify the ChannelName as "model" in the InputDataConfig parameter. Set the ContentType for the model channel to application/x-sagemaker-model. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the num_layers, image_shape and num_classes input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker image classification algorithm, see the End-to-End Incremental Training Image Classification Example. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 285).

Inference with the Image Format Algorithm

The generated models can be hosted for inference and support encoded .jpg and .png image formats as image/png, image/jpeg, and application/x-image content-type. The output is the probability values for all classes encoded in JSON format, or in JSON Lines text format for batch transform. The image classification model processes a single image per request and so outputs only one line in the JSON or JSON Lines format. The following is an example of a response in JSON Lines format:

```json
accept: application/jsonlines

{"prediction": [prob_0, prob_1, prob_2, prob_3, ...]}
```

For more details on training and inference, see the image classification sample notebook instances referenced in the introduction.

EC2 Instance Recommendation for the Image Classification Algorithm

For image classification, we support the following GPU instances for training: ml.p2.xlarge, ml.p2.8xlarge, ml.p2.16xlarge, ml.p3.2xlarge, ml.p3.8xlarge and ml.p3.16xlarge. We recommend using GPU instances with more memory for training with large batch sizes. However, both CPU (such as C4) and GPU (such as P2 and P3) instances can be used for the inference. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training.

Both P2 and P3 instances are supported in the image classification algorithm.

Image Classification Sample Notebooks

For a sample notebook that uses the Amazon SageMaker image classification algorithm to train a model on the caltech-256 dataset and then to deploy it to perform inferences, see the End-to-End Multiclass Image Classification Example. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list.
of all the Amazon SageMaker samples. The example image classification notebooks are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

How Image Classification Works

The image classification algorithm takes an image as input and classifies it into one of the output categories. Deep learning has revolutionized the image classification domain and has achieved great performance. Various deep learning networks such as ResNet [1], DenseNet, inception, and so on, have been developed to be highly accurate for image classification. At the same time, there have been efforts to collect labeled image data that are essential for training these networks. ImageNet[2] is one such large dataset that has more than 11 million images with about 11,000 categories. Once a network is trained with ImageNet data, it can then be used to generalize with other datasets as well, by simple re-adjustment or fine-tuning. In this transfer learning approach, a network is initialized with weights (in this example, trained on ImageNet), which can be later fine-tuned for an image classification task in a different dataset.

Image classification in Amazon SageMaker can be run in two modes: full training and transfer learning. In full training mode, the network is initialized with random weights and trained on user data from scratch. In transfer learning mode, the network is initialized with pre-trained weights and just the top fully connected layer is initialized with random weights. Then, the whole network is fine-tuned with new data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data.

Image Classification Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes</td>
<td>Number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>num_training_samples</td>
<td>Number of training examples in the input dataset.</td>
</tr>
<tr>
<td></td>
<td>If there is a mismatch between this value and the number of samples in the training set, then the behavior of the lr_scheduler_step parameter is undefined and distributed training accuracy might be affected.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>augmentation_type</td>
<td>Data augmentation type. The input images can be augmented in multiple ways as specified below.</td>
</tr>
<tr>
<td></td>
<td>• crop: Randomly crop the image and flip the image horizontally</td>
</tr>
<tr>
<td></td>
<td>• crop_color: In addition to ‘crop’, three random values in the range [-36, 36], [-50, 50], and [-50, 50] are added to the corresponding Hue-Saturation-Lightness channels respectively</td>
</tr>
<tr>
<td></td>
<td>• crop_color_transform: In addition to crop_color, random transformations, including rotation, shear, and aspect ratio variations are applied to the image. The maximum angle of</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>rotation is 10 degrees, the maximum shear ratio is 0.1, and the maximum aspect changing ratio is 0.25.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: crop, crop_color, or crop_color_transform.</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>beta_1</td>
<td>The beta1 for adam, that is the exponential decay rate for the first moment estimates.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>beta_2</td>
<td>The beta2 for adam, that is the exponential decay rate for the second moment estimates.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.999</td>
</tr>
<tr>
<td>checkpoint_frequency</td>
<td>Period to store model parameters (in number of epochs).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer no greater than epochs.</td>
</tr>
<tr>
<td></td>
<td>Default value: None (Save checkpoint at the epoch that has the best validation accuracy.)</td>
</tr>
<tr>
<td>early_stopping</td>
<td>True to use early stopping logic during training. False not to use it.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>early_stopping_min_epochs</td>
<td>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of epochs to wait before ending training if no improvement is made in the relevant metric. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>Relative tolerance to measure an improvement in accuracy validation metric. If the ratio of the improvement in accuracy divided by the previous best accuracy is smaller than the early_stopping_tolerance value set, early stopping considers there is no improvement. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>epochs</td>
<td>Number of training epochs.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 30</td>
</tr>
<tr>
<td>eps</td>
<td>The epsilon for adam and rmsprop. It is usually set to a small value to avoid division by 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 1e-8</td>
</tr>
<tr>
<td>gamma</td>
<td>The gamma for rmsprop, the decay factor for the moving average of the squared gradient.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>image_shape</td>
<td>The input image dimensions, which is the same size as the input layer of the network. The format is defined as 'num_channels, height, width'. The image dimension can take on any value as the network can handle varied dimensions of the input. However, there may be memory constraints if a larger image dimension is used. Typical image dimensions for image classification are '3, 224, 224'. This is similar to the ImageNet dataset.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: '3, 224, 224'</td>
</tr>
<tr>
<td>kv_store</td>
<td>Weight update synchronization mode during distributed training. The weight updates can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See distributed training in MXNet for more details. This parameter is not applicable to single machine training.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: dist_sync or dist_async</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Initial learning rate.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>lr_scheduler_factor</td>
<td>The ratio to reduce learning rate used in conjunction with the lr_scheduler_step parameter, defined as lr_new = lr_old * lr_scheduler_factor.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lr_scheduler_step</td>
<td>The epochs at which to reduce the learning rate. As explained in the lr_scheduler_factor parameter, the learning rate is reduced by lr_scheduler_factor at these epochs. For example, if the value is set to “10, 20”, then the learning rate is reduced by lr_scheduler_factor after 10th epoch and again by lr_scheduler_factor after 20th epoch. The epochs are delimited by “,”.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. In a single-machine multi-GPU setting, each GPU handles mini_batch_size/num_gpu training samples. For the multi-machine training in dist_sync mode, the actual batch size is mini_batch_size*number of machines. See MXNet docs for more details.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 32</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for sgd and nag, ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>multi_label</td>
<td>Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num_layers</td>
<td>Number of layers for the network. For data with large image size (for example, 224x224 - like ImageNet), we suggest selecting the number of layers from the set [18, 34, 50, 101, 152, 200]. For data with small image size (for example, 28x28 - like CIFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200].</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110]</td>
</tr>
<tr>
<td></td>
<td>Default value: 152</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: One of sgd, adam, rmsprop, or nag.</td>
</tr>
<tr>
<td></td>
<td>• sgd: Stochastic gradient descent</td>
</tr>
<tr>
<td></td>
<td>• adam: Adaptive momentum estimation</td>
</tr>
<tr>
<td></td>
<td>• rmsprop: Root mean square propagation</td>
</tr>
<tr>
<td></td>
<td>• nag: Nesterov accelerated gradient</td>
</tr>
<tr>
<td></td>
<td>Default value: sgd</td>
</tr>
<tr>
<td>precision_dtype</td>
<td>The precision of the weights used for training. The algorithm can use either single precision (float32) or half precision (float16) for the weights. Using half-precision for weights results in reduced memory consumption.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float32 or float16</td>
</tr>
<tr>
<td></td>
<td>Default value: float32</td>
</tr>
<tr>
<td>resize</td>
<td>Resizes the image before using it for training. The images are resized so that the shortest side has the number of pixels specified by this parameter. If the parameter is not set, then the training data is used without resizing.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>top_k</td>
<td>Reports the top-k accuracy during training. This parameter has to be greater than 1, since the top-1 training accuracy is the same as the regular training accuracy that has already been reported.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer larger than 1.</td>
</tr>
<tr>
<td></td>
<td>Default value: no default value</td>
</tr>
<tr>
<td>use_pretrained_model</td>
<td>Flag to use pre-trained model for training. If set to 1, then the pretrained model with the corresponding number of layers is loaded and used for training. Only the top FC layer are reinitialized with random weights. Otherwise, the network is trained from scratch.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>use_weighted_loss</td>
<td>Flag to use weighted cross-entropy loss for multi-label classification (used only when multi_label = 1), where the weights are calculated based on the distribution of classes.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The coefficient weight decay for sgd and nag, ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
</tbody>
</table>

**Tune an Image Classification Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 290).

**Metrics Computed by the Image Classification Algorithm**

The image classification algorithm is a supervised algorithm. It reports an accuracy metric that is computed during training. When tuning the model, choose this metric as the objective metric.
### Tunable Image Classification Hyperparameters

Tune an image classification model with the following hyperparameters. The hyperparameters that have the greatest impact on image classification objective metrics are: mini\_batch\_size, learning\_rate, and optimizer. Tune the optimizer-related hyperparameters, such as momentum, weight\_decay, beta\_1, beta\_2, eps, and gamma, based on the selected optimizer. For example, use beta\_1 and beta\_2 only when adam is the optimizer.

For more information about which hyperparameters are used in each optimizer, see Image Classification Hyperparameters (p. 120).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta_1</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.999</td>
</tr>
<tr>
<td>beta_2</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.999</td>
</tr>
<tr>
<td>eps</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-8, MaxValue: 1.0</td>
</tr>
<tr>
<td>gamma</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-8, MaxValue: 0.999</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.5</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 512</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'rmsprop', 'nag']</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
</tbody>
</table>

### IP Insights Algorithm

Amazon SageMaker IP Insights is an unsupervised learning algorithm that learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers. You can use it to identify a user attempting to log into a web service from an anomalous IP address, for example. Or you can use it to identify an account that is attempting to create computing resources from an unusual IP address. Trained IP Insight models can be hosted at an endpoint for making real-time predictions or used for processing batch transforms.

Amazon SageMaker IP insights ingests historical data as (entity, IPv4 Address) pairs and learns the IP usage patterns of each entity. When queried with an (entity, IPv4 Address) event, an Amazon SageMaker
IP Insights model returns a score that infers how anomalous the pattern of the event is. For example, when a user attempts to log in from an IP address, if the IP Insights score is high enough, a web login server might decide to trigger a multi-factor authentication system. In more advanced solutions, you can feed the IP Insights score into another machine learning model. For example, you can combine the IP Insight score with other features to rank the findings of another security system, such as those from Amazon GuardDuty.

The Amazon SageMaker IP Insights algorithm can also learn vector representations of IP addresses, known as embeddings. You can use vector-encoded embeddings as features in downstream machine learning tasks that use the information observed in the IP addresses. For example, you can use them in tasks such as measuring similarities between IP addresses in clustering and visualization tasks.

Topics
- Input/Output Interface for the IP Insights Algorithm (p. 128)
- EC2 Instance Recommendation for the IP Insights Algorithm (p. 128)
- IP Insights Sample Notebooks (p. 129)
- How IP Insights Works (p. 129)
- IP Insights Hyperparameters (p. 130)
- Tune an IP Insights Model (p. 133)
- IP Insights Data Formats (p. 135)

Input/Output Interface for the IP Insights Algorithm

Training and Validation

The Amazon SageMaker IP Insights algorithm supports training and validation data channels. It uses the optional validation channel to compute an area-under-curve (AUC) score on a predefined negative sampling strategy. The AUC metric validates how well the model discriminates between positive and negative samples. Training and validation data content types need to be in text/csv format. The first column of the CSV data is an opaque string that provides a unique identifier for the entity. The second column is an IPv4 address in decimal-dot notation. IP Insights currently supports only File mode. For more information and some examples, see IP Insights Training Data Formats (p. 135).

Inference

For inference, IP Insights supports text/csv, application/json, and application/jsonlines data content types. For more information about the common data formats for inference provided by Amazon SageMaker, see Common Data Formats for Inference (p. 67). IP Insights inference returns output formatted as either application/json or application/jsonlines. Each record in the output data contains the corresponding dot_product (or compatibility score) for each input data point. For more information and some examples, see IP Insights Inference Data Formats (p. 135).

EC2 Instance Recommendation for the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm can run on both GPU and CPU instances. For training jobs, we recommend using GPU instances. However, for certain workloads with large training datasets, distributed CPU instances might reduce training costs. For inference, we recommend using CPU instances.

GPU Instances for the IP Insights Algorithm

IP Insights supports all available GPUs. If you need to speed up training, we recommend starting with a single GPU instance, such as ml.p3.2xlarge, and then moving to a multi-GPU environment, such as ml.p3.8xlarge and ml.p3.16xlarge. Multi-GPUs automatically divide the mini batches of training data across themselves. If you switch from a single GPU to multiple GPUs, the mini_batch_size is divided
equally into the number of GPUs used. You may want to increase the value of the mini_batch_size to compensate for this.

**CPU Instances for the IP Insights Algorithm**

The type of CPU instance that we recommend depends largely on the instance’s available memory and the model size. The model size is determined by two hyperparameters: vector_dim and num_entity_vectors. The maximum supported model size is 8 GB. The following table lists typical EC2 instance types that you would deploy based on these input parameters for various model sizes. In Table 1, the value for vector_dim in the first column range from 32 to 2048 and the values for num_entity_vectors in the first row range from 10,000 to 50,000,000.

<table>
<thead>
<tr>
<th>vector_dim \ num_entity</th>
<th>10,000</th>
<th>50,000</th>
<th>100,000</th>
<th>500,000</th>
<th>1,000,000</th>
<th>5,000,000</th>
<th>10,000,000</th>
<th>50,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.4xlarge</td>
</tr>
<tr>
<td>64</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.4xlarge</td>
</tr>
<tr>
<td>128</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.4xlarge</td>
</tr>
<tr>
<td>256</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.4xlarge</td>
</tr>
<tr>
<td>512</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.2xlarge</td>
<td>ml.m5.xlarge</td>
</tr>
<tr>
<td>1024</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.xlarge</td>
</tr>
<tr>
<td>2048</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.large</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.xlarge</td>
<td>ml.m5.xlarge</td>
</tr>
</tbody>
</table>

The values for the mini_batch_size, num_ip_encoder_layers, random_negative_sampling_rate, and shuffled_negative_sampling_rate hyperparameters also affect the amount of memory required. If these values are large, you might need to use a larger instance type than normal.

**IP Insights Sample Notebooks**

For a sample notebook that shows how to train the Amazon SageMaker IP Insights algorithm and perform inferences with it, see [An Introduction to the Amazon SageMaker IP Insights Algorithm](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Notebook Instances](#). After creating a notebook instance, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker examples. To open a notebook, choose its Use tab and choose Create copy.

**How IP Insights Works**

Amazon SageMaker IP Insights is an unsupervised algorithm that consumes observed data in the form of (entity, IPv4 address) pairs that associates entities with IP addresses. IP Insights determines how likely it is that an entity would use a particular IP address by learning latent vector representations for both entities and IP addresses. The distance between these two representations can then serve as the proxy for how likely this association is.

The IP Insights algorithm uses a neural network to learn the latent vector representations for entities and IP addresses. Entities are first hashed to a large but fixed hash space and then encoded by a simple embedding layer. Character strings such as user names or account IDs can be fed directly into IP Insights as they appear in log files. You don’t need to preprocess the data for entity identifiers. You can provide entities as an arbitrary string value during both training and inference. The hash size should be configured with a value that is high enough to insure that the number of collisions, which occur
when distinct entities are mapped to the same latent vector, remain insignificant. For more information about how to select appropriate hash sizes, see Feature Hashing for Large Scale Multitask Learning. For representing IP addresses, on the other hand, IP Insights uses a specially designed encoder network to uniquely represent each possible IPv4 address by exploiting the prefix structure of IP addresses.

During training, IP Insights automatically generates negative samples by randomly pairing entities and IP addresses. These negative samples represent data that is less likely to occur in reality. The model is trained to discriminate between positive samples that are observed in the training data and these generated negative samples. More specifically, the model is trained to minimize the cross entropy, also known as the \( \log \text{ loss} \), defined as follows:

\[
L = \frac{1}{N} \sum_{n} [y_n \log p_n + (1 - y_n) \log (1 - p_n)]
\]

\( y_n \) is the label that indicates whether the sample is from the real distribution governing observed data \( (y_{n=1}) \) or from the distribution generating negative samples \( (y_{n=0}) \). \( p_n \) is the probability that the sample is from the real distribution, as predicted by the model.

Generating negative samples is an important process that is used to achieve an accurate model of the observed data. If negative samples are extremely unlikely, for example, if all of the IP addresses in negative samples are 10.0.0.0, then the model trivially learns to distinguish negative samples and fails to accurately characterize the actual observed dataset. To keep negative samples more realistic, IP Insights generates negative samples both by randomly generating IP addresses and randomly picking IP addresses from training data. You can configure the type of negative sampling and the rates at which negative samples are generated with the \textit{random_negative_sampling_rate} and \textit{shuffled_negative_sampling_rate} hyperparameters.

Given an \( nth \) (entity, IP address) pair, the IP Insights model outputs a score, \( S_n \), that indicates how compatible the entity is with the IP address. This score corresponds to the log odds ratio for a given (entity, IP address) of the pair coming from a real distribution as compared to coming from a negative distribution. It is defined as follows:

\[
S_n = \log \left( \frac{P_{\text{real}}(n)}{P_{\text{neg}}(n)} \right)
\]

The score is essentially a measure of the similarity between the vector representations of the \( nth \) entity and IP address. It can be interpreted as how much more likely it would be to observe this event in reality than in a randomly generated dataset. During training, the algorithm uses this score to calculate an estimate of the probability of a sample coming from the real distribution, \( p_n \), to use in the cross entropy minimization, where:

\[
p_n = \frac{1}{1 + e^{-S_n}}
\]

**IP Insights Hyperparameters**

In the CreateTransformJob (p. 562) request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker IP Insights algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_entity_vectors</td>
<td>The number of entity vector representations (entity embedding vectors) to train. Each entity in the training set is randomly assigned to one of these vectors using a hash function. Because of hash collisions, it might be possible to have multiple entities assigned to the same vector. This would cause the same vector to represent multiple entities. This generally has a negligible effect on</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>model performance, as long as the collision rate is not too severe. To keep the collision rate low, set this value as high as possible. However, the model size, and, therefore, the memory requirement, for both training and inference, scales linearly with this hyperparameter. We recommend that you set this value to twice the number of unique entity identifiers.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 ≤ positive integer ≤ 250,000,000</td>
</tr>
<tr>
<td>vector_dim</td>
<td>The size of embedding vectors to represent entities and IP addresses. The larger the value, the more information that can be encoded using these representations. In practice, model size scales linearly with this parameter and limits how large the dimension can be. In addition, using vector representations that are too large can cause the model to overfit, especially for small training datasets. Overfitting occurs when a model doesn't learn any pattern in the data but effectively memorizes the training data and, therefore, cannot generalize well and performs poorly during inference. The recommended value is 128.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: 4 ≤ positive integer ≤ 4096</td>
</tr>
<tr>
<td>batch_metrics_publish_interval</td>
<td>The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second).</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer ≥ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 1,000</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of passes over the training data. The optimal value depends on your data size and learning rate. Typical values range from 5 to 100.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer ≥ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate for the optimizer. IP Insights use a gradient-descent-based Adam optimizer. The learning rate effectively controls the step size to update model parameters at each iteration. Too large a learning rate can cause the model to diverge because the training is likely to overshoot a minima. On the other hand, too small a learning rate slows down convergence. Typical values range from 1e-4 to 1e-1.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: 1e-6 ≤ float ≤ 10.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The number of examples in each mini batch. The training procedure processes data in mini batches. The optimal value depends on the number of unique account identifiers in the dataset. In general, the larger the mini_batch_size, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large mini_batch_size, the training is more likely to converge to a poor local minimum and perform relatively worse for inference.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: 1 ≤ positive integer ≤ 500000</td>
</tr>
<tr>
<td></td>
<td>Default value: 10,000</td>
</tr>
<tr>
<td>num_ip_encoder_layers</td>
<td>The number of fully connected layers used to encode the IP address embedding. The larger the number of layers, the greater the model's capacity to capture patterns among IP addresses. However, using a large number of layers increases the chance of overfitting.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: 0 ≤ positive integer ≤ 100</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
**random_negative_sampling_rate** | The number of random negative samples, R, to generate per input example. The training procedure relies on negative samples to prevent the vector representations of the model collapsing to a single point. Random negative sampling generates R random IP addresses for each input account in the mini batch. The sum of the `random_negative_sampling_rate` (R) and `shuffled_negative_sampling_rate` (S) must be in the interval: $1 \leq R + S \leq 500$.
- **Optional**
- Valid values: $0 \leq$ positive integer $\leq 500$
- Default value: 1

**shuffled_negative_sampling_rate** | The number of shuffled negative samples, S, to generate per input example. In some cases, it helps to use more realistic negative samples that are randomly picked from the training data itself. This kind of negative sampling is achieved by shuffling the data within a mini batch. Shuffled negative sampling generates S negative IP addresses by shuffling the IP address and account pairings within a mini batch. The sum of the `random_negative_sampling_rate` (R) and `shuffled_negative_sampling_rate` (S) must be in the interval: $1 \leq R + S \leq 500$.
- **Optional**
- Valid values: $0 \leq$ positive integer $\leq 500$
- Default value: 1

**weight_decay** | The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data.
- **Optional**
- Valid values: $0.0 \leq$ float $\leq 10.0$
- Default value: 0.00001

### Tune an IP Insights Model

**Automatic model tuning**, also called hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 290).
Metrics Computed by the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm is an unsupervised learning algorithm that learns associations between IP addresses and entities. The algorithm trains a discriminator model, which learns to separate observed data points (positive samples) from randomly generated data points (negative samples). Automatic model tuning on IP Insights helps you find the model that can most accurately distinguish between unlabeled validation data and automatically generated negative samples. The model accuracy on the validation dataset is measured by the area under the receiver operating characteristic (ROC) curve. This validation:discriminator_auc metric can take values between 0.0 and 1.0, where 1.0 indicates perfect accuracy.

The IP Insights algorithm computes a validation:discriminator_auc metric during validation, the value of which is used as the objective function to optimize for hyperparameter tuning.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:discriminator_auc</td>
<td>Area under the ROC curve on the validation dataset. The validation dataset is not labeled. AUC is a metric that describes the model's ability to discriminate validation data points from randomly generated data points.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable IP Insights Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker IP Insights algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 100</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-4, MaxValue: 0.1</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 100, MaxValue: 50000</td>
</tr>
<tr>
<td>num_entity_vectors</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 10000, MaxValue: 1000000</td>
</tr>
<tr>
<td>num_ip_encoder_layer</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 10</td>
</tr>
<tr>
<td>random_negative_sampling</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 0, MaxValue: 10</td>
</tr>
<tr>
<td>shuffled_negative_sampling</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 0, MaxValue: 10</td>
</tr>
<tr>
<td>vector_dim</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 256</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
</tbody>
</table>
IP Insights Data Formats

This section provides examples of the available input and output data formats used by the IP Insights algorithm during training and inference.

Topics
- IP Insights Training Data Formats (p. 135)
- IP Insights Inference Data Formats (p. 135)

IP Insights Training Data Formats

The following are the available data input formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input training format described in Common Data Formats for Training (p. 64). However, the Amazon SageMaker IP Insights algorithm currently supports only the CSV data input format.

IP Insights Training Data Input Formats

INPUT: CSV

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

```
content-type: text/csv

entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

IP Insights Inference Data Formats

The following are the available input and output formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats for Inference (p. 67). However, the Amazon SageMaker IP Insights algorithm does not currently support RecordIO format.

IP Insights Input Request Formats

INPUT: CSV Format

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

```
content-type: text/csv

entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

INPUT: JSON Format

JSON data can be provided in different formats. IP Insights follows the common Amazon SageMaker formats. For more information about inference formats, see Common Data Formats for Inference (p. 67).
content-type: application/json

```
{
   "instances": [
      {
         "data": {
            "features": {
               "values": ["entity_id_1", "192.168.1.2"]
            },
            "features": {
               "values": ["entity_id_2", "10.10.1.2"]
            }
         }
      }
   ]
}
```

**INPUT: JSONLINES Format**

The JSON Lines content type is useful for running batch transform jobs. For more information on Amazon SageMaker inference formats, see Common Data Formats for Inference (p. 67). For more information on running batch transform jobs, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

content-type: application/jsonlines

```
{"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]},
{"features": ["entity_id_2", "10.10.1.2"]}}
```

**IP Insights Output Response Formats**

**OUTPUT: JSON Response Format**

The default output of the Amazon SageMaker IP Insights algorithm is the dot_product between the input entity and IP address. The dot_product signifies how compatible the model considers the entity and IP address. The dot_product is unbounded. To make predictions about whether an event is anomalous, you need to set a threshold based on your defined distribution. For information about how to use the dot_product for anomaly detection, see the An Introduction to the Amazon SageMakerIP Insights Algorithm.

accept: application/json

```
{
   "predictions": [
      {
         "dot_product": 0.0,
         "entity_embedding": [1.0, 0.0, 0.0],
         "ip_embedding": [0.0, 1.0, 0.0]
      },
      {
         "dot_product": 2.0,
         "entity_embedding": [2.0, 0.0, 0.0],
         "ip_embedding": [0.0, 2.0, 0.0]
      }
   ]
}
```

Advanced users can access the model's learned entity and IP embeddings by providing the additional content-type parameter verbose=True to the Accept heading. You can use the entity_embedding and ip_embedding for debugging, visualizing, and understanding the model. Additionally, you can use these embeddings in other machine learning techniques, such as classification or clustering.

accept: application/json;verbose=True

```
{
   "predictions": [
      {
         "dot_product": 0.0,
         "entity_embedding": [1.0, 0.0, 0.0],
         "ip_embedding": [0.0, 1.0, 0.0]
      },
      {
         "dot_product": 2.0,
         "entity_embedding": [2.0, 0.0, 0.0],
         "ip_embedding": [0.0, 2.0, 0.0]
      }
   ]
}
```

136
K-Means Algorithm

K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

Amazon SageMaker uses a modified version of the web-scale k-means clustering algorithm. Compared with the original version of the algorithm, the version used by Amazon SageMaker is more accurate. Like the original algorithm, it scales to massive datasets and delivers improvements in training time. To do this, the version used by Amazon SageMaker streams mini-batches (small, random subsets) of the training data. For more information about mini-batch k-means, see Web-scale k-means Clustering.

The k-means algorithm expects tabular data, where rows represent the observations that you want to cluster, and the columns represent attributes of the observations. The $n$ attributes in each row represent a point in $n$-dimensional space. The Euclidean distance between these points represents the similarity of the corresponding observations. The algorithm groups observations with similar attribute values (the points corresponding to these observations are closer together). For more information about how k-means works in Amazon SageMaker, see How K-Means Clustering Works (p. 138).

Topics
- Input/Output Interface for the K-Means Algorithm (p. 137)
- EC2 Instance Recommendation for the K-Means Algorithm (p. 138)
- K-Means Sample Notebooks (p. 138)
- How K-Means Clustering Works (p. 138)
- K-Means Hyperparameters (p. 141)
- Tune a K-Means Model (p. 143)
- K-Means Response Formats (p. 144)

Input/Output Interface for the K-Means Algorithm

For training, the k-means algorithm expects data to be provided in the *train* channel (recommended S3DataDistributionType=ShardedByS3Key), with an optional test channel (recommended S3DataDistributionType=FullyReplicated) to score the data on. Both recordIO-wrapped-
protobuf and CSV formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference, text/csv, application/json, and application/x-recordio-protobuf are supported. k-means returns a closest_cluster label and the distance_to_cluster for each observation.

For more information on input and output file formats, see K-Means Response Formats (p. 144) for inference and the K-Means Sample Notebooks (p. 138). The k-means algorithm does not support multiple instance learning, in which the training set consists of labeled “bags”, each of which is a collection of unlabeled instances.

**EC2 Instance Recommendation for the K-Means Algorithm**

We recommend training k-means on CPU instances. You can train on GPU instances, but should limit GPU training to p*.xlarge instances because only one GPU per instance is used.

**K-Means Sample Notebooks**

For a sample notebook that uses the Amazon SageMaker K-means algorithm to segment the population of counties in the United States by attributes identified using principle component analysis, see Analyze US census data for population segmentation using Amazon SageMaker. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its Use tab and select Create copy.

**How K-Means Clustering Works**

K-means is an algorithm that trains a model that groups similar objects together. The k-means algorithm accomplishes this by mapping each observation in the input dataset to a point in the \( n \)-dimensional space (where \( n \) is the number of attributes of the observation). For example, your dataset might contain observations of temperature and humidity in a particular location, which are mapped to points \((t, h)\) in 2-dimensional space.

**Note**

Clustering algorithms are unsupervised. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

In k-means clustering, each cluster has a center. During model training, the k-means algorithm uses the distance of the point that corresponds to each observation in the dataset to the cluster centers as the basis for clustering. You choose the number of clusters \((k)\) to create.

For example, suppose that you want to create a model to recognize handwritten digits and you choose the MNIST dataset for training. The dataset provides thousands of images of handwritten digits (0 through 9). In this example, you might choose to create 10 clusters, one for each digit (0, 1, ..., 9). As part of model training, the k-means algorithm groups the input images into 10 clusters.

Each image in the MINST dataset is a 28x28-pixel image, with a total of 784 pixels. Each image corresponds to a point in a 784-dimensional space, similar to a point in a 2-dimensional space \((x, y)\). To find a cluster to which a point belongs, the k-means algorithm finds the distance of that point from all of the cluster centers. It then chooses the cluster with the closest center as the cluster to which the image belongs.

**Note**

Amazon SageMaker uses a customized version of the algorithm where, instead of specifying that the algorithm create \(k\) clusters, you might choose to improve model accuracy by specifying extra cluster centers \((K = k'x)\). However, the algorithm ultimately reduces these to \(k\) clusters.
In Amazon SageMaker, you specify the number of clusters when creating a training job. For more information, see CreateTrainingJob (p. 556). In the request body, you add the HyperParameters string map to specify the k and extra_center_factor strings.

The following is a summary of how k-means works for model training in Amazon SageMaker:

1. It determines the initial K cluster centers.

   **Note**
   In the following topics, K clusters refer to k * x, where you specify k and x when creating a model training job.

2. It iterates over input training data and recalculates cluster centers.

3. It reduces resulting clusters to k (if the data scientist specified the creation of k*x clusters in the request).

The following sections also explain some of the parameters that a data scientist might specify to configure a model training job as part of the HyperParameters string map.

**Topics**

- Step 1: Determine the Initial Cluster Centers (p. 139)
- Step 2: Iterate over the Training Dataset and Calculate Cluster Centers (p. 140)
- Step 3: Reduce the Clusters from K to k (p. 141)

**Step 1: Determine the Initial Cluster Centers**

When using k-means in Amazon SageMaker, the initial cluster centers are chosen from the observations in a small, randomly sampled batch. Choose one of the following strategies to determine how these initial cluster centers are selected:

- The random approach—Randomly choose K observations in your input dataset as cluster centers. For example, you might choose a cluster center that points to the 784-dimensional space that corresponds to any 10 images in the MNIST training dataset.

- The k-means++ approach, which works as follows:
  1. Start with one cluster and determine its center. You randomly select an observation from your training dataset and use the point corresponding to the observation as the cluster center. For example, in the MNIST dataset, randomly choose a handwritten digit image. Then choose the point in the 784-dimensional space that corresponds to the image as your cluster center. This is cluster center 1.
  2. Determine the center for cluster 2. From the remaining observations in the training dataset, pick an observation at random. Choose one that is different than the one you previously selected. This observation corresponds to a point that is far away from cluster center 1. Using the MNIST dataset as an example, you do the following:
     - For each of the remaining images, find the distance of the corresponding point from cluster center 1. Square the distance and assign a probability that is proportional to the square of the distance. That way, an image that is different from the one that you previously selected has a higher probability of getting selected as cluster center 2.
     - Choose one of the images randomly, based on probabilities assigned in the previous step. The point that corresponds to the image is cluster center 2.
  3. Repeat Step 2 to find cluster center 3. This time, find the distances of the remaining images from cluster center 2.
  4. Repeat the process until you have the K cluster centers.
To train a model in Amazon SageMaker, you create a training job. In the request, you provide configuration information by specifying the following HyperParameters string maps:

- To specify the number of clusters to create, add the k string.
- For greater accuracy, add the optional extra_center_factor string.
- To specify the strategy that you want to use to determine the initial cluster centers, add the init_method string and set its value to random or k-means++.

For more information, see CreateTrainingJob (p. 556). For an example, see Step 2.3.2: Create a Training Job (p. 26).

You now have an initial set of cluster centers.

**Step 2: Iterate over the Training Dataset and Calculate Cluster Centers**

The cluster centers that you created in the preceding step are mostly random, with some consideration for the training dataset. In this step, you use the training dataset to move these centers toward the true cluster centers. The algorithm iterates over the training dataset, and recalculates the K cluster centers.

1. Read a mini-batch of observations (a small, randomly chosen subset of all records) from the training dataset and do the following.

   **Note**
   When creating a model training job, you specify the batch size in the mini_batch_size string in the HyperParameters string map.

   a. Assign all of the observations in the mini-batch to one of the clusters with the closest cluster center.
   b. Calculate the number of observations assigned to each cluster. Then, calculate the proportion of new points assigned per cluster.

   For example, consider the following clusters:

   Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.
   Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.
   Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

   Calculate the proportion of new points assigned to each of clusters as follows:

   \[
   \begin{align*}
   p_1 &= \text{proportion of points assigned to } c_1 = \frac{25}{100+25} \\
   p_2 &= \text{proportion of points assigned to } c_2 = \frac{40}{150+40} \\
   p_3 &= \text{proportion of points assigned to } c_3 = \frac{5}{450+5}
   \end{align*}
   \]

   c. Compute the center of the new points added to each cluster:

   \[
   \begin{align*}
   d_1 &= \text{center of the new points added to cluster } 1 \\
   d_2 &= \text{center of the new points added to cluster } 2 \\
   d_3 &= \text{center of the new points added to cluster } 3
   \end{align*}
   \]

   d. Compute the weighted average to find the updated cluster centers as follows:

   \[
   \begin{align*}
   \text{Center of cluster } 1 &= ((1 - p_1) \times \text{center of cluster } 1) + (p_1 \times d_1) \\
   \text{Center of cluster } 2 &= ((1 - p_2) \times \text{center of cluster } 2) + (p_2 \times d_2)
   \end{align*}
   \]
K-Means Algorithm

Center of cluster 3 = ((1 - p3) * center of cluster 3) + (p3 * d3)

2. Read the next mini-batch, and repeat Step 1 to recalculate the cluster centers.
3. For more information about mini-batch k-means, see Web-Scale k-means Clustering.

Step 3: Reduce the Clusters from $K$ to $k$

If the algorithm created $K$ clusters—($K = k \times x$) where $x$ is greater than 1—then it reduces the $K$ clusters to $k$ clusters. (For more information, see extra_center_factor in the preceding discussion.) It does this by applying Lloyd's method with kmeans++ initialization to the $K$ cluster centers. For more information about Lloyd's method, see k-means clustering.

K-Means Hyperparameters

In the CreateTrainingJob (p. 556) request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the k-means training algorithm provided by Amazon SageMaker. For more information about how k-means clustering works, see How K-Means Clustering Works (p. 138).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The number of features in the input data.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td>k</td>
<td>The number of required clusters.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of passes done over the training data.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>eval_metrics</td>
<td>A JSON list of metric types used to report a score for the model.</td>
</tr>
<tr>
<td></td>
<td>Allowed values are msd for Means Square Error and ssd for Sum of Square Distance. If test data is provided, the score is reported for each of the metrics requested.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either [&quot;msd&quot;] or [&quot;ssd&quot;] or [&quot;msd&quot;, &quot;ssd&quot;].</td>
</tr>
<tr>
<td></td>
<td>Default value: [&quot;msd&quot;]</td>
</tr>
<tr>
<td>extra_center_factor</td>
<td>The algorithm creates $K$ centers = num_clusters * extra_center_factor as it runs and reduces the number of centers from $K$ to $k$ when finalizing the model.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either a positive integer or auto.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td><strong>half_life_time_size</strong></td>
<td>Used to determine the weight given to an observation when computing a cluster mean. This weight decays exponentially as more points are observed. When a point is first observed, it is assigned a weight of 1 when computing the cluster mean. The decay constant for the exponential decay function is chosen so that after observing half_life_time_size points, its weight is 1/2. If set to 0, there is no decay.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><strong>init_method</strong></td>
<td>Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An alternative k-means++ method chooses the first cluster center at random. Then it spreads out the position of the remaining initial clusters by weighting the selection of centers with a probability distribution that is proportional to the square of the distance of the remaining data points from existing centers.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either random or kmeans++.</td>
</tr>
<tr>
<td></td>
<td>Default value: random</td>
</tr>
<tr>
<td><strong>local_lloyd_init_method</strong></td>
<td>The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either random or kmeans++.</td>
</tr>
<tr>
<td></td>
<td>Default value: kmeans++</td>
</tr>
<tr>
<td><strong>local_lloyd_max_iter</strong></td>
<td>The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing k centers.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
</tbody>
</table>
### Parameter Name

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local_lloyd_num_trials</td>
<td>The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing ( k ) centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either a positive integer or auto.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>local_lloyd_tol</td>
<td>The tolerance for change in loss for early stopping of Lloyd's expectation-maximization (EM) procedure used to build the final model containing ( k ) centers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The number of observations per mini-batch for the data iterator.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5000</td>
</tr>
</tbody>
</table>

### Tune a K-Means Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker k-means algorithm is an unsupervised algorithm that groups data into clusters whose members are as similar as possible. Because it is unsupervised, it doesn't use a validation dataset that hyperparameters can optimize against. But it does take a test dataset and emits metrics that depend on the squared distance between the data points and the final cluster centroids at the end of each training run. To find the model that reports the tightest clusters on the test dataset, you can use a hyperparameter tuning job. The clusters optimize the similarity of their members.

For more information about model tuning, see [Automatic Model Tuning (p. 290)](https://docs.aws.amazon.com/sagemaker/latest/dg/model-tuning.html).

### Metrics Computed by the K-Means Algorithm

The k-means algorithm computes the following metrics during training. When tuning a model, choose one of these metrics as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:msd</td>
<td>Mean squared distances between each record in the test set and the closest center of the model.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
### K-Means Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker k-means algorithm.

#### JSON Response Format

```json
{
    "predictions": [
        {
            "closest_cluster": 1.0,
            "distance_to_cluster": 3.0,
        },
        {
            "closest_cluster": 2.0,
            "distance_to_cluster": 5.0,
        },
        ....
    ]
}
```

#### JSONLINES Response Format

```json
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```
K-Nearest Neighbors (k-NN) Algorithm

Amazon SageMaker k-nearest neighbors (k-NN) algorithm is an index-based algorithm. It uses a non-parametric method for classification or regression. For classification problems, the algorithm queries the $k$ points that are closest to the sample point and returns the most frequently used label of their class as the predicted label. For regression problems, the algorithm queries the $k$ closest points to the sample point and returns the average of their feature values as the predicted value.

Training with the k-NN algorithm has three steps: sampling, dimension reduction, and index building. Sampling reduces the size of the initial dataset so that it fits into memory. For dimension reduction, the algorithm decreases the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency. We provide two methods of dimension reduction methods: random projection and the fast Johnson-Lindenstrauss transform. Typically, you use dimension reduction for high-dimensional ($d > 1000$) datasets to avoid the “curse of dimensionality” that troubles the statistical analysis of data that becomes sparse as dimensionality increases. The main objective of k-NN’s training is to construct the index. The index enables efficient lookups of distances between points whose values or class labels have not yet been determined and the k nearest points to use for inference.

Topics

- Input/Output Interface for the k-NN Algorithm (p. 145)
- k-NN Sample Notebooks (p. 146)
- How the K-nn Algorithm Works (p. 146)
- EC2 Instance Recommendation for the K-nn Algorithm (p. 147)
- K-nn Hyperparameters (p. 147)
- Tune a K-nn Model (p. 149)
- Data Formats for K-nn Training Input (p. 150)
- K-nn Request and Response Formats (p. 151)

Input/Output Interface for the k-NN Algorithm

Amazon SageMaker k-NN supports train and test data channels.

- Use a train channel for data that you want to sample and construct into the k-NN index.
- Use a test channel to emit scores in log files. Scores are listed as one line per mini-batch: accuracy for classifier, mean-squared error (mse) for regressor for score.
For training inputs, k-NN supports text/csv and application/x-recordio-protobuf data formats. For input type text/csv, the first label_size columns are interpreted as the label vector for that row. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference inputs, k-NN supports the application/json, application/x-recordio-protobuf, and text/csv data formats. The text/csv format accepts a label_size and encoding parameter. It assumes a label_size of 0 and a UTF-8 encoding.

For inference outputs, k-NN supports the application/json and application/x-recordio-protobuf data formats. These two data formats also support a verbose output mode. In verbose output mode, the API provides the search results with the distances vector sorted from smallest to largest, and corresponding elements in the labels vector.

For batch transform, k-NN supports the application/jsonlines data format for both input and output. An example input is as follows:

```
content-type: application/jsonlines
{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

An example output is as follows:

```
accept: application/jsonlines
{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

For more information on input and output file formats, see Data Formats for K-nn Training Input (p. 150) for training, K-nn Request and Response Formats (p. 151) for inference, and the k-NN Sample Notebooks (p. 146).

**k-NN Sample Notebooks**

For a sample notebook that uses the Amazon SageMaker k-nearest neighbor algorithm to predict wilderness cover types from geological and forest service data, see the K-Nearest Neighbor Covertype. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**How the K-nn Algorithm Works**

**Step 1: Sample**

To specify the total number of data points to be sampled from the training dataset, use the sample_size parameter. For example, if the initial dataset has 1,000 data points and the sample_size is set to 100, where the total number of instances is 2, each worker would sample 50 points. A total set of 100 data points would be collected. Sampling runs in linear time with respect to the number of data points.

**Step 2: Perform Dimension Reduction**

The current implementation of k-NN has two methods of dimension reduction. You specify the method in the dimension_reduction_type hyperparameter. The sign method specifies a random projection, which uses a linear projection using a matrix of random signs, and the fjit method specifies a fast
Johnson-Lindenstrauss transform, a method based on the Fourier transform. Both methods preserve the L2 and inner product distances. The \texttt{sign} method should be used when the target dimension is large and has better performance with CPU inference. The methods differ in their computational complexity. The \texttt{sign} method requires \(O(ndk)\) time to reduce the dimension of a batch of \(n\) points of dimension \(d\) into a target dimension \(k\). The \texttt{fjlst} method requires \(O(nd \log(d))\) time, but the constants involved are larger. Using dimension reduction introduces noise into the data and this noise can reduce prediction accuracy.

**Step 3: Build an Index**

During inference, the algorithm queries the index for the k-nearest-neighbors of a sample point. Based on the references to the points, the algorithm makes the classification or regression prediction. It makes the prediction based on the class labels or values provided. k-NN provides three different types of indexes: a flat index, an inverted index, and an inverted index with product quantization. You specify the type with the \texttt{index_type} parameter.

**Serialize the Model**

When the k-NN algorithm finishes training, it serializes three files to prepare for inference.

- \texttt{model algo-1}: Contains the serialized index for computing the nearest neighbors.
- \texttt{model algo-1.labels}: Contains serialized labels (np.float32 binary format) for computing the predicted label based on the query result from the index.
- \texttt{model algo-1.json}: Contains the JSON-formatted model metadata which stores the \(k\) and \texttt{predictor_type} hyper-parameters from training for inference along with other relevant state.

With the current implementation of k-NN, you can modify the metadata file to change the way predictions are computed. For example, you can change \(k\) to 10 or change \texttt{predictor_type} to \texttt{regressor}.

```json
{
    "k": 5,
    "predictor_type": "classifier",
    "dimension_reduction": {
        "type": "sign",
        "seed": 3,
        "target_dim": 10,
        "input_dim": 20
    },
    "normalize": False,
    "version": "1.0"
}
```

**EC2 Instance Recommendation for the K-nn Algorithm**

**Instance Recommendation for Training with the K-nn Algorithm**

To start, try running training on a CPU, using, for example, an \texttt{ml.m5.2xlarge} instance, or on a GPU using, for example, an \texttt{ml.p2.xlarge} instance.

**Instance Recommendation for Inference with the K-nn Algorithm**

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.

**K-nn Hyperparameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The number of features in the input data.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Required</strong></td>
<td>Valid values: positive integer.</td>
</tr>
<tr>
<td>k</td>
<td>The number of nearest neighbors.</td>
</tr>
<tr>
<td>predictor_type</td>
<td>The type of inference to use on the data labels.</td>
</tr>
<tr>
<td>sample_size</td>
<td>The number of data points to be sampled from the training data set.</td>
</tr>
<tr>
<td>dimension_reduction_target</td>
<td>The target dimension to reduce to.</td>
</tr>
<tr>
<td>dimension_reduction_type</td>
<td>The type of dimension reduction method.</td>
</tr>
<tr>
<td>faiss_index_ivf_nlists</td>
<td>The number of centroids to construct in the index when index_type is faiss.IVFFlat or faiss.IVFPQ.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>faiss_index_pq_m</td>
<td>The number of vector sub-components to construct in the index when index_type is set to faiss.IVFPQ. The FaceBook AI Similarity Search (FAISS) library requires that the value of faiss_index_pq_m is a divisor of the data dimension. If faiss_index_pq_m is not a divisor of the data dimension, we increase the data dimension to smallest integer divisible by faiss_index_pq_m. If no dimension reduction is applied, the algorithm adds a padding of zeros. If dimension reduction is applied, the algorithm increase the value of the dimension_reduction_target hyper-parameter. Optional Valid values: One of the following positive integers: 1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 96</td>
</tr>
<tr>
<td>index_metric</td>
<td>The metric to measure the distance between points when finding nearest neighbors. When training with index_type set to faiss.IVFPQ, the INNER_PRODUCT distance and COSINE similarity are not supported. Optional Valid values: L2 for Euclidean-distance, INNER_PRODUCT for inner-product distance, COSINE for cosine similarity. Default value: L2</td>
</tr>
<tr>
<td>index_type</td>
<td>The type of index. Optional Valid values: faiss.Flat, faiss.IVFFlat, faiss.IVFPQ. Default values: faiss.Flat</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The number of observations per mini-batch for the data iterator. Optional Valid values: positive integer Default value: 5000</td>
</tr>
</tbody>
</table>

**Tune a K-nn Model**

The Amazon SageMaker k-nearest neighbors algorithm is a supervised algorithm. The algorithm consumes a test data set and emits a metric about the accuracy for a classification task or about the mean squared error for a regression task. These accuracy metrics compare the model predictions for their respective task to the ground truth provided by the empirical test data. To find the best model that reports the highest accuracy or lowest error on the test dataset, run a hyperparameter tuning job for k-NN.

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric appropriate for the prediction task of the algorithm. Automatic model tuning searches the
hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric. The hyperparameters are used only to help estimate model parameters and are not used by the trained model to make predictions.

For more information about model tuning, see Automatic Model Tuning (p. 290).

**Metrics Computed by the K-nn Algorithm**

The k-nearest neighbors algorithm computes one of two metrics in the following table during training depending on the type of task specified by the `predictor_type` hyper-parameter.

- `classifier` specifies a classification task and computes `test:accuracy`
- `regressor` specifies a regression task and computes `test:mse`.

Choose the `predictor_type` value appropriate for the type of task undertaken to calculate the relevant objective metric when tuning a model.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>test:accuracy</strong></td>
<td>When <code>predictor_type</code> is set to <code>classifier</code>, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The accuracy reported ranges from 0.0 (0%) to 1.0 (100%).</td>
<td>Maximize</td>
</tr>
<tr>
<td><strong>test:mse</strong></td>
<td>When <code>predictor_type</code> is set to <code>regressor</code>, k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The mean squared error is computed by comparing the two labels.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

**Tunable K-nn Hyperparameters**

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>k</code></td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 1024</td>
</tr>
<tr>
<td><code>sample_size</code></td>
<td>IntegerParameterRanges</td>
<td>MinValue: 256, MaxValue: 2000000</td>
</tr>
</tbody>
</table>

**Data Formats for K-nn Training Input**

All Amazon SageMaker built-in algorithms adhere to the common input training formats described in Common Data Formats - Training. This topic contains a list of the available input formats for the Amazon SageMaker k-nearest-neighbor algorithm.

**CSV Data Format**

content-type: text/csv; label_size=1
The first label_size columns are interpreted as the label vector for that row.

**RECORDIO Data Format**

```json
[ Record = {
  features = {
    'values': {
      values: [1.2, 1.3, 9.6, 20.3]  # float32
    }
  },
  label = {
    'values': {
      values: [4]  # float32
    }
  }
}
]
```

**K-nn Request and Response Formats**

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker k-nearest-neighbor algorithm.

**INPUT: CSV Request Format**

```plaintext
1.2,1.3,9.6,20.3
```

This accepts a label_size or encoding parameter. It assumes a label_size of 0 and a utf-8 encoding.

**INPUT: JSON Request Format**

```json
{
  "instances": [
    {
      "data": {
        "features": {
          "values": [-3, -1, -4, 2]
        }
      }
    },
    {
      "features": [3.0, 0.1, 0.04, 0.002]
    }
  ]
}
```

**INPUT: JSONLINES Request Format**

```json
{"features": [1.5, 16.0, 14.0, 23.0]}
```
INPUT: RECORDIO Request Format

content-type: application/x-recordio-protobuf

```
[  
  Record = { 
    features = { 
      'values': { 
        values: [-3, -1, -4, 2]  # float32 
      }, 
      label = {} 
    }, 
    Record = { 
      features = { 
        'values': { 
          values: [3.0, 0.1, 0.04, 0.002]  # float32 
        }, 
        label = {} 
      }, 
    } 
  } 
]
```

OUTPUT: JSON Response Format

accept: application/json

```
{
  "predictions": [ 
    {"predicted_label": 0.0}, 
    {"predicted_label": 2.0} 
  ]
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"predicted_label": 0.0} 
{"predicted_label": 2.0}
```

OUTPUT: VERBOSE JSON Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/json; verbose=true

```
{
  "predictions": [ 
    { 
      "predicted_label": 0.0, 
      "distances": [3.11792408, 3.89746071, 6.32548437], 
      "labels": [0.0, 1.0, 0.0] 
    }, 
  ]
}
```
K-Nearest Neighbors (k-NN) Algorithm

OUTPUT: RECORDIO-PROTOBUF Response Format

```
{
    "predicted_label": 2.0,
    "distances": [1.08470316, 3.04917915, 5.25393973],
    "labels": [2.0, 2.0, 0.0]
}
```

OUTPUT: VERBOSE RECORDIO-PROTOBUF Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

```
[ Record = {
    features = {},
    label = {
        'predicted_label': {
            values: [0.0]  # float32
        }
    },
    Record = {
        features = {},
        label = {
            'predicted_label': {
                values: [2.0]  # float32
            }
        }
    }
}
```

accept: application/x-recordio-protobuf; verbose=true
Latent Dirichlet Allocation (LDA) Algorithm

The Amazon SageMaker Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

The exact content of two documents with similar topic mixtures will not be the same. But overall, you would expect these documents to more frequently use a shared subset of words, than when compared with a document from a different topic mixture. This allows LDA to discover these word groups and use them to form topics. As an extremely simple example, given a set of documents where the only words that occur within them are: *eat, sleep, play, meow, and bark*, LDA might produce topics like the following:

<table>
<thead>
<tr>
<th>Topic</th>
<th>eat</th>
<th>sleep</th>
<th>play</th>
<th>meow</th>
<th>bark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic 1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.2</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Topic 2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.4</td>
<td>0.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

You can infer that documents that are more likely to fall into Topic 1 are about cats (who are more likely to *meow* and *sleep*), and documents that fall into Topic 2 are about dogs (who prefer to *play* and *bark*). These topics can be found even though the words dog and cat never appear in any of the texts.

**Topics**
- Input/Output Interface for the LDA Algorithm (p. 155)
- EC2 Instance Recommendation for the LDA Algorithm (p. 155)
- LDA Sample LDA Notebooks (p. 155)
- How LDA Works (p. 155)
- LDA Hyperparameters (p. 157)
Input/Output Interface for the LDA Algorithm

LDA expects data to be provided on the train channel, and optionally supports a test channel, which is scored by the final model. LDA supports both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats. For CSV, the data must be dense and have dimension equal to number of records * vocabulary size. LDA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

For inference, text/csv, application/json, and application/x-recordio-protobuf content types are supported. Sparse data can also be passed for application/json and application/x-recordio-protobuf predictions, which include the topic_mixture vector for each observation.

Please see the LDA Sample LDA Notebooks (p. 155) for more detail on training and inference formats.

EC2 Instance Recommendation for the LDA Algorithm

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

LDA Sample LDA Notebooks

For a sample notebook that shows how to train the Amazon SageMaker Latent Dirichlet Allocation algorithm on a dataset and then how to deploy the trained model to perform inferences about the topic mixtures in input documents, see the An Introduction to SageMaker LDA. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

How LDA Works

Amazon SageMaker LDA is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of different categories. These categories are themselves a probability distribution over the features. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. This is opposed to discriminative models, which attempt to learn how inputs map to outputs.

You can use LDA for a variety of tasks, from clustering customers based on product purchases to automatic harmonic analysis in music. However, it is most commonly associated with topic modeling in text corpuses. Observations are referred to as documents. The feature set is referred to as vocabulary. A feature is referred to as a word. And the resulting categories are referred to as topics.

Note
Lemmatization significantly increases algorithm performance and accuracy. Consider pre-processing any input text data.

An LDA model is defined by two parameters:

- $\alpha$—A prior estimate on topic probability (in other words, the average frequency that each topic within a given document occurs).
- $\beta$—a collection of k topics where each topic is given a probability distribution over the vocabulary used in a document corpus, also called a “topic-word distribution.”
LDA is a "bag-of-words" model, which means that the order of words does not matter. LDA is a generative model where each document is generated word-by-word by choosing a topic mixture $\theta \sim \text{Dirichlet}(\alpha)$.

For each word in the document:
- Choose a topic $z \sim \text{Multinomial}(\theta)$
- Choose the corresponding topic-word distribution $\beta_z$.
- Draw a word $w \sim \text{Multinomial}(\beta_z)$.

When training the model, the goal is to find parameters $\alpha$ and $\beta$, which maximize the probability that the text corpus is generated by the model.

The most popular methods for estimating the LDA model use Gibbs sampling or Expectation Maximization (EM) techniques. The Amazon SageMaker LDA uses tensor spectral decomposition. This provides several advantages:

- **Theoretical guarantees on results.** The standard EM-method is guaranteed to converge only to local optima, which are often of poor quality.
- **Embarrassingly parallelizable.** The work can be trivially divided over input documents in both training and inference. The EM-method and Gibbs Sampling approaches can be parallelized, but not as easily.
- **Fast.** Although the EM-method has low iteration cost it is prone to slow convergence rates. Gibbs Sampling is also subject to slow convergence rates and also requires a large number of samples.

At a high-level, the tensor decomposition algorithm follows this process:

1. The goal is to calculate the spectral decomposition of a $V \times V \times V$ tensor, which summarizes the moments of the documents in our corpus. $V$ is vocabulary size (in other words, the number of distinct words in all of the documents). The spectral components of this tensor are the LDA parameters $\alpha$ and $\beta$, which maximize the overall likelihood of the document corpus. However, because vocabulary size tends to be large, this $V \times V \times V$ tensor is prohibitively large to store in memory.
2. Instead, it uses a $V \times V$ moment matrix, which is the two-dimensional analog of the tensor from step 1, to find a whitening matrix of dimension $V \times k$. This matrix can be used to convert the $V \times V$ moment matrix into a $k \times k$ identity matrix. $k$ is the number of topics in the model.
3. This same whitening matrix can then be used to find a smaller $k \times k \times k$ tensor. When spectrally decomposed, this tensor has components that have a simple relationship with the components of the $V \times V \times V$ tensor.
4. **Alternating Least Squares** is used to decompose the smaller $k \times k \times k$ tensor. This provides a substantial improvement in memory consumption and speed. The parameters $\alpha$ and $\beta$ can be found by “unwhitening” these outputs in the spectral decomposition.

After the LDA model's parameters have been found, you can find the topic mixtures for each document. You use stochastic gradient descent to maximize the likelihood function of observing a given topic mixture corresponding to these data.

Topic quality can be improved by increasing the number of topics to look for in training and then filtering out poor quality ones. This is in fact done automatically in Amazon SageMaker LDA: 25% more topics are computed and only the ones with largest associated Dirichlet priors are returned. To perform further topic filtering and analysis, you can increase the topic count and modify the resulting LDA model as follows:

```python
> import mxnet as mx
> alpha, beta = mx.ndarray.load('model.tar.gz')
```
For more information about algorithms for LDA and the Amazon SageMaker implementation, see the following:


### LDA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the LDA training algorithm provided by Amazon SageMaker. For more information, see How LDA Works (p. 155).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_topics</td>
<td>The number of topics for LDA to find within the data.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>feature_dim</td>
<td>The size of the vocabulary of the input document corpus.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The total number of documents in the input document corpus.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>alpha0</td>
<td>Initial guess for the concentration parameter: the sum of the elements of the Dirichlet prior. Small values are more likely to generate sparse topic mixtures and large values (greater than 1.0) produce more uniform mixtures.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>max_restarts</td>
<td>The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>max_iterations</td>
<td>The maximum number of iterations to perform during the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.</td>
</tr>
<tr>
<td>tol</td>
<td>Target error tolerance for the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.</td>
</tr>
</tbody>
</table>

### Tune an LDA Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

LDA is an unsupervised topic modeling algorithm that attempts to describe a set of observations (documents) as a mixture of different categories (topics). The “per-word log-likelihood” (PWLL) metric measures the likelihood that a learned set of topics (an LDA model) accurately describes a test document dataset. Larger values of PWLL indicate that the test data is more likely to be described by the LDA model.

For more information about model tuning, see Automatic Model Tuning (p. 290).

### Metrics Computed by the LDA Algorithm

The LDA algorithm reports on a single metric during training: test:pwll. When tuning a model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:pwll</td>
<td>Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>
Tunable LDA Hyperparameters

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, $\alpha_0$ and $num\_topics$, can affect the LDA objective metric ($test:\text{pwll}$). If you don’t already know the optimal values for these hyperparameters, which maximize per-word log-likelihood and produce an accurate LDA model, automatic model tuning can help find them.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 10</td>
</tr>
<tr>
<td>$num_topics$</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 150</td>
</tr>
</tbody>
</table>

Linear Learner Algorithm

Linear models are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples $(x, y)$. $x$ is a high-dimensional vector and $y$ is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to $num\_classes$ - 1. For regression problems, $y$ is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector $x$ to an approximation of the label $y$.

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the Amazon SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from a validation set. You can also explore a large number of models and choose the best. The best model optimizes either of the following:

- Continuous objective, such as mean square error, cross entropy loss, absolute error, and so on
- Discrete objectives suited for classification, such as F1 measure, precision@recall, or accuracy

Compared with methods that provide a solution for only continuous objectives, the Amazon SageMaker linear learner algorithm provides a significant increase in speed over naive hyperparameter optimization techniques. It is also more convenient.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification or regression) as arguments. The feature dimension is also required. For more information, see CreateTrainingJob (p. 556). You can specify additional parameters in the HyperParameters string map of the request body. These parameters control the optimization procedure, or specifics of the objective function that you train on. For example, the number of epochs, regularization, and loss type.

Topics

- Input/Output Interface for the Linear Learner Algorithm (p. 160)
- EC2 Instance Recommendation for the Linear Learner Algorithm (p. 160)
- Linear Learner Sample Notebooks (p. 160)
- How Linear Learner Works (p. 160)
- Linear Learner Hyperparameters (p. 161)
- Tune a Linear Learner Model (p. 170)
- Linear Learner Response Formats (p. 172)
Input/Output Interface for the Linear Learner Algorithm

The Amazon SageMaker linear learner algorithm supports three data channels: train, validation (optional), and test (optional). If you provide validation data, it should be FullyReplicated. The algorithm logs validation loss at every epoch, and uses a sample of the validation data to calibrate and select the best model. If you don't provide validation data, the algorithm uses a sample of the training data to calibrate and select the model. If you provide test data, the algorithm logs include the test score for the final model.

For training, the linear learner algorithm supports both recordIO-wrapped protobuf and CSV formats. For the application/x-recordio-protobuf input type, only Float32 tensors are supported. For the text/csv input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference, the linear learner algorithm supports the application/json, application/x-recordio-protobuf, and text/csv formats. For binary classification models, it returns both the score and the predicted label. For regression, it returns only the score.

For more information on input and output file formats, see Linear Learner Response Formats (p. 172) for inference, and the Linear Learner Sample Notebooks (p. 160).

EC2 Instance Recommendation for the Linear Learner Algorithm

You can train the linear learner algorithm on single- or multi-machine CPU and GPU instances. During testing, we have not found substantial evidence that multi-GPU computers are faster than single-GPU computers. Results can vary, depending on your specific use case.

Linear Learner Sample Notebooks

For a sample notebook that uses the Amazon SageMaker linear learner algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to Linear Learner with MNIST. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). After you have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all of the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, choose its Use tab and choose Create copy.

How Linear Learner Works

Note

Input data for the linear learner algorithm must be shuffled. If it isn't—for example, if the data is ordered by label—the method fails.

Step 1: Preprocess

If the option is turned on, the algorithm first goes over a small sample of the data to learn its characteristics. The algorithm learns the mean value and the standard deviation for every feature and label. It uses this information during training.

You can configure the algorithm to normalize the data. Normalization shifts the label to have a mean of zero and scales it to have unit standard deviation. If you want the algorithm to normalize the data for you, specify the auto (default) value. When auto normalization is enabled, the algorithm does the following:

- For regression problems, shifts and scales the label. For classification problems, leaves the label as is.
- Always scales the features.
• Shifts the features only for dense data.

Step 2: Train

With the linear learner algorithm, you train with a distributed implementation of stochastic gradient descent (SGD). You can control the optimization process by choosing the optimization algorithm. For example, you can choose to use Adam, AdaGrad, stochastic gradient descent, or other optimization algorithms. You also specify their hyperparameters, such as momentum, learning rate, and the learning rate schedule. If you aren't sure which algorithm or hyperparameter value to use, choose a default that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives. For example, you vary L1 or L2 regularization and try out different optimizer settings.

Step 3: Validate and Set the Threshold

When training is done, you can evaluate the models on a validation set. For regression problems, output the model that gets the best score on the validation set. For classification problems, use a sample of (raw prediction, label) pairs to tune the threshold for a provided objective. The raw prediction is the output of the trained linear function. Allow classification objectives based on the predicted label, such as F1 measure, accuracy, precision@recall, and so on. Choose the model that achieves the best score on the validation set.

Note
If you don’t provide a validation set, the algorithm optimizes over the training set. In that case, don't explore different regularization procedures.

Linear Learner Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The number of features in the input data. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td>num_classes</td>
<td>The number of classes for the response variable. The algorithm assumes that classes are labeled 0, ..., num_classes - 1. Required when predictor_type is multiclass_classifier. Otherwise, the algorithm ignores it. Valid values: Integers from 3 to 1,000,000</td>
</tr>
<tr>
<td>predictor_type</td>
<td>Specifies the type of target variable as a binary classification, multiclass classification, or regression. Required Valid values: binary_classifier, multiclass_classifier, or regressor</td>
</tr>
<tr>
<td>accuracy_top_k</td>
<td>When computing the top-k accuracy metric for multiclass classification, the value of k. If the model assigns one of the top-k scores to the true label, an example is scored as correct. Optional</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>balance_multiclass_weights</td>
<td>Specifies whether to use class weights, which give each class equal importance in the loss function. Used only when the predictor_type is multiclass_classifier.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: true, false</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>beta_1</td>
<td>The exponential decay rate for first-moment estimates. Applies only when the optimizer value is adam.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or floating-point value between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>beta_2</td>
<td>The exponential decay rate for second-moment estimates. Applies only when the optimizer value is adam.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or floating-point integer between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>bias_lr_mult</td>
<td>Allows a different learning rate for the bias term. The actual learning rate for the bias is learning_rate * bias_lr_mult.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>bias_wd_mult</td>
<td>Allows different regularization for the bias term. The actual L2 regularization weight for the bias is wd * bias_wd_mult. By default, there is no regularization on the bias term.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or non-negative floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| binary_classifier_model_selection_criteria | When predictor_type is set to binary_classifier, the model evaluation criteria for the validation dataset (or for the training dataset if you don't provide a validation dataset). Criteria include:  
  - accuracy—The model with the highest accuracy.  
  - f_beta—The model with the highest F1 score. The default is F1.  
  - precision_at_target_recall—The model with the highest precision at a given recall target.  
  - recall_at_target_precision—The model with the highest recall at a given precision target.  
  - loss_function—The model with the lowest value of the loss function used in training.  

Optional  
Valid values: accuracy, f_beta, precision_at_target_recall, recall_at_target_precision, or loss_function  
Default value: accuracy |
| early_stopping_patience | If no improvement is made in the relevant metric, the number of epochs to wait before ending training. If you have provided a value for binary_classifier_model_selection_criteria, the metric is that value. Otherwise, the metric is the same as the value specified for the loss hyperparameter.  
  
The metric is evaluated on the validation data. If you haven't provided validation data, the metric is always the same as the value specified for the loss hyperparameter and is evaluated on the training data. To disable early stopping, set early_stopping_patience to a value greater than the value specified for epochs.  

Optional  
Valid values: Positive integer  
Default value: 3 |
| early_stopping_tolerance | The relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping considers the improvement to be zero.  

Optional  
Valid values: Positive floating-point integer  
Default value: 0.001 |
| epochs | The maximum number of passes over the training data.  

Optional  
Valid values: Positive integer  
Default value: 15 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| f_beta         | The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if the value specified for binary_classifier_model_selection_criteria is f_beta.  
**Optional**  
Valid values: Positive floating-point integers  
Default value: 1.0 |
| huber_delta    | The parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than delta and L1 loss for errors larger than delta.  
**Optional**  
Valid values: Positive floating-point integer  
Default value: 1.0 |
| init_bias      | Initial weight for the bias term.  
**Optional**  
Valid values: Floating-point integer  
Default value: 0 |
| init_method    | Sets the initial distribution function used for model weights. Functions include:  
• uniform—Uniformly distributed between (-scale, +scale)  
• normal—Normal distribution, with mean 0 and sigma  
**Optional**  
Valid values: uniform or normal  
Default value: uniform |
| init_scale     | Scales an initial uniform distribution for model weights. Applies only when the init_method hyperparameter is set to uniform.  
**Optional**  
Valid values: Positive floating-point integer  
Default value: 0.07 |
| init_sigma     | The initial standard deviation for the normal distribution. Applies only when the init_method hyperparameter is set to normal.  
**Optional**  
Valid values: Positive floating-point integer  
Default value: 0.01 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>l1</strong></td>
<td>The L1 regularization parameter. If you don’t want to use L1 regularization, set the value to 0.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or non-negative float</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td><strong>learning_rate</strong></td>
<td>The step size used by the optimizer for parameter updates.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto, whose value depends on the optimizer chosen.</td>
</tr>
<tr>
<td><strong>loss</strong></td>
<td>Specifies the loss function.</td>
</tr>
</tbody>
</table>
|                   | The available loss functions and their default values depend on the value of **predictor_type**:
<p>|                   | • If the <strong>predictor_type</strong> is set to <strong>regressor</strong>, the available options are auto, squared_loss, absolute_loss, eps_insensitive_squared_loss, eps_insensitive_absolute_loss, quantile_loss, and huber_loss. The default value for auto is squared_loss. |
|                   | • If the <strong>predictor_type</strong> is set to <strong>binary_classifier</strong>, the available options are auto, logistic, and hinge_loss. The default value for auto is logistic. |
|                   | • If the <strong>predictor_type</strong> is set to <strong>multiclass_classifier</strong>, the available options are auto and softmax_loss. The default value for auto is softmax_loss. |
|                   | Valid values: auto, logistic, squared_loss, absolute_loss, hinge_loss, eps_insensitive_squared_loss, eps_insensitive_absolute_loss, quantile_loss, or huber_loss |
|                   | <strong>Optional</strong>|
|                   | Default value: auto |
| <strong>loss_insensitivity</strong> | The parameter for the epsilon-insensitive loss type. During training and metric evaluation, any error smaller than this value is considered to be zero. |
|                   | <strong>Optional</strong>|
|                   | Valid values: Positive floating-point integer |
|                   | Default value: 0.01 |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr_scheduler_factor</td>
<td>For every lr_scheduler_step hyperparameter, the learning rate decreases by this quantity. Applies only when the use_lr_scheduler hyperparameter is set to true.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive floating-point integer between 0 and 1</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>lr_scheduler_minimum_lr</td>
<td>The learning rate never decreases to a value lower than the value set for lr_scheduler_minimum_lr. Applies only when the use_lr_scheduler hyperparameter is set to true.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default values: auto</td>
</tr>
<tr>
<td>lr_scheduler_step</td>
<td>The number of steps between decreases of the learning rate. Applies only when the use_lr_scheduler hyperparameter is set to true.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>margin</td>
<td>The margin for the hinge_loss function.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.0</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The number of observations per mini-batch for the data iterator.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum of the sgd optimizer.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or a floating-point integer between 0 and 1</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>normalize_data</td>
<td>Normalizes the features before training to achieve a std_dev value of 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto, true, or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>normalize_label</td>
<td>Normalizes the label. For regression problems, the label is normalized. For classification problems, it is not normalized. If you set the normalize_label hyperparameter to true for classification problems, the algorithm ignores it.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto, true, or false</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>num_calibration_samples</td>
<td>The number of observations from the validation dataset to use for model calibration (when finding the best threshold).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>num_models</td>
<td>The number of models to train in parallel. For the default, auto, the algorithm decides the number of parallel models to train. One model is trained according to the given training parameter (regularization, optimizer, loss), and the rest by close parameters.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or positive integer</td>
</tr>
<tr>
<td></td>
<td>Default values: auto</td>
</tr>
<tr>
<td>num_point_for_scaler</td>
<td>The number of data points to use for calculating normalization or unbiasing of terms.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10,000</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimization algorithm to use.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• auto—The default value.</td>
</tr>
<tr>
<td></td>
<td>• sgd—Stochastic gradient descent.</td>
</tr>
<tr>
<td></td>
<td>• adam— Adaptive momentum estimation.</td>
</tr>
<tr>
<td></td>
<td>• rmsprop—A gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient.</td>
</tr>
<tr>
<td></td>
<td>Default value: auto. The default setting for auto is adam.</td>
</tr>
<tr>
<td>positive_example_weight_mult</td>
<td>The weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If you want the algorithm to choose a weight so that errors in classifying negative vs. positive examples have equal impact on training loss, specify balanced. If you want the algorithm to choose the weight that optimizes performance, specify auto.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: balanced, auto, or a positive floating-point integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.0</td>
</tr>
<tr>
<td>quantile</td>
<td>The quantile for quantile loss. For quantile q, the model attempts to produce predictions so that the value of true_label is greater than the prediction with probability q.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer between 0 and 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>target_precision</td>
<td>The target precision. If binary_classifier_model_selection_criteria is recall_at_target_precision, then precision is held at this value while recall is maximized.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Floating-point integer between 0 and 1.0</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.8</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| **target_recall** | The target recall. If `binary_classifier_model_selection_criteria` is `precision_at_target_recall`, then recall is held at this value while precision is maximized.  
Optional  
Valid values: Floating-point integer between 0 and 1.0  
Default value: 0.8 |
| **unbias_data** | Unbiases the features before training so that the mean is 0. By default, data is unbiased if the `use_bias` hyperparameter is set to true.  
Optional  
Valid values: `auto`, `true`, or `false`  
Default value: `auto` |
| **unbias_label** | Unbiases labels before training so that the mean is 0. Applies to regression only if the `use_bias` hyperparameter is set to true.  
Optional  
Valid values: `auto`, `true`, or `false`  
Default value: `auto` |
| **use_bias** | Specifies whether the model should include a bias term, which is the intercept term in the linear equation.  
Optional  
Valid values: `true` or `false`  
Default value: `true` |
| **use_lr_scheduler** | Whether to use a scheduler for the learning rate. If you want to use a scheduler, specify `true`.  
Optional  
Valid values: `true` or `false`  
Default value: `true` |
| **wd** | The weight decay parameter, also known as the L2 regularization parameter. If you don't want to use L2 regularization, set the value to 0.  
Optional  
Valid values: `auto` or non-negative floating-point integer  
Default value: `auto` |
Tune a Linear Learner Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The linear learner algorithm also has an internal mechanism for tuning hyperparameters separate from the automatic model tuning feature described here. By default, the linear learner algorithm tunes hyperparameters by training multiple models in parallel. When you use automatic model tuning, the linear learner internal tuning mechanism is turned off automatically. This sets the number of parallel models, num_models, to 1. The algorithm ignores any value that you set for num_models.

For more information about model tuning, see Automatic Model Tuning (p. 290).

Metrics Computed by the Linear Learner Algorithm

The linear learner algorithm reports five metrics, which are computed during training. Choose one of them as the objective metric. To avoid overfitting, we recommend tuning the model against a validation metric instead of a training metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:objective_loss</td>
<td>The mean value of the objective loss function on the test dataset after the model is trained. By default, the loss is logistic loss for binary classification and squared loss for regression. To set the loss to other types, use the loss hyperparameter.</td>
<td>Minimize</td>
</tr>
<tr>
<td>test:binary_classification_accuracy</td>
<td>The accuracy of the final model on the test dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:binary_f_beta</td>
<td>The F_beta score of the final model on the test dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall.</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:precision</td>
<td>The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the binary_classifier_model_selection hyperparameter to precision_at_target_recall and setting the value for the target_recall hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:recall</td>
<td>The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the binary_classifier_model_selection hyperparameter to recall_at_target_precision and setting the value for the target_precision hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:objective_loss</td>
<td>The mean value of the objective loss function on the validation dataset every epoch. By default, the loss is logistic loss for binary classification and</td>
<td>Minimize</td>
</tr>
<tr>
<td>Metric Name</td>
<td>Description</td>
<td>Optimization Direction</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>validation:binary_classification_accuracy</td>
<td>The accuracy of the final model on the validation dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:binary_f_beta</td>
<td>The F_beta score of the final model on the validation dataset. By default, the F_beta score is the F1 score, which is the harmonic mean of the validation:precision and validation:recall metrics.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:precision</td>
<td>The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the binary_classifier_model_selection hyperparameter to precision_at_target_recall and setting the value for the target_recall hyperparameter.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:recall</td>
<td>The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the binary_classifier_model_selection hyperparameter to recall_at_target_precision and setting the value for the target_precision hyperparameter.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

**Tuning Linear Learner Hyperparameters**

You can tune a linear learner model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>wd</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-7, MaxValue: 1</td>
</tr>
<tr>
<td>l1</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-7, MaxValue: 1</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-5, MaxValue: 1</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 100, MaxValue: 5000</td>
</tr>
<tr>
<td>use_bias</td>
<td>CategoricalParameterRanges</td>
<td>[True, False]</td>
</tr>
<tr>
<td>positive_example_weight</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-5, MaxValue: 1e5</td>
</tr>
</tbody>
</table>
Amazon SageMaker Developer Guide
Linear Learner Algorithm

Linear Learner Response Formats
JSON Response Format
All Amazon SageMaker built-in algorithms adhere to the common input inference format described
in Common Data Formats - Inference. The following are the available output formats for the Amazon
SageMaker linear learner algorithm.
Binary Classiﬁcation
let response =
{
"predictions":
[
{
"score": 0.4,
"predicted_label": 0
}
]
}

Multiclass Classiﬁcation
let response =
{
"predictions":
[
{
"score": [0.1, 0.2, 0.4, 0.3],
"predicted_label": 2
}
]
}

Regression
let response =
{
"predictions":
[
{
"score": 0.4
}
]
}

JSONLINES Response Format
Binary Classiﬁcation
{"score": 0.4, "predicted_label": 0}

Multiclass Classiﬁcation
{"score": 0.4, "predicted_label": 0}

Regression
{"score": 0.4}

RECORDIO Response Format
Binary Classiﬁcation

172


Neural Topic Model (NTM) Algorithm

Amazon SageMaker NTM is an unsupervised learning algorithm that is used to organize a corpus of documents into topics that contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as "bike", "car", "train", "mileage", and "speed" are likely to share a topic on "transportation" for example. Topic modeling can be used to classify or summarize documents based on the topics detected or to retrieve information or recommend content based on topic similarities. The topics from documents that NTM learns are characterized as a latent representation because the topics are inferred from the observed word distributions in the corpus. The semantics of topics are usually inferred by examining the top ranking words they contain. Because the method is unsupervised, only the number of topics, not the topics themselves, are prespecified. In addition, the topics are not guaranteed to align with how a human might naturally categorize documents.
Topic modeling provides a way to visualize the contents of a large document corpus in terms of the learned topics. Documents relevant to each topic might be indexed or searched for based on their soft topic labels. The latent representations of documents might also be used to find similar documents in the topic space. You can also use the latent representations of documents that the topic model learns for input to another supervised algorithm such as a document classifier. Because the latent representations of documents are expected to capture the semantics of the underlying documents, algorithms based in part on these representations are expected to perform better than those based on lexical features alone.

Although you can use both the Amazon SageMaker NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results on the same input data.

For more information on the mathematics behind NTM, see Neural Variational Inference for Text Processing.

Topics

- Input/Output Interface for the NTM Algorithm (p. 174)
- EC2 Instance Recommendation for the NTM Algorithm (p. 175)
- NTM Sample Notebooks (p. 175)
- NTM Hyperparameters (p. 175)
- Tune an NTM Model (p. 177)
- NTM Response Formats (p. 178)

Input/Output Interface for the NTM Algorithm

Amazon SageMaker Neural Topic Model supports four data channels: train, validation, test, and auxiliary. The validation, test, and auxiliary data channels are optional. If you specify any of these optional channels, set the value of the S3DataDistributionType parameter for them to FullyReplicated. If you provide validation data, the loss on this data is logged at every epoch, and the model stops training as soon as it detects that the validation loss is not improving. If you don't provide validation data, the algorithm stops early based on the training data, but this can be less efficient. If you provide test data, the algorithm reports the test loss from the final model.

The train, validation, and test data channels for NTM support both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats. For CSV format, each row must be represented densely with zero counts for words not present in the corresponding document, and have dimension equal to: (number of records) \* (vocabulary size). You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV. The auxiliary channel is used to supply a text file that contains vocabulary. By supplying the vocabulary file, users are able to see the top words for each of the topics printed in the log instead of their integer IDs. Having the vocabulary file also allows NTM to compute the Word Embedding Topic Coherence (WETC) scores, a new metric displayed in the log that captures similarity among the top words in each topic effectively. The ContentType for the auxiliary channel is text/plain, with each line containing a single word, in the order corresponding to the integer IDs provided in the data. The vocabulary file must be named vocab.txt and currently only UTF-8 encoding is supported.

For inference, text/csv, application/json, application/jsonlines, and application/x-recordio-protobuf content types are supported. Sparse data can also be passed for application/json and application/x-recordio-protobuf. NTM inference returns application/json or application/x-recordio-protobuf predictions, which include the topic_weights vector for each observation.

See the blog post and the companion notebook for more details on using the auxiliary channel and the WETC scores. For more information on how to compute the WETC score, see Coherence-Aware Neural Topic Modeling. We used the pairwise WETC described in this paper for the Amazon SageMaker Neural Topic Model.
EC2 Instance Recommendation for the NTM Algorithm

NTM training supports both GPU and CPU instance types. We recommend GPU instances, but for certain workloads, CPU instances may result in lower training costs. CPU instances should be sufficient for inference.

NTM Sample Notebooks

For a sample notebook that uses the Amazon SageMaker NTM algorithm to uncover topics in documents from a synthetic data source where the topic distributions are known, see the Introduction to Basic Functionality of NTM. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

NTM Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>The vocabulary size of the dataset. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1, max: 1,000,000)</td>
</tr>
<tr>
<td>num_topics</td>
<td>The number of required topics. Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 2, max: 1000)</td>
</tr>
<tr>
<td>batch_norm</td>
<td>Whether to use batch normalization during training. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>The maximum magnitude for each gradient component. Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 1e-3)</td>
</tr>
<tr>
<td></td>
<td>Default value: Infinity</td>
</tr>
<tr>
<td>encoder_layers</td>
<td>The number of layers in the encoder and the output size of each layer. When set to auto, the algorithm uses two layers of sizes 3 x num_topics and 2 x num_topics respectively. Optional</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
**encoder_layers_activation** | The activation function to use in the encoder layers.  
**Optional**  
Valid values:  
- **sigmoid**: Sigmoid function  
- **tanh**: Hyperbolic tangent  
- **relu**: Rectified linear unit  
Default value: **sigmoid**

**epochs** | The maximum number of passes over the training data.  
**Optional**  
Valid values: Positive integer (min: 1)  
Default value: 50

**learning_rate** | The learning rate for the optimizer.  
**Optional**  
Valid values: Float (min: 1e-6, max: 1.0)  
Default value: 0.001

**mini_batch_size** | The number of examples in each mini batch.  
**Optional**  
Valid values: Positive integer (min: 1, max: 10000)  
Default value: 256

**num_patience_epochs** | The number of successive epochs over which early stopping criterion is evaluated. Early stopping is triggered when the change in the loss function drops below the specified tolerance within the last num_patience_epochs number of epochs. To disable early stopping, set num_patience_epochs to a value larger than epochs.  
**Optional**  
Valid values: Positive integer (min: 1)  
Default value: 3
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimizer</td>
<td>The optimizer to use for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• sgd: Stochastic gradient descent</td>
</tr>
<tr>
<td></td>
<td>• adam: Adaptive momentum estimation</td>
</tr>
<tr>
<td></td>
<td>• adagrad: Adaptive gradient algorithm</td>
</tr>
<tr>
<td></td>
<td>• adadelta: An adaptive learning rate algorithm</td>
</tr>
<tr>
<td></td>
<td>• rmsprop: Root mean square propagation</td>
</tr>
<tr>
<td></td>
<td>Default value: adadelta</td>
</tr>
<tr>
<td>rescale_gradient</td>
<td>The rescale factor for gradient.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float (min: 1e-3, max: 1.0)</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.0</td>
</tr>
<tr>
<td>sub_sample</td>
<td>The fraction of the training data to sample for training per epoch.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 0.0, max: 1.0)</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.0</td>
</tr>
<tr>
<td>tolerance</td>
<td>The maximum relative change in the loss function. Early stopping is triggered when change in the loss function drops below this value within the last num_patience_epochs number of epochs.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 1e-6, max: 0.1)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The weight decay coefficient. Adds L2 regularization.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float (min: 0.0, max: 1.0)</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
</tbody>
</table>

**Tune an NTM Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.
Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. Latent representations use inferred variables that are not directly measured to model the observations in a dataset. Automatic model tuning on NTM helps you find the model that minimizes loss over the training or validation data. Training loss measures how well the model fits the training data. Validation loss measures how well the model can generalize to data that it is not trained on. Low training loss indicates that a model is a good fit to the training data. Low validation loss indicates that a model has not overfit the training data and so should be able to model documents on which it has not been trained successfully. Usually, it's preferable to have both losses be small. However, minimizing training loss too much might result in overfitting and increase validation loss, which would reduce the generality of the model.

For more information about model tuning, see Automatic Model Tuning (p. 290).

Metrics Computed by the NTM Algorithm

The NTM algorithm reports a single metric that is computed during training: validation:total_loss. The total loss is the sum of the reconstruction loss and Kullback-Leibler divergence. When tuning hyperparameter values, choose this metric as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:total_loss</td>
<td>Total Loss on validation set</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

Tunable NTM Hyperparameters

You can tune the following hyperparameters for the NTM algorithm. Usually setting low mini_batch_size and small learning_rate values results in lower validation losses, although it might take longer to train. Low validation losses don't necessarily produce more coherent topics as interpreted by humans. The effect of other hyperparameters on training and validation loss can vary from dataset to dataset. To see which values are compatible, see NTM Hyperparameters (p. 175).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoder_layers_activation</td>
<td>CategoricalParameterRanges</td>
<td>['sigmoid', 'tanh', 'relu']</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-4, MaxValue: 0.1</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 16, MaxValue: 2048</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'adadelta']</td>
</tr>
<tr>
<td>rescale_gradient</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.1, MaxValue: 1.0</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
</tbody>
</table>

NTM Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in Common Data Formats - Inference. This topic contains a list of the available output formats for the Amazon SageMaker NTM algorithm.
Object2Vec Algorithm

Object2Vec is a general-purpose neural embedding algorithm that is highly customizable. It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in such a way that the semantics of the relationship between pairs of objects in the original space are preserved in the embedding space. You can use the learned embeddings, for example, to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space. You can also use the embeddings as features of the corresponding objects in downstream supervised tasks, such as classification or regression.

Object2Vec generalizes the well-known Word2Vec embedding technique for words that is optimized in the Amazon SageMaker BlazingText Algorithm (p. 73). For a blog post that discusses how to apply Object2Vec to some practical use cases, see Introduction to Amazon SageMaker Object2Vec. For

Topics

- Input/Output Interface for the Object2Vec Algorithm (p. 180)
- EC2 Instance Recommendation for the Object2Vec Algorithm (p. 180)
- Object2Vec Sample Notebooks (p. 181)
- How Object2Vec Works (p. 181)
Input/Output Interface for the Object2Vec Algorithm

You can use Object2Vec on many different input data types, including the following:

- Sentence-sentence pairs
- Labels-sequence pairs
- Customer-customer pairs
- Product-product pairs
- Item review user-item pairs

Natively, Object2Vec currently supports two types of input:

- Discrete tokens, which are represented as a list consisting of a single integer-id. For example, [10].
- Sequences of discrete tokens, which are represented as lists of integer-ids. For example, [0, 12, 10, 13].

To transform the input data into the supported formats, you must preprocess it.

The object in each pair can be asymmetric. For example, they can be (token, sequence) pairs or (token, token) pairs or (sequence, sequence) pairs. For token inputs, the algorithm supports simple embeddings as compatible encoders. For sequences of token vectors, the algorithm supports the following as encoders:

- Average-pooled embeddings
- Hierarchical convolutional neural networks (CNNs),
- Multi-layered bidirectional long short-term memory (BiLSTMs)

The input label for each pair can be a categorical label that expresses the relationship between the objects in the pair or it can be a rating/score that expresses the strength of the similarity between the two objects. For categorical labels used in classification, the algorithm supports the cross-entropy loss function. For ratings/score-based labels used in regression, the algorithm supports the mean squared error (MSE) loss function. You specify these loss functions with the `output_layer` hyperparameter.

EC2 Instance Recommendation for the Object2Vec Algorithm

Instance Recommendation for Training

To start, try running training on a CPU, using, for example, an ml.m5.2xlarge instance, or on a GPU using, for example, an ml.p2.xlarge instance. Currently, the Object2Vec algorithm is only set up to train on a single machine. However, it does offer support for multiple GPUs.

Instance Recommendation for Inference

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.
Object2Vec Sample Notebooks

For a sample notebook that uses the Amazon SageMaker Object2Vec algorithm to encode sequences into fixed-length embeddings, see Using Object2Vec to Encode Sentences into Fixed Length Embeddings. For a sample notebook that uses the Amazon SageMaker Object2Vec algorithm in the multi label prediction setting to predict the genre of a movie from its plot description, see Movie genre prediction with Object2Vec Algorithm. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). After you have created a notebook instance and opened it, choose SageMaker Examples to see a list of Amazon SageMaker samples. To open a notebook, choose its Use tab and choose Create copy.

How Object2Vec Works

There are three tasks to complete when using the Amazon SageMaker Object2Vec algorithm: process the data, train the model, and produce inferences.

Topics
- Step 1: Process Data (p. 181)
- Step 2: Train a Model (p. 181)
- Step 3: Produce Inferences (p. 182)

Step 1: Process Data

The data must be converted to the JSON Lines text file format specified in the Data Formats for Object2Vec Training (p. 190) section during preprocessing. Also, to ensure obtaining the best accuracy from training, the data needs to be randomly shuffled before feeding it into the model.

Step 2: Train a Model

The architecture of Amazon SageMaker Object2Vec consists of following main components:

- Two input channels—The 2 input channels take a pair of objects of same or different types as inputs, and pass them to independent and customizable encoders. An example of the input objects could be sequence pairs, tokens pairs, or sequence and tokens pairs.

- Two encoders: The enc0 and enc1 encoders convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair, which are then passed into a comparator.

- Comparator—The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects for each user-specified relationship-type. The output scores can vary between 1, indicating a strong relationship between a sentence pair, and 0, representing a weak relationship.

At training time, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types. They are passed through independent, customizable encoders that are compatible with the input types of corresponding objects. The encoders convert each object in a pair into a fixed-length embedding vector of equal length. The encoded embeddings of the objects in the pair are then passed into a comparator that compares the embeddings in different ways and outputs scores. The scores correspond to the strength of the relationship between the objects in the pair for each relationship-type specified by the user. The training loss function can then minimize the disagreement between the relationships predicted by the model and those specified by the user-provided labels in the training data.
General Architecture of Object2Vec: The network accepts two inputs and their relationship label at training time. Each of the inputs is passed into a compatible encoder, and the embeddings output by the encoders are passed into a comparator that emits the confidence score corresponding to the relationship label.

**Step 3: Produce Inferences**

After the model has been trained, you can use the trained encoder to perform inference in two modes:

- To convert singleton input objects into fixed length embeddings using the corresponding encoder
- To predict the relationship label or score between a pair of input objects

The inference server automatically figures out which of the modes is requested based on the input data. To get the embeddings as output, provide only one input in each instance. To predict the relationship label or score, provide both inputs in the pair.
## Object2Vec Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enc0_max_seq_len</td>
<td>The maximum sequence length for the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 5000$</td>
</tr>
<tr>
<td>enc0_vocab_size</td>
<td>The vocabulary size of enc0 tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 3000000$</td>
</tr>
<tr>
<td>bucket_width</td>
<td>The allowed difference between data sequence length when bucketing is enabled. Bucketing is enabled when a non-zero value is specified for this parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{integer} \leq 100$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0 (no bucketing)</td>
</tr>
<tr>
<td>dropout</td>
<td>The dropout probability for network layers. Dropout is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0.0 \leq \text{float} \leq 1.0$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by the early_stopping_tolerance.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 5$</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after an early_stopping_patience number of consecutive epochs.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0.000001 \leq \text{float} \leq 0.1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</td>
</tr>
<tr>
<td>enc_dim</td>
<td>The dimension of the output of the embedding layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enc0_network</td>
<td>Network model for the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: hcnn, bilstm, or pooled_embedding</td>
</tr>
<tr>
<td></td>
<td>• hcnn: A heterogeneous convolutional neural network</td>
</tr>
<tr>
<td></td>
<td>• bilstm: A bidirectional long short-term memory network (LSTM), in which</td>
</tr>
<tr>
<td></td>
<td>the signal propagates backward as well as forward in time. This is an</td>
</tr>
<tr>
<td></td>
<td>appropriate recurrent neural network (RNN) architecture for sequential</td>
</tr>
<tr>
<td></td>
<td>learning tasks.</td>
</tr>
<tr>
<td></td>
<td>• pooled_embedding: Averages the embeddings of all the tokens in the input.</td>
</tr>
<tr>
<td></td>
<td>Default value: hcnn</td>
</tr>
<tr>
<td>enc0_cnn_filter_width</td>
<td>The filter width of the convolutional neural network (CNN) enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 1 ≤ integer ≤ 9</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>enc0_freeze_pretrained_embedding</td>
<td>Whether to freeze enc0 pretrained embedding weights.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: True</td>
</tr>
<tr>
<td>enc0_layers</td>
<td>The number of layers in the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or 1 ≤ integer ≤ 4</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>enc0_pretrained_embedding_file</td>
<td>The filename of pretrained enc0 token embedding file in the auxiliary data channel.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period.</td>
</tr>
<tr>
<td></td>
<td>[A-Za-z0-9_/__]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>enc0_token_embedding_dim</code></td>
<td>The output dimension of the enc0 token embedding layer.</td>
</tr>
<tr>
<td><strong>Conditional</strong></td>
<td>Valid values: 2 ≤ integer ≤ 1000</td>
</tr>
<tr>
<td>Default value: 300</td>
<td></td>
</tr>
<tr>
<td><code>enc0_vocab_file</code></td>
<td>The vocabulary file for mapping pretrained enc0 token embeddings to vocabulary IDs.</td>
</tr>
<tr>
<td><strong>Conditional</strong></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [a-zA-Z0-9._]</td>
</tr>
<tr>
<td>Default value: &quot;&quot; (empty string)</td>
<td></td>
</tr>
<tr>
<td><code>enc1_network</code></td>
<td>The network model for the enc1 encoder. If its value is set to enc0, then enc1 uses the same network model as enc0, including the hyperparameter values. Note that although the enc0 and enc1 encoder networks may have symmetric architecture, shared parameter values for these networks is not supported.</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>Valid values: <code>hcnn</code>, <code>bilstm</code>, or <code>pooled_embedding</code></td>
</tr>
<tr>
<td></td>
<td>• enc0: Network model for the enc0 encoder.</td>
</tr>
<tr>
<td></td>
<td>• hcnn: A heterogeneous convolutional neural network</td>
</tr>
<tr>
<td></td>
<td>• bilstm: A bidirectional LSTM, in which the signal propagates backward as well as forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks.</td>
</tr>
<tr>
<td></td>
<td>• pooled_embedding: Averages the embeddings of all the tokens in the input.</td>
</tr>
<tr>
<td>Default value: <code>enc0</code></td>
<td></td>
</tr>
<tr>
<td><code>enc1_cnn_filter_width</code></td>
<td>The filter width of the CNN enc1 encoder.</td>
</tr>
<tr>
<td><strong>Conditional</strong></td>
<td>Valid values: 1 ≤ integer ≤ 9</td>
</tr>
<tr>
<td>Default value: 3</td>
<td></td>
</tr>
<tr>
<td><code>enc1_freeze_pretrained_embedding</code></td>
<td>Whether to freeze enc1 pretrained embedding weights.</td>
</tr>
<tr>
<td><strong>Conditional</strong></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td>Default value: True</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enc1_layers</td>
<td>The number of layers in the enc1 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: auto or $1 \leq \text{integer} \leq 4$</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>enc1_max_seq_len</td>
<td>The maximum sequence length for the enc1 encoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 5000$</td>
</tr>
<tr>
<td>enc1_pretrained_embedding_file</td>
<td>The filename of the enc1 pretrained token embedding file in the auxiliary data channel.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9_.]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>enc1_token_embedding_dim</td>
<td>The output dimension of enc1 token embedding layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 1000$</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
<tr>
<td>enc1_vocab_file</td>
<td>The vocabulary file for mapping pretrained enc1 token embeddings to vocabulary IDs.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9_.]</td>
</tr>
<tr>
<td></td>
<td>Default value: &quot;&quot; (empty string)</td>
</tr>
<tr>
<td>enc1_vocab_size</td>
<td>The vocabulary size of enc0 tokens.</td>
</tr>
<tr>
<td></td>
<td><strong>Conditional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 3000000$</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of epochs to run for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 100$</td>
</tr>
<tr>
<td></td>
<td>Default value: 30</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1.0\times 10^{-6} \leq \text{float} \leq 1.0$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0004</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size that the data set is split into for an optimizer during training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $1 \leq \text{integer} \leq 10000$</td>
</tr>
<tr>
<td></td>
<td>Default value: 32</td>
</tr>
<tr>
<td>mlp_activation</td>
<td>The type of activation function for the multilayer perceptron (MLP) layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: \text{tanh}, \text{relu}, or \text{linear}</td>
</tr>
<tr>
<td></td>
<td>• \text{tanh}: Hyperbolic tangent</td>
</tr>
<tr>
<td></td>
<td>• \text{relu}: Rectified linear unit (ReLU)</td>
</tr>
<tr>
<td></td>
<td>• \text{linear}: Linear function</td>
</tr>
<tr>
<td></td>
<td>Default value: \text{linear}</td>
</tr>
<tr>
<td>mlp_dim</td>
<td>The dimension of the output from multilayer perceptron (MLP) layers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 10000$</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td>mlp_layers</td>
<td>The number of multilayer perceptron (MLP) layers in the network.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 \leq \text{integer} \leq 10$</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>num_classes</td>
<td>The number of classes for classification training. Ignored for regression problems.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $2 \leq \text{integer} \leq 30$</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer type.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: One of adadelta, adagrad, adam, sgd, or rmsprop.</td>
</tr>
<tr>
<td></td>
<td>• adadelta: A per-dimension learning rate method for gradient descent</td>
</tr>
<tr>
<td></td>
<td>• adagrad: Adaptive gradient algorithm</td>
</tr>
<tr>
<td></td>
<td>• adam: Adaptive moment estimation algorithm</td>
</tr>
<tr>
<td></td>
<td>• sgd: Stochastic gradient descent</td>
</tr>
<tr>
<td></td>
<td>• rmsprop: Root mean square propagation</td>
</tr>
<tr>
<td></td>
<td>Default value: sgd</td>
</tr>
<tr>
<td>output_layer</td>
<td>The type of output layer.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: softmax or mean_squared_error:</td>
</tr>
<tr>
<td></td>
<td>• softmax: The Softmax function used for classification.</td>
</tr>
<tr>
<td></td>
<td>• mean_squared_error: The MSE used for regression.</td>
</tr>
<tr>
<td></td>
<td>Default value: softmax</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The weight decay parameter used for optimization.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 10000</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>

### Tune an Object2Vec Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the chosen hyperparameters to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [*Automatic Model Tuning* (p. 290)].

### Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm has both classification and regression metrics. The **output_layer** type determines which metric you can use for automatic model tuning.

**Regressor Metrics Computed by the Object2Vec Algorithm**

The algorithm reports a mean squared error regressor metric, which is computed during testing and validation. When tuning the model for regression tasks, choose this metric as the objective.
## Classification Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm reports accuracy and cross-entropy classification metrics, which are computed during test and validation. When tuning the model for classification tasks, choose one of these as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:mean_squared_error</td>
<td>Root Mean Square Error</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mean_squared_error</td>
<td>Root Mean Square Error</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>test:cross_entropy</td>
<td>Cross-entropy</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:accuracy</td>
<td>Accuracy</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:cross_entropy</td>
<td>Cross-entropy</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

## Tunable Object2Vec Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>dropout</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.001, MaxValue: 0.1</td>
</tr>
<tr>
<td>enc_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 4, MaxValue: 4096</td>
</tr>
<tr>
<td>enc0_cnn_filter_width</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>enc0_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>enc0_token_embedding_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 5, MaxValue: 300</td>
</tr>
<tr>
<td>enc1_cnn_filter_width</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 5</td>
</tr>
<tr>
<td>enc1_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter Type</td>
<td>Recommended Ranges</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>enc1_token_embedding_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 5, MaxValue: 300</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRange</td>
<td>MinValue: 4, MaxValue: 20</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-6, MaxValue: 1.0</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 8192</td>
</tr>
<tr>
<td>mlp_activation</td>
<td>CategoricalParameterRanges</td>
<td>[tanh, relu, linear]</td>
</tr>
<tr>
<td>mlp_dim</td>
<td>IntegerParameterRange</td>
<td>MinValue: 16, MaxValue: 1024</td>
</tr>
<tr>
<td>mlp_layers</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 4</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>[adagrad, adam, rmsprop, sgd, adadelta]</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 1.0</td>
</tr>
</tbody>
</table>

Data Formats for Object2Vec Training

**INPUT: JSONLINES Request Format**

Content-type: application/jsonlines

```
{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}
{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}
{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

The "in0" and "in1" are the inputs for encoder0 and encoder1, respectively. The same format is valid for both classification and regression problems. For regression, the field "label" can accept real valued inputs.

Data Formats for Object2Vec Inference

**INPUT: CLASSIFICATION OR REGRESSION Request Format**

Content-type: application/json

```
{
    "instances" : [
        {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]},
        {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]},
        {"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
    ]
}```
For classification problems, the length of the scores vector corresponds to `num_classes`. For regression problems, the length is 1.

**OUTPUT: CLASSIFICATION OR REGRESSION Response Format**

Accept: application/json

```json
{
  "predictions": [
    {
      "scores": [0.6533935070037842, 0.07582679390907288, 0.2707797586917877]
    },
    {
      "scores": [0.026291321963071823, 0.6577019095420837, 0.31600672006607056]
    }
  ]
}
```

Accept: application/jsonlines

```json
{"scores":[0.195667684078216,0.395351558923721,0.408980727195739]}
{"scores":[0.251988261938095,0.258233487606048,0.489778339862823]}
{"scores":[0.280087798833847,0.368331134319305,0.35158109649169]}
```

In both the classification and regression formats, the scores correspond to each of the labels.

**Encoder Embeddings for Object2Vec**

**INPUT: ENCODER EMBEDDINGS**

Content-type: application/json

```json
{
  "instances" : [
    {
      "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]},
    {
      "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]},
    {
      "in0": [774, 14, 21, 206]}
  ]
}
```
In both these formats, you specify only one input type, either “in0” or “in1.” The inference service then invokes the corresponding encoder and outputs the embeddings for each of the instances.

**OUTPUT: ENCODER EMBEDDINGS**

```
Content-type: application/json

{
  "predictions": [
    {
      "embeddings":
      [0.057368703186511,0.030703511089086,0.099890425801277,0.063688032329082,0.026327300816774,0.003637571120634,0.021305780857801,0.004316598642617,0.0,0.003397724591195,0.0,0.000378780066967,0.0,0.0,0.0,0.007419463712722],

    },
    {
      "embeddings":
      [0.150190666317939,0.05145975202322,0.098204270005226,0.064249359071254,0.056249320507049,0.01513972133]
    }
  ]
}
```

The vector length of the embeddings output by the inference service is equal to the value of one of the hyperparameters, which you specify at training time: `enc0_token_embedding_dim`, `enc1_token_embedding_dim`, or `enc_dim`.

## Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene. The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box. It uses the Single Shot multibox Detector (SSD) framework and supports two base networks: VGG and ResNet. The network can be trained from scratch, or trained with models that have been pre-trained on the ImageNet dataset.

### Topics
- Input/Output Interface for the Object Detection Algorithm (p. 193)
- EC2 Instance Recommendation for the Object Detection Algorithm (p. 206)
- Object Detection Sample Notebooks (p. 206)
- How Object Detection Works (p. 206)
- Object Detection Hyperparameters (p. 206)
- Tune an Object Detection Model (p. 211)
- Object Detection Request and Response Formats (p. 212)
Input/Output Interface for the Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm supports both RecordIO (application/x-recordio) and image (image/png, image/jpeg, and application/x-image) content types for training in file mode and supports RecordIO (application/x-recordio) for training in pipe mode. However you can also train in pipe mode using the image files (image/png, image/jpeg, and application/x-image), without creating RecordIO files, by using the augmented manifest format. The recommended input format for the Amazon SageMaker object detection algorithms is Apache MXNet RecordIO. However, you can also use raw images in .jpg or .png format. The algorithm supports only application/x-image for inference.

**Note**
To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

See the Object Detection Sample Notebooks (p. 206) for more details on data formats.

Train with the RecordIO Format

If you use the RecordIO format for training, specify both train and validation channels as values for the InputDataConfig parameter of the CreateTrainingJob (p. 556) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to application/x-recordio. An example of how to generate RecordIO file can be found in the object detection sample notebook. You can also use tools from the MXNet example to generate RecordIO files for popular datasets like the PASCAL Visual Object Classes and Common Objects in Context (COCO).

Train with the Image Format

If you use the image format for training, specify train, validation, train_annotation, and validation_annotation channels as values for the InputDataConfig parameter of CreateTrainingJob (p. 556) request. Specify the individual image data (.jpg or .png) files for the train and validation channels. For annotation data, you can use the JSON format. Specify the corresponding .json files in the train_annotation and validation_annotation channels. Set the content type for all four channels to image/png or image/jpeg based on the image type. You can also use the content type application/x-image when your dataset contains both .jpg and .png images. The following is an example of a .json file.

```json
{
    "file": "your_image_directory/sample_image1.jpg",
    "image_size": [
        {
            "width": 500,
            "height": 400,
            "depth": 3
        }
    ],
    "annotations": [
        {
            "class_id": 0,
            "left": 111,
            "top": 134,
            "width": 61,
            "height": 128
        },
        {
            "class_id": 0,
            "left": 161,
            "top": 250,
            "width": 79,
            "height": 193
        }
    ]
}
```
Each image needs a .json file for annotation, and the .json file should have the same name as the corresponding image. The name of above .json file should be "sample_image1.json". There are four properties in the annotation .json file. The property "file" specifies the relative path of the image file. For example, if your training images and corresponding .json files are stored in s3://your_bucket/train/sample_image and s3://your_bucket/train_annotation, specify the path for your train and train_annotation channels as s3://your_bucket/train and s3://your_bucket/train_annotation, respectively.

In the .json file, the relative path for an image named sample_image1.jpg should be sample_image/sample_image1.jpg. The "image_size" property specifies the overall image dimensions. The SageMaker object detection algorithm currently only supports 3-channel images. The "annotations" property specifies the categories and bounding boxes for objects within the image. Each object is annotated by a "class_id" index and by four bounding box coordinates ("left", "top", "width", "height"). The "left" (x-coordinate) and "top" (y-coordinate) values represent the upper-left corner of the bounding box. The "width" (x-coordinate) and "height" (y-coordinate) values represent the dimensions of the bounding box. The origin (0, 0) is the upper-left corner of the entire image. If you have multiple objects within one image, all the annotations should be included in a single .json file. The "categories" property stores the mapping between the class index and class name. The class indices should be numbered successively and the numbering should start with 0. The "categories" property is optional for the annotation .json file.

### Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the InputDataConfig parameter of the Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a deep learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- `AlgorithmSpecification` - Identifies the training algorithm to use.
• **HyperParameters** - Specify these algorithm-specific parameters to influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

• **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.

• **OutputDataConfig** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results of model training.

• **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

• **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.

• **StoppingCondition** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see How It Works.

### Request Syntax

```json
{
   "AlgorithmSpecification": {
      "AlgorithmName": "string",
      "MetricDefinitions": [
         {
            "Name": "string",
            "Regex": "string"
         }
      ],
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "EnableInterContainerTrafficEncryption": boolean,
   "EnableNetworkIsolation": boolean,
   "HyperParameters": {
      "string": "string"
   },
   "InputDataConfig": [
      {
         "ChannelName": "string",
         "CompressionType": "string",
```
"S3OutputPath": "string"

},

"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number

},

"RoleArn": "string",

"StoppingCondition": {
  "MaxRuntimeInSeconds": number

},

"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
],

"TrainingJobName": "string",

"VpcConfig": {
  "SecurityGroupId": [ "string" ],
  "Subnets": [ "string" ]

}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 556)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 744) object
Note
The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean
Required: No

HyperParameters (p. 556)
Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map
Key Length Constraints: Maximum length of 256.
Value Length Constraints: Maximum length of 256.
Required: No

InputDataConfig (p. 556)
An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3 location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams.

Type: Array of Channel (p. 756) objects
Array Members: Minimum number of 1 item. Maximum number of 8 items.
Required: No

OutputDataConfig (p. 556)
Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object
Required: Yes

ResourceConfig (p. 556)
The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.
Type: ResourceConfig (p. 866) object

Required: Yes

RoleArn (p. 556)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_]+$`

Required: Yes

StoppingCondition (p. 556)

Sets a duration for training. Use this parameter to cap model training costs. To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the `CreateModel` API.

Type: StoppingCondition (p. 879) object

Required: Yes

Tags (p. 556)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobName (p. 556)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: `^[a-zA-Z0-9\-]*[a-zA-Z0-9]+$`

Required: Yes
VpcConfig (p. 556)

A VpcConfig (p. 913) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

Response Syntax

```
{
  "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 560)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
Object Detection Algorithm

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

(request) request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in JSON Lines format in which each line represents one sample. The images are specified using the 'source-ref' tag that points to the S3 location of the image. The annotations are provided under the "AttributeName" parameter value as specified in the

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a deep learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.

- **HyperParameters** - Specify these algorithm-specific parameters to influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

- **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.

- **OutputDataConfig** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results of model training.

- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.

- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.

- **StoppingCondition** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see How It Works.

**Request Syntax**

```json
{
   "AlgorithmSpecification": {
       "AlgorithmName": "string",
       "MetricDefinitions": [
           {
               "Name": "string", 200
           }
       ]
   }
}
```
"Regex": "string"
}
],
"TrainingImage": "string",
"TrainingInputMode": "string"
},
"EnableInterContainerTrafficEncryption": boolean,
"EnableNetworkIsolation": boolean,
"HyperParameters": {
  "string": "string"
},
"InputDataConfig": [
{
  "ChannelName": "string",
  "CompressionType": "string",
  "ContentType": "string",
  "DataSource": {
    "S3DataSource": {
      "AttributeNames": [ "string" ],
      "S3DataDistributionType": "string",
      "S3DataType": "string",
      "S3Uri": "string"
    }
  }
},
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
  "Seed": number
}
]

"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
}
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
}
"RoleArn": "string",
"StoppingCondition": {
  "MaxRuntimeInSeconds": number
}
"Tags": [{
  "Key": "string",
  "Value": "string"
}]
"TrainingJobName": "string",
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
}
AlgorithmSpecification (p. 556)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 744) object

Required: Yes

EnableInterContainerTrafficEncryption (p. 556)

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableNetworkIsolation (p. 556)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note
The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 556)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

InputDataConfig (p. 556)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3 location where the input data is stored.
It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

OutputDataConfig (p. 556)

Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object

Required: Yes

ResourceConfig (p. 556)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 866) object

Required: Yes

RoleArn (p. 556)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

**Note**
To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@-_\\/]+$`

Required: Yes

StoppingCondition (p. 556)

Sets a duration for training. Use this parameter to cap model training costs. To stop a job, Amazon SageMaker sends the algorithm the `SIGTERM` signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.
When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the CreateModel API.

Type: StoppingCondition (p. 879) object

Required: Yes

**Tags (p. 556)**

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**TrainingJobName (p. 556)**

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

**VpcConfig (p. 556)**

A VpcConfig (p. 913) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

**Response Syntax**

```
{
  "TrainingJobArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**TrainingJobArn (p. 560)**

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:training-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with Amazon SageMaker. Incremental training saves training time when you want to train a...
new model with the same or similar data. Amazon SageMaker object detection models can be seeded only with another build-in object detection model trained in Amazon SageMaker.

To use a pretrained model, in the CreateTrainingJob request, specify the ChannelName as "model" in the InputDataConfig parameter. Set the ContentType for the model channel to application/x-sagemaker-model. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the base_network and num_classes input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker object detection algorithm, see Amazon SageMaker Object Detection Incremental Training sample notebook. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker.

**EC2 Instance Recommendation for the Object Detection Algorithm**

For object detection, we support the following GPU instances for training: ml.p2.xlarge, ml.p2.8xlarge, ml.p2.16xlarge, ml.p3.2xlarge, ml.p3.8xlarge and ml.p3.16xlarge. We recommend using GPU instances with more memory for training with large batch sizes. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training. However, both CPU (such as C5 and M5) and GPU (such as P2 and P3) instances can be used for the inference. All the supported instance types for inference are itemized on Amazon SageMaker ML Instance Types.

**Object Detection Sample Notebooks**

For a sample notebook that shows how to use the Amazon SageMaker Object Detection algorithm to train and host a model on the COCO dataset using the Single Shot multibox Detector algorithm, see Object Detection using the Image and JSON format. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances. Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**How Object Detection Works**

The object detection algorithm identifies and locates all instances of objects in an image from a known collection of object categories. The algorithm takes an image as input and outputs the category that the object belongs to, along with a confidence score that it belongs to the category. The algorithm also predicts the object's location and scale with a rectangular bounding box. Amazon SageMaker Object Detection uses the Single Shot multibox Detector (SSD) algorithm that takes a convolutional neural network (CNN) pretrained for classification task as the base network. SSD uses the output of intermediate layers as features for detection. Various CNNs such as VGG and ResNet have achieved great performance on the image classification task. Object detection in Amazon SageMaker supports both VGG-16 and ResNet-50 as a base network for SSD. The algorithm can be trained in full training mode or in transfer learning mode. In full training mode, the base network is initialized with random weights and then trained on user data. In transfer learning mode, the base network weights are loaded from pretrained models.

**Object Detection Hyperparameters**

In the CreateTrainingJob request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters that are used to help estimate the parameters of the model from a training dataset. The following table lists the hyperparameters provided by Amazon.
SageMaker for training the object detection algorithm. For more information about how object training works, see How Object Detection Works (p. 206).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes</td>
<td>The number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>num_training_samples</td>
<td>The number of training examples in the input dataset.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>If there is a mismatch between this value and the number of samples in the training set, then the behavior of the lr_scheduler_step parameter will be undefined and distributed training accuracy may be affected.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>base_network</td>
<td>The base network architecture to use.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 'vgg-16' or 'resnet-50'</td>
</tr>
<tr>
<td></td>
<td>Default value: 'vgg-16'</td>
</tr>
<tr>
<td>early_stopping</td>
<td>True to use early stopping logic during training. False not to use it.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: True or False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>early_stopping_min_epochs</td>
<td>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of epochs to wait before ending training if no improvement, as defined by the early_stopping_tolerance hyperparameter, is made in the relevant metric. It is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| early_stopping_tolerance | The tolerance value that the relative improvement in validation:mAP, the mean average precision (mAP), is required to exceed to avoid early stopping. If the ratio of the change in the mAP divided by the previous best mAP is smaller than the `early_stopping_tolerance` value set, early stopping considers that there is no improvement. It is used only when `early_stopping` = True. **Optional**
  - Valid values: $0 \leq \text{float} \leq 1$
  - Default value: 0.0 |
| image_shape | The image size for input images. We rescale the input image to a square image with this size. We recommend using 300 and 512 for better performance. **Optional**
  - Valid values: positive integer $\geq 300$
  - Default: 300 |
| epochs | The number of training epochs. **Optional**
  - Valid values: positive integer
  - Default: 30 |
| freeze_layer_pattern | The regular expression (regex) for freezing layers in the base network. For example, if we set `freeze_layer_pattern = "^\(\text{conv1_|conv2_}\).\.*"`, then any layers with a name that contains "conv1_" or "conv2_" are frozen, which means that the weights for these layers are not updated during training. The layer names can be found in the network symbol files [here](#) and [here](#). **Optional**
  - Valid values: string
  - Default: No layers frozen. |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **kv_store**  | The weight update synchronization mode used for distributed training. The weights can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See the Distributed Training MXNet tutorial for details.  
**Note**  
This parameter is not applicable to single machine training.  
**Optional**  
Valid values: 'dist_sync' or 'dist_async'  
- 'dist_sync': The gradients are synchronized after every batch with all the workers. With 'dist_sync', batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like a single machine with batch size n*b.  
- 'dist_async': Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed.  
Default: - |
| **label_width** | The force padding label width used to sync across training and validation data. For example, if one image in the data contains at most 10 objects, and each object's annotation is specified with 5 numbers, [class_id, left, top, width, height], then the label_width should be no smaller than (10*5 + header information length). The header information length is usually 2. We recommend using a slightly larger label_width for the training, such as 60 for this example.  
**Optional**  
Valid values: Positive integer large enough to accommodate the largest annotation information length in the data.  
Default: 350 |
| **learning_rate** | The initial learning rate.  
**Optional**  
Valid values: float in (0, 1]  
Default: 0.001 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr_scheduler_factor</td>
<td>The ratio to reduce learning rate. Used in conjunction with the <code>lr_scheduler_step</code> parameter defined as $lr_{new} = lr_{old} \times lr_{scheduler_factor}$.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1)</td>
</tr>
<tr>
<td></td>
<td>Default: 0.1</td>
</tr>
<tr>
<td>lr_scheduler_step</td>
<td>The epochs at which to reduce the learning rate. The learning rate is reduced by <code>lr_scheduler_factor</code> at epochs listed in a comma-delimited string: &quot;epoch1, epoch2, ...&quot;. For example, if the value is set to &quot;10, 20&quot; and the <code>lr_scheduler_factor</code> is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default: -</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. In a single-machine multi-gpu setting, each GPU handles <code>mini_batch_size/num\_gpu</code> training samples. For the multi-machine training in <code>dist\_sync</code> mode, the actual batch size is <code>mini_batch_size\times number\_of\_machines</code>. A large <code>mini_batch_size</code> usually leads to faster training, but it may cause out of memory problem. The memory usage is related to <code>mini_batch_size</code>, <code>image\_shape</code>, and <code>base\_network</code> architecture. For example, on a single p3.2xlarge instance, the largest <code>mini_batch_size</code> without an out of memory error is 32 with the base network set to &quot;resnet-50&quot; and an <code>image\_shape</code> of 300. With the same instance, you can use 64 as the <code>mini_batch_size</code> with the base network vgg-16 and an <code>image\_shape</code> of 300.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default: 32</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for sgd. Ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1)</td>
</tr>
<tr>
<td></td>
<td>Default: 0.9</td>
</tr>
<tr>
<td>nms_threshold</td>
<td>The non-maximum suppression threshold.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1)</td>
</tr>
<tr>
<td></td>
<td>Default: 0.45</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimizer</td>
<td>The optimizer types. For details on optimizer values, see MXNet's API.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta']</td>
</tr>
<tr>
<td></td>
<td>Default: 'sgd'</td>
</tr>
<tr>
<td>overlap_threshold</td>
<td>The evaluation overlap threshold.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1]</td>
</tr>
<tr>
<td></td>
<td>Default: 0.5</td>
</tr>
<tr>
<td>use_pretrained_model</td>
<td>Indicates whether to use a pre-trained model for training. If set to 1, then the pre-trained model with corresponding architecture is loaded and used for training. Otherwise, the network is trained from scratch.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default: 1</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The weight decay coefficient for sgd and rmsprop. Ignored for other optimizers.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: float in (0, 1)</td>
</tr>
<tr>
<td></td>
<td>Default: 0.0005</td>
</tr>
</tbody>
</table>

**Tune an Object Detection Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Automatic Model Tuning (p. 290)](#).

**Metrics Computed by the Object Detection Algorithm**

The object detection algorithm reports on a single metric during training: `validation:mAP`. When tuning a model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:mAP</td>
<td>Mean Average Precision (mAP) computed on the validation set.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

211
Tunable Object Detection Hyperparameters

Tune the Amazon SageMaker object detection model with the following hyperparameters. The hyperparameters that have the greatest impact on the object detection objective metric are: mini_batch_size, learning_rate, and optimizer.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-6, MaxValue: 0.5</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 8, MaxValue: 64</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'rmsprop', 'adadelta']</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
</tbody>
</table>

Object Detection Request and Response Formats

Request Format

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with image/jpeg and image/png content-types.

Response Formats

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

```json
{"prediction": [ [4.0, 0.8641945540949988, 0.3088374733924866, 0.7030484080314636, 0.7110607028007507, 0.9345266819000244], [0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571, 0.9712159633636475], [4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057, 0.596758782863169], [8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051, 0.8985623121261597], [3.0, 0.42260299175977707, 0.0.19305512309074402, 0.08386176824569702, 0.39093565940856934, 0.9574796557426453] ]}
```

Each row in this .json file contains an array that represents a detected object. Each of these object arrays consists of a list of six numbers. The first number is the predicted class label. The second number is the associated confidence score for the detection. The last four numbers represent the bounding box coordinates [xmin, ymin, xmax, ymax]. These output bounding box corner indices are normalized by the overall image size. Note that this encoding is different than that use by the input .json format. For example, in the first entry of the detection result, 0.3088374733924866 is the left coordinate (x-coordinate of upper-left corner) of the bounding box as a ratio of the overall image width, 0.7030484080314636 is the top coordinate (y-coordinate of upper-left corner) of the bounding box as a ratio of the overall image height, 0.7110607028007507 is the right coordinate (x-coordinate of
Principal Component Analysis (PCA) Algorithm

PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called components, which are composites of the original features that are
uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

In Amazon SageMaker, PCA operates in two modes, depending on the scenario:

- **regular**: For datasets with sparse data and a moderate number of observations and features.
- **randomized**: For datasets with both a large number of observations and features. This mode uses an approximation algorithm.

PCA uses tabular data.

The rows represent observations you want to embed in a lower dimensional space. The columns represent features that you want to find a reduced approximation for. The algorithm calculates the covariance matrix (or an approximation thereof in a distributed manner), and then performs the singular value decomposition on this summary to produce the principal components.

**Topics**

- Input/Output Interface for the PCA Algorithm (p. 214)
- EC2 Instance Recommendation for the PCA Algorithm (p. 214)
- PCA Sample Notebooks (p. 214)
- How PCA Works (p. 215)
- PCA Hyperparameters (p. 216)
- PCA Response Formats (p. 217)

**Input/Output Interface for the PCA Algorithm**

For training, PCA expects data provided in the train channel, and optionally supports a dataset passed to the test dataset, which is scored by the final algorithm. Both recordIO-wrapped-protobuf and CSV formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

For inference, PCA supports text/csv, application/json, and application/x-recordio-protobuf. Results are returned in either application/json or application/x-recordio-protobuf format with a vector of “projections.”

For more details on training and inference file formats, see the PCA Sample Notebooks (p. 214) and the .

For more information on input and output file formats, see PCA Response Formats (p. 217) for inference and the PCA Sample Notebooks (p. 214).

**EC2 Instance Recommendation for the PCA Algorithm**

PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

**PCA Sample Notebooks**

For a sample notebook that shows how to use the Amazon SageMaker Principal Component Analysis algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see An Introduction to PCA with MNIST. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.
Principal Component Analysis (PCA) Algorithm

How PCA Works

Principal Component Analysis (PCA) is a learning algorithm that reduces the dimensionality (number of features) within a dataset while still retaining as much information as possible.

PCA reduces dimensionality by finding a new set of features called components, which are composites of the original features, but are uncorrelated with one another. The first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

It is an unsupervised dimensionality reduction algorithm. In unsupervised learning, labels that might be associated with the objects in the training dataset aren’t used.

Given the input of a matrix with rows \( x_1, \ldots, x_n \) each of dimension \( 1 \times d \), the data is partitioned into mini-batches of rows and distributed among the training nodes (workers). Each worker then computes a summary of its data. The summaries of the different workers are then unified into a single solution at the end of the computation.

Modes

The Amazon SageMaker PCA algorithm uses either of two modes to calculate these summaries, depending on the situation:

- **regular**: for datasets with sparse data and a moderate number of observations and features.
- **randomized**: for datasets with both a large number of observations and features. This mode uses an approximation algorithm.

As the algorithm’s last step, it performs the singular value decomposition on the unified solution, from which the principal components are then derived.

**Mode 1: Regular**

The workers jointly compute both \( \sum x_i^T x_i \) and \( \sum x_i \).

**Note**

Because \( x_i \) are \( 1 \times d \) row vectors, \( x_i^T x_i \) is a matrix (not a scalar). Using row vectors within the code allows us to obtain efficient caching.

The covariance matrix is computed as \( \sum x_i^T x_i - (1/n)(\sum x_i)^T \sum x_i \), and its top \( \text{num\_components} \) singular vectors form the model.

**Note**

If \( \text{subtract\_mean} \) is False, we avoid computing and subtracting \( \sum x_i \).

Use this algorithm when the dimension \( d \) of the vectors is small enough so that \( d^3 \) can fit in memory.

**Mode 2: Randomized**

When the number of features in the input dataset is large, we use a method to approximate the covariance metric. For every mini-batch \( X_b \) of dimension \( b \times d \), we randomly initialize a \( (\text{num\_components} + \text{extra\_components}) \times b \) matrix that we multiply by each mini-batch, to create a \( (\text{num\_components} + \text{extra\_components}) \times d \) matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final \( (\text{num\_components} + \text{extra\_components}) \times d \) matrix. The top right \( \text{num\_components} \) singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let \( \ell = \text{num\_components} + \text{extra\_components} \). Given a mini-batch \( X_b \) of dimension \( b \times d \), the worker draws a random matrix \( H_b \) of dimension \( \ell \times b \). Depending on whether the environment uses a GPU or CPU and the dimension size, the matrix is either a random sign matrix where each entry is \( \pm 1 \) or a FJLT (fast Johnson Lindenstrauss transform; for information, see FJLT Transforms and the follow-
up papers). The worker then computes $H_i X_i$ and maintains $B = \sum H_i X_i$. The worker also maintains $h^T$, the sum of columns of $H_1, \ldots, H_T$ ($T$ being the total number of mini-batches), and $a$, the sum of all input rows. After processing the entire shard of data, the worker sends the server $B$, $h$, $s$, and $n$ (the number of input rows).

Denote the different inputs to the server as $B^i, h^i, s^i, n^i$. The server computes $B$, $h$, $s$, $n$ the sums of the respective inputs. It then computes $C = B - (1/n)h^T s$, and finds its singular value decomposition. The top-right singular vectors and singular values of $C$ are used as the approximate solution to the problem.

**PCA Hyperparameters**

In the CreateTrainingJob request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see How PCA Works (p. 215).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>Input dimension. Required, Valid values: positive integer</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of rows in a mini-batch. Required, Valid values: positive integer</td>
</tr>
<tr>
<td>num_components</td>
<td>The number of principal components to compute. Required, Valid values: positive integer</td>
</tr>
<tr>
<td>algorithm_mode</td>
<td>Mode for computing the principal components. Optional, Valid values: regular or randomized</td>
</tr>
<tr>
<td>extra_components</td>
<td>As the value increases, the solution becomes more accurate but the runtime and memory consumption increase linearly. The default, -1, means the maximum of 10 and num_components. Valid for randomized mode only. Optional, Valid values: Non-negative integer or -1</td>
</tr>
<tr>
<td>subtract_mean</td>
<td>Indicates whether the data should be unbiased both during training and at inference. Optional</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>Valid values: One of true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
</tbody>
</table>

## PCA Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker PCA algorithm.

### JSON Response Format

Accept—application/json

```json
{
  "projections": [
    {
      "projection": [1.0, 2.0, 3.0, 4.0, 5.0]
    },
    {
      "projection": [6.0, 7.0, 8.0, 9.0, 0.0]
    },
    ....
  ]
}
```

### JSONLINES Response Format

Accept—application/jsonlines

```json
{ "projection": [1.0, 2.0, 3.0, 4.0, 5.0] }
{ "projection": [6.0, 7.0, 8.0, 9.0, 0.0] }
```

### RECORDIO Response Format

Accept—application/x-recordio-protobuf

```json
[
  Record = {
    features = {},
    label = {
      'projection': {
        keys: [],
        values: [1.0, 2.0, 3.0, 4.0, 5.0]
      }
    }
  },
  Record = {
    features = {},
    label = {
      'projection': {
        keys: [],
        values: [1.0, 2.0, 3.0, 4.0, 5.0]
      }
    }
  }
]```
Random Cut Forest (RCF) Algorithm

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a data set can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

With each data point, RCF associates an anomaly score. Low score values indicate that the data point is considered "normal." High values indicate the presence of an anomaly in the data. The definitions of "low" and "high" depend on the application but common practice suggests that scores beyond three standard deviations from the mean score are considered anomalous.

While there are many applications of anomaly detection algorithms to one-dimensional time series data such as traffic volume analysis or sound volume spike detection, RCF is designed to work with arbitrary-dimensional input. Amazon SageMaker RCF scales well with respect to number of features, data set size, and number of instances.

Topics
- Input/Output Interface for the RCF Algorithm (p. 218)
- Instance Recommendations for the RCF Algorithm (p. 219)
- RCF Sample Notebooks (p. 219)
- How RCF Works (p. 219)
- RCF Hyperparameters (p. 222)
- Tune an RCF Model (p. 223)
- RCF Response Formats (p. 223)

Input/Output Interface for the RCF Algorithm

Amazon SageMaker Random Cut Forest supports the train and test data channels. The optional test channel is used to compute accuracy, precision, recall, and F1-score metrics on labeled data. Train and test data content types can be either application/x-recordio-protobuf or text/csv formats. For the test data, when using text/csv format, the content must be specified as text/csv;label_size=1 where the first column of each row represents the anomaly label: "1" for an anomalous data point and "0" for a normal data point. You can use either File mode or Pipe mode to train RCF models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

Also note that the train channel only supports S3DataDistributionType=ShardedByS3Key and the test channel only supports S3DataDistributionType=FullyReplicated. The S3 distribution type can be specified using the Python SDK as follows:

```python
import sagemaker

# specify Random Cut Forest training job information and hyperparameters
rcf = sagemaker.estimator.Estimator(...)

# explicitly specify "SharededByS3Key" distribution type
train_data = sagemaker.s3_input(
    s3_data=s3_training_data_location,
    content_type='text/csv;label_size=0',
    distribution='ShardedByS3Key')

# run the training job on input data stored in S3
rcf.fit({'train': train_data})
```
See the Amazon SageMaker Data Types documentation for more information on customizing the S3 data source attributes. Finally, in order to take advantage of multi-instance training the training data must be partitioned into at least as many files as instances.

For inference, RCF supports application/x-recordio-protobuf, text/csv and application/json input data content types. See the Common Data Formats for Built-in Algorithms (p. 64) documentation for more information. RCF inference returns application/x-recordio-protobuf or application/json formatted output. Each record in these output data contains the corresponding anomaly scores for each input data point. See Common Data Formats--Inference for more information.

For more information on input and output file formats, see RCF Response Formats (p. 223) for inference and the RCF Sample Notebooks (p. 219).

**Instance Recommendations for the RCF Algorithm**

For training, we recommend the ml.m4, ml.c4, and ml.c5 instance families. For inference we recommend using a ml.c5.xl instance type in particular, for maximum performance as well as minimized cost per hour of usage. Although the algorithm could technically run on GPU instance types it does not take advantage of GPU hardware.

**RCF Sample Notebooks**

For an example of how to train an RCF model and perform inferences with it, see the Introduction to SageMaker Random Cut Forests notebook. For a sample notebook that uses the Amazon SageMaker Random Cut Forest algorithm for anomaly detection, see An Introduction to SageMaker Random Cut Forests. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its Use tab and select Create copy.

**How RCF Works**

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree. The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree. The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size. The RCF algorithm is based on the one described in reference [1].

**Sample Data Randomly**

The first step in the RCF algorithm is to obtain a random sample of the training data. In particular, suppose we want a sample of size \( \kappa \) from \( N \) total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw \( \kappa \) elements from this set. However, frequently the training data is too large to fit all at once, and this approach isn't feasible. Instead, we use a technique called reservoir sampling.
**Reservoir sampling** is an algorithm for efficiently drawing random samples from a dataset \( S = \{S_1, \ldots, S_N\} \) where the elements in the dataset can only be observed one at a time or in batches. In fact, reservoir sampling works even when \( N \) is not known *a priori*. If only one sample is requested, such as when \( K = 1 \), the algorithm is like this:

**Algorithm: Reservoir Sampling**

- **Input:** dataset or data stream \( S = \{S_1, \ldots, S_N\} \)
- **Initialize the random sample** \( X = S_1 \)
- **For each observed sample** \( S_n, n = 2, \ldots, N \):
  - Pick a uniform random number \( \xi \in [0, 1] \)
  - If \( \xi < 1/n \)
    - Set \( X = S_n \)
  - Return \( X \)

This algorithm selects a random sample such that \( P(X = S_n) = 1/N \) for all \( n = 1, \ldots, N \). When \( K > 1 \) the algorithm is more complicated. Additionally, a distinction must be made between random sampling that is with and without replacement. RCF performs an augmented reservoir sampling without replacement on the training data based on the algorithms described in [2].

**Train a RFC Model and Produce Inferences**

The next step in RCF is to construct a random cut forest using the random sample of data. First, the sample is partitioned into a number of equal-sized partitions equal to the number of trees in the forest. Then, each partition is sent to an individual tree. The tree recursively organizes its partition into a binary tree by partitioning the data domain into bounding boxes.

This procedure is best illustrated with an example. Suppose a tree is given the following two-dimensional dataset. The corresponding tree is initialized to the root node:

A two-dimensional dataset where the majority of data lies in a cluster (blue) except for one anomalous data point (orange). The tree is initialized with a root node.

The RCF algorithm organizes these data in a tree by first computing a bounding box of the data, selecting a random dimension (giving more weight to dimensions with higher "variance"), and then
randomly determining the position of a hyperplane "cut" through that dimension. The two resulting subspaces define their own sub tree. In this example, the cut happens to separate a lone point from the remainder of the sample. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.

A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

Bounding boxes are then computed for the left and right halves of the data and the process is repeated until every leaf of the tree represents a single data point from the sample. Note that if the lone point is sufficiently far away then it is more likely that a random cut would result in point isolation. This observation provides the intuition that tree depth is, loosely speaking, inversely proportional to the anomaly score.

When performing inference using a trained RCF model the final anomaly score is reported as the average across scores reported by each tree. Note that it is often the case that the new data point does not already reside in the tree. To determine the score associated with the new point the data point is inserted into the given tree and the tree is efficiently (and temporarily) reassembled in a manner equivalent to the training process described above. That is, the resulting tree is as if the input data point were a member of the sample used to construct the tree in the first place. The reported score is inversely proportional to the depth of the input point within the tree.

Choose Hyperparameters

The primary hyperparameters used to tune the RCF model are num_trees and num_samples_per_tree. Increasing num_trees has the effect of reducing the noise observed in anomaly scores since the final score is the average of the scores reported by each tree. While the optimal value is application-dependent we recommend using 100 trees to begin with as a balance between score noise and model complexity. Note that inference time is proportional to the number of trees. Although training time is also affected it is dominated by the reservoir sampling algorithm described above.

The parameter num_samples_per_tree is related to the expected density of anomalies in the dataset. In particular, num_samples_per_tree should be chosen such that 1/num_samples_per_tree approximates the ratio of anomalous data to normal data. For example, if 256 samples are used in each tree then we expect our data to contain anomalies 1/256 or approximately 0.4% of the time. Again, an optimal value for this hyperparameter is dependent on the application.
References


RCF Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker RCF algorithm. For more information, including recommendations on how to choose hyperparameters, see *How RCF Works* (p. 219).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| feature_dim             | The number of features in the data set. (If you are using the client libraries through a notebook, this value is calculated for you and need not be specified.)  
**Required** (When the job is run through the console.)  
Valid values: Positive integer (min: 1, max: 10000) |
| eval_metrics            | A list of metrics used to score a labeled test data set. The following metrics can be selected for output:  
• accuracy - returns fraction of correct predictions.  
• precision_recall_fscore - returns the positive and negative precision, recall, and F1-scores.  
**Optional**  
Valid values: a list with possible values taken from accuracy or precision_recall_fscore.  
Default value: Both accuracy, precision_recall_fscore are calculated. |
| num_samples_per_tree    | Number of random samples given to each tree from the training data set.  
**Optional**  
Valid values: Positive integer (min: 1, max: 2048)  
Default value: 256 |
| num_trees               | Number of trees in the forest.  
**Optional**  
Valid values: Positive integer (min: 50, max: 1000)  
Default value: 100 |
Tune an RCF Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker RCF algorithm is an unsupervised anomaly-detection algorithm that requires a labeled test dataset for hyperparameter optimization. It calculates anomaly scores for test datapoints and then labels the datapoints as anomalous if their scores are beyond three standard deviations from the mean score. This is known as the three-sigma limit heuristic. The F1 score is emitted based on the difference between calculated labels and actual labels. The hyperparameter tuning job finds the model that maximizes that score. The success of hyperparameter optimization depends on the applicability of the three-sigma limit heuristic to the test dataset.

For more information about model tuning, see *Automatic Model Tuning* (p. 290).

Metrics Computed by the RCF Algorithm

The RCF algorithm computes the following metric during training. When tuning the model, choose this metric as the objective metric.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>test:f1</td>
<td>F1 score on the test dataset, based on the difference between calculated labels and actual labels.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable RCF Hyperparameters

You can tune a RCF model with the following hyperparameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_samples_per_tree</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 2048</td>
</tr>
<tr>
<td>num_trees</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 50, MaxValue: 1000</td>
</tr>
</tbody>
</table>

RCF Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in *Common Data Formats - Inference*. Note that Amazon SageMaker Random Cut Forest supports both dense and sparse JSON and RecordIO formats. This topic contains a list of the available output formats for the Amazon SageMaker RCF algorithm.

JSON Response Format

ACCEPT: application/json.

```json
{
}
```
"scores": [

    {"score": 0.02},

    {"score": 0.25}

]
Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm provides a fine-grained, pixel-level approach to developing computer vision applications. It tags every pixel in an image with a class label from a predefined set of classes. Tagging is fundamental for understanding scenes, which is critical to an increasing number of computer vision applications, such as self-driving vehicles, medical imaging diagnostics, and robot sensing.

For comparison, the Amazon SageMaker Image Classification Algorithm (p. 106) is a supervised learning algorithm that analyzes only whole images, classifying them into one of multiple output categories. The Object Detection Algorithm (p. 192) is a supervised learning algorithm that detects and classifies all instances of an object in an image. It indicates the location and scale of each object in the image with a rectangular bounding box.

Because the semantic segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects contained in the image. The segmentation output is
represented as an RGB or grayscale image, called a segmentation mask. A segmentation mask is an RGB (or grayscale) image with the same shape as the input image.

Amazon SageMaker semantic segmentation algorithm is built using the MXNet Gluon framework and the Gluon CV toolkit provides you with a choice of three build-in algorithms to train a deep neural network. You can use the Fully-Convolutional Network (FCN) algorithm, Pyramid Scene Parsing (PSP) algorithm, or DeepLabV3.

Each of the three algorithms has two distinct components:

- The backbone (or encoder)—A network that produces reliable activation maps of features.
- The decoder—A network that constructs the segmentation mask from the encoded activation maps.

You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: ResNet50 or ResNet101. These backbones include pretrained artifacts that were originally trained on the ImageNet classification task. You can fine-tune these backbones for segmentation using your own data. Or, you can initialize and train these networks from scratch using only your own data. The decoders are never pretrained.

To deploy the trained model for inference, use the Amazon SageMaker hosting service. During inference, you can request the segmentation mask either as a PNG image or as a set of probabilities for each class for each pixel. You can use these masks as part of a larger pipeline that includes additional downstream image processing or other applications.

Topics

- Semantic Segmentation Sample Notebooks (p. 226)
- Input/Output Interface for the Semantic Segmentation Algorithm (p. 226)
- EC2 Instance Recommendation for the Semantic Segmentation Algorithm (p. 229)
- Semantic Segmentation Hyperparameters (p. 229)

Semantic Segmentation Sample Notebooks

For a sample Jupyter notebook that uses the Amazon SageMaker semantic segmentation algorithm to train a model and deploy it to perform inferences, see the Semantic Segmentation Example. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42).

To see a list of all of the Amazon SageMaker samples, create and open a notebook instance, and choose the SageMaker Examples tab. The example semantic segmentation notebooks are located under Introduction to Amazon algorithms. To open a notebook, choose its Use tab, and choose Create copy.

Input/Output Interface for the Semantic Segmentation Algorithm

Amazon SageMaker semantic segmentation expects the customer's training dataset to be on Amazon Simple Storage Service (Amazon S3). Once trained, it produces the resulting model artifacts on Amazon S3. The input interface format for the Amazon SageMaker semantic segmentation is similar to that of most standardized semantic segmentation benchmarking datasets. The dataset in Amazon S3 is expected to be presented in two channels, one for train and one for validation using four directories, two for images and two for annotations. Annotations are expected to be uncompressed PNG images. The dataset might also have a label map that describes how the annotation mappings are established. If not, the algorithm uses a default. It also supports the augmented manifest image format (application/x-image) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an image/jpeg content type.
How Training Works

The training data is split into four directories: train, train_annotation, validation, and validation_annotation. There is a channel for each of these directories. The dataset also expected to have one label_map.json file per channel for train_annotation and validation_annotation respectively. If you don’t provide these JSON files, Amazon SageMaker provides the default set label map.

The dataset specifying these files should look similar to the following example:

```
  s3://bucket_name
  | - train
  |   | - 0000.jpg
  |   | - coffee.jpg
  | - validation
  |   | - 00a0.jpg
  |   | - banana.jpg
  | - train_annotation
  |   | - 0000.png
  |   | - coffee.png
  | - validation_annotation
  |   | - 00a0.png
  |   | - banana.png
  | - label_map
  |   | - train_label_map.json
  |   | - validation_label_map.json
```

Every JPG image in the train and validation directories has a corresponding PNG label image with the same name in the train_annotation and validation_annotation directories. This naming convention helps the algorithm to associate a label with its corresponding image during training. The train, train_annotation, validation, and validation_annotation channels are mandatory. The annotations are single-channel PNG images. The format works as long as the metadata (modes) in the image helps the algorithm read the annotation images into a single-channel 8-bit unsigned integer. For more information on our support for modes, see the Python Image Library documentation. We recommend using the 8-bit pixel, true color P mode.

The image that is encoded is a simple 8-bit integer when using modes. To get from this mapping to a map of a label, the algorithm uses one mapping file per channel, called the label map. The label map is used to map the values in the image with actual label indices. In the default label map, which is provided by default if you don’t provide one, the pixel value in an annotation matrix (image) directly index the label. These images can be grayscale PNG files or 8-bit indexed PNG files. The label map file for the unscaled default case is the following:

```
{
  "scale": "1"
}
```

To provide some contrast for viewing, some annotation software scales the label images by a constant amount. To support this, the Amazon SageMaker semantic segmentation algorithm provides a rescaling option to scale down the values to actual label values. When scaling down doesn’t convert the value to an appropriate integer, the algorithm defaults to the greatest integer less than or equal to the scale value. The following code shows how to set the scale value to rescale the label values:

```
{
  "scale": "3"
}
```
The following example shows how this "scale" value is used to rescale the `encoded_label` values of the input annotation image when they are mapped to the `mapped_label` values to be used in training. The label values in the input annotation image are 0, 3, 6, with scale 3, so they are mapped to 0, 1, 2 for training:

```
encoded_label = [0, 3, 6]
mapped_label = [0, 1, 2]
```

In some cases, you might need to specify a particular color mapping for each class. Use the map option in the label mapping as shown in the following example of a label_map file:

```
{
  "map": {
    "0": 5,
    "1": 0,
    "2": 2
  }
}
```

This label mapping for this example is:

```
encoded_label = [0, 5, 2]
mapped_label = [1, 0, 2]
```

With label mappings, you can use different annotation systems and annotation software to obtain data without a lot of preprocessing. You can provide one label map per channel. The files for a label map in the label_map channel must follow the naming conventions for the four directory structure. If you don't provide a label map, the algorithm assumes a scale of 1 (the default).

Amazon SageMaker Semantic Segmentation requires access to the internet and does not support Network Isolation Mode. If a VpcConfig was used, the subnet and security groups must allow outbound access to the internet for the algorithm to work.

### Training with the Augmented Manifest Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. The augmented manifest file contains data objects and should be in JSON Lines format, as described in the CreateTrainingJob (p. 556) request API. Each line in the manifest is an entry containing the Amazon S3 URI for the image and the URI for the annotation image.

Each JSON object in the manifest file must contain a `source-ref` key. The `source-ref` key should contain the value of the Amazon S3 URI to the image. The labels are provided under the `AttributeName` parameter value as specified in the CreateTrainingJob (p. 556) request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the example below, the `AttributeName` is `city-streets`.

```
{"source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-streets-metadata": {"job-name": "label-city-streets", }}
```

For more information on augmented manifest files, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

### Incremental Training

You can also seed the training of a new model with a model that you trained previously using Amazon SageMaker. This incremental training saves training time when you want to train a new model with the
same or similar data. Currently, incremental training is supported only for models trained with the built-in Amazon SageMaker Semantic Segmentation.

To use your own pre-trained model, specify the ChannelName as "model" in the InputDataConfig for the CreateTrainingJob (p. 556) request. Set the ContentType for the model channel to application/x-sagemaker-model. The backbone, algorithm, crop_size, and num_classes input parameters that define the network architecture must be consistently specified in the input hyperparameters of the new model and the pre-trained model that you upload to the model channel. For the pretrained model file, you can use the compressed (.tar.gz) artifacts from Amazon SageMaker outputs. You can use either RecordIO or Image formats for input data. For more information on incremental training and for instructions on how to use it, see Incremental Training in Amazon SageMaker (p. 285).

Produce Inferences

To query a trained model that is deployed to an endpoint, you need to provide an image and an AcceptType that denotes the type of output required. The endpoint takes JPEG images with an image/jpeg content type. If you request an AcceptType of image/png, the algorithm outputs a PNG file with a segmentation mask in the same format as the labels themselves. If you request an accept type of application/x-recordio-protobuf, the algorithm returns class probabilities encoded in recordio-protobuf format. The latter format outputs a 3D tensor where the third dimension is the same size as the number of classes. This component denotes the probability of each class label for each pixel.

EC2 Instance Recommendation for the Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes. The algorithm can be trained using P2/P3 EC2 Amazon Elastic Compute Cloud (Amazon EC2) instances in single machine configurations. It supports the following GPU instances for training:

- ml.p2.xlarge
- ml.p2.8xlarge
- ml.p2.16xlarge
- ml.p3.2xlarge
- ml.p3.8xlarge
- ml.p3.16xlarge

For inference, you can use either CPU instances (such as c5 and m5) and GPU instances (such as p2 and p3) or both. For information about the instance types that provide varying combinations of CPU, GPU, memory, and networking capacity for inference, see Amazon SageMaker ML Instance Types.

Semantic Segmentation Hyperparameters

The following tables list the hyperparameters supported by the Amazon SageMaker semantic segmentation algorithm for network architecture, data inputs, and training. You specify Semantic Segmentation for training in the AlgorithmName of the CreateTrainingJob (p. 556) request.

Network Architecture Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backbone</td>
<td>The backbone to use for the algorithm's encoder component.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: resnet-50, resnet-101</td>
</tr>
<tr>
<td></td>
<td>Default value: resnet-50</td>
</tr>
<tr>
<td>use_pretrained_model</td>
<td>Whether a pretrained model is to be used for the backbone.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: True, False</td>
</tr>
<tr>
<td></td>
<td>Default value: True</td>
</tr>
<tr>
<td>algorithm</td>
<td>The algorithm to use for semantic segmentation.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• fcn: Fully-Convolutional Network (FCN) algorithm</td>
</tr>
<tr>
<td></td>
<td>• psp: Pyramid Scene Parsing (PSP) algorithm</td>
</tr>
<tr>
<td></td>
<td>• deeplab: DeepLab V3 algorithm</td>
</tr>
<tr>
<td></td>
<td>Default value: fcn</td>
</tr>
</tbody>
</table>

Data Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_classes</td>
<td>The number of classes to segment.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: 2 \leq \text{positive integer} \leq 254</td>
</tr>
<tr>
<td>num_training_samples</td>
<td>The number of samples in the training data. The algorithm uses this value</td>
</tr>
<tr>
<td></td>
<td>to set up the learning rate scheduler.</td>
</tr>
<tr>
<td></td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>crop_size</td>
<td>The image size for input images. We rescale the input image to a square</td>
</tr>
<tr>
<td></td>
<td>image to this crop_size. We do this by rescaling the shorter side to match</td>
</tr>
<tr>
<td></td>
<td>this parameter while maintaining the aspect ratio, and then take a random</td>
</tr>
<tr>
<td></td>
<td>crop along the longer side.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer &gt; 16</td>
</tr>
<tr>
<td></td>
<td>Default value: 480</td>
</tr>
</tbody>
</table>

Training Hyperparameters
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>early_stopping</td>
<td>Whether to use early stopping logic during training.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: True, False</td>
</tr>
<tr>
<td></td>
<td>Default value: False</td>
</tr>
<tr>
<td>early_stopping_min_epochs</td>
<td>The minimum number of epochs that must be run.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>The number of epochs that meet the tolerance for lower performance before the algorithm enforces an early stop.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 4</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>If the relative improvement of the mIOU is smaller than this value, early stopping considers the epoch as not improved. This is used only when early_stopping = True.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of epochs with which to train.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 30</td>
</tr>
<tr>
<td>gamma1</td>
<td>The decay factor for the moving average of the squared gradient for rmsprop. Used only for rmsprop.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>gamma2</td>
<td>The momentum factor for rmsprop.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 ≤ float ≤ 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The initial learning rate.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 &lt; \text{float} \leq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>lr_scheduler</td>
<td>The shape of the learning rate schedule that controls its decrease over time.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values:</td>
</tr>
<tr>
<td></td>
<td>• step: A stepwise decay, where the learning rate is reduced by a factor at certain intervals.</td>
</tr>
<tr>
<td></td>
<td>• poly: A smooth decay using a polynomial function.</td>
</tr>
<tr>
<td></td>
<td>• cosine: A smooth decay using a cosine function.</td>
</tr>
<tr>
<td></td>
<td>Default value: poly</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. Using a large mini_batch_size usually results in faster training, but it might cause you to run out of memory.</td>
</tr>
<tr>
<td></td>
<td>Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 4</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for the sgd optimizer. When you use other optimizers, the semantic segmentation algorithm ignores this parameter.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: $0 &lt; \text{float} \leq 1$</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.9</td>
</tr>
</tbody>
</table>
## Parameter Name | Description
--- | ---
**optimizer** | The type of optimizer. For more information about an optimizer, choose the appropriate link:  
- adam: Adaptive momentum estimation  
- adagrad: Adaptive gradient descent  
- nag: Nesterov accelerated gradient  
- rmsprop: Root mean square propagation  
- sgd: Stochastic gradient descent  
**Optional**  
Valid values: adam, adagrad, nag, rmsprop, sgd  
Default value: sgd

**validation_mini_batch_size** | The batch size for validation. A large mini_batch_size usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture.  
- To score the validation on the entire image without cropping the images, set this parameter to 1. Use this option if you want to measure performance on the entire image as a whole.  
  **Note**  
  Setting the validation_mini_batch_size parameter to 1 causes the algorithm to create a new network model for every image. This might slow validation and training.  
- To crop images to the size specified in the crop_size parameter, even during evaluation, set this parameter to a value greater than 1.  
**Optional**  
Valid values: positive integer  
Default value: 4

**weight_decay** | The weight decay coefficient for the sgd optimizer. When you use other optimizers, the algorithm ignores this parameter.  
**Optional**  
Valid values: 0 < float < 1  
Default value: 0.0001

### Sequence-to-Sequence Algorithm

Amazon SageMaker Sequence to Sequence is a supervised learning algorithm where the input is a sequence of tokens (for example, text, audio) and the output generated is another sequence of tokens. Example applications include: machine translation (input a sentence from one language and predict what that sentence would be in another language), text summarization (input a longer string of words and predict a shorter string of words that is a summary), speech-to-text (audio clips converted into output
sentences in tokens). Recently, problems in this domain have been successfully modeled with deep neural networks that show a significant performance boost over previous methodologies. Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

**Topics**

- Input/Output Interface for the Sequence-to-Sequence Algorithm (p. 234)
- EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm (p. 235)
- Sequence-to-Sequence Sample Notebooks (p. 235)
- How Sequence-to-Sequence Works (p. 235)
- Sequence-to-Sequence Hyperparameters (p. 236)
- Tune a Sequence-to-Sequence Model (p. 244)

**Input/Output Interface for the Sequence-to-Sequence Algorithm**

**Training**

Amazon SageMaker seq2seq expects data in RecordIO-Protobuf format. However, the tokens are expected as integers, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in the seq2seq example notebook. In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

- **train**: It should contain the training data (for example, the train.rec file generated by the preprocessing script).
- **validation**: It should contain the validation data (for example, the val.rec file generated by the preprocessing script).
- **vocab**: It should contain two vocabulary files (vocab.src.json and vocab.trg.json)

If the algorithm doesn't find data in any of these three channels, training results in an error.

**Inference**

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the application/json format. Otherwise, use the recordio-protobuf format to work with the integer encoded data. Both mode supports batching of input data. application/json format also allows you to visualize the attention matrix.

- **application/json**: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be application/json. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:

  - `configuration`: (attribute_matrix: true): Returns the attention matrix for the particular input sequence.

- **application/x-recordio-protobuf**: Expects the input in recordio-protobuf format and returns the output in recordio-protobuf format. Both content and accept types should be applications/x-recordio-protobuf. For this format, the source sequences must be converted
into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For batch transform, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be application/jsonlines. The format for input is as follows:

```
content-type: application/jsonlines

{"source": "source_sequence_0"}
{"source": "source_sequence_1"}
```

The format for response is as follows:

```
accept: application/jsonlines

{"target": "predicted_sequence_0"}
{"target": "predicted_sequence_1"}
```

For additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference, see the Sequence-to-Sequence Sample Notebooks (p. 235).

**EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm**

Currently Amazon SageMaker seq2seq is only supported on GPU instance types and is only set up to train on a single machine. But it does also offer support for multiple GPUs.

**Sequence-to-Sequence Sample Notebooks**

For a sample notebook that shows how to use the Amazon SageMaker Sequence to Sequence algorithm to train a English-German translation model, see Machine Translation English-German Example Using SageMaker Seq2Seq. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the Introduction to Amazon algorithms section. To open a notebook, click on its Use tab and select Create copy.

**How Sequence-to-Sequence Works**

Typically, a neural network for sequence-to-sequence modeling consists of a few layers, including:

- **An embedding layer.** In this layer, the input matrix, which is input tokens encoded in a sparse way (for example, one-hot encoded) are mapped to a dense feature layer. This is required because a high-dimensional feature vector is more capable of encoding information regarding a particular token (word for text corpora) than a simple one-hot-encoded vector. It is also a standard practice to initialize this embedding layer with a pre-trained word vector like FastText or Glove or to initialize it randomly and learn the parameters during training.

- **An encoder layer.** After the input tokens are mapped into a high-dimensional feature space, the sequence is passed through an encoder layer to compress all the information from the input embedding layer (of the entire sequence) into a fixed-length feature vector. Typically, an encoder is made of RNN-type networks like long short-term memory (LSTM) or gated recurrent units (GRU). (Colah's blog explains LSTM in a great detail.)

- **A decoder layer.** The decoder layer takes this encoded feature vector and produces the output sequence of tokens. This layer is also usually built with RNN architectures (LSTM and GRU).
The whole model is trained jointly to maximize the probability of the target sequence given the source sequence. This model was first introduced by Sutskever et al. in 2014.

**Attention mechanism.** The disadvantage of an encoder-decoder framework is that model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain. To tackle this problem, in 2015, Bahdanau et al. proposed the attention mechanism. In an attention mechanism, the decoder tries to find the location in the encoder sequence where the most important information could be located and uses that information and previously decoded words to predict the next token in the sequence.

For more in details, see the whitepaper *Effective Approaches to Attention-based Neural Machine Translation* by Luong, et al. that explains and simplifies calculations for various attention mechanisms. Additionally, the whitepaper *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation* by Wu, et al. describes Google’s architecture for machine translation, which uses skip connections between encoder and decoder layers.

### Sequence-to-Sequence Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>batch_size</strong></td>
<td>Mini batch size for gradient descent.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Default value: 64</td>
<td></td>
</tr>
<tr>
<td><strong>beam_size</strong></td>
<td>Length of the beam for beam search. Used during training for computing bleu and used during inference.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td>Default value: 5</td>
<td></td>
</tr>
<tr>
<td><strong>bleu_sample_size</strong></td>
<td>Number of instances to pick from validation dataset to decode and compute bleu score during training. Set to -1 to use full validation set (if bleu is chosen as optimized_metric).</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
<tr>
<td><strong>bucket_width</strong></td>
<td>Returns (source,target) buckets up to (max_seq_len_source, max_seq_len_target). The longer side of the data uses steps of bucket_width while the shorter side uses steps scaled down by the average target/source length ratio. If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of max_len.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------</td>
</tr>
<tr>
<td><strong>Default value: 10</strong></td>
<td></td>
</tr>
</tbody>
</table>
| **bucketing_enabled** | Set to **false** to disable bucketing, unroll to maximum length. **Optional**  
Valid values: **true** or **false**  
Default value: **true** |
| **checkpoint_frequency_num_batches** | Checkpoint and evaluate every x batches. **Optional**  
Valid values: positive integer  
Default value: **1000** |
| **checkpoint_threshold** | Maximum number of checkpoints model is allowed to not improve in **optimized_metric** on validation dataset before training is stopped. **Optional**  
Valid values: positive integer  
Default value: **3** |
| **clip_gradient** | Clip absolute gradient values greater than this. Set to negative to disable. **Optional**  
Valid values: float  
Default value: **1** |
| **cnn_activation_type** | The **cnn** activation type to be used. **Optional**  
Valid values: String. One of **glu**, **relu**, **softrelu**, **sigmoid**, or **tanh**.  
Default value: **glu** |
| **cnn_hidden_dropout** | Dropout probability for dropout between convolutional layers. **Optional**  
Valid values: Float. Range in \([0,1]\).  
Default value: **0** |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cnn_kernel_width_decoder</code></td>
<td>Kernel width for the cnn decoder.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td><code>cnn_kernel_width_encoder</code></td>
<td>Kernel width for the cnn encoder.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td><code>cnn_num_hidden</code></td>
<td>Number of cnn hidden units for encoder and decoder.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><code>decoder_type</code></td>
<td>Decoder type.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td><code>embed_dropout_source</code></td>
<td>Dropout probability for source side embeddings.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>embed_dropout_target</code></td>
<td>Dropout probability for target side embeddings.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>encoder_type</code></td>
<td>Encoder type. The rnn architecture is based on attention mechanism by Bahdanau et al. and cnn architecture is based on Gehring et al.</td>
</tr>
<tr>
<td></td>
<td><em>Optional</em></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>fixed_rate_lr_half_life</strong></td>
<td>Half life for learning rate in terms of number of checkpoints for fixed_rate_* schedulers.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td><strong>learning_rate</strong></td>
<td>Initial learning rate.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0003</td>
</tr>
<tr>
<td><strong>loss_type</strong></td>
<td>Loss function for training.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. cross-entropy</td>
</tr>
<tr>
<td></td>
<td>Default value: cross-entropy</td>
</tr>
<tr>
<td><strong>lr_scheduler_type</strong></td>
<td>Learning rate scheduler type. plateau_reduce means reduce the learning rate whenever optimized_metric on validation_accuracy plateaus. inv_t is inverse time decay. learning_rate/(1+decay_rate*t)</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of plateau_reduce, fixed_rate_inv_t, or fixed_rate_inv_sqrt_t.</td>
</tr>
<tr>
<td></td>
<td>Default value: plateau_reduce</td>
</tr>
<tr>
<td><strong>max_num_batches</strong></td>
<td>Maximum number of updates/batches to process. -1 for infinite.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -1</td>
</tr>
<tr>
<td><strong>max_num_epochs</strong></td>
<td>Maximum number of epochs to pass through training data before fitting is stopped. Training continues until this number of epochs even if validation accuracy is not improving if this parameter is passed. Ignored if not passed.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer and less than or equal to max_num_epochs.</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>max_seq_len_source</td>
<td>Maximum length for the source sequence length. Sequences longer than this length are truncated to this length.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>max_seq_len_target</td>
<td>Maximum length for the target sequence length. Sequences longer than this length are truncated to this length.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>min_num_epochs</td>
<td>Minimum number of epochs the training must run before it is stopped via early_stopping conditions.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>momentum</td>
<td>Momentum constant used for sgd. Don't pass this parameter if you are using adam or rmsprop.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td>num_embed_source</td>
<td>Embedding size for source tokens.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td>num_embed_target</td>
<td>Embedding size for target tokens.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num_layers_decoder</td>
<td>Number of layers for Decoder <em>rnn</em> or <em>cnn</em>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>num_layers_encoder</td>
<td>Number of layers for Encoder <em>rnn</em> or <em>cnn</em>.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>optimized_metric</td>
<td>Metrics to optimize with early stopping.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <em>perplexity</em>, <em>accuracy</em>, or <em>bleu</em>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>perplexity</em></td>
</tr>
<tr>
<td>optimizer_type</td>
<td>Optimizer to choose from.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <em>adam</em>, <em>sgd</em>, or <em>rmsprop</em>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>adam</em></td>
</tr>
<tr>
<td>plateau_reduce_lr_factor</td>
<td>Factor to multiply learning rate with (for <em>plateau_reduce</em>).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>plateau_reduce_lr_threshold</td>
<td>For <em>plateau_reduce</em> scheduler, multiply learning rate with reduce factor if <em>optimized_metric</em> didn't improve for this many checkpoints.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rnn_attention_in_upper_layers</td>
<td>Pass the attention to upper layers of rnn, like Google NMT paper. Only applicable if more than one layer is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: boolean (true or false)</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>rnn_attention_num_hidden</td>
<td>Number of hidden units for attention layers. defaults to rnn_num_hidden.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn_num_hidden</td>
</tr>
<tr>
<td>rnn_attention_type</td>
<td>Attention model for encoders. mlp refers to concat and bilinear refers to general from the Luong et al. paper.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of dot, fixed, mlp, or bilinear.</td>
</tr>
<tr>
<td></td>
<td>Default value: mlp</td>
</tr>
<tr>
<td>rnn_cell_type</td>
<td>Specific type of rnn architecture.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either lstm or gru.</td>
</tr>
<tr>
<td></td>
<td>Default value: lstm</td>
</tr>
<tr>
<td>rnn_decoder_state_init</td>
<td>How to initialize rnn decoder states from encoders.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of last, avg, or zero.</td>
</tr>
<tr>
<td></td>
<td>Default value: last</td>
</tr>
<tr>
<td>rnn_first_residual_layer</td>
<td>First rnn layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>rnn_num_hidden</code></td>
<td>The number of <code>rnn</code> hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional Long Term Short Term Memory (LSTM) by default.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: positive even integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1024</td>
</tr>
<tr>
<td><code>rnn_residual_connections</code></td>
<td>Add residual connection to stacked <code>rnn</code>. Number of layers should be more than 1.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: boolean (true or false)</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
</tr>
<tr>
<td><code>rnn_decoder_hidden_dropout</code></td>
<td>Dropout probability for hidden state that combines the context with the <code>rnn</code> hidden state in the decoder.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>training_metric</code></td>
<td>Metrics to track on training on validation data.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either perplexity or accuracy.</td>
</tr>
<tr>
<td></td>
<td>Default value: perplexity</td>
</tr>
<tr>
<td><code>weight_decay</code></td>
<td>Weight decay constant.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td><code>weight_init_scale</code></td>
<td>Weight initialization scale (for uniform and xavier initialization).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 2.34</td>
</tr>
</tbody>
</table>
Tune a Sequence-to-Sequence Model

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see Automatic Model Tuning (p. 290).

Metrics Computed by the Sequence-to-Sequence Algorithm

The sequence to sequence algorithm reports three metrics that are computed during training. Choose one of them as an objective to optimize when tuning the hyperparameter values.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:accuracy</td>
<td>Accuracy computed on the validation dataset.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:bleu</td>
<td>Bleu score computed on the validation dataset. Because BLEU computation is</td>
<td>Maximize</td>
</tr>
<tr>
<td></td>
<td>expensive, you can choose to compute BLEU on a random subsample of the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>validation dataset to speed up the overall training process. Use the bleu</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sample_size parameter to specify the subsample.</td>
<td></td>
</tr>
<tr>
<td>validation:perplexity</td>
<td>Perplexity, is a loss function computed on the validation dataset.</td>
<td>Minimize</td>
</tr>
<tr>
<td></td>
<td>Perplexity measures the cross-entropy between an empirical sample and the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>distribution predicted by a model and so provides a measure of how well a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>model predicts the sample values, Models that are good at predicting a</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sample have a low perplexity.</td>
<td></td>
</tr>
</tbody>
</table>

Tunable Sequence-to-Sequence Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker Sequence to Sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight_init_type</td>
<td>Type of weight initialization.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either uniform or xavier.</td>
</tr>
<tr>
<td></td>
<td>Default value: xavier</td>
</tr>
<tr>
<td>xavier_factor_type</td>
<td>Xavier factor type.</td>
</tr>
<tr>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of in, out, or avg.</td>
</tr>
<tr>
<td></td>
<td>Default value: in</td>
</tr>
</tbody>
</table>
Sequence to Sequence (seq2seq) metrics are: `batch_size`, `optimizer_type`, `learning_rate`, `num_layers_encoder`, and `num_layers_decoder`.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_layers_encoder</td>
<td>IntegerParameterRange</td>
<td>[1-10]</td>
</tr>
<tr>
<td>num_layers_decoder</td>
<td>IntegerParameterRange</td>
<td>[1-10]</td>
</tr>
<tr>
<td>batch_size</td>
<td>CategoricalParameterRange</td>
<td>[16,32,64,128,256,512,1024,2048]</td>
</tr>
<tr>
<td>optimizer_type</td>
<td>CategoricalParameterRange</td>
<td>['adam', 'sgd', 'rmsprop']</td>
</tr>
<tr>
<td>weight_init_type</td>
<td>CategoricalParameterRange</td>
<td>['xavier', 'uniform']</td>
</tr>
<tr>
<td>weight_init_scale</td>
<td>ContinuousParameterRange</td>
<td>For the xavier type: MinValue: 2.0, MaxValue: 3.0 For the uniform type: MinValue: -1.0, MaxValue: 1.0</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.00005, MaxValue: 0.2</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.1</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.5, MaxValue: 0.9</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1.0, MaxValue: 5.0</td>
</tr>
<tr>
<td>rnn_num_hidden</td>
<td>CategoricalParameterRange</td>
<td>Applicable only to recurrent neural networks (RNNs). [128,256,512,1024,2048]</td>
</tr>
<tr>
<td>cnn_num_hidden</td>
<td>CategoricalParameterRange</td>
<td>Applicable only to convolutional neural networks (CNNs). [128,256,512,1024,2048]</td>
</tr>
<tr>
<td>num_embed_source</td>
<td>IntegerParameterRange</td>
<td>[256-512]</td>
</tr>
<tr>
<td>num_embed_target</td>
<td>IntegerParameterRange</td>
<td>[256-512]</td>
</tr>
<tr>
<td>embed_dropout_source</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>embed_dropout_target</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>rnn_decoder_hidden_dropout</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
<tr>
<td>cnn_hidden_dropout</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.0, MaxValue: 0.5</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr_scheduler_type</td>
<td>CategoricalParameterRange</td>
<td>['plateau_reduce', 'fixed_rate_inv_t', 'fixed_rate_inv_sqrt_t']</td>
</tr>
<tr>
<td>plateau_reduce_lr_factor</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 0.1, MaxValue: 0.5</td>
</tr>
<tr>
<td>plateau_reduce_lr_threshold</td>
<td>IntegerParameterRange</td>
<td>[1-5]</td>
</tr>
<tr>
<td>fixed_rate_lr_half_life</td>
<td>IntegerParameterRange</td>
<td>[10-30]</td>
</tr>
</tbody>
</table>

### XGBoost Algorithm

**XGBoost** (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

#### Topics
- Input/Output Interface for the XGBoost Algorithm (p. 246)
- EC2 Instance Recommendation for the XGBoost Algorithm (p. 247)
- XGBoost Sample Notebooks (p. 247)
- How XGBoost Works (p. 247)
- XGBoost Hyperparameters (p. 248)
- Tune an XGBoost Model (p. 254)

### Input/Output Interface for the XGBoost Algorithm

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

#### Note

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the index value pairs for features. So each row has the format: `<label> <index1>:<value1> <index2>:<value2> ...`. Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.
For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the InstanceType) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

**To use a model trained with SageMaker XGBoost in open source XGBoost**

- Use the following Python code:

```python
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

**To differentiate the importance of labelled data points use Instance Weight Supports**

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For text/libsvm input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...`. For text/csv input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

**EC2 Instance Recommendation for the XGBoost Algorithm**

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

**XGBoost Sample Notebooks**

For a sample notebook that shows how to use the Amazon SageMaker XGBoost algorithm to train and host a regression model, see [Regression with Amazon SageMaker XGBoost algorithm](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Notebook Instances](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the [Introduction to Amazon algorithms](#) section. To open a notebook, click on its **Use** tab and select **Create copy**.

**How XGBoost Works**

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leaves that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called
gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

**For more detail on XGBoost, see:**
- XGBoost: A Scalable Tree Boosting System
- Introduction to Boosted Trees

## XGBoost Hyperparameters

The Amazon SageMaker XGBoost algorithm is an implementation of the open-source XGBoost package. Currently Amazon SageMaker supports version 0.72. For more detail about hyperparameter configuration for this version of XGBoost, see Release 0.72 XGBoost Parameters.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_class</td>
<td>The number of classes.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong> if objective is set to <code>multi:softmax</code> or <code>multi:softprob</code>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>num_round</td>
<td>The number of rounds to run the training.</td>
</tr>
<tr>
<td></td>
<td><strong>Required</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>alpha</td>
<td>L1 regularization term on weights. Increasing this value makes models more conservative.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>base_score</td>
<td>The initial prediction score of all instances, global bias.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>booster</td>
<td>Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <code>gbtree</code>, <code>gblinear</code>, or <code>dart</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>gbtree</code></td>
</tr>
<tr>
<td>colsample_bylevel</td>
<td>Subsample ratio of columns for each split, in each level.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,1].</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>colsample_bytree</td>
<td>Subsample ratio of columns when constructing each tree.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>csv_weights</td>
<td>When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>early_stopping_rounds</td>
<td>The model trains until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training. Amazon SageMaker hosting uses the best model for inference.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>eta</td>
<td>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0,1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.3</td>
</tr>
<tr>
<td>eval_metric</td>
<td>Evaluation metrics for validation data. A default metric is assigned according to the objective:</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>For a list of valid inputs, see <a href="https://xgboost.readthedocs.io/en/latest/parameter.html">XGBoost Parameters</a>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: Default according to objective.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| gamma          | Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.  
Optional  
Valid values: Float. Range: [0,∞).  
Default value: 0 |
| grow_policy    | Controls the way that new nodes are added to the tree. Currently supported only if tree_method is set to hist.  
Optional  
Valid values: String. Either depthwise or lossguide.  
Default value: depthwise |
| lambda         | L2 regularization term on weights. Increasing this value makes models more conservative.  
Optional  
Valid values: float  
Default value: 1 |
| lambda_bias    | L2 regularization term on bias.  
Optional  
Valid values: Float. Range: [0.0, 1.0].  
Default value: 0 |
| max_bin        | Maximum number of discrete bins to bucket continuous features. Used only if tree_method is set to hist.  
Optional  
Valid values: integer  
Default value: 256 |
| max_delta_step | Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.  
Optional  
Valid values: Integer. Range: [0,∞).  
Default value: 0 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_depth</td>
<td>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfitted. 0 indicates no limit. A limit is required when grow_policy=depth-wise.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Integer. Range: [0,∞)</td>
</tr>
<tr>
<td>Default value: 6</td>
<td></td>
</tr>
<tr>
<td>max_leaves</td>
<td>Maximum number of nodes to be added. Relevant only if grow_policy is set to lossguide.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>Default value: 0</td>
<td></td>
</tr>
<tr>
<td>min_child_weight</td>
<td>Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Float. Range: [0,∞).</td>
</tr>
<tr>
<td>Default value: 1</td>
<td></td>
</tr>
<tr>
<td>normalize_type</td>
<td>Type of normalization algorithm.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: Either tree or forest.</td>
</tr>
<tr>
<td>Default value: tree</td>
<td></td>
</tr>
<tr>
<td>nthread</td>
<td>Number of parallel threads used to run xgboost.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: integer</td>
</tr>
<tr>
<td>Default value:</td>
<td>Maximum number of threads.</td>
</tr>
<tr>
<td>objective</td>
<td>Specifies the learning task and the corresponding learning objective. Examples: reg:linear, reg:logistic, multi:softmax. For a full list of valid inputs, please refer to XGBoost Parameters.</td>
</tr>
<tr>
<td>Optional</td>
<td>Valid values: string</td>
</tr>
<tr>
<td>Default value: reg:linear</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>one_drop</td>
<td>When this flag is enabled, at least one tree is always dropped during the dropout.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>process_type</td>
<td>The type of boosting process to run.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either default or update.</td>
</tr>
<tr>
<td></td>
<td>Default value: default</td>
</tr>
<tr>
<td>rate_drop</td>
<td>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0</td>
</tr>
<tr>
<td>refresh_leaf</td>
<td>This is a parameter of the 'refresh' updater plugin. When set to true (1), tree leaves and tree node stats are updated. When set to false(0), only tree node stats are updated.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: 0/1</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>sample_type</td>
<td>Type of sampling algorithm.</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: Either uniform or weighted.</td>
</tr>
<tr>
<td></td>
<td>Default value: uniform</td>
</tr>
<tr>
<td>scale_pos_weight</td>
<td>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: sum(negative cases) / sum(positive cases).</td>
</tr>
<tr>
<td></td>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| seed           | Random number seed.  
                   | **Optional**  
                   | Valid values: integer  
                   | Default value: 0 |
| silent         | 0 means print running messages, 1 means silent mode.  
                   | **Optional**  
                   | Valid values: 0 or 1  
                   | Default value: 0 |
| sketch_eps     | Used only for approximate greedy algorithm. This translates into $O(1 / \text{sketch\_eps})$ number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.  
                   | **Optional**  
                   | Valid values: Float, Range: [0, 1].  
                   | Default value: 0.03 |
| skip_drop      | Probability of skipping the dropout procedure during a boosting iteration.  
                   | **Optional**  
                   | Valid values: Float. Range: [0.0, 1.0].  
                   | Default value: 0.0 |
| subsample      | Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.  
                   | **Optional**  
                   | Valid values: Float. Range: [0,1].  
                   | Default value: 1 |
| tree_method    | The tree construction algorithm used in XGBoost.  
                   | **Optional**  
                   | Valid values: One of auto, exact, approx, or hist.  
                   | Default value: auto |
### Parameter Name | Description
--- | ---
tweedie_variance_power | Parameter that controls the variance of the Tweedie distribution.  
Optional  
Valid values: Float. Range: (1, 2).  
Default value: 1.5
updater | A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees.  
For a full list of valid inputs, please refer to XGBoost Parameters.  
Optional  
Valid values: comma-separated string.  
Default value: grow_colmaker, prune

---

**Tune an XGBoost Model**

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see *Automatic Model Tuning (p. 290).*

**Metrics Computed by the XGBoost Algorithm**

The XGBoost algorithm computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Description</th>
<th>Optimization Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation:auc</td>
<td>Area under the curve.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:error</td>
<td>Binary classification error rate, calculated as #(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:logloss</td>
<td>Negative log-likelihood.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mae</td>
<td>Mean absolute error.</td>
<td></td>
</tr>
<tr>
<td>validation:map</td>
<td>Mean average precision.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:merror</td>
<td>Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
Use Your Own Algorithms with Amazon SageMaker

You can easily package your own algorithms for use with Amazon SageMaker, regardless of programming language or framework. Amazon SageMaker is highly flexible. It allows you to do the following:

- Use a suitable algorithm provided by Amazon SageMaker for model training and your own inference code, such as code for embedded applications, or devices, or both.
- Use your own training algorithm and inference code provided by Amazon SageMaker.
• Use your own training algorithm and your own inference code. You package the algorithm and inference code in Docker images, and use the images to train a model and deploy it with Amazon SageMaker.

• Use deep learning containers provided by Amazon SageMaker for model training and your own inference code. You provide a script written for the deep learning framework, such as Apache MXNet or TensorFlow. For more information about training, see Use Apache MXNet with Amazon SageMaker (p. 390) and Use TensorFlow with Amazon SageMaker (p. 390).

Amazon SageMaker algorithms are packaged as Docker images. This gives you the flexibility to use almost any algorithm code with Amazon SageMaker, regardless of implementation language, dependent libraries, frameworks, and so on. For more information on creating Docker images, see The Dockerfile instructions.

You can provide separate Docker images for the training algorithm and inference code, or you can combine them into a single Docker image. When creating Docker images for use with Amazon SageMaker, consider the following:

• Providing two Docker images can increase storage requirements and cost because common libraries might be duplicated.

• In general, smaller containers start faster for both training and hosting. Models train faster and the hosting service can react to increases in traffic by automatically scaling more quickly.

• You might be able to write an inference container that is significantly smaller than the training container. This is especially common when you use GPUs for training, but your inference code is optimized for CPUs.

• Amazon SageMaker requires that Docker containers run without privileged access.

• Docker containers might send messages to the Stdout and Stderr files. Amazon SageMaker sends these messages to Amazon CloudWatch logs in your AWS account.

The following sections provide detailed information about how Amazon SageMaker interacts with Docker containers and explain Amazon SageMaker requirements for Docker images. Use this information when creating your own containers. For general information about Docker containers, see Docker Basics in the Amazon Elastic Container Service Developer Guide.

Topics
• Using Your Own Training Algorithms (p. 256)
• Using Your Own Inference Code (p. 260)
• Example: Using Your Own Algorithms (p. 265)
• Create Algorithm and Model Package Resources (p. 265)
• Use Algorithm and Model Package Resources (p. 271)

Using Your Own Training Algorithms

This section explains how Amazon SageMaker interacts with a Docker container that runs your custom training algorithm. Use this information to write training code and create a Docker image for your training algorithms.

Topics
• How Amazon SageMaker Runs Your Training Image (p. 257)
• How Amazon SageMaker Provides Training Information (p. 258)
• How Amazon SageMaker Signals Algorithm Success and Failure (p. 260)
• How Amazon SageMaker Processes Training Output (p. 260)
How Amazon SageMaker Runs Your Training Image

To configure a Docker container to run as an executable, use an ENTRYPOINT instruction in a Dockerfile. Note the following:

• For model training, Amazon SageMaker runs the container as follows:

```
docker run image train
```

Amazon SageMaker overrides any default CMD statement in a container by specifying the train argument after the image name. The train argument also overrides arguments that you provide using CMD in the Dockerfile.

• Use the exec form of the ENTRYPOINT instruction:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

For example:

```
ENTRYPOINT ["python", "k-means-algorithm.py"]
```

The exec form of the ENTRYPOINT instruction starts the executable directly, not as a child of /bin/sh. This enables it to receive signals like SIGTERM and SIGKILL from Amazon SageMaker APIs. Note the following:

• The CreateTrainingJob (p. 556) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.

• The StopTrainingJob (p. 716) API issues the equivalent of the docker stop, with a 2 minute timeout, command to gracefully stop the specified container:

```
docker stop -t120
```

The command attempts to stop the running container by sending a SIGTERM signal. After the 2 minute timeout, SIGKILL is sent and the containers are forcibly stopped. If the container handles the SIGTERM gracefully and exits within 120 seconds from receiving it, no SIGKILL is sent.

Note
If you want access to the intermediate model artifacts after Amazon SageMaker stops the training, add code to handle saving artifacts in your SIGTERM handler.

• If you plan to use GPU devices for model training, make sure that your containers are nvidia-docker compatible. Only the CUDA toolkit should be included on containers; don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see NVIDIA/nvidia-docker.

• You can't use the tini initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

• /opt/ml and all sub-directories are reserved by Amazon SageMaker training. When building your algorithm's docker image, please ensure you don't place any data required by your algorithm under them as the data may no longer be visible during training.
How Amazon SageMaker Provides Training Information

This section explains how Amazon SageMaker makes training information, such as training data, hyperparameters, and other configuration information, available to your Docker container.

When you send a CreateTrainingJob (p. 556) request to Amazon SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contain the training algorithm. You also specify the Amazon Simple Storage Service (Amazon S3) location where training data is stored and algorithm-specific parameters. Amazon SageMaker makes this information available to the Docker container so that your training algorithm can use it. This section explains how we make this information available to your Docker container. For information about creating a training job, see CreateTrainingJob.

Topics
- Hyperparameters (p. 258)
- Environment Variables (p. 258)
- Input Data Configuration (p. 258)
- Training Data (p. 259)
- Distributed Training Configuration (p. 259)

Hyperparameters

Amazon SageMaker makes the hyperparameters in a CreateTrainingJob request available in the Docker container in the /opt/ml/input/config/hyperparameters.json file.

Environment Variables

- TRAINING_JOB_NAME—The training job name stored in the TrainingJobName parameter in a CreateTrainingJob (p. 556) request.

Input Data Configuration

You specify data channel information in the InputDataConfig parameter in a CreateTrainingJob request. Amazon SageMaker makes this information available in the /opt/ml/input/config/inputdataconfig.json file in the Docker container.

For example, suppose that you specify three data channels (train, evaluation, and validation) in your request. Amazon SageMaker provides the following JSON:

```json
{
    "train" : {
        "ContentType": "trainingContentType",
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"},
    "evaluation" : {
        "ContentType": "evalContentType",
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"},
    "validation" : {
        "TrainingInputMode": "File",
        "S3DistributionType": "FullyReplicated",
        "RecordWrapperType": "None"}
}
```

Note

Amazon SageMaker provides only relevant information about each data channel (for example, the channel name and the content type) to the container, as shown.
Training Data

The TrainingInputMode parameter in a CreateTrainingJob request specifies how to make data available for model training: in FILE mode or PIPE mode. Depending on the specified input mode, Amazon SageMaker does the following:

- **FILE mode**—Amazon SageMaker makes the data for the channel available in the /opt/ml/input/data/channel_name directory in the Docker container. For example, if you have three channels named training, validation, and testing, Amazon SageMaker makes three directories in the Docker container:
  
  • /opt/ml/input/data/training
  • /opt/ml/input/data/validation
  • /opt/ml/input/data/testing

- **PIPE mode**—Amazon SageMaker makes data for the channel available from the named pipe: /opt/ml/input/data/channel_name_epoch_number. For example, if you have three channels named training, validation, and testing, you will need to read from the following pipes:
  
  • /opt/ml/input/data/training_0, /opt/ml/input/data/training_1, ...
  • /opt/ml/input/data/validation_0, /opt/ml/input/data/validation_1, ...
  • /opt/ml/input/data/testing_0, /opt/ml/input/data/testing_1, ...

Read the pipes sequentially. For example, if you have a channel called training, read the pipes in this sequence:

1. Open /opt/ml/input/data/training_0 in read mode and read it to EOF (or if you are done with the first epoch, close the file early).
2. After closing the first pipe file, look for /opt/ml/input/data/training_1 and read it to go through the second epoch, and so on.

If the file for a given epoch doesn't exist yet, your code may need to retry until the pipe is created. There is no sequencing restriction across channel types. That is, you can read multiple epochs for the training channel, for example, and only start reading the validation channel when you are ready. Or, you can read them simultaneously if your algorithm requires that.

Distributed Training Configuration

If you're performing distributed training with multiple containers, Amazon SageMaker makes information about all containers available in the /opt/ml/input/config/resourceconfig.json file.

To enable inter-container communication, this JSON file contains information for all containers. Amazon SageMaker makes this file available for both FILE and PIPE mode algorithms. The file provides the following information:

- **current_host**—The name of the current container on the container network. For example, algo-1. Host values can change at any time. Don't write code with specific values for this variable.

- **hosts**—The list of names of all containers on the container network, sorted lexicographically. For example, ["algo-1", "algo-2", "algo-3"] for a three-node cluster. Containers can use these names to address other containers on the container network. Host values can change at any time. Don't write code with specific values for these variables.

- **Do not use the information in /etc/hostname or /etc/hosts because it might be inaccurate.**

- **Hostname information may not be immediately available to the algorithm container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.**
The following is an example file on node 1 in a three-node cluster:

```json
{
  "current_host": "algo-1",
  "hosts": ["algo-1","algo-2","algo-3"]
}
```

### How Amazon SageMaker Signals Algorithm Success and Failure

A training algorithm indicates whether it succeeded or failed using the exit code of its process.

A successful training execution should exit with an exit code of 0 and an unsuccessful training execution should exit with a non-zero exit code. These will be converted to "Completed" and "Failed" in the `TrainingJobStatus` returned by `DescribeTrainingJob`. This exit code convention is standard and is easily implemented in all languages. For example, in Python, you can use `sys.exit(1)` to signal a failure exit and simply running to the end of the main routine will cause Python to exit with code 0.

In the case of failure, the algorithm can write a description of the failure to the failure file. See next section for details.

### How Amazon SageMaker Processes Training Output

As your algorithm runs in a container, it generates output including the status of the training job and model and output artifacts. Your algorithm should write this information to the following files, which are located in the container’s `/output` directory. Amazon SageMaker processes the information contained in this directory as follows:

- `/opt/ml/output/failure`—If training fails, after all algorithm output (for example, logging) completes, your algorithm should write the failure description to this file. In a `DescribeTrainingJob` response, Amazon SageMaker returns the first 1024 characters from this file as `FailureReason`.

- `/opt/ml/model`—Your algorithm should write all final model artifacts to this directory. Amazon SageMaker copies this data as a single object in compressed tar format to the S3 location that you specified in the `CreateTrainingJob` request. If multiple containers in a single training job write to this directory they should ensure no file/directory names clash. Amazon SageMaker aggregates the result in a tar file and uploads to S3.

### Next Step

**Using Your Own Inference Code (p. 260)**

### Using Your Own Inference Code

You can use Amazon SageMaker to interact with Docker containers and run your own inference code in one of two ways:

- To use your own inference code with a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To use your own inference code to get predictions for an entire dataset, use Amazon SageMaker batch transform.

### Topics

- Using Your Own Inference Code with Amazon SageMaker Hosting Services (p. 261)
Using Your Own Inference Code with Amazon SageMaker Hosting Services

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for hosting services. Use this information to write inference code and create a Docker image.

Topics

- How Amazon SageMaker Runs Your Inference Image (p. 261)
- How Amazon SageMaker Loads Your Model Artifacts (p. 262)
- How Containers Serve Requests (p. 262)
- How Your Container Should Respond to Inference Requests (p. 262)
- How Your Container Should Respond to Health Check (Ping) Requests (p. 263)

How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model inference, Amazon SageMaker runs the container as:

```
docker run image serve
```

Amazon SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the image name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

For example:

```
ENTRYPOINT ["python", "k_means_inference.py"]
```

The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from the Amazon SageMaker APIs, which is a requirement.

For example, when you use the `CreateEndpoint` (p. 522) API to create an endpoint, Amazon SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. Amazon SageMaker runs the Docker container on those instances.

If you reduce the number of instances backing the endpoint (by calling the `UpdateEndpointWeightsAndCapacities` (p. 724) APIs), Amazon SageMaker runs a command to stop
the Docker container on the instances being terminated. The command sends the \texttt{SIGTERM} signal, then it sends the \texttt{SIGKILL} signal thirty seconds later.

If you update the endpoint (by calling the \texttt{UpdateEndpoint (p. 722)} API), Amazon SageMaker launches another set of ML compute instances and runs the Docker containers that contain your inference code on them. Then it runs a command to stop the previous Docker container. To stop the Docker container, command sends the \texttt{SIGTERM} signal, then it sends the \texttt{SIGKILL} signal thirty seconds later.

- Amazon SageMaker uses the container definition that you provided in your \texttt{CreateModel (p. 538)} request to set environment variables and the DNS hostname for the container as follows:

  - It sets environment variables using the \texttt{ContainerDefinition.Environment} string-to-string map.
  - It sets the DNS hostname using the \texttt{ContainerDefinition.ContainerHostname}.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your \texttt{CreateEndpointConfig} request), make sure that your containers are \texttt{nvidia-docker} compatible. Don't bundle NVIDIA drivers with the image. For more information about \texttt{nvidia-docker}, see \texttt{NVIDIA/nvidia-docker}.

- You can't use the \texttt{tini} initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

### How Amazon SageMaker Loads Your Model Artifacts

In your \texttt{CreateModel (p. 538)} request, the container definition includes the \texttt{ModelDataUrl} parameter, which identifies the S3 location where model artifacts are stored. Amazon SageMaker uses this information to determine where to copy the model artifacts from. It copies the artifacts to the /\texttt{opt/ml/model} directory for use by your inference code.

The \texttt{ModelDataUrl} must point to a tar.gz file. Otherwise, Amazon SageMaker won't download the file.

If you trained your model in Amazon SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside Amazon SageMaker, you need to create this single compressed tar file and save it in a S3 location. Amazon SageMaker decompresses this tar file into /\texttt{opt/ml/model} directory before your container starts.

### How Containers Serve Requests

Containers need to implement a web server that responds to /\texttt{invocations} and /\texttt{ping} on port 8080.

### How Your Container Should Respond to Inference Requests

To obtain inferences, the client application sends a POST request to the Amazon SageMaker endpoint. For more information, see the \texttt{InvokeEndpoint (p. 735)} API. Amazon SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

- Amazon SageMaker strips all POST headers except those supported by \texttt{InvokeEndpoint}. Amazon SageMaker might add additional headers. Inference containers must be able to safely ignore these additional headers.
• To receive inference requests, the container must have a web server listening on port 8080 and must accept POST requests to the /invocations endpoint.
• A customer’s model containers must accept socket connection requests within 250 ms.
• A customer’s model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

How Your Container Should Respond to Health Check (Ping) Requests

The CreateEndpoint and UpdateEndpoint API calls result in Amazon SageMaker starting new inference containers. Soon after container startup, Amazon SageMaker starts sending periodic GET requests to the /ping endpoint.

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

If the container does not begin to consistently respond with 200s during the first 30 seconds after startup, the CreateEndPoint and UpdateEndpoint APIs will fail.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

Using Your Own Inference Code with Batch Transform

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for batch transform. Use this information to write inference code and create a Docker image.

Topics
• How Amazon SageMaker Runs Your Inference Image (p. 263)
• How Amazon SageMaker Loads Your Model Artifacts (p. 264)
• How Containers Serve Requests (p. 264)
• How Your Container Should Respond to Health Check (Ping) Requests (p. 265)

How Amazon SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an ENTRYPOINT instruction in a Dockerfile. Note the following:

• For batch transform, Amazon SageMaker runs the container as:

```bash
docker run image serve
```

Amazon SageMaker overrides default CMD statements in a container by specifying the serve argument after the image name. The serve argument overrides arguments that you provide with the CMD command in the Dockerfile.

• We recommend that you use the exec form of the ENTRYPOINT instruction:

```bash
ENTRYPOINT ["executable", "param1", "param2"]
```
For example:

```sh
ENTRYPOINT ["python", "k_means_inference.py"]
```

- Amazon SageMaker sets environment variables specified in `CreateModel (p. 538)` and `CreateTransformJob (p. 562)` on your container. Additionally, the following environment variables will be populated:
  - `SAGEMAKER_BATCH` is always set to `true` when the container runs in Batch Transform.
  - `SAGEMAKER_MAX_PAYLOAD_IN_MB` is set to the largest size payload that will be sent to the container via HTTP.
  - `SAGEMAKER_BATCH_STRATEGY` will be set to `SINGLE_RECORD` when the container will be sent a single record per call to invocations and `MULTI_RECORD` when the container will get as many records as will fit in the payload.
  - `SAGEMAKER_MAX_CONCURRENT_TRANSFORMS` is set to the maximum number of /invocations requests that can be opened simultaneously.

  **Note**
  The last three environment variables come from the API call made by the user. If the user doesn't set values for them, they aren't passed. In that case, either the default values or the values requested by the algorithm (in response to the `/execution-parameters`) are used.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateTransformJob` request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see NVIDIA/nvidia-docker.

- You can't use the tini initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.

### How Amazon SageMaker Loads Your Model Artifacts

In your `CreateModel (p. 538)` request, the container definition includes the `ModelDataUrl` parameter, which identifies the S3 location where model artifacts are stored. Amazon SageMaker uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory for use by your inference code.

The `ModelDataUrl` must point to a tar.gz file. Otherwise, it Amazon SageMaker doesn't download the file.

If you trained your model in Amazon SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside Amazon SageMaker, you need to create this single compressed tar file and save it in a S3 location. Amazon SageMaker decompresses this tar file into `/opt/ml/model` directory before your container starts.

### How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

Containers can optionally provide dynamic information to Amazon SageMaker for better throughput. If you want to provide this information dynamically, implement `/execution-parameters` on port 8080 in addition to `/invocations` and `/ping`. The response for GET `/execution-parameters` should be a simple JSON object with keys for MaxConcurrentTransforms, BatchStrategy, and MaxPayloadInMB. Here is an example valid response:
Example: Using Your Own Algorithms

Amazon SageMaker will attempt invoking /execution-parameters before calling /invocations. If the container does not implement /execution-parameters, Amazon SageMaker applies these defaults:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Possible Values</th>
<th>Default Values</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxConcurrentTransforms</td>
<td>1-100</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>BatchStrategy</td>
<td>SINGLE_RECORD, MULTI_RECORD</td>
<td>MULTI_RECORD</td>
<td></td>
</tr>
<tr>
<td>MaxPayloadInMB</td>
<td>≧0</td>
<td>6</td>
<td>0 signifies that the payload can be arbitrarily large and is transmitted in HTTP chunked encoding.</td>
</tr>
</tbody>
</table>

How Your Container Should Respond to Health Check (Ping) Requests

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

Example: Using Your Own Algorithms

An example of packaging scikit-learn for use with Amazon SageMaker is available at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb

Create Algorithm and Model Package Resources

After your training and/or inference code is packaged in Docker containers, create algorithm and model package resources that you can use in your Amazon SageMaker account and, optionally, publish on AWS Marketplace.

Topics
- Create an Algorithm Resource (p. 265)
- Create a Model Package Resource (p. 269)

Create an Algorithm Resource

To create an algorithm resource that you can use to run training jobs in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker containers that contains the training and, optionally, inference code.
- The configuration of the input data that your algorithm expects for training.
- The hyperparameters that your algorithm supports.
• Metrics that your algorithm sends to Amazon CloudWatch during training jobs.
• The instance types that your algorithm supports for training and inference, and whether it supports distributed training across multiple instances.
• Validation profiles, which are training jobs that Amazon SageMaker uses to test your algorithm's training code and batch transform jobs that Amazon SageMaker runs to test your algorithm's inference code.

To ensure that buyers and sellers can be confident that products work in Amazon SageMaker, we require that you validate your algorithms before listing them on AWS Marketplace. You can list products in the AWS Marketplace only if validation succeeds. To validate your algorithms, Amazon SageMaker uses your validation profile and sample data to run the following validations tasks:
1. Create a training job in your account to verify that your training image works with Amazon SageMaker.
2. If you included inference code in your algorithm, create a model in your account using the algorithm's inference image and the model artifacts produced by the training job.
3. If you included inference code in your algorithm, create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

When you list your product on AWS Marketplace, the inputs and outputs of this validation process persist as part of your product and are made available to your buyers. This helps buyers understand and evaluate the product before they buy it. For example, buyers can inspect the input data that you used, the outputs generated, and the logs and metrics emitted by your code. The more comprehensive your validation specification, the easier it is for customers to evaluate your product.

Note
In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the Training jobs and Transform jobs pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. If any issues are found, you will have to create the algorithm again.

Note
To publish your algorithm on AWS Marketplace, at least one validation profile is required.

You can create an algorithm by using either the Amazon SageMaker console or the Amazon SageMaker API.

Topics
• Create an Algorithm Resource (Console) (p. 266)
• Create an Algorithm Resource (API) (p. 269)

Create an Algorithm Resource (Console)

To create an algorithm resource (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms, then choose Create algorithm.
3. On the Training specifications page, provide the following information:
   a. For Algorithm name, type a name for your algorithm. The algorithm name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   b. Type a description for your algorithm. This description appears in the Amazon SageMaker console and in the AWS Marketplace.
c. For **Training image**, type the path in Amazon ECR where your training container is stored.

d. For **Support distributed training**, Choose **Yes** if your algorithm supports training on multiple instances. Otherwise, choose **No**.

e. For **Support instance types for training**, choose the instance types that your algorithm supports.

f. For **Channel specification**, specify up to 8 channels of input data for your algorithm. For example, you might specify 3 input channels named `train`, `validation`, and `test`. For each channel, specify the following information:

   i. For **Channel name**, type a name for the channel. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   
   ii. To require the channel for your algorithm, choose **Channel required**.
   
   iii. Type a description for the channel.
   
   iv. For **Supported input modes**, choose **Pipe mode** if your algorithm supports streaming the input data, and **File mode** if your algorithm supports downloading the input data as a file. You can choose both.
   
   v. For **Supported content types**, type the MIME type that your algorithm expects for input data.
   
   vi. For **Supported compression type**, choose **Gzip** if your algorithm supports Gzip compression. Otherwise, choose **None**.
   
   vii. Choose **Add channel** to add another data input channel, or choose **Next** if you are done adding channels.

4. On the **Tuning specifications** page, provide the following information:

   a. For **Hyperparameter specification**, specify the hyperparameters that your algorithm supports by editing the JSON object. For each hyperparameter that your algorithm supports, construct a JSON block similar to the following:

```json
{
  "DefaultValue": "5",
  "Description": "The first hyperparameter",
  "IsRequired": true,
  "IsTunable": false,
  "Name": "intRange",
  "Range": {
   "IntegerParameterRangeSpecification": {
     "MaxValue": "10",
     "MinValue": "1"
   }
  },
  "Type": "Integer"
}
```

In the JSON, supply the following:

   i. For **DefaultValue**, specify a default value for the hyperparameter, if there is one.
   
   ii. For **Description**, specify a description for the hyperparameter.
   
   iii. For **IsRequired**, specify whether the hyperparameter is required.
   
   iv. For **IsTunable**, specify **true** if this hyperparameter can be tuned when a user runs a hyperparameter tuning job that uses this algorithm. For information, see Automatic Model Tuning (p. 290).
   
   v. For **Name**, specify a name for the hyperparameter.
   
   vi. For **Range**, specify one of the following:

   - **IntegerParameterRangeSpecification** - the values of the hyperparameter are integers. Specify minimum and maximum values for the hyperparameter.
• ContinuousParameterRangeSpecification - the values of the hyperparameter are floating-point values. Specify minimum and maximum values for the hyperparameter.
• CategoricalParameterRangeSpecification - the values of the hyperparameter are categorical values. Specify a list of all of the possible values.

vii. For Type, specify Integer, Continuous, or Categorical. The value must correspond to the type of Range that you specified.

b. For Metric definitions, specify any training metrics that you want your algorithm to emit. Amazon SageMaker uses the regular expression that you specify to find the metrics by parsing the logs from your training container during training. Users can view these metrics when they run training jobs with your algorithm, and they can monitor and plot the metrics in Amazon CloudWatch. For information, see Monitor and Analyze Training Jobs Using Metrics (p. 279). For each metric, provide the following information:

i. For Metric name, type a name for the metric.
ii. For Regex, type the regular expression that Amazon SageMaker uses to parse training logs so that it can find the metric value.
iii. For Objective metric support choose Yes if this metric can be used as the objective metric for a hyperparameter tuning job. For information, see Automatic Model Tuning (p. 290).

iv. Choose Add metric to add another metric, or choose Next if you are done adding metrics.

5. On the Inference specifications page, provide the following information if your algorithm supports inference:

a. For Container definition, type path in Amazon ECR where your inference container is stored.

b. For Container DNS host name, type the name of a DNS host for your image.

c. For Supported instance types for real-time inference, choose the instance types that your algorithm supports for models deployed as hosted endpoints in Amazon SageMaker. For information, see Deploy a Model on Amazon SageMaker Hosting Services (p. 7).

d. For Supported instance types for batch transform jobs, choose the instance types that your algorithm supports for batch transform jobs. For information, see Get Inferences for an Entire Dataset with Batch Transform (p. 10).

e. For Supported content types, type the type of input data that your algorithm expects for inference requests.

f. For Supported response MIME types, type the MIME types that your algorithm supports for inference responses.

g. Choose Next.

6. On the Validation specifications page, provide the following information:

a. For Publish this algorithm on AWS Marketplace, choose Yes to publish the algorithm on AWS Marketplace.

b. For Validate this algorithm, choose Yes if you want Amazon SageMaker to run training jobs and/or batch transform jobs that you specify to test the training and/or inference code of your algorithm.

Note
To publish your algorithm on AWS Marketplace, your algorithm must be validated.

c. For IAM role, choose an IAM role that has the required permissions to run training jobs and batch transform jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 415).

d. For Validation profile, specify the following:

• A name for the validation profile.
• A **Training job definition.** This is a JSON block that describes a training job. This is in the same format as the TrainingJobDefinition (p. 889) input parameter of the CreateAlgorithm (p. 512) API.

• A **Transform job definition.** This is a JSON block that describes a batch transform job. This is in the same format as the TransformJobDefinition (p. 900) input parameter of the CreateAlgorithm (p. 512) API.

e. Choose **Create algorithm.**

### Create an Algorithm Resource (API)

To create an algorithm resource by using the Amazon SageMaker API, call the [CreateAlgorithm](p. 512) API.

### Create a Model Package Resource

To create a model package resource that you can use to create deployable models in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker container that contains the inference code, or the algorithm resource that was used to train the model.
- The location of the model artifacts. Model artifacts can either be packaged in the same Docker container as the inference code or stored in Amazon S3.
- The instance types that your model package supports for both real-time inference and batch transform jobs.
- Validation profiles, which are batch transform jobs that Amazon SageMaker runs to test your model package's inference code.

Before listing model packages on AWS Marketplace, you must validate them. This ensures that buyers and sellers can be confident that products work in Amazon SageMaker. You can list products on AWS Marketplace only if validation succeeds.

The validation procedure uses your validation profile and sample data to run the following validations tasks:
1. Create a model in your account using the model package's inference image and the optional model artifacts that are stored in Amazon S3.

    **Note**
    A model package is specific to the region in which you create it. The S3 bucket where the model artifacts are stored must be in the same region where your created the model package.

2. Create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.
3. Create a validation profile.

    **Note**
    In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the [Transform jobs](p. 311) pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. After fixing issues, recreate the algorithm. When the status of the algorithm is **COMPLETED**, find it in the Amazon SageMaker console and start the listing process.

    **Note**
    To publish your model package on AWS Marketplace, at least one validation profile is required.
You can create an model package either by using the Amazon SageMaker console or by using the Amazon SageMaker API.

Topics
- Create a Model Package Resource (Console) (p. 270)
- Create a Model Package Resource (API) (p. 271)

Create a Model Package Resource (Console)

To create a model package in the Amazon SageMaker console:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Model packages, then choose Create model package.
3. On the Inference specifications page, provide the following information:
   a. For Model package name, type a name for your model package. The model package name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
   b. Type a description for your model package. This description appears in the Amazon SageMaker console and in the AWS Marketplace.
   c. For Inference specification options, choose Provide the location of the inference image and model artifacts to create a model package by using an inference container and model artifacts. Choose Provide the algorithm used for training and its model artifacts to create a model package from an algorithm resource that you created or subscribe to from AWS Marketplace.
   d. If you chose Provide the location of the inference image and model artifacts for Inference specification options, provide the following information for Container definition and Supported resources:
      i. For Location of inference image, type the path to the image that contains your inference code. The image must be stored as a Docker container in Amazon ECR.
      ii. For Location of model data artifacts, type the location in S3 where your model artifacts are stored.
      iii. For Container DNS host name, type the name of the DNS host to use for your container.
      iv. For Supported instance types for real-time inference, choose the instance types that your model package supports for real-time inference from Amazon SageMaker hosted endpoints.
      v. For Supported instance types for batch transform jobs, choose the instance types that your model package supports for batch transform jobs.
      vi. Supported content types, type the content types that your model package expects for inference requests.
      vii. For Supported response MIME types, type the MIME types that your model package uses to provide inferences.
   e. If you chose Provide the algorithm used for training and its model artifacts for Inference specification options, provide the following information:
      i. For Algorithm ARN, type the Amazon Resource Name (ARN) of the algorithm resource to use to create the model package.
      ii. For Location of model data artifacts, type the location in S3 where your model artifacts are stored.
   f. Choose Next.
4. On the Validation and scanning page, provide the following information:
a. For **Publish this model package on AWS Marketplace**, choose **Yes** to publish the model package on AWS Marketplace.

b. For **Validate this model package**, choose **Yes** if you want Amazon SageMaker to run batch transform jobs that you specify to test the inference code of your model package.

   **Note**
   To publish your model package on AWS Marketplace, your model package must be validated.

c. For **IAM role**, choose an IAM role that has the required permissions to run batch transform jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see *Amazon SageMaker Roles* (p. 415).

d. For **Validation profile**, specify the following:
   - A name for the validation profile.
   - A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition` (p. 900) input parameter of the `CreateAlgorithm` (p. 512) API.

5. Choose **Create model package**.

### Create a Model Package Resource (API)

To create a model package by using the Amazon SageMaker API, call the `CreateModelPackage` (p. 542) API.

### Use Algorithm and Model Package Resources

You can create algorithms and model packages as resources in your Amazon SageMaker account, and you can find and subscribe to algorithms and model packages on AWS Marketplace.

Use algorithms to:

- Run training jobs. For information, see *Use an Algorithm to Run a Training Job* (p. 272).
- Run hyperparameter tuning jobs. For information, see *Use an Algorithm to Run a Hyperparameter Tuning Job* (p. 274).
- Create model packages. After you use an algorithm resource to run a training job or a hyperparameter tuning job, you can use the model artifacts that these jobs output along with the algorithm to create a model package. For information, see *Create a Model Package Resource* (p. 269).

   **Note**
   If you subscribe to an algorithm on AWS Marketplace, you must create a model package before you can use it to get inferences by creating hosted endpoint or running a batch transform job.
Use model packages to:

- Create models that you can use to get real-time inference or run batch transform jobs. For information, see Use a Model Package to Create a Model (p. 277).
- Create hosted endpoints to get real-time inference. For information, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29).
- Create batch transform jobs. For information, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).

**Topics**

- Use an Algorithm to Run a Training Job (p. 272)
- Use an Algorithm to Run a Hyperparameter Tuning Job (p. 274)
- Use a Model Package to Create a Model (p. 277)

**Use an Algorithm to Run a Training Job**

You can create use an algorithm resource to create a training job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

**Topics**

- Use an Algorithm to Run a Training Job (Console) (p. 273)
- Use an Algorithm to Run a Training Job (API) (p. 274)
• Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK) (p. 274)

Use an Algorithm to Run a Training Job (Console)

To use an algorithm to run a training job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Algorithms.
3. Choose an algorithm that you created from the list on the My algorithms tab or choose an algorithm that you subscribed to on the AWS Marketplace subscriptions tab.
4. Choose Create training job.

The algorithm you chose will automatically be selected.
5. On the Create training job page, provide the following information:
   a. For Job name, type a name for the training job.
   b. For IAM role, choose an IAM role that has the required permissions to run training jobs in Amazon SageMaker, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 415).
   c. For Resource configuration, provide the following information:
      i. For Instance type, choose the instance type to use for training.
      ii. For Instance count, type the number of ML instances to use for the training job.
      iii. For Additional volume per instance (GB), type the size of the ML storage volume that you want to provision. ML storage volumes store model artifacts and incremental states.
      iv. For Encryption key, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instance, specify the key.
      v. For Stopping condition, specify the maximum amount of time in seconds, minutes, hours, or days, that you want the training job to run.
   d. For VPC, choose an Amazon VPC that you want to allow your training container to access. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud (p. 435).
   e. For Hyperparameters, specify the values of the hyperparameters to use for the training job.
   f. For Input data configuration, specify the following values for each channel of input data to use for the training job. You can see what channels the algorithm you're using for training sports, and the content type, supported compression type, and supported input modes for each channel, under Channel specification section of the Algorithm summary page for the algorithm.
      i. For Channel name, type the name of the input channel.
      ii. For Content type, type the content type of the data that the algorithm expects for the channel.
      iii. For Compression type, choose the data compression type to use, if any.
      iv. For Record wrapper, choose RecordIO if the algorithm expects data in the RecordIO format.
      v. For S3 data type, S3 data distribution type, and S3 location, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 869).
      vi. For Input mode, choose File to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose Pipe To stream data directly from Amazon S3 to the container.
vii. To add another input channel, choose Add channel. If you are finished adding input channels, choose Done.

g. For Output location, specify the following values:

i. For S3 output path, choose the S3 location where the training job stores output, such as model artifacts.

   Note
   You use the model artifacts stored at this location to create a model or model package from this training job.

ii. For Encryption key, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

h. For Tags, specify one or more tags to manage the training job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

i. Choose Create training job to run the training job.

Use an Algorithm to Run a Training Job (API)

To use an algorithm to run a training job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) as the AlgorithmName field of the AlgorithmSpecification (p. 744) object that you pass to CreateTrainingJob (p. 556). For information about training models in Amazon SageMaker, see Train a Model with Amazon SageMaker (p. 4).

Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a training job, create an AlgorithmEstimator object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the algorithm_arn argument. Then call the fit method of the estimator. For example:

```python
from sagemaker import AlgorithmEstimator
data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

Use an Algorithm to Run a Hyperparameter Tuning Job

A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. For more information, see Automatic Model Tuning (p. 290).

You can create use an algorithm resource to create a hyperparameter tuning job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.
Use an Algorithm to Run a Hyperparameter Tuning Job (Console)

To use an algorithm to run a hyperparameter tuning job (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**.
3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create hyperparameter tuning job**.

The algorithm you chose will automatically be selected.
5. On the **Create hyperparameter tuning job** page, provide the following information:
   a. For **Warm start**, choose **Enable warm start** to use the information from previous hyperparameter tuning jobs as a starting point for this hyperparameter tuning job. For more information, see Run a Warm Start Hyperparameter Tuning Job (p. 303).
      i. Choose **Identical data and algorithm** if your input data is the same as the input data for the parent jobs of this hyperparameter tuning job, or choose **Transfer learning** to use additional or different input data for this hyperparameter tuning job.
      ii. For **Parent hyperparameter tuning job(s)**, choose up to 5 hyperparameter tuning jobs to use as parents to this hyperparameter tuning job.
   b. For **Hyperparameter tuning job name**, type a name for the tuning job.
   c. For **IAM role**, choose an IAM role that has the required permissions to run hyperparameter tuning jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the **AmazonSageMakerFullAccess** managed policy attached. For information, see Amazon SageMaker Roles (p. 415).
   d. For **VPC**, choose a Amazon VPC that you want to allow the training jobs that the tuning job launches to access. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud (p. 435).
   e. Choose **Next**.
   f. For **Objective metric**, choose the metric that the hyperparameter tuning job uses to determine the best combination of hyperparameters, and choose whether to minimize or maximize this metric. For more information, see View the Best Training Job (p. 300).
   g. For **Hyperparameter configuration**, choose ranges for the tunable hyperparameters that you want the tuning job to search, and set static values for hyperparameters that you want to remain constant in all training jobs that the hyperparameter tuning job launches. For more information, see Define Hyperparameter Ranges (p. 292).
   h. Choose **Next**.
      i. For **Input data configuration**, specify the following values for each channel of input data to use for the hyperparameter tuning job. You can see what channels the algorithm you're using for hyperparameter tuning supports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.
         i. For **Channel name**, type the name of the input channel.
         ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.
iii. For **Compression type**, choose the data compression type to use, if any.

iv. For **Record wrapper**, choose RecordIO if the algorithm expects data in the RecordIO format.

v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see S3DataSource (p. 869).

vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe** to stream data directly from Amazon S3 to the container.

vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.

j. For **Output** location, specify the following values:

i. For **S3 output path**, choose the S3 location where the training jobs that this hyperparameter tuning job launches store output, such as model artifacts.

   **Note**
   You use the model artifacts stored at this location to create a model or model package from this hyperparameter tuning job.

ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.

k. For **Resource configuration**, provide the following information:

i. For **Instance type**, choose the instance type to use for each training job that the hyperparameter tuning job launches.

ii. For **Instance count**, type the number of ML instances to use for each training job that the hyperparameter tuning job launches.

iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision each training job that the hyperparameter tuning job launches. ML storage volumes store model artifacts and incremental states.

iv. For **Encryption key**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instances, specify the key.

l. For **Resource limits**, provide the following information:

i. For **Maximum training jobs**, specify the maximum number of training jobs that you want the hyperparameter tuning job to launch. A hyperparameter tuning job can launch a maximum of 500 training jobs.

ii. For **Maximum parallel training jobs**, specify the maximum number of concurrent training jobs that the hyperparameter tuning job can launch. A hyperparameter tuning job can launch a maximum of 10 concurrent training jobs.

iii. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want each training job that the hyperparameter tuning job launches to run.

m. For **Tags**, specify one or more tags to manage the hyperparameter tuning job. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see For more information, see AWS Tagging Strategies.

n. Choose **Create jobs** to run the hyperparameter tuning job.

---

**Use an Algorithm to Run a Hyperparameter Tuning Job (API)**

To use an algorithm to run a hyperparameter tuning job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) of the algorithm as the **AlgorithmName** field of the **AlgorithmSpecification** (p. 744) object that you pass to **CreateHyperParameterTuningJob** (p. 528).
For information about hyperparameter tuning in Amazon SageMaker, see Automatic Model Tuning (p. 290).

**Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK)**

Use an algorithm that you created or subscribed to on AWS Marketplace to create a hyperparameter tuning job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then initialize a `HyperparameterTuner` object with the `AlgorithmEstimator` you created as the value of the `estimator` argument. Finally, call the `fit` method of the `AlgorithmEstimator`. For example:

```python
from sagemaker import AlgorithmEstimator
from sagemaker.tuner import HyperparameterTuner

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.set_hyperparameters(max_leaf_nodes=10)
tuner = HyperparameterTuner(estimator=algo, base_tuning_job_name='some-name',
    objective_metric_name='validation:accuracy',
    hyperparameter_ranges=hyperparameter_ranges,
    max_jobs=2, max_parallel_jobs=2)

tuner.fit({'training': train_input}, include_cls_metadata=False)
tuner.wait()
```

**Use a Model Package to Create a Model**

Use a model package to create a deployable model that you can use to get real-time inferences by creating a hosted endpoint or to run batch transform jobs. You can create a deployable model from a model package by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

**Topics**

- Use a Model Package to Create a Model (Console) (p. 277)
- Use a Model Package to Create a Model (API) (p. 278)
- Use a Model Package to Create a Model (Amazon SageMaker Python SDK) (p. 278)

**Use a Model Package to Create a Model (Console)**

To create a deployable model from a model package (console)

2. Choose Model packages.
3. Choose a model package that you created from the list on the My model packages tab or choose a model package that you subscribed to on the AWS Marketplace subscriptions tab.
4. Choose Create model.
5. For Model name, type a name for the model.
6. For IAM role, choose an IAM role that has the required permissions to call other services on your behalf, or choose Create a new role to allow Amazon SageMaker to create a role that has the AmazonSageMakerFullAccess managed policy attached. For information, see Amazon SageMaker Roles (p. 415).
7. For VPC, choose a Amazon VPC that you want to allow the model to access. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud (p. 438).
8. Leave the default values for Container input options and Choose model package.
9. For environment variables, provide the names and values of environment variables you want to pass to the model container.
10. For Tags, specify one or more tags to manage the model. Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see AWS Tagging Strategies.
11. Choose Create model.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29). For information about batch transform jobs, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).

Use a Model Package to Create a Model (API)

To use a model package to create a deployable model by using the Amazon SageMaker API, specify the name or the Amazon Resource Name (ARN) of the model package as the ModelPackageName field of the ContainerDefinition (p. 765) object that you pass to the CreateModel (p. 538) API.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29). For information about batch transform jobs, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).

Use a Model Package to Create a Model (Amazon SageMaker Python SDK)

To use a model package to create a deployable model by using the Amazon SageMaker Python SDK, initialize a ModelPackage object, and pass the Amazon Resource Name (ARN) of the model package as the model_package_arn argument. For example:

```python
from sagemaker import ModelPackage
model = ModelPackage(role='SageMakerRole',
                      model_package_arn='training-job-scikit-decision-trees-1542660466-6f92',
                      sagemaker_session=sagemaker_session)
```

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29). For information about batch transform jobs, see Step 2.4.2: Deploy the Model to Batch Transform (p. 31).
Train a Model

For an overview on training a model with Amazon SageMaker, see Train a Model with Amazon SageMaker (p. 4).

Amazon SageMaker provides features to monitor and manage the training and validation of machine learning models. For guidance on metrics available, incremental training, automatic model tuning, and the use of augmented manifest files to label training data, see the following topics.

- For guidance on metrics used to monitor and train models, see Monitor and Analyze Training Jobs Using Metrics (p. 279).
- For guidance on incremental training in Amazon SageMaker, see Incremental Training in Amazon SageMaker (p. 285).
- For guidance on automatic model tuning, also known as hyperparameter tuning, see Automatic Model Tuning (p. 290).
- For guidance on using an augmented manifest file to label training data, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

Monitor and Analyze Training Jobs Using Metrics

An Amazon SageMaker training job is an iterative process that teaches a model to make predictions by presenting examples from a training dataset. Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help diagnose whether the model is learning well and will generalize well for making predictions on unseen data. The training algorithm writes the values of these metrics to logs, which Amazon SageMaker monitors and sends to Amazon CloudWatch in real time. To analyze the performance of your training job, you can view graphs of these metrics in CloudWatch. When a training job has completed, you can also get a list of the metric values that it computes in its final iteration by calling the DescribeTrainingJob (p. 629) operation.

Topics
- Training Metrics Sample Notebooks (p. 279)
- Defining Training Metrics (p. 280)
- Monitoring Training Job Metrics (Console) (p. 281)
- Monitoring Training Job Metrics (Amazon SageMaker Console) (p. 281)
- Example: Viewing a Training and Validation Curve (p. 284)

Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics:

- An Introduction to the Amazon SageMaker ObjectToVec Model for Sequence-to-sequence Embedding (object2vec_sentence_similarity.ipynb)
- Regression with the Amazon SageMaker XGBoost Algorithm (xgboost_abalone.ipynb)

For instructions how to create and access Jupyter notebook instances that you can use to run the examples in Amazon SageMaker, see Use Example Notebooks (p. 47). To see a list of all
the Amazon SageMaker samples, after creating and opening a notebook instance, choose the "SageMaker Examples" tab. To access the example notebooks that show how to use training metrics, `object2vec_sentence_similarity.ipynb` and `xgboost_abalone.ipynb`, from the Introduction to Amazon algorithms section. To open a notebook, choose its "Use" tab, then choose "Create copy."

## Defining Training Metrics

Amazon SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you want a custom algorithm to emit metrics to CloudWatch, you have to specify the metrics that you want Amazon SageMaker to send to CloudWatch when you configure the training job. You specify the name of the metrics that you want to send and the regular expressions that Amazon SageMaker uses to parse the logs that your algorithm emits to find those metrics.

You can specify the metrics that you want to track with the AWS Management Console, the Amazon SageMaker Python SDK (https://github.com/aws/sagemaker-python-sdk), or the low-level Amazon SageMaker API.

### Topics
- Defining Training Metrics (Low-level Amazon SageMaker API) (p. 280)
- Defining Training Metrics (Amazon SageMaker Python SDK) (p. 280)
- Define Training Metrics (Console) (p. 281)
- Defining Regular Expressions for Metrics (p. 281)

### Defining Training Metrics (Low-level Amazon SageMaker API)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions in the `MetricDefinitions` field of the `AlgorithmSpecification` input parameter that you pass to the `CreateTrainingJob` operation. For example, if your algorithm emits metrics for training error and validation error, and you want to monitor both of those metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```json
"AlgorithmSpecification": {
    "TrainingImage": "ContainerName",
    "TrainingInputMode": "File",
    "MetricDefinitions": [
        {
            "Name": "train:error",
            "Regex": ".*\[[0-9]+\]#011train-error:(\S+).*"
        },
        {
            "Name": "validation:error",
            "Regex": ".*\[[0-9]+\]#011validation-error:(\S+).*"
        }
    ]
}
```

For more information about defining and running a training job by using the low-level Amazon SageMaker API, see Step 2.3.2: Create a Training Job (p. 26).

### Defining Training Metrics (Amazon SageMaker Python SDK)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions as the `metric_definitions` argument when you initialize an `Estimator` object. For example, if your algorithm emits metrics for training error and validation error, and you want to monitor both of those metrics in CloudWatch, your `Estimator` initialization would look like the following:
estimator = Estimator(image_name=ImageName,
    role='SageMakerRole', train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    train_instance_type='ml.c4.xlarge',
    k=10,
    sagemaker_session=sagemaker_session,
    metric_definitions=[
        {'Name': 'train:error', 'Regex': '\[[0-9]+\] train-error:.*'},
        {'Name': 'validation:error', 'Regex': '\[[0-9]+\] validation-error:.*'}
    ]
)

For more information about training by using Amazon SageMaker Python SDK estimators, see https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

Define Training Metrics (Console)

You can define metrics for a custom algorithm in the console when you create a training job by providing the name and regular expression (regex) for Metrics.

Defining Regular Expressions for Metrics

To find a metric, Amazon SageMaker searches the logs that your algorithm emits and finds logs that match the regular expression that you specify for that metric. If you are using your own algorithm, do the following:

• Make sure that the algorithm writes the metrics that you want to capture to logs
• Define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

Monitoring Training Job Metrics (Console)

You can monitor the metrics that a training job emits in real time in the CloudWatch console.

To monitor training job metrics (CloudWatch console)

2. Choose Metrics, then choose /aws/sagemaker/TrainingJobs.
3. Choose TrainingJobName.
4. On the All metrics tab, choose the names of the training metrics that you want to monitor.
5. On the Graphed metrics tab, configure the graph options. For more information about using CloudWatch graphs, see Graph Metrics in the Amazon CloudWatch User Guide.

Monitoring Training Job Metrics (Amazon SageMaker Console)

You can monitor the metrics that a training job emits in real time by using the Amazon SageMaker console.
To monitor training job metrics (Amazon SageMaker console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Training jobs, then choose the training job whose metrics you want to see.
3. Choose TrainingJobName.
4. In the Monitor section, you can review the graphs of instance utilization and algorithm metrics.
Monitoring Training Job Metrics
(Amazon SageMaker Console)
Example: Viewing a Training and Validation Curve

Typically, you split the data that you train your model on into training and validation datasets. You use the training set to train the model parameters that are used to make predictions on the training dataset. Then you test how well the model makes predictions by calculating predictions for the validation set. To analyze the performance of a training job, you commonly plot a training curve against a validation curve. Viewing a graph that shows the accuracy for both the training and validation sets over time can help you to improve the performance of your model. For example, if training accuracy continues to increase over time, but, at some point, validation accuracy starts to decrease, you are likely overfitting your model. To address this, you can make adjustments to your model, such as increasing regularization.

For this example, you can use the `Image-classification-full-training` example that is in the `Example notebooks` section of your Amazon SageMaker notebook instance. If you don't have an Amazon SageMaker notebook instance, create one by following the instructions at Step 1: Create an Amazon SageMaker Notebook Instance (p. 17). If you prefer, you can follow along with the End-to-End Multiclass Image Classification Example in the example notebook on GitHub. You also need an Amazon S3 bucket to store the training data and for the model output. If you haven't created a bucket to use with Amazon SageMaker, create one by following the instructions at Step 3: Create an Amazon S3 Bucket (p. 16).

To view training and validation error curves

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Notebooks, and then choose Notebook instances.
3. Choose the notebook instance that you want to use, and then choose Open.
4. On the dashboard for your notebook instance, choose SageMaker Examples.
5. Expand the Introduction to Amazon Algorithms section, and then choose Use next to `Image-classification-full-training.ipynb`.
7. In the first code cell of the notebook, replace `<<bucket-name>>` with the name of your S3 bucket.
8. Run all of the cells in the notebook up to the Deploy section. You don't need to deploy an endpoint or get inference for this example.
10. Choose Metrics, then choose `/aws/sagemaker/TrainingJobs`.
11. Choose TrainingJobName.
12. On the All metrics tab, choose the `train:accuracy` and `validation:accuracy` metrics for the training job that you created in the notebook.
13. On the graph, choose an area that the metric's values to zoom in. You should see something like the following:
Incremental Training in Amazon SageMaker

Over time, you might find that a model generates inference that are not as good as they were in the past. With incremental training, you can use the artifacts from an existing model and use an expanded dataset to train a new model. Incremental training saves both time and resources.

Use incremental training to:

- Train a new model using an expanded dataset that contains an underlying pattern that was not accounted for in the previous training and which resulted in poor model performance.
- Use the model artifacts or a portion of the model artifacts from a popular publicly available model in a training job. You don’t need to train a new model from scratch.
Perform Incremental Training (Console)

To complete this procedure, you need:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see Common Parameters for Built-In Algorithms (p. 59).
- The URL of the S3 bucket where you've stored the model artifacts that you want to use in incremental training. To find the URL for the model artifacts, see the details page of the training job used to create the model. To find the details page, in the Amazon SageMaker console, choose Inference, choose Models, and then choose the model.

To restart a stopped training job, use the URL to the model artifacts that are stored in the details page as you would with a model or a completed training job.

To perform incremental training (console)

2. In the navigation pane, choose Training, then choose Training jobs.
3. Choose Create training job.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. The training job name must have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56).
6. (Optional) For Resource configuration, either leave the default values or increase the resource consumption to reduce computation time.
   a. (Optional) For Instance type, choose the ML compute instance type that you want to use. In most cases, ml.m4.xlarge is sufficient.
   b. For Instance count, use the default, 1.
   c. (Optional) For Additional volume per instance (GB), choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
   a. For Channel name, either leave the default (train) or enter a more meaningful name for the training dataset, such as expanded-training-dataset.
   b. For InputMode, choose File. For incremental training, you need to use file input mode.
   c. For S3 data distribution type, choose FullyReplicated. This causes each ML compute instance to use a full replicate of the expanded dataset when training incrementally.
   d. If the expanded dataset is uncompressed, set the Compression type to None. If the expanded dataset is compressed using Gzip, set it to Gzip.
   e. (Optional) If you are using File input mode, leave Content type empty. For Pipe input mode, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.
   f. For Record wrapper, if the dataset is saved in RecordIO format, choose RecordIO. If your dataset is not saved as a RecordIO formatted file, choose None.
   g. For S3 data type, if the dataset us stored as a single file, choose S3Prefix. If the dataset is stored as several files in a folder, choose Manifest.
   h. For S3 location, provide the URL to the path where you stored the expanded dataset.
   i. Choose Done.

8. To use model artifacts in a training job, you need to add a new channel and provide the needed information about the model artifacts.
   a. For Input data configuration, choose Add channel.
   b. For Channel name, enter model to identify this channel as the source of the model artifacts.
   c. For InputMode, choose File. Model artifacts are stored as files.
   d. For S3 data distribution type, choose FullyReplicated. This indicates that each ML compute instance should use all of the model artifacts for training.
   e. For Compression type, choose None because we are using a model for the channel.
   f. Leave Content type empty. Content type is the multipurpose internet mail extension (MIME) type of the data. For model artifacts, we leave it empty.
   g. Set Record wrapper to None because model artifacts are not stored in RecordIO format.
   h. For S3 data type, if you are using a built-in algorithm or an algorithm that stores the model as a single file, choose S3Prefix. If you are using an algorithm that stores the model as several files, choose Manifest.
   i. For S3 location, provide the URL to the path where you stored the model artifacts. Typically, the model is stored with the name model.tar.gz. To find the URL for the model artifacts, in the navigation pane, choose Inference, then choose Models. From the list of models, choose a model to display its details page. The URL for the model artifacts is listed under Primary container.
   j. Choose Done.

9. For Output data configuration, provide the following information:
   a. For S3 location, type the path to the S3 bucket where you want to store the output data.
   b. (Optional) For Encryption key, you can add your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

10. (Optional) For Tags, add one or more tags to the training job. A tag is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with Project as a key and a value referring to a project that is related to the training job, such as Home value forecasts.

11. Choose Create training job. Amazon SageMaker creates and runs training job.
After the training job has completed, the newly trained model artifacts are stored under the **S3 output path** that you provided in the **Output data configuration** field. To deploy the model to get predictions, see Step 2.4: Deploy the Model to Amazon SageMaker (p. 29).

### Perform Incremental Training (API)

This example shows how to use Amazon SageMaker APIs to train a model using the Amazon SageMaker image classification algorithm and the Caltech 256 Image Dataset, then train a new model using the first one. Please see the [incremental training sample notebook](#) for more details on using incremental training.

**Note**

In this example we used the original datasets in the incremental training, however you can use different datasets, such as ones that contain newly added samples. Upload the new datasets to S3 and make adjustments to the `data_channels` variable used to train the new model.

Get an AWS Identity and Access Management (IAM) role that grants required permissions and initialize environment variables:

```python
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

sess = sagemaker.Session()
print(sess)

bucket = sess.default_bucket()
print(bucket)

prefix = 'ic-incr-training'
```

Get the training image for the image classification algorithm:

```python
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sess.boto_region_name, 'image-classification', repo_version="latest")

# Display the training image
print (training_image)
```

Download the training and validation datasets, then upload them to Amazon Simple Storage Service (Amazon S3):

```python
import os
import urllib.request
import boto3

# Define a download function
def download(url):
    filename = url.split("/")[-1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)

# Download the caltech-256 training and validation datasets
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')

# Create four channels: train, validation, train_lst, and validation_lst
s3train = 's3://{}//{}//train/'.format(bucket, prefix)
s3validation = 's3://{}//{}//validation/'.format(bucket, prefix)
```

288
# Upload the first files to the train and validation channels

```bash
!aws s3 cp caltech-256-60-train.rec $s3train --quiet
!aws s3 cp caltech-256-60-val.rec $s3validation --quiet
```

Define the training hyperparameters:

```python
hyperparams = {
    "num_layers": "18",
    "resize": "32",
    "num_training_samples": "50000",
    "num_classes": "10",
    "image_shape": "3,28,28",
    "mini_batch_size": "128",
    "epochs": "3",
    "learning_rate": "0.1",
    "lr_scheduler_step": "2,3",
    "lr_scheduler_factor": "0.1",
    "augmentation_type": "crop_color",
    "optimizer": "sgd",
    "momentum": "0.9",
    "weight_decay": "0.0001",
    "beta_1": "0.9",
    "beta_2": "0.999",
    "gamma": "0.9",
    "eps": "1e-8",
    "top_k": "5",
    "checkpoint_frequency": "1",
    "use_pretrained_model": "0",
    "model_prefix": ""
}
```

Create an estimator object and train the first model using the training and validation datasets:

```python
# Define hyperparameters for the estimator
hyperparams = {
    "num_layers": "18",
    "resize": "32",
    "num_training_samples": "50000",
    "num_classes": "10",
    "image_shape": "3,28,28",
    "mini_batch_size": "128",
    "epochs": "3",
    "learning_rate": "0.1",
    "lr_scheduler_step": "2,3",
    "lr_scheduler_factor": "0.1",
    "augmentation_type": "crop_color",
    "optimizer": "sgd",
    "momentum": "0.9",
    "weight_decay": "0.0001",
    "beta_1": "0.9",
    "beta_2": "0.999",
    "gamma": "0.9",
    "eps": "1e-8",
    "top_k": "5",
    "checkpoint_frequency": "1",
    "use_pretrained_model": "0",
    "model_prefix": ""
}
```

```python
# Fit the base estimator
s3_output_location = 's3://{}/{}/output'.format(bucket, prefix)
ic = sagemaker.estimator.Estimator(training_image,
    role,
    train_instance_count=1,
    train_instance_type='ml.p2.xlarge',
    train_volume_size=50,
    train_max_run=360000,
    input_mode='File',
    output_path=s3_output_location,
    sagemaker_session=sess,
    hyperparameters=hyperparams)
train_data = sagemaker.session.s3_input(s3train, distribution='FullyReplicated',
    content_type='application/x-recordio',
    s3_data_type='S3Prefix')
validation_data = sagemaker.session.s3_input(s3validation, distribution='FullyReplicated',
    content_type='application/x-recordio',
    s3_data_type='S3Prefix')
data_channels = {'train': train_data, 'validation': validation_data}
ic.fit(inputs=data_channels, logs=True)
```

To use the model to incrementally train another model, create a new estimator object and use the model artifacts (ic.model_data, in this example) for the model_uri input argument:

```python
# Given the base estimator, create a new one for incremental training
incr_ic = sagemaker.estimator.Estimator(training_image,
    role,
```

289
Automatic Model Tuning

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a binary classification problem on a marketing dataset. Your goal is to maximize the area under the curve (auc) metric of the algorithm by training an XGBoost Algorithm (p. 246) model. You don't know which values of the eta, alpha, min_child_weight, and max_depth hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that Amazon SageMaker hyperparameter tuning searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

You can use Amazon SageMaker automatic model tuning with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for machine learning frameworks.

Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

You should also prepare your dataset and algorithm so that they work in Amazon SageMaker and successfully run a training job at least once. For information about setting up and running a training job, see Train a Model with a Built-in Algorithm and Deploy It (p. 22).

Topics

- How Hyperparameter Tuning Works (p. 291)
- Define Objective Metrics (p. 291)
- Define Hyperparameter Ranges (p. 292)
- Example: Hyperparameter Tuning Job (p. 293)
- Stop Training Jobs Early (p. 301)
- Run a Warm Start Hyperparameter Tuning Job (p. 303)
- Best Practices for Hyperparameter Optimization (p. 306)
How Hyperparameter Tuning Works

Hyperparameter tuning is a supervised machine learning regression problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. You can choose any metric that the algorithm you use defines. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these guesses. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Hyperparameter tuning uses an Amazon SageMaker implementation of Bayesian optimization.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything it knows about this problem so far. Sometimes it chooses a point that is likely to produce an incremental improvement on the best result found so far. This allows hyperparameter tuning to exploit the best known results. Other times it chooses a set of hyperparameters far removed from those it has tried. This allows it to explore the space to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

For more information about Bayesian optimization, see the following:

- A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning
- Practical Bayesian Optimization of Machine Learning Algorithms
- Taking the Human Out of the Loop: A Review of Bayesian Optimization

Note
Hyperparameter tuning might not improve your model. It is an advanced tool for building machine solutions, and, as such, should be considered part of the scientific development process.

When you build complex machine learning systems like deep learning neural networks, exploring all of the possible combinations is impractical. Hyperparameter tuning can accelerate your productivity by trying many variations of a model, focusing on the most promising combinations of hyperparameter values within the ranges that you specify. To get good results, you need to choose the right ranges to explore. Because the algorithm itself is stochastic, it's possible that the hyperparameter tuning model will fail to converge on the best answer, even if the ranges specified are correct.

Define Objective Metrics

Note
When you use one of the Amazon SageMaker built-in algorithms, you don't need to define metrics. Built-in algorithms automatically send metrics to hyperparameter tuning. You do need to choose one of the metrics that the built-in algorithm emits as the objective metric for the tuning job. For a list of metrics that a built-in algorithm emits, see the Metrics table for the algorithm in Use Amazon SageMaker Built-in Algorithms (p. 56).

To optimize hyperparameters for a machine learning model, a tuning job evaluates the training jobs it launches by using a metric that the training algorithm writes to logs. Amazon SageMaker hyperparameter tuning parses your algorithm's stdout and stderr streams to find algorithm metrics, such as loss or validation-accuracy, that show how well the model is performing on the dataset.

Note
These are the same metrics that Amazon SageMaker sends to CloudWatch Logs. For more information, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 429).

If you use your own algorithm for hyperparameter tuning, make sure that your algorithm emits at least one metric by writing evaluation data to stderr or stdout.
You can define up to 20 metrics for your tuning job to monitor. You choose one of those metrics to be the objective metric, which hyperparameter tuning uses to evaluate the training jobs. The hyperparameter tuning job returns the training job that returned the best value for the objective metric as the best training job.

You define metrics for a tuning job by specifying a name and a regular expression for each metric that your tuning job monitors. Design the regular expressions to match metrics that your algorithm emits. You pass these metrics to the CreateHyperParameterTuningJob operation in the TrainingJobDefinition parameter as the MetricDefinitions field of the AlgorithmSpecification field.

Metric definitions have the following structure:

```json
[
  {
    "Name": "validation:rmse",
    "Regex": ".*\[[0-9]+\].*\#011validation-rmse:\(\S+\)"
  },
  {
    "Name": "validation:auc",
    "Regex": ".*\[[0-9]+\].*\#011validation-auc:\(\S+\)"
  },
  {
    "Name": "train:auc",
    "Regex": ".*\[[0-9]+\]\#011train-auc:\(\S+\)"
  }
]
```

A regular expression (regex) matches what is in the log, like a search function. Special characters in the regex affect the search. For example, in the regex for the valid-acc metric defined above, the parentheses tell the regex to capture what's inside them.

Choose one of the metrics that you define as the objective metric for the tuning job. If you are using the Specify the value of the name key in the HyperParameterTuningJobObjective field of the HyperParameterTuningJobConfig parameter that you send to the CreateHyperParameterTuningJob operation.

### Define Hyperparameter Ranges

Hyperparameter tuning finds the best hyperparameter values for your model by searching over ranges of hyperparameters. You specify the hyperparameters and range of values over which to search by defining hyperparameter ranges for your tuning job. Choosing hyperparameters and ranges significantly affects the performance of your tuning job. For guidance on choosing hyperparameters and ranges, see Best Practices for Hyperparameter Optimization.

To define hyperparameter ranges by using the low-level API, you specify the names of hyperparameters and ranges of values in the ParameterRanges field of the HyperParameterTuningJobConfig parameter that you pass to the CreateHyperParameterTuningJob operation. The ParameterRanges field has three subfields, one for each of the categorical, integer, and continuous hyperparameter ranges. You can define up to 20 hyperparameters to search over. Each value of a categorical hyperparameter range counts as a hyperparameter against the limit. Hyperparameter ranges have the following structure:

```json
"ParameterRanges": {
  "CategoricalParameterRanges": [
    {
      "Name": "tree_method",
      "Values": ["auto", "exact", "approx", "hist"]
    }
  ]
},
```

292
"ContinuousParameterRanges": [
  {
    "Name": "eta",
    "MaxValue": "0.5",
    "MinValue": "0"
  }
],
"IntegerParameterRanges": [
  {
    "Name": "max_depth",
    "MaxValue": "10",
    "MinValue": "1"
  }
]

Example: Hyperparameter Tuning Job

This example shows how to create a new notebook for configuring and launching a hyperparameter tuning job. The tuning job uses the XGBoost Algorithm (p. 246) to train a model to predict whether a customer will enroll for a term deposit at a bank after being contacted by phone.

You use the low-level AWS SDK for Python (Boto) to configure and launch the hyperparameter tuning job, and the AWS Management Console to monitor the status of hyperparameter training jobs. You can also use the Amazon SageMaker high-level Amazon SageMaker Python SDK to configure, run, monitor, and analyze hyperparameter tuning jobs. For more information, see https://github.com/aws/sagemaker-python-sdk.

Prerequisites

To run the code in this example, you need

- An AWS account and an administrator user (p. 15)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training (p. 16)
- A running Amazon SageMaker notebook instance (p. 17)

Topics

- Create a Notebook (p. 293)
- Get the an Amazon Sagemaker Boto 3 Client (p. 294)
- Get the Amazon SageMaker Execution Role (p. 294)
- Specify a Bucket and Data Output Location (p. 295)
- Download, Prepare, and Upload Training Data (p. 295)
- Configure and Launch a Hyperparameter Tuning Job (p. 296)
- Monitor the Progress of a Hyperparameter Tuning Job (p. 299)
- Clean up (p. 301)

Create a Notebook

Create a Jupyter notebook that contains a preinstalled environment with the default Anaconda installation and Python3.

To create a Jupyter notebook

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/
2. Open a running notebook instance, by choosing **Open** next to its name. The Jupyter notebook server page appears:

![Jupyter Notebook Server Page]

3. To create a notebook, choose **Files**, **New**, and **conda_python3**.
4. Name the notebook.

**Next Step**

**Get the an Amazon Sagemaker Boto 3 Client (p. 294)**

**Get the an Amazon Sagemaker Boto 3 Client**

Import libraries and get a boto3 client, which you use to call the hyperparameter tuning APIs.

In the new Jupyter notebook, type the following code:

```python
import sagemaker
import boto3
from sagemaker.predictor import csv_serializer  # Converts strings for HTTP POST requests on inference
import numpy as np  # For performing matrix operations and numerical processing
import pandas as pd  # For manipulating tabular data
import os

region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')
```

**Next Step**

**Get the Amazon SageMaker Execution Role (p. 294)**

**Get the Amazon SageMaker Execution Role**

Get the execution role for the notebook instance. This is the IAM role that you created when you created your notebook instance. You pass the role to the tuning job.

```python
from sagemaker import get_execution_role

role = sagemaker.get_execution_role()
print(role)
```

**Next Step**

**Specify a Bucket and Data Output Location (p. 295)**
Specify a Bucket and Data Output Location

Specify the name of the Amazon S3 bucket where you want to store the output of the training jobs that the tuning job launches. The name of the bucket must start with \texttt{sagemaker}, and be globally unique. The bucket must be in the same AWS Region as the notebook instance that you use for this example. You can use the bucket that you created when you set up Amazon SageMaker, or you can create a new bucket. For information, see Step 3: Create an Amazon S3 Bucket (p. 16).

\texttt{prefix} is the path within the bucket where Amazon SageMaker stores the output from training jobs.

\begin{verbatim}
bucket = 'sagemaker-MyBucket'  # Replace with the name of your S3 bucket
prefix = 'sagemaker/DEMO-automatic-model-tuning-xgboost-dm'
\end{verbatim}

Next Step

Download, Prepare, and Upload Training Data (p. 295)

Download, Prepare, and Upload Training Data

For this example, you use a training dataset of information about bank customers that includes the customer's job, marital status, and how they were contacted during the bank's direct marketing campaign. To use a dataset for a hyperparameter tuning job, you download it, transform the data, and then upload it to an Amazon S3 bucket.

For more information about the dataset and the data transformation that the example performs, see the \texttt{hpo_xgboost_direct_marketing_sagemaker_APIs} notebook in the Hyperparameter Tuning section of the SageMaker Examples tab in your notebook instance.

Download and Explore the Training Dataset

To download and explore the dataset, run the following code in your notebook:

\begin{verbatim}
!unzip -o bank-additional.zip
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)  # Make sure we can see all of the columns
pd.set_option('display.max_rows', 5)      # Keep the output on one page
data
\end{verbatim}

Prepare and Upload Data

Before creating the hyperparameter tuning job, prepare the data and upload it to an S3 bucket where the hyperparameter tuning job can access it.

Run the following code in your notebook:

\begin{verbatim}
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)  # Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'unemployed']), 1, 0)  # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)  # Convert categorical variables to sets of indicators
model_data
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)
\end{verbatim}
Example: Hyperparameter Tuning Job

```
train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data)), int(0.9*len(model_data))])

pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('validation.csv', index=False, header=False)
pd.concat([test_data['y_yes'], test_data.drop(['y_no', 'y_yes'], axis=1)], axis=1).to_csv('test.csv', index=False, header=False)
```

Next Step

Configure and Launch a Hyperparameter Tuning Job (p. 296)

**Configure and Launch a Hyperparameter Tuning Job**

To configure and launch a hyperparameter tuning job, complete the following steps.

**Topics**

- Specify the Hyperparameter Tuning Job Settings (p. 296)
- Configure the Training Jobs (p. 297)
- Name and Launch the Hyperparameter Tuning Job (p. 298)
- Next Step (p. 299)

**Specify the Hyperparameter Tuning Job Settings**

To specify settings for the hyperparameter tuning job, you define a JSON object. You pass the object as the value of the `HyperParameterTuningJobConfig` parameter to `CreateHyperParameterTuningJob` (p. 528) when you create the tuning job.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune. For more information, see Define Hyperparameter Ranges (p. 292)
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An objective metric is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the XGBoost Algorithm (p. 246) built-in algorithm. The objective metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```
tuning_job_config = {
  "ParameterRanges": {
    "CategoricalParameterRanges": [],
    "ContinuousParameterRanges": [
      {
        "MaxValue": "1",
        "MinValue": "0",
        "Name": "eta"
      },
```

296
Example: Hyperparameter Tuning Job

```json
{
    "MaxValue": "2",
    "MinValue": "0",
    "Name": "alpha"
},
{
    "MaxValue": "10",
    "MinValue": "1",
    "Name": "min_child_weight"
},
"IntegerParameterRanges": [
    {
    "MaxValue": "10",
    "MinValue": "1",
    "Name": "max_depth"
    }
],
"ResourceLimits": {
    "MaxNumberOfTrainingJobs": 20,
    "MaxParallelTrainingJobs": 3
},
"Strategy": "Bayesian",
"HyperParameterTuningJobObjective": {
    "MetricName": "validation:auc",
    "Type": "Maximize"
}
}
```

Configure the Training Jobs

To configure the training jobs that the tuning job launches, define a JSON object that you pass as the value of the `TrainingJobDefinition` parameter of the `CreateHyperParameterTuningJob` call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.

  **Note**
  
  Specify metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don’t specify metrics.

- The container image that specifies the training algorithm.
- The input configuration for your training and test data.
- The storage location for the algorithm’s output. Specify the S3 bucket where you want to store the output of the training jobs.
- The values of algorithm hyperparameters that are not tuned in the tuning job.
- The type of instance to use for the training jobs.
- The stopping condition for the training jobs. This is the maximum duration for each training job.

In this example, we set static values for the `eval_metric`, `num_round`, `objective`, `rate_drop`, and `tweedie_variance_power` parameters of the XGBoost Algorithm built-in algorithm.

```python
from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(boto3.Session().region_name, 'xgboost')
s3_input_train = 's3://{}/{}train'.format(bucket, prefix)
s3_input_validation = 's3://{}/{}validation/'.format(bucket, prefix)
```
training_job_definition = {
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train
                }
            }
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation
                }
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://{}//{}//output".format(bucket, prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.2xlarge",
        "VolumeSizeInGB": 10
    },
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 43200
    }
}

### Name and Launch the Hyperparameter Tuning Job

Now you can provide a name for the hyperparameter tuning job and then launch it by calling the `CreateHyperParameterTuningJob` (p. 528) API. Pass `tuning_job_config`, and `training_job_definition` that you created in previous steps as the values of the parameters.

tuning_job_name = "MyTuningJob"
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = tuning_job_name,
                                            HyperParameterTuningJobConfig =
                                            tuning_job_config,
                                            TrainingJobDefinition = training_job_definition)
Next Step

Monitor the Progress of a Hyperparameter Tuning Job (p. 299)

Monitor the Progress of a Hyperparameter Tuning Job

To monitor the progress of a hyperparameter tuning job and the training jobs that it launches, use the Amazon SageMaker console.

Topics

- View the Status of the Hyperparameter Tuning Job (p. 299)
- View the Status of the Training Jobs (p. 299)
- View the Best Training Job (p. 300)

View the Status of the Hyperparameter Tuning Job

To view the status of the hyperparameter tuning job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Hyperparameter tuning jobs.
3. In the list of hyperparameter tuning jobs, check the status of the hyperparameter tuning job you launched. A tuning job can be:
   - Completed—The hyperparameter tuning job successfully completed.
   - InProgress—The hyperparameter tuning job is in progress. One or more training jobs are still running.
   - Failed—The hyperparameter tuning job failed.
   - Stopped—The hyperparameter tuning job was manually stopped before it completed. All training jobs that the hyperparameter tuning job launched are stopped.
   - Stopping—The hyperparameter tuning job is in the process of stopping.

View the Status of the Training Jobs

To view the status of the training jobs that the hyperparameter tuning job launched

1. In the list of hyperparameter tuning jobs, choose the job that you launched.
2. Choose Training jobs.
3. View the status of each training job. To see more details about a job, choose it in the list of training jobs. To view a summary of the status of all of the training jobs that the hyperparameter tuning job launched, see Training job status counter.

A training job can be:

- **Completed**—The training job successfully completed.
- **InProgress**—The training job is in progress.
- **Stopped**—The training job was manually stopped before it completed.
- **Failed (Retriable)**—The training job failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
- **Failed (Non-retriable)**—The training job failed and can't be retried. A failed training job can't be retried when a client error occurs.

**View the Best Training Job**

A hyperparameter tuning job uses the objective metric that each training job returns to evaluate training jobs. While the hyperparameter tuning job is in progress, the best training job is the one that has returned the best objective metric so far. After the hyperparameter tuning job is complete, the best training job is the one that returned the best objective metric.

To view the best training job, choose Best training job.
To deploy the best training job as a model that you can host at an Amazon SageMaker endpoint, choose **Create model**.

**Next Step**

**Clean up**

To avoid incurring unnecessary charges, when you are done with the example, use the AWS Management Console to delete the resources that you created for it.

**Note**

If you plan to explore other examples, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at [https://console.aws.amazon.com/sagemaker/](https://console.aws.amazon.com/sagemaker/) and delete the notebook instance. Stop the instance before deleting it.
2. Open the Amazon S3 console at [https://console.aws.amazon.com/s3/](https://console.aws.amazon.com/s3/) and delete the bucket that you created to store model artifacts and the training dataset.
3. Open the IAM console at [https://console.aws.amazon.com/iam/](https://console.aws.amazon.com/iam/) and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at [https://console.aws.amazon.com/cloudwatch/](https://console.aws.amazon.com/cloudwatch/) and delete all of the log groups that have names starting with `/aws/sagemaker/`.

**Stop Training Jobs Early**

Stop the training jobs that a hyperparameter tuning job launches early when they are not improving significantly as measured by the objective metric. Stopping training jobs early can help reduce compute time and helps you avoid overfitting your model. To configure a hyperparameter tuning job to stop training jobs early, set the `TrainingJobEarlyStoppingType` field of the `HyperParameterTuningJobConfig` object that you use to configure the tuning job to
For information about configuring and launching a hyperparameter tuning job, see Example: Hyperparameter Tuning Job (p. 293).

**How Early Stopping Works**

When you enable early stopping for a hyperparameter tuning job, Amazon SageMaker evaluates each training job the hyperparameter tuning job launches as follows:

- After each epoch of training, get the value of the objective metric.
- Compute the running average of the objective metric for all previous training jobs up to the same epoch, and then compute the median of all of the running averages.
- If the value of the objective metric for the current training job is worse (higher when minimizing or lower when maximizing the objective metric) than the median value of running averages of the objective metric for previous training jobs up to the same epoch, Amazon SageMaker stops the current training job.

**Algorithms That Support Early Stopping**

To support early stopping, an algorithm must emit objective metrics for each epoch. The following built-in Amazon SageMaker algorithms support early stopping:

- **Linear Learner Algorithm (p. 159)**—Supported only if you use `objective_loss` as the objective metric.
- **XGBoost Algorithm (p. 246)**
- **Image Classification Algorithm (p. 106)**
- **Object Detection Algorithm (p. 192)**
- **Sequence-to-Sequence Algorithm (p. 233)**
- **IP Insights Algorithm (p. 127)**

**Note**

This list of built-in algorithms that support early stopping is current as of December 13, 2018. Other built-in algorithms might support early stopping in the future. If an algorithm emits a metric that can be used as an objective metric for a hyperparameter tuning job (preferably a validation metric), then it supports early stopping.

To use early stopping with your own algorithm, you must write your algorithms such that it emits the value of the objective metric after each epoch. The following list shows how you can do that in different frameworks:

**TensorFlow**

Use the `tf.contrib.learn.monitors.ValidationMonitor` class. For information, see https://www.tensorflow.org/api_docs/python/tf/contrib/learn/monitors.

**MXNet**

Use the `mxnet.callback.LogValidationMetricsCallback`. For information, see https://mxnet.apache.org/api/python/callback/callback.html.

**Chainer**

Extend chainer by using the `extensions.Evaluator` class. For information, see https://docs.chainer.org/en/v1.24.0/reference/extensions.html#evaluator.

**PyTorch and Spark**

There is no high-level support. You must explicitly write your training code so that it computes objective metrics and writes them to logs after each epoch.
Run a Warm Start Hyperparameter Tuning Job

Use warm start to start a hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job. Hyperparameter tuning uses Bayesian search to choose combinations of hyperparameter values from ranges that you specify. For more information, see How Hyperparameter Tuning Works (p. 291). Using information from previous hyperparameter tuning jobs can help increase the performance of the new hyperparameter tuning job by making the search for the best combination of hyperparameters more efficient.

Note
Warm start tuning jobs typically take longer to start than standard hyperparameter tuning jobs, because the results from the parent jobs have to be loaded before the job can start. The increased time depends on the total number of training jobs launched by the parent jobs.

Reasons you might want to consider warm start include:

- You want to gradually increase the number of training jobs over several tuning jobs based on the results you see after each iteration.
- You get new data, and want to tune a model using the new data.
- You want to change the ranges of hyperparameters that you used in a previous tuning job, change static hyperparameters to tunable, or change tunable hyperparameters to static values.
- You stopped a previous hyperparameter job early or it stopped unexpectedly.

Topics
- Types of Warm Start Tuning Jobs (p. 303)
- Warm Start Tuning Restrictions (p. 304)
- Warm Start Tuning Sample Notebook (p. 304)
- Create a Warm Start Tuning Job (p. 305)

Types of Warm Start Tuning Jobs

There are two different types of warm start tuning jobs:

**IDENTICAL_DATA_AND_ALGORITHM**

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed.

Use identical data and algorithm when you use the same training data as you used in a previous hyperparameter tuning job, but you want to increase the total number of training jobs or change ranges or values of hyperparameters.

When you run an warm start tuning job of type **IDENTICAL_DATA_AND_ALGORITHM**, there is an additional field in the response to DescribeHyperParameterTuningJob (p. 601) named OverallBestTrainingJob. The value of this field is the TrainingJobSummary (p. 893) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.
TRANSFER_LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The training algorithm image can also be a different version from the version used in the parent hyperparameter tuning job. When you use transfer learning, changes in the dataset or the algorithm that significantly affect the value of the objective metric might reduce the usefulness of using warm start tuning.

Warm Start Tuning Restrictions

The following restrictions apply to all warm start tuning jobs:

- A tuning job can have a maximum of 5 parent jobs, and all parent jobs must be in a terminal state (Completed, Stopped, or Failed) before you start the new tuning job.
- The objective metric used in the new tuning job must be the same as the objective metric used in the parent jobs.
- The total number of static plus tunable hyperparameters must remain the same between parent jobs and the new tuning job. Because of this, if you think you might want to use a hyperparameter as tunable in a future warm start tuning job, you should add it as a static hyperparameter when you create a tuning job.
- The type of each hyperparameter (continuous, integer, categorical) must not change between parent jobs and the new tuning job.
- The number of total changes from tunable hyperparameters in the parent jobs to static hyperparameters in the new tuning job, plus the number of changes in the values of static hyperparameters cannot be more than 10. Each value in a categorical hyperparameter counts against this limit. For example, if the parent job has a tunable categorical hyperparameter with the possible values red and blue, you change that hyperparameter to static in the new tuning job, that counts as 2 changes against the allowed total of 10. If the same hyperparameter had a static value of red in the parent job, and you change the static value to blue in the new tuning job, it also counts as 2 changes.
- Warm start tuning is not recursive. For example, if you create MyTuningJob3 as a warm start tuning job with MyTuningJob2 as a parent job, and MyTuningJob2 is itself an warm start tuning job with a parent job MyTuningJob1, the information that was learned when running MyTuningJob1 is not used for MyTuningJob3. If you want to use the information from MyTuningJob1, you must explicitly add it as a parent for MyTuningJob3.
- The training jobs launched by every parent job in a warm start tuning job count against the 500 maximum training jobs for a tuning job.
- Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Warm Start Tuning Sample Notebook

For a sample notebook that shows how to use warm start tuning, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_warmstart/hpo_image_classification_warmstart.ipynb. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Example Notebooks (p. 47). Once you have created a notebook instance and opened it, select the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The warm start tuning example notebook is located in the Hyperparameter tuning section, and is named hpo_image_classification_warmstart.ipynb. To open a notebook, click on its Use tab and select Create copy.
Create a Warm Start Tuning Job

You can use either the low-level AWS SDK for Python (Boto 3) or the high-level Amazon SageMaker Python SDK to create a warm start tuning job.

Topics

• Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3)) (p. 305)
• Create a Warm Start Tuning Job (Amazon SageMaker Python SDK) (p. 305)

Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3))

To use warm start tuning, you specify the values of a HyperParameterTuningJobWarmStartConfig (p. 804) object, and pass that as the WarmStartConfig field in a call to CreateHyperParameterTuningJob (p. 528).

The following code shows how to create a HyperParameterTuningJobWarmStartConfig (p. 804) object and pass it to CreateHyperParameterTuningJob (p. 528) job by using the low-level Amazon SageMaker API for Python (Boto 3).

Create the HyperParameterTuningJobWarmStartConfig object:

```python
warm_start_config = {
    "ParentHyperParameterTuningJobs" : [
        {"HyperParameterTuningJobName" : 'MyParentTuningJob'}
    ],
    "WarmStartType" : "IdenticalDataAndAlgorithm"
}
```

Create the warm start tuning job:

```python
smclient = boto3.Session().client('sagemaker')
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = 'MyWarmStartTuningJob',
                                        HyperParameterTuningJobConfig = tuning_job_config, # See notebook for tuning configuration
                                        TrainingJobDefinition = training_job_definition, # See notebook for job definition
                                        WarmStartConfig = warm_start_config)
```

Create a Warm Start Tuning Job (Amazon SageMaker Python SDK)

To use the Amazon SageMaker Python SDK to run a warm start tuning job, you:

• Specify the parent jobs and the warm start type by using a WarmStartConfig object.
• Pass the WarmStartConfig object as the value of the warm_start_config argument of a HyperparameterTuner object.
• Call the fit method of the HyperparameterTuner object.

For more information about using the Amazon SageMaker Python SDK for hyperparameter tuning, see https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning.

This example uses an estimator that uses the Image Classification Algorithm (p. 106) algorithm for training. The following code sets the hyperparameter ranges that the warm start tuning job searches within to find the best combination of values. For information about setting hyperparameter ranges, see Define Hyperparameter Ranges (p. 292).
hyperparameter_ranges = {'learning_rate': ContinuousParameter(0.0, 0.1),
                          'momentum': ContinuousParameter(0.0, 0.99)}

The following code configures the warm start tuning job by creating a `WarmStartConfig` object.

```python
from sagemaker.tuner import WarmStartConfig,
                      WarmStartTypes
parent_tuning_job_name = "MyParentTuningJob"
warm_start_config = WarmStartConfig(type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
                                     parents={parent_tuning_job_name})
```

Now set the values for static hyperparameters, which are hyperparameters that keep the same value for every training job that the warm start tuning job launches. In the following code, `imageclassification` is an estimator that was created previously.

```python
imageclassification.set_hyperparameters(num_layers=18,
                                         image_shape='3,224,224',
                                         num_classes=257,
                                         num_training_samples=15420,
                                         mini_batch_size=128,
                                         epochs=30,
                                         optimizer='sgd',
                                         top_k='2',
                                         precision_dtype='float32',
                                         augmentation_type='crop')
```

Now create the `HyperparameterTuner` object and pass the `WarmStartConfig` object that you previously created as the `warm_start_config` argument.

```python
tuner_warm_start = HyperparameterTuner(imageclassification,
                                         'validation:accuracy',
                                         hyperparameter_ranges,
                                         objective_type='Maximize',
                                         max_jobs=10,
                                         max_parallel_jobs=2,
                                         base_tuning_job_name='warmstart',
                                         warm_start_config=warm_start_config)
```

Finally, call the `fit` method of the `HyperparameterTuner` object to launch the warm start tuning job.

```python
tuner_warm_start.fit(
    {'train': s3_input_train, 'validation': s3_input_validation},
    include_cls_metadata=False)
```

**Best Practices for Hyperparameter Optimization**

Hyperparameter optimization is not a fully-automated process. To improve optimization, use the following guidelines when you create hyperparameters.

**Topics**

- Choosing the Number of Hyperparameters (p. 307)
- Choosing Hyperparameter Ranges (p. 307)
- Using Logarithmic Scales for Hyperparameters (p. 307)
- Choosing the Best Number of Concurrent Training Jobs (p. 307)
• Running Training Jobs on Multiple Instances (p. 307)

Choosing the Number of Hyperparameters

The difficulty of a hyperparameter tuning job depends primarily on the number of hyperparameters that Amazon SageMaker has to search. Although you can simultaneously use up to 20 variables in a hyperparameter tuning job, limiting your search to a much smaller number is likely to give better results.

Choosing Hyperparameter Ranges

The ranges for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. Although you might want to specify a very large range that covers every possible value, you will get better results by limiting your search to a small range where all possible values in the range are reasonable. If you get the best metric values within a part of a range, consider limiting the range to that part.

Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, Amazon SageMaker attempts to figure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they should be log-scaled, it might take some time for Amazon SageMaker to discover that. If you know that a hyperparameter should be log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

Choosing the Best Number of Concurrent Training Jobs

Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.

Running Training Jobs on Multiple Instances

When a training job runs on multiple instances, hyperparameter tuning uses the last-reported objective metric from all instances of that training job. Design distributed training jobs so that you get the metric report you want.

Provide Dataset Metadata to Training Jobs with an Augmented Manifest File

To classify data into different groupings, you train a model by using a dataset and metadata that act as labels. To include metadata with your dataset in a training job, use an augmented manifest file. When using an augmented manifest file, your dataset must be stored in Amazon Simple Storage Service (Amazon S3) and you must configure your training job to use dataset stored there. You specify the location and format of this dataset for one or more channels.

When specifying a channel's parameters, you specify a path to the file, called a S3Uri. Amazon SageMaker interprets this URI based on the specified S3DataType. The AugmentedManifestFile option defines a manifest format that includes metadata with the input data. Using an augmented manifest file is an alternative to preprocessing when you have labeled data. For training jobs using labeled data, you typically need to preprocess the dataset to combine input data with metadata before training. If your training dataset is large, preprocessing can be time consuming and expensive.
Augmented Manifest File Format

An augmented manifest file must be formatted in JSON Lines format. In JSON Lines format, each line in the file is a complete JSON object followed by a newline separator.

During training, Amazon SageMaker parses each JSON line and sends some or all of its attributes on to the training algorithm. You specify which attribute contents to pass and the order in which to pass them with the `AttributeNames` parameter of the `CreateTrainingJob` API. The `AttributeNames` parameter is an ordered list of attribute names that Amazon SageMaker looks for in the JSON object to use as training input.

For example, if you list `"line", "book"` for `AttributeNames`, the input data must include the attribute names of `line` and `book` in the specified order. For this example, the following augmented manifest file content is valid:

```json
{"author": "Herman Melville", "line": "Call me Ishmael", "book": "Moby Dick"}
{"line": "It was love at first sight.", "author": "Joseph Heller", "book": "Catch-22"}
```

Amazon SageMaker ignores unlisted attribute names even if they precede, follow, or are in between listed attributes.

When using augmented manifest files, observe the following guidelines:

- The order of the attributes listed in the `AttributeNames` parameter determines the order of the attributes passed to the algorithm in the training job.
- The listed `AttributeNames` can be a subset of all of the attributes in the JSON line. Amazon SageMaker ignores unlisted attributes in the file.
- You can specify any type of data allowed by the JSON format in `AttributeNames`, including text, numerical, data arrays, or objects.
- To include an S3 URI as an attribute name, add the suffix `-ref` to it.

If an attribute name contains the suffix `-ref`, the attribute's value must be an S3 URI to a data file that is accessible to the training job. For example, if `AttributeNames` contains `"image-ref", "is-a-cat"`, a valid augmented manifest file might contain these lines:

```json
{"image-ref": "s3://mybucket/sample01/image1.jpg", "is-a-cat": 1}
{"image-ref": "s3://mybucket/sample02/image2.jpg", "is-a-cat": 0}
```

For the first line of this manifest, Amazon SageMaker retrieves the contents of the S3 object `s3://mybucket/foo/image1.jpg` and streams it to the algorithm for training. The second line is the string representation of the `is-a-cat` attribute "1", which is followed by the contents of the second line.

Stream Augmented Manifest File Data

Augmented manifest files are supported only for channels using Pipe input mode. For each channel, the data is extracted from its augmented manifest file and streamed (in order) to the algorithm through the channel's named pipe. Pipe mode uses the first in first out (FIFO) method, so records are processed in the order in which they are queued. For information about Pipe input mode, see `InputMode`.

Attribute names with a `-ref` suffix point to preformatted binary data. In some cases, the algorithm knows how to parse the data. In other cases, you might need to wrap the data so that records are delimited for the algorithm. If the algorithm is compatible with RecordIO-formatted data, specifying `RecordIO` for `RecordWrapperType` solves this issue. If the algorithm is not compatible with RecordIO format, specify `None` for `RecordWrapperType` and make sure that your data is parsed correctly for...
your algorithm. Using the ["image-ref", "is-a-cat"] example, if you use RecordIO wrapping, the following stream of data is sent to the queue:

```
recordio_formatted(s3://mybucket/foo/image1.jpg)recordio_formatted("1")
recordio_formatted(s3://mybucket/bar/image2.jpg)recordio_formatted("0")
```

Images that aren't wrapped with RecordIO format, are streamed with the corresponding is-a-cat attribute value as one record. This can cause a problem because the algorithm might not delimit the images and attributes correctly.

For more information about Pipe mode and how to use it, see Using Your Own Training Algorithms - Input Data Configuration.

Use an Augmented Manifest File (Console)

To complete this procedure, you need:

- The URL of the S3 bucket where you've stored the augmented manifest file.
- To store the data that is listed in the augmented manifest file in an S3 bucket.
- The URL of the S3 bucket where you want to store the output of the job.

To use an augmented manifest file in a training job (console)

2. In the navigation pane, choose Training, then choose Training jobs.
3. Choose Create training job.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. It can have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see Use Amazon SageMaker Built-in Algorithms (p. 56). If you want to use a custom algorithm, make sure that it is compatible with Pipe mode.
6. (Optional) For Resource configuration, either accept the default values or, to reduce computation time, increase the resource consumption.
   a. (Optional) For Instance type, choose the ML compute instance type that you want to use. In most cases, ml.m4.xlarge is sufficient.
   b. For Instance count, use the default, 1.
   c. (Optional) For Additional volume per instance (GB), choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
   a. For Channel name, either accept the default (train) or enter a more meaningful name, such as training-augmented-manifest-file.
   b. For InputMode, choose Pipe.
   c. For S3 data distribution type, choose FullyReplicated. When training incrementally, fully replicating causes each ML compute instance to use a complete copy of the expanded dataset. For neural-based algorithms, such as Neural Topic Model (NTM) Algorithm (p. 173), choose ShardedByS3Key.
   d. If the data specified in the augmented manifest file is uncompressed, set the Compression type to None. If the data is compressed using gzip, set it to Gzip.
   e. (Optional) For Content type, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.
f. For **Record wrapper**, if the dataset specified in the augmented manifest file is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO-formatted file, choose **None**.

g. For **S3 data type**, choose **AugmentedManifestFile**.

h. For **S3 location**, provide the path to the bucket where you stored the augmented manifest file.

i. For **AugmentedManifestFile attribute names**, specify the name of an attribute that you want to use. The attribute name must be present within the augmented manifest file, and is case-sensitive.

j. (Optional) To add more attribute names, choose **Add row** and specify another attribute name for each attribute.

k. (Optional) To adjust the order of attribute names, choose the up or down buttons next to the names. When using an augmented manifest file, the order of the specified attribute names is important.

l. Choose **Done**.

8. For **Output data configuration**, provide the following information:

   a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.

   b. (Optional) You can use your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. For **Encryption key**, provide the key ID or its Amazon Resource Number (ARN). For more information, see KMS-Managed Encryption Keys.

9. (Optional) For **Tags**, add one or more tags to the training job. A tag is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with **Project** as a key and a value that refers to a project that is related to the training job, such as **Home value forecasts**.

10. Choose **Create training job**. Amazon SageMaker creates and runs the training job.

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see Step 2.4: Deploy the Model to Amazon SageMaker (p. 29).

**Use an Augmented Manifest File (API)**

The following shows how to train a model with an augmented manifest file using the Amazon SageMaker high-level Python library:

```python
# Create a model object set to using "Pipe" mode.
model = sagemaker.estimator.Estimator(training_image,
           role,
           train_instance_count=1,
           train_instance_type='ml.p3.2xlarge',
           train_volume_size = 50,
           train_max_run = 360000,
           input_mode = 'Pipe',
           output_path=s3_output_location,
           sagemaker_session=session)

# Create a train data channel with S3_data_type as 'AugmentedManifestFile' and attribute names.
train_data = sagemaker.session.s3_input(your_augmented_manifest_file,
                                        distribution='FullyReplicated',
                                        content_type='image/jpeg',
                                        s3_data_type='AugmentedManifestFile',
                                        attribute_names=['source-ref', 'annotations'])

data_channels = {'train': train_data}
```
# Train a model.
model.fit(inputs=data_channels, logs=True)

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for S3 output path in the Output data configuration field. To deploy the model to get predictions, see Step 2.4: Deploy the Model to Amazon SageMaker (p. 29).
Deploy a Model

For an overview on deploying a model with Amazon SageMaker, see Deploy a Model in Amazon SageMaker (p. 7).

Amazon SageMaker provides features to manage resources and optimize inference performance when deploying machine learning models. For guidance on using inference pipelines, Neo, Elastic inference, automatic scaling, and on best practices, see the following topics.

- For guidance using Amazon SageMaker inference pipeline to manage data processing and real-time predictions or process batch transforms in a series of Docker containers, see Amazon SageMaker Inference Pipelines (p. 312).
- For guidance using Amazon SageMaker Neo to enable machine learning models to train once and run anywhere in the cloud, see Amazon SageMaker Neo (p. 325).
- For guidance using Elastic Inference (EI) to speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models using a GPU instance for your endpoint, see Amazon SageMaker Elastic Inference (EI) (p. 346).
- For guidance on using automatic scaling to dynamically adjust the number of instances provisioned for a production variant in response to changes in your workload, see Automatically Scale Amazon SageMaker Models (p. 356).
- For guidance on best practices to use for model deployment, see Best Practices for Deploying Amazon SageMaker Models (p. 370).

Amazon SageMaker Inference Pipelines

An inference pipeline is an Amazon SageMaker model composed of a linear sequence of two to five containers that process requests for inferences on data. Amazon SageMaker Inference Pipelines enable the definition and deployment of any combination of pretrained Amazon SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. These inference pipelines are fully managed and can combine preprocessing, predictions, and post-processing as part of a data science process. Pipeline model invocations are handled as a sequence of HTTP requests. The initial request is handled by the first container in the pipeline, then the intermediate response is sent as a request to the second container, and so on, for each container in the pipeline with the final response returned to the client. In particular, you can add SparkML Model Serving and scikit-learn containers that reuse the data-transformers developed for training models. The entire assembled inference pipeline itself constitutes an Amazon SageMaker model that can be used for making either real-time predictions or for processing batch transforms directly without any external pre-processing.

When the pipeline model is deployed, the full set of containers installs and runs on each EC2 instance in the endpoint or transform job. Feature processing and inferences are executed with low latency because the containers deployed in an inference pipeline are co-located on the same EC2 instance. You define the containers for a pipeline model using the CreateModel (p. 538). Instead of setting one PrimaryContainer, you now can set multiple Containers that make up the pipeline. You specify the order in which the containers are executed when creating the inference pipeline model. Although the pipeline model is immutable, you can update the inference pipeline by redeploying a new one using the regular UpdateEndpoint (p. 722) process. This provides modularity in the development of your machine...
learning workflows and greater flexibility during experimentation. There are no additional costs for using this feature and you only pay for the instances running behind an endpoint.

### Topics

- Sample Notebooks for Amazon SageMaker Inference Pipelines (p. 313)
- Feature Processing with SparkML and Scikit-learn (p. 313)
- Create a Pipeline Model (p. 314)
- Make Real-time Predictions with an Inference Pipeline (p. 316)
- Run Batch Transforms with an Inference Pipeline (p. 318)
- Use Logs and Metrics to Monitor an Inference Pipeline (p. 319)
- Troubleshoot Inference Pipelines (p. 324)

### Sample Notebooks for Amazon SageMaker Inference Pipelines

For an end-to-end sample that uploads and processes a dataset, trains a model, and builds a pipeline model, see the Inference Pipelines with SparkML and XGBoost on Abalone sample notebook. This notebook demonstrates how you can build your machine learning pipeline by using Spark feature Transformers and the Amazon SageMaker XGBoost algorithm. When the model has been trained, the sample shows how to deploy the pipeline (Feature Transformer and XGBoost) for real-time predictions and also performs a batch transform job against the same pipeline. For additional examples that create and deploy inference pipelines, see the Inference Pipelines with SparkML and BlazingText on DBPedia and Training using SparkML on EMR and hosting on SageMaker sample notebooks. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see Use Notebook Instances (p. 42). Once you have created a notebook instance and opened it, choose the SageMaker Examples tab to see a list of all the Amazon SageMaker samples. The first two inference pipeline notebooks are located in the advanced_functionality folder and the third notebook is in the sagemaker-python-sdk folder. To open a notebook, choose its Use tab and choose Create copy.

### Feature Processing with SparkML and Scikit-learn

You typically have to explore, preprocess, and featurize your data before using it to train a model. (For information on how to do this in Amazon SageMaker, see Explore and Preprocess Data (p. 4)). You can use Spark and scikit-learn preprocessors to transform your data and engineer features before you train your models with Amazon SageMaker built-in algorithms or with your own custom algorithms. You can execute your SparkML jobs with AWS Glue, a serverless ETL service from your Amazon SageMaker notebook. You can also connect to existing EMR clusters to run their SparkML jobs using Amazon EMR. This requires that you create an IAM role that grants permission for making calls from your Amazon SageMaker notebook to AWS Glue. AWS Glue currently only supports Python 2.7.

These jobs are then packaged and serialized with MLeap into MLeap containers that can be added to an inference pipeline. This removes the need for externally managed Spark clusters and allows you to seamlessly scale from a sample of rows to terabytes of data. The same transformers work at both training and inference time, so you no longer need to duplicate preprocessing and feature engineering logic or come up with your own one-time solution to persist these models. The end-to-end solutions that Amazon SageMaker Inference Pipelines provides eliminates the need to maintain outside infrastructure and enables making predictions directly from data inputs.

As part of the SparkML jobs run on AWS Glue, a SparkML pipeline is serialized into MLeap format. It can then be used with the SparkML Model Serving Container with Amazon SageMaker Inference Pipelines. MLeap is a serialization format and execution engine for machine learning pipelines. It supports Spark,
Scikit-learn and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. Bundles can be deserialized back into Spark for batch-mode scoring or the MLeap runtime to power real-time API services.

Scikit-learn jobs can be executed and packaged into containers directly on Amazon SageMaker. For a similar example of Python code for building a scikit-learn featurizer model that trains on Fisher's Iris flower data set and predicts the species of Iris based on morphological measurements, see IRIS Training and Prediction with Sagemaker Scikit-learn.

Create a Pipeline Model

You can create a pipeline model that can be deployed to an endpoint or for a transform job with the Amazon SageMaker console.

Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose Models, and then choose Create models from the Inference group. On the Create model page, complete the Model name, IAM role, and, if needed, VPC fields.

To add information about the remaining containers in the inference pipeline, choose Add container, then choose Next. Specify each of the containers to be added to the Amazon SageMaker inference pipeline. We support up to five containers in a pipeline. They must be listed in the order of execution. Complete the Container input options, Location of inference code image, and, optionally, Location of model artifacts, Container host name, and Environmental variables fields. Confirm that the information for the containers is accurate, and then choose Create model.
The `MyInferencePipelineModel` page summarizes the model input. If you provide the environment variables in a corresponding container definition, Amazon SageMaker shows them in Environment variables.

Make Real-time Predictions with an Inference Pipeline

Trained models can be used in an inference pipeline for making real-time predictions directly without any external pre-processing. As part of the pipeline, customers can choose to use the built-in feature transformers already available in Amazon SageMaker, or they can implement their own transformation logic using just a few lines of scikit-learn or Spark code. This section contains examples for each of these cases.

MLeap is a serialization format and execution engine for machine learning pipelines. It supports Spark, scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. Bundles can be deserialized back into Spark for batch-mode scoring or the MLeap runtime to power real-time API services.
The containers in the pipeline need to listen on the port specified by the SAGEMAKER_BIND_TO_PORT environment variable (instead of 8080). This environment variable is automatically provided to containers when running in an inference pipeline. If this environment variable is not present, containers should default to 8080. Use the following command to add a label to your Dockerfile that indicates that your container complies with this requirement:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

If your container needs to listen on a second port, choose a port in the range specified by the SAGEMAKER_SAFE_PORT_RANGE environment variable. The value is specify an inclusive range in the format "XXXX-YYYY", where XXXX and YYYY are two multi-digit integers. This is also provided automatically when running in a multicontainer pipeline.

**Note**

When your own custom Docker images are used in a pipeline that includes Amazon SageMaker built-in algorithms, we require an Amazon ECR policy. In particular, your repo needs to grant permission for Amazon SageMaker to pull the image. For information on how to implement this ECR policy, see Troubleshoot ECR Permissions for Inference Pipelines (p. 324).

### Create and Deploy an Inference Pipeline Endpoint

The following code shows how to create and deploy a real-time inference pipeline model with SparkML and XGBoost models in series using the Amazon SageMaker SDK.

```python
from sagemaker.model import Model
from sagemaker.pipeline_model import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = 's3://{}/{}/{}'.format(s3_model_bucket, s3_model_key_prefix, 'model.tar.gz')
sparkml_model = SparkMLModel(model_data=sparkml_data)
xgb_model = Model(model_data=xgb_model.model_data, image=training_image)

model_name = 'serial-inference-' + timestamp_prefix
endpoint_name = 'serial-inference-ep-' + timestamp_prefix
sm_model = PipelineModel(name=model_name, role=role, models=[sparkml_model, xgb_model])
sm_model.deploy(initial_instance_count=1, instance_type='ml.c4.xlarge', endpoint_name=endpoint_name)
```

### Request an Inference from an Endpoint

The following code shows how to make predictions by calling a real-time inference endpoint and passing a request payload in JSON format:

```python
from sagemaker.predictor import json_serializer, json_deserializer, RealTimePredictor
from sagemaker.content_types import CONTENT_TYPE_CSV, CONTENT_TYPE_JSON

payload = {
    "input": [
        {
            "name": "Pclass",
            "type": "float",
            "val": "1.0"
        },
        {
            "name": "Embarked",
            "type": "string",
            "val": "Q"
        }
    ]
}
```
Run Batch Transforms with an Inference Pipeline

Batch transform uses a trained model to get inferences on an entire dataset that is stored in Amazon Simple Storage Service (Amazon S3). For information about Amazon SageMaker batch transforms, see Get Inferences for an Entire Dataset with Batch Transform (p. 10). The same inference pipeline model created and deployed to an endpoint for real-time processing can also be used in a batch transform job to process inferences on a full dataset. For a batch transform job in a pipeline, the input data is downloaded from Amazon S3 and sent in one or more HTTP requests to the inference pipeline model. For an example that prepares data for a batch transform, see the Preparing data for Batch Transform section of the ML Pipeline with SparkML and XGBoost - Training and Inference sample notebook.

Note
When you use your own custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you must have an Amazon ECR policy. In particular, your repo needs to grant permission for Amazon SageMaker to pull the image. For information on how to implement this ECR policy, see Troubleshoot ECR Permissions for Inference Pipelines (p. 324).

The following code shows how to run a transform job using the Amazon SageMaker Python SDK. In this example, model_name refers to the inference pipeline that combines SparkML and XGBoost models, which was created in previous examples. The Amazon S3 location specified by input_data_path contains input CSV data to be downloaded and sent to the SparkML model. After the transform job completes, the Amazon S3 location specified by output_data_path will contain the output CSV data returned by the XGBoost model.

```python
input_data_path = 's3://{}//{}/{}'.format(default_bucket, 'key', 'file_name')
output_data_path = 's3://{}//{}/{}'.format(default_bucket, 'key')
transform_job = sagemaker.transformer.Transformer(
    predictor = RealTimePredictor(endpoint=endpoint_name, sagemaker_session=sess,
        serializer=json_serializer,
        content_type=CONTENT_TYPE_JSON, accept=CONTENT_TYPE_CSV)
    print(predictor.predict(payload))
```
Use Logs and Metrics to Monitor an Inference Pipeline

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SageMaker resources. For information about the monitoring tools that Amazon SageMaker provides to watch your resources, report when something is wrong, and take automatic actions when appropriate, see Monitor Amazon SageMaker (p. 425).

Use Metrics to Monitor Multi-container Models

You can monitor multi-container models in Amazon SageMaker Inference Pipelines using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. Amazon SageMaker model training jobs and endpoints write Amazon CloudWatch metrics and logs. For more information on monitoring with Amazon CloudWatch, see Monitor Amazon SageMaker with Amazon CloudWatch (p. 425). The following tables list the metrics and dimensions for Amazon SageMaker.

Endpoint Invocation Metrics

The AWS/SageMaker namespace includes the following request metrics from calls to `InvokeEndpoint` (p. 735).

Metrics are available at a 1-minute frequency.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation4XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocation5XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocations</td>
<td>The number of <code>InvokeEndpoint</code> requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>InvocationsPerInstance</td>
<td>The number of invocations sent to a model, normalized by InstanceCount in each ProductionVariant. $1/\text{numberOfInstances}$ is sent as the value on each request, where numberOfInstances is the number of active instances for the ProductionVariant behind the endpoint at the time of the request.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ModelLatency</td>
<td>The interval of time taken by the model or models to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container. For an inference pipeline endpoint, the ModelLatency is the total interval of time taken by all the containers.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>OverheadLatency</td>
<td>The interval of time added to the time taken to respond to a client request by Amazon SageMaker overheads. This interval is measured from the time Amazon SageMaker receives the request until it returns a response to the client, minus the ModelLatency. Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication or authorization of the request.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ContainerLatency</td>
<td>The interval of time taken by an Inference Pipelines container to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container.</td>
</tr>
</tbody>
</table>

### Dimensions for Endpoint Invocation Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndpointName, VariantName, ContainerName</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
</tbody>
</table>

For an inference pipeline endpoint, per-container latency metrics in your account show as **Endpoint Container Metrics** and **Endpoint Variant Metrics** under the **SageMaker** namespace. **ContainerLatency** shows up only if it's an inference pipeline.
The per-endpoint, per-container latency metrics display the **ContainerName**, **EndpointName**, **VariantName**, and **Metric Name**.

### Training Job, Batch Transform Job, and Endpoint Instance Metrics

The `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs`, and `/aws/sagemaker/Endpoints` namespaces include the following metrics for the training jobs and endpoint instances.

Metrics are available at a 1-minute frequency.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUtilization</td>
<td>The percentage of CPU units that are used by the containers on an instance. The value can range between 0 and 100, and is multiplied by the number of CPUs. For example, if there are four CPUs, <code>CPUUtilization</code> can range from 0% to 400%. For training jobs, the value is the CPU utilization of the algorithm container on the instance. For batch transform jobs, the value is the CPU utilization of the transform container on the instance. For multi-container models, the value is sum of CPU utilization by all containers. For endpoint variants, the value is the sum of the CPU utilization of all provided containers on the instance. Units: Percent</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%. For training jobs, the value is the memory utilization of the algorithm container on the instance. For batch transform jobs, the value is the memory utilization of the transform container on the instance. For multi-container models, the value is sum of memory utilization by all containers.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **GPUUtilization**    | The percentage of GPU units that are used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUUtilization can range from 0% to 400%.  
For training jobs, the value is the GPU utilization of the algorithm container on the instance.  
For batch transform jobs, the value is the GPU utilization of the transform container on the instance.  
For multi-container models, the value is sum of GPU utilization by all containers.  
For endpoint variants, the value is the sum of the GPU utilization of all provided containers on the instance.  
Units: Percent |
| **GPUMemoryUtilization** | The percentage of GPU memory used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUMemoryUtilization can range from 0% to 400%.  
For training jobs, the value is the GPU memory utilization of the algorithm container on the instance.  
For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance.  
For multi-container models, the value is sum of GPU utilization by all containers.  
For endpoint variants, the value is the sum of the GPU memory utilization of all provided containers on the instance.  
Units: Percent |
| **DiskUtilization**   | The percentage of disk space used by the containers on an instance uses. This value can range between 0% and 100%. This metric is not supported for batch transform jobs.  
For training jobs, the value is the disk space utilization of the algorithm container on the instance.  
For endpoint variants, the value is the sum of the disk space utilization of all the provided containers on the instance.  
Units: Percent |
To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to stdout or stderr is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.

## Use Logs to Monitor an Inference Pipeline

The following table lists all of the logs provided by Amazon SageMaker.

### Logs

<table>
<thead>
<tr>
<th>Log Group Name</th>
<th>Log Stream Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aws/sagemaker/TrainingJobs</td>
<td>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</td>
</tr>
<tr>
<td>/aws/sagemaker/Endpoints/ [EndpointName]</td>
<td>[production-variant-name]/[instance-id]</td>
</tr>
<tr>
<td></td>
<td>[production-variant-name]/[instance-id]</td>
</tr>
<tr>
<td></td>
<td>[production-variant-name]/[instance-id]/[container-name provided in SageMaker model] (For Inference Pipelines)</td>
</tr>
<tr>
<td>/aws/sagemaker/NotebookInstances</td>
<td>[notebook-instance-name]/[LifecycleConfigHook]</td>
</tr>
<tr>
<td>/aws/sagemaker/TransformJobs</td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]</td>
</tr>
<tr>
<td></td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]/data-log</td>
</tr>
<tr>
<td></td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]/[container-name provided in SageMaker model] (For Inference Pipelines)</td>
</tr>
</tbody>
</table>
Note
1. The /aws/sagemaker/NotebookInstances log group is created when you create a notebook instance with a lifecycle configuration. For more information, see Step 1.1: (Optional) Customize a Notebook Instance (p. 20).
2. For Inference Pipelines, if you don't provide container names, the platform uses **container-1, container-2**, and so on, corresponding to the order provided in the Amazon SageMaker model.

For more information on Amazon SageMaker logging, see Log Amazon SageMaker Events with Amazon CloudWatch (p. 429).

Troubleshoot Inference Pipelines

Amazon SageMaker provides logs and error message to troubleshoot issues that arise with Inference Pipelines.

Troubleshoot ECR Permissions for Inference Pipelines

When you use your own custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you need an Amazon ECR policy. In particular, your repo needs to grant permission for Amazon SageMaker to pull the image. The ECR policy must add the following permissions:

```json
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "allowSageMakerToPull",
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}
```

Troubleshoot with Inference Pipelines Logs

We publish the customer container logs for endpoints deploying an inference pipeline to CloudWatch under the following path for each container.

/aws/sagemaker/Endpoints/{EndpointName}/{Variant}/{InstanceId}/{ContainerHostname}

For example, if you want logs for the following endpoint:

| EndpointName: MyInferencePipelinesEndpoint |
| Variant: MyInferencePipelinesVariant     |
| InstanceId: i-0179208609ff7e488        |
| ContainerHostname: MyContainerName1 and MyContainerName2 |

The logs will be in:

logGroup: /aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint
To see the logs, filter on `MyInferencePipelinesEndpoint` in the CloudWatch Log Groups console.

To see the log streams, choose `MyInferencePipelinesEndpoint` in the CloudWatch Log Groups console and then Search Log Group.

For a listing on all of the logs provided by Amazon SageMaker, see Use Logs and Metrics to Monitor an Inference Pipeline (p. 319).

Troubleshoot with Inference Pipelines Error Messages

We have updated the error messages to indicate which containers failed.

- During `InvokeEndpoint`, if the error is a ModelError (error code 424), we indicate which container failed instead of generically indicating the error is from *model*.
- During `InvokeEndpoint`, if the request payload (the response from the previous container) exceeds the limit of 5 MB, we throw as `ModelError` with a detailed error message such as:

  Received response from MyContainerName1 with status code 200. However, the request payload from MyContainerName1 to MyContainerName2 is 6000000 bytes, which has exceeded the maximum limit of 5 MB. See https://us-west-2.console.aws.amazon.com/cloudwatch/home?region=us-west-2#logEventViewer:group=/aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint in account 123456789012 for more information.

- While running the `CreateEndpoint` (p. 522), if a container fails the ping health check, Amazon SageMaker returns a `ClientError` and indicates all the containers that failed the ping check in the last health check attempt instead of just reporting on the `PrimaryContainer`.

Amazon SageMaker Neo

Neo is a new capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

Ordinarily, optimizing machine learning models for inference on multiple platforms is extremely difficult because you need to hand-tune models for the specific hardware and software configuration of each platform. If you want to get optimal performance for a given workload, you need to know the hardware
architecture, instruction set, memory access patterns, and input data shapes among other factors. For traditional software development, tools such as compilers and profilers simplify the process. For machine learning, most tools are specific to the framework or to the hardware. This forces you into a manual trial-and-error process that is unreliable and unproductive.

Neo eliminates the time and effort required to do this by automatically optimizing TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models for deployment on ARM, Intel, and Nvidia processors. Neo consists of a compiler and a runtime. First, the Neo compilation API reads models exported from various frameworks. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimizations. Then it generates binary code for the optimized operations, writes them to a shared object library, and saves the model definition and parameters into separate files. Neo also provides a runtime for each target platform that loads and executes the compiled model.

You can create a Neo compilation job from either the Amazon SageMaker console, AWS Command Line Interface (AWS CLI), Python notebook, or the Amazon SageMaker SDK. With a few CLI commands, an API invocation, or a few clicks, you can convert a model for your chosen platform. You can deploy the model to an Amazon SageMaker endpoint or on an AWS IoT Greengrass device quickly. Amazon SageMaker provides Neo container images for Amazon SageMaker XGBoost and Image Classification models, and supports Amazon SageMaker-compatible containers for your own compiled models.

Note
Neo currently supports image classification models exported as frozen graphs from TensorFlow, MXNet, PyTorch, and XGBoost models. Neo is available in the following AWS Regions:

- US East (N. Virginia), us-east-1
- US West (Oregon), us-west-2
- EU (Ireland), eu-west-1

Topics
- Amazon SageMaker Neo Sample Notebooks (p. 326)
- Use Neo to Compile a Model (p. 327)
- Deploy a Model (p. 331)
- Request Inferences from a Deployed Service (p. 340)
- Troubleshooting Neo Compilation Errors (p. 340)

Amazon SageMaker Neo Sample Notebooks

For sample notebooks that uses Amazon SageMaker Neo to train, compile, optimize, and deploy machine learning models to make inferences, see:

- MNIST Training, Compilation and Deployment with MXNet Module
- MNIST Training, Compilation and Deployment with Tensorflow Module
- Model Optimization with an Image Classification Example
- Model Optimization with XGBoost Example

For instructions on how to run these example notebooks in Amazon SageMaker, see Use Example Notebooks (p. 47). If you need instructions on how to create a notebook instance to run these examples, see Amazon SageMaker, see Use Notebook Instances (p. 42). To navigate to the relevant example in your notebook instance, choose the Amazon SageMaker Examples tab to see a list of all of the Amazon SageMaker samples. To open a notebook, choose its Use tab, then choose Create copy.
Use Neo to Compile a Model

This section show how to create, describe, stop, and list compilation jobs. There are three options available in Neo for managing the compilation jobs for machine learning models: Using the Neo CLI, the Amazon SageMaker console, or the Amazon SageMaker SDK.

Topics

- Compile a Model (API) (p. 327)
- Compile a Model (Console) (p. 328)
- Compile a Model (Amazon SageMaker SDK) (p. 331)

Compile a Model (API)

This section shows how to manage compilation jobs for machine learning models. You can create, describe, stop, and list compilation jobs.

Create a Compilation Job

As shown in the following JSON file, you specify the data input format, the S3 bucket where you stored your model, the S3 bucket where you want to write the compiled model, and the target hardware:

```
job.json
{
    "CompilationJobName": "job002",
    "RoleArn": "arn:aws:iam::<your-account>:role/service-role/AmazonSageMaker-
    ExecutionRole-20180829T140091",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/train",
        "DataInputConfig": "{\"data\": [1,3,1024,1024]}",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/compile",
        "TargetDevice": "ml_c5"
    },
    "StoppingCondition": {"MaxRuntimeInSeconds": 300}
}
```

```
aws sagemaker create-compilation-job \
--cli-input-json file://job.json \
--region us-west-2
# You should get CompilationJobArn
```

Describe a Compilation Job

```
aws sagemaker describe-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2
```

Stop a Compilation Job

```
aws sagemaker stop-compilation-job \
```
```
--compilation-job-name $JOB_NM \
--region us-west-2

# There is no output for compilation-job operation
```

**List a Compilation Job**

```
aws sagemaker list-compilation-jobs \
--region us-west-2
```

**Compile a Model (Console)**

You can create a Neo compilation job in the Amazon SageMaker console. In the Amazon SageMaker console, choose **Compilation jobs**, and then choose **Create compilation job**.

On the **Create compilation job** page, for **Job name**, enter a name. Then select an **IAM role**.

If you don’t have an IAM role, choose **Create a new role**.
On the **Create an IAM role** page, choose **Any S3 bucket**, and choose **Create role**.

In the **Input configuration** section, for **Location of model artifacts**, enter the path of the S3 bucket that contains your model artifacts. For **Data input configuration**, enter the JSON string that specifies how many data matrix inputs you and the shape of each data matrices. For **Machine learning framework**, choose the framework.
In the **Output configuration** section, for **S3 Output location**, enter the path to the S3 bucket or folder where you want to store the model. For **Target device**, choose which device you want to deploy your model to, and choose **Create job**.

Check the status of the compilation job when started.

Check the status of the compilation job when completed.
Compile a Model (Amazon SageMaker SDK)

Follow the steps described in the Neo MXNet Part 1 sample to produce a machine learning model train using Amazon SageMaker. Then you can use Neo to further optimize the model with the following code:

```python
output_path = '/'.join(mnist_estimator.output_path.split('/')[-1])
compiled_model = mnist_estimator.compile_model(target_instance_family='ml_c5', input_shape={'data':[1, 784]}, role=role, output_path=output_path)
```

This code compiles the model and saves the optimized model in `output_path`. Sample notebooks of using SDK are provided in the Amazon SageMaker Neo Sample Notebooks (p. 326) section.

Deploy a Model

You can deploy the compact module to performance-critical cloud services with Amazon SageMaker Hosting Services or to resource-constrained edge devices with AWS IoT Greengrass.

Topics

- Deploy a Model Compiled with Neo with Hosting Services (p. 331)
- Deploy a Model Compiled with Neo (AWS IoT Greengrass) (p. 339)

Deploy a Model Compiled with Neo with Hosting Services

To deploy a Neo-compiled model to an HTTPS endpoint, you must configure and create the endpoint for the model using Amazon SageMaker hosting services. Currently developers can use Amazon SageMaker APIs to deploy modules on to ml.c5, ml.c4, ml.m5, ml.m4, ml.p3, and ml.p2 instances.

When you deploy a compiled model, you need to use the same instance for the target that you used for compilation. This creates an Amazon SageMaker endpoint that you can use to perform inferences. There are three options available for deploying Neo-compiled models:

Topics

- Deploy a Model Compiled with Neo (AWS CLI) (p. 332)
- Deploy a Model Compiled with Neo (Console) (p. 333)
- Deploy a Model Compiled with Neo (Amazon SageMaker SDK) (p. 339)
Deploy Models

Deploy a Model Compiled with Neo (AWS CLI)

The deployment of a Neo-compiled model with the CLI has three steps.

Topics
- Create a Model That Was Compiled with Neo (AWS CLI) (p. 332)
- Create the Endpoint Configuration (AWS CLI) (p. 333)
- Create an Endpoint (AWS CLI) (p. 333)

Create a Model That Was Compiled with Neo (AWS CLI)

For the full syntax of the `CreateModel` API, see `CreateModel` (p. 538).

For Neo-compiled models, use one of the following values for `PrimaryContainer/ContainerHostname`, depending on your region and applications:

- **Amazon SageMaker Image Classification**
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest`
  - `78557368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest`

- **Amazon SageMaker XGBoost**
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest`
  - `78557368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest`

- **TensorFlow** The TensorFlow version used must be in TensorFlow SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3`

- **MXNet** The MXNet version used must be in MXNet SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3`

- **Pytorch** The Pytorch version used must be in Pytorch SageMaker Estimators list.
  - `301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
  - `802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3`
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3

Also, if you are using **TensorFlow**, **Pytorch**, or **MXNet**, add the following key-value pair to **PrimaryContainer/Environment**:

```
"Environment": {
"SAGEMAKER_SUBMIT_DIRECTORY" : "[Full S3 path for *.tar.gz file containing the training script]"
}
```

The script must be packaged as a *.tar.gz file. The *.tar.gz file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

- **neo_preprocess(payload, content_type)**: Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
- **neo_postprocess(result)**: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the Amazon SageMaker Neo Sample Notebooks (p. 326) for examples using these functions.

**Create the Endpoint Configuration (AWS CLI)**

For the full syntax of the `CreateEndpointConfig` API, see `CreateEndpointConfig` (p. 525). You must specify the correct instance type in `ProductionVariants/InstanceType`. It is imperative that this value matches the instance type specified in your compilation job.

**Create an Endpoint (AWS CLI)**

For the full syntax of the `CreateEndpoint` API, see `CreateEndpoint` (p. 522).

**Deploy a Model Compiled with Neo (Console)**

You can create a Neo endpoint in the Amazon SageMaker console. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

Choose **Models**, and then choose **Create models** from the **Inference** group. On the **Create model** page, complete the **Model name**, **IAM role**, and, if needed, **VPC** fields.
To add information about the container used to deploy your model, choose Add container, then choose Next. Complete the Container input options, Location of inference code image, and Location of model artifacts, and optionally, Container host name, and Environmental variables fields.
To deploy Neo-compiled models, choose the following:

- **Container input options**: Provide model artifacts and inference image.
- **Location of inference code image**: Choose one of the following images, depending on the region and kind of application:
  - **Amazon SageMaker Image Classification**
    - 301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest
    - 78557368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest
    - 007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest
    - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest
  - **Amazon SageMaker XGBoost**
    - 301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest
    - 78557368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest
    - 007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest
    - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest
  - **TensorFlow**: The TensorFlow version used must be in TensorFlow SageMaker Estimators list.
• 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3
• 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow: [tensorflow-version]-[cpu/gpu]-py3

• MXNet The MXNet version used must be in MXNet SageMaker Estimators list.
  • 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
  • 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
  • 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
  • 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3

• Pytorch The Pytorch version used must be in Pytorch SageMaker Estimators list.
  • 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
  • 78557368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
  • 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3
  • 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch: [pytorch-version]-[cpu/gpu]-py3

• Location of model artifact: the full S3 path of the compiled model artifact generated by the Neo compilation API.

• Environmental variables:
  • Omit this field for SageMaker Image Classification and SageMaker XGBoost.
  • For TensorFlow, Pytorch, and MXNet, specify the environment variable SAGEMAKER_SUBMIT_DIRECTORY as the full S3 path that contains the training script.

The script must be packaged as a *.tar.gz file. The *.tar.gz file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

• neo_preprocess(payload, content_type): Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
• neo_postprocess(result): Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the Amazon SageMaker Neo Sample Notebooks (p. 326) for examples using these functions.

Confirm that the information for the containers is accurate, and then choose Create model. This takes you to the create model landing page. Select the Create endpoint button there.
In **Create and configure endpoint** diagram, specify the **Endpoint name**. Choose **Create a new endpoint configuration** in **Attach endpoint configuration**.

In **New endpoint configuration** page, specify the **Endpoint configuration name**.
Then press **Edit** next to the name of the model and specify the correct **Instance type** on the **Edit Production Variant** page. It is imperative that the **Instance type** value match the one specified in your compilation job.
When you're done click **Save**, then click **Create endpoint configuration** on the **New endpoint configuration** page, and then click **Create endpoint**.

**Deploy a Model Compiled with Neo (Amazon SageMaker SDK)**

The object handle for the compiled model supplies the `deploy` function, which allows you to create an endpoint to serve inference requests. The function lets you set the number and type of instances that are used for the endpoint. You must choose an instance for which you have compiled your model. For example, in the job compiled in Compile a Model (Amazon SageMaker SDK) (p. 331) section, this is `ml_c5`. The Neo API uses a special runtime, the **Neo runtime**, to run Neo-optimized models.

```python
predictor = compiled_model.deploy(initial_instance_count = 1, instance_type = 'ml.c5.4xlarge')
```

After the command is done, the name of the newly created endpoint is printed in the jupyter notebook.

**Deploy a Model Compiled with Neo (AWS IoT Greengrass)**

**AWS IoT Greengrass** extends cloud capabilities to local devices. It enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. With AWS IoT Greengrass, you can perform machine learning inference at the edge on locally generated data using cloud-trained models. Currently, you can deploy models on to all AWS IoT Greengrass devices based on ARM® Cortex-A™, Intel® Atom™, and Nvidia® Jetson™ series processors. For more information on deploying a Lambda inference application to perform machine learning inferences with AWS IoT Greengrass, see Perform Machine Learning Inference.
Request Inferences from a Deployed Service

If you have followed instructions in Deploy a Model Compiled with Neo with Hosting Services (p. 331), you should have an Amazon SageMaker endpoint set up and running. You can now submit inference requests using Boto3 client. Here is an example of sending an image for inference:

```python
import boto3
import json
endpoint = '<insert name of your endpoint here>'
runtime = boto3.Session().client('sagemaker-runtime')

# Read image into memory
with open(image, 'rb') as f:
    payload = f.read()
# Send image via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='application/x-image', Body=payload)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

For XGBoost application, you should submit a CSV text instead:

```python
import boto3
import json
endpoint = '<insert your endpoint here>'
runtime = boto3.Session().client('sagemaker-runtime')
csv_text = '1,-1.0,1.0,1.5,2.6'
# Send CSV text via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv', Body=csv_text)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

Note that BYOM allows for a custom content type. For more information, see InvokeEndpoint (p. 735).

Troubleshooting Neo Compilation Errors

This section contains information about how to understand and prevent common errors, the error messages they generate, and guidance on how to resolve these errors. It also contains lists of the frameworks and the operations in each of those frameworks that Neo supports.

**Topics**
- Prevent Neo Input Errors (p. 340)
- Neo Error Messages (p. 344)
- Resolve Neo Errors (p. 346)

**Prevent Neo Input Errors**

Some of the most common errors are due to invalid inputs. This section contains information arranged in question and answer form to help you avoid these errors.

*Which frameworks does Neo support?*
Which operators does Amazon SageMaker Neo support for these frameworks?

The following table lists the supported operations for each framework.

<table>
<thead>
<tr>
<th>MXNet</th>
<th>TensorFlow</th>
<th>PyTorch/ONNX</th>
</tr>
</thead>
<tbody>
<tr>
<td>'_add_scalar'</td>
<td>'Add'</td>
<td>'Abs'</td>
</tr>
<tr>
<td>'_add_symbol'</td>
<td>'ArgMax'</td>
<td>'Add'</td>
</tr>
<tr>
<td>'_contrib_MultiBoxDetection'</td>
<td>'ArgMin'</td>
<td>'ArgMax'</td>
</tr>
<tr>
<td>'_contrib_MultiBoxPrior'</td>
<td>'AvgPool'</td>
<td>'ArgMin'</td>
</tr>
<tr>
<td>'_copy'</td>
<td>'BatchNormWithGlobalNormalization'</td>
<td>'AveragePool'</td>
</tr>
<tr>
<td>'_div_scalar'</td>
<td>'BiasAdd'</td>
<td>'BatchNormalization'</td>
</tr>
<tr>
<td>'_div_symbol'</td>
<td>'Cast'</td>
<td>'Cast'</td>
</tr>
<tr>
<td>'_minus_scalar'</td>
<td>'Ceil'</td>
<td>'Ceil'</td>
</tr>
<tr>
<td>'_mul_symbol'</td>
<td>'CheckNumerics'</td>
<td>'Clip'</td>
</tr>
<tr>
<td>'_pow_scalar'</td>
<td>'Conv2D'</td>
<td>'ConstantFill'</td>
</tr>
<tr>
<td>'_rdiv_scalar'</td>
<td>'DecodeJpeg'</td>
<td>'Conv'</td>
</tr>
<tr>
<td>'_rminus_scalar'</td>
<td>'Elu'</td>
<td>'Dropout'</td>
</tr>
<tr>
<td>'_rpow_scalar'</td>
<td>'Equal'</td>
<td>'Dropout'</td>
</tr>
<tr>
<td>'_rsub_scalar'</td>
<td>'ExpandDims'</td>
<td>'Elu'</td>
</tr>
<tr>
<td>'_sub_scalar'</td>
<td>'Fill'</td>
<td>'Exp'</td>
</tr>
<tr>
<td>'_sub_symbol'</td>
<td>'Floor'</td>
<td>'FC'</td>
</tr>
<tr>
<td>'Activation'</td>
<td>'FusedBatchNorm'</td>
<td>'Flatten'</td>
</tr>
<tr>
<td>'add_n'</td>
<td>'FusedBatchNormV2'</td>
<td>'Floor'</td>
</tr>
<tr>
<td>'argmax'</td>
<td>'GatherV2'</td>
<td>'Gather'</td>
</tr>
<tr>
<td>'BatchNorm'</td>
<td>'Greater'</td>
<td>'Gemm'</td>
</tr>
<tr>
<td>'BatchNorm_v1'</td>
<td>'GreaterEqual'</td>
<td>'GlobalAveragePool'</td>
</tr>
<tr>
<td>MXNet</td>
<td>TensorFlow</td>
<td>PyTorch/ONNX</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>'broadcast_add'</td>
<td>'Identity'</td>
<td>'GlobalMaxPool'</td>
</tr>
<tr>
<td>'broadcast_div'</td>
<td>'LeakyRelu'</td>
<td>'HardSigmoid'</td>
</tr>
<tr>
<td>'broadcast_mul'</td>
<td>'Less'</td>
<td>'Identity'</td>
</tr>
<tr>
<td>'broadcast_sub'</td>
<td>'LessEqual'</td>
<td>'ImageScaler'</td>
</tr>
<tr>
<td>'broadcast_to'</td>
<td>'LRN'</td>
<td>'LeakyRelu'</td>
</tr>
<tr>
<td>'cast'</td>
<td>'MatMul'</td>
<td>'Log'</td>
</tr>
<tr>
<td>'Cast'</td>
<td>'Maximum'</td>
<td>'LogSoftmax'</td>
</tr>
<tr>
<td>'clip'</td>
<td>'MaxPool'</td>
<td>'LRN'</td>
</tr>
<tr>
<td>'Concat'</td>
<td>'Mean'</td>
<td>'MatMul'</td>
</tr>
<tr>
<td>'concat'</td>
<td>'Minimum'</td>
<td>'Max'</td>
</tr>
<tr>
<td>'Convolution'</td>
<td>'Mul'</td>
<td>'MaxPool'</td>
</tr>
<tr>
<td>'Convolution_v1'</td>
<td>'NotEqual'</td>
<td>'Mean'</td>
</tr>
<tr>
<td>'Crop'</td>
<td>'Pack'</td>
<td>'Min'</td>
</tr>
<tr>
<td>'Deconvolution'</td>
<td>'Pad'</td>
<td>'Mul'</td>
</tr>
<tr>
<td>'Dropout'</td>
<td>'PadV2'</td>
<td>'Neg'</td>
</tr>
<tr>
<td>'elemwise_add'</td>
<td>'Range'</td>
<td>'Pad'</td>
</tr>
<tr>
<td>'elemwise_div'</td>
<td>'Rank'</td>
<td>'ParametricSoftplus'</td>
</tr>
<tr>
<td>'elemwise_mul'</td>
<td>'Relu'</td>
<td>'Pow'</td>
</tr>
<tr>
<td>'elemwise_sub'</td>
<td>'Relu6'</td>
<td>'PRelu'</td>
</tr>
<tr>
<td>'exp'</td>
<td>'Reshape'</td>
<td>'Reciprocal'</td>
</tr>
<tr>
<td>'expand_dims'</td>
<td>'ResizeBilinear'</td>
<td>'ReduceMax'</td>
</tr>
<tr>
<td>'flatten'</td>
<td>'Rsqrt'</td>
<td>'ReduceMean'</td>
</tr>
<tr>
<td>'Flatten'</td>
<td>'Selu'</td>
<td>'ReduceMin'</td>
</tr>
<tr>
<td>'FullyConnected'</td>
<td>'Shape'</td>
<td>'ReduceProd'</td>
</tr>
<tr>
<td>'LeakyReLU'</td>
<td>'Sigmoid'</td>
<td>'ReduceSum'</td>
</tr>
<tr>
<td>'LinearRegressionOutput'</td>
<td>'Softmax'</td>
<td>'Relu'</td>
</tr>
<tr>
<td>'log'</td>
<td>'Square'</td>
<td>'Reshape'</td>
</tr>
<tr>
<td>'log_softmax'</td>
<td>'Squeeze'</td>
<td>'Scale'</td>
</tr>
<tr>
<td>'LRN'</td>
<td>'StridedSlice'</td>
<td>'ScaledTanh'</td>
</tr>
<tr>
<td>'max'</td>
<td>'Sub'</td>
<td>'Selu'</td>
</tr>
<tr>
<td>'max_axis'</td>
<td>'Sum'</td>
<td>'Shape'</td>
</tr>
</tbody>
</table>
### Which model architectures does Neo support?

Neo supports image classification models.

### Which model format files does Neo read in?

The file needs to be formatted as a tar.gz file that includes additional files that depend on the type of framework.

- **TensorFlow**: Neo supports saved models and frozen models.
  - For **saved models**, Neo expects one .pb or one .pbtxt file and a variables directory that contains variables.
  - For **frozen models**, Neo expect only one .pb or .pbtxt file.
- **PyTorch**: Neo expects one .pth file containing the model definition.
Amazon SageMaker Developer Guide
Troubleshoot Errors

• MXNET: Neo expects one symbol ﬁle (.json) and one parameter ﬁle (.params).
• XGBoost: Neo expects one XGBoost model ﬁle (.model) where the number of nodes in a tree can't
exceed 2^31.
• ONNX: Neo expects one .onnx ﬁle.
What input data shapes does Neo expect?
Neo expects the name and shape of the expected data inputs for your trained model with a JSON
dictionary form or list form. The data inputs are framework speciﬁc.
• TensorFlow: You must specify the name and shape (NHWC format) of the expected data inputs using
a dictionary format for your trained model. The dictionary formats required for the console and CLI are
diﬀerent.
• Examples for one input:
• If using the console, {"input":[1,1024,1024,3]}
• If using the CLI, {\"input\":[1,1024,1024,3]}
• Examples for two inputs:
• If using the console, {"data1": [1,28,28,1], "data2":[1,28,28,1]}
• If using the CLI, {\"data1\": [1,28,28,1], \"data2\":[1,28,28,1]}
• MXNET/ONNX: You must specify the name and shape (NCHW format) of the expected data inputs
in order using a dictionary format for your trained model. The dictionary formats required for the
console and CLI are diﬀerent.
• Examples for one input:
• If using the console, {"data":[1,3,1024,1024]}
• If using the CLI, {\"data\":[1,3,1024,1024]}
• Examples for two inputs:
• If using the console, {"var1": [1,1,28,28], "var2":[1,1,28,28]}
• If using the CLI, {\"var1\": [1,1,28,28], \"var2\":[1,1,28,28]}
• PyTorch: You can either specify the name and shape (NCHW format) of expected data inputs in order
using a dictionary format for your trained model or you can specify the shape only using a list format.
The dictionary formats required for the console and CLI are diﬀerent. The list formats for the console
and CLI are the same.
• Examples for one input in dictionary format:
• If using the console, {"input0":[1,3,224,224]}
• If using the CLI, {\"input0\":[1,3,224,224]}
• Example for one input in list format: [[1,3,224,224]]
• Examples for two inputs in dictionary format:
• If using the console, {"input0":[1,3,224,224], "input1":[1,3,224,224]}
• If using the CLI, {\"input0\":[1,3,224,224], \"input1\":[1,3,224,224]}
• Example for two inputs in list format: [[1,3,224,224], [1,3,224,224]]
• XGBOOST: input data name and shape are not needed.

Neo Error Messages
This section lists and classiﬁes Neo errors and error messages.

Neo Error Messages
This list catalogs the user and system error messages you can receive from Neo deployments.
344


• **User error messages**
  • **Client permission error**: Neo passes the errors for these straight through from the dependent service.
    
    *Access Denied* when calling *sts:AssumeRole*

    *Any 400 error* when calling S3 to download or upload a client model.

  • **PassRole error**
  • **Load error**: Keywords in error messages, 'InputConfiguration','ModelSizeTooBig'.
    
    *Load Error: InputConfiguration*: Exactly one *.xxx* file is allowed for {yyy} model.

    *Load Error: ModelSizeTooBig*: number of nodes in a tree can't exceed \(2^{31}\)

  • **Compilation error**: Keywords in error messages, 'OperatorNotImplemented','OperatorAttributeNotImplemented','OperatorAttributeRequired','OperatorAttributeValueNotValid'.
    
    *OperatorNotImplemented*: (xxx) is not supported.

    *OperatorAttributeNotImplemented*: (xxx) is not supported in {yyy}.

    *OperatorAttributeRequired*: Required attribute {xxx} not found in {yyy}.

    *OperatorAttributeValueNotValid*: The value of attribute {xxx} in operator {yyy} cannot be negative.

  • **Any Malformed Input Errors**

  • **System error messages**
    
    For system errors, Neo shows only one error message similar to the following: There was an unexpected error during compilation, check your inputs and try again in a few minutes.

    This covers all unexpected errors and errors that are not user errors.

**Neo Error Classifications**

This list classifies the *user errors* you can receive from Neo. These include access and permission errors and load errors for each of the supported frameworks. All other errors are *system errors*.

• **Client permission error**: Neo passes the errors for these straight through from the dependent service.

  *Access Denied* when calling *sts:AssumeRole*

  *Any 400 error* when calling Amazon S3 to download or upload a client model.

  • **PassRole error**
  • **Load error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, check whether the tarball contains the necessary files for compilation. The checking criteria is framework-specific:
    
    • **TensorFlow**: Expects only protobuf file (*.pb or *.pbtxt). For *saved models*, expects one variables folder.
    
    • **Pytorch**: Expect only one pytorch file (*.pth).
    
    • **MXNET**: Expect only one symbol file (*.json) and one parameter file (*.params).
    
    • **XGBoost**: Expect only one XGBoost model file (*.model). The input model has size limitation.

  • **Compilation error**: Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, and that the tarball contains necessary files for compilation. The checking criteria is:
    
    • **OperatorNotImplemented**: An operator has not been implemented.
    
    • **OperatorAttributeNotImplemented**: The attribute in the specified operator has not been implemented.
• **OperatorAttributeRequired**: An attribute is required for an internal symbol graph, but it is not listed in the user input model graph.
• **OperatorAttributeValueNotValid**: The value of the attribute in the specific operator is not valid.

### Resolve Neo Errors

This section provides guidance on troubleshooting common issues with Neo. These include permission, load, compilation, and system errors and errors involving invalid inputs and unsupported operations.

• **Catalog of Known Issues**:
  • If you see **Client Permission Error**, review the set up documentation and make sure that you have correctly granted the permissions that are failing.
  • If you see **Load Error**, check the model format files that Neo expects for different frameworks.
  • If you see **Compilation Error**, check and address the details error message in your input model graph.
  • If you see **System Error**, try again in a few minutes. If that fails, file a ticket.
• **Lack of Roles and Permissions**: Review the set up documentation and make sure that you have correctly granted the permissions that are failing.
• **Invalid API and Console Inputs**: Fix your input as described in the validation error.
• **Unsupported Operators**:
  • Check the failure reason where Neo has listed all unsupported operators with the keyword ‘OperatorNotImplemented’.
  • For example: Compilation Error: OperatorNotImplemented: The following operators are not implemented: {'_sample_multinomial', 'RNN'}
  • Remove the unsupported operators from your input model graph and test it again.

### Amazon SageMaker Elastic Inference (EI)

By using Amazon Elastic Inference (EI), you can speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models, but at a fraction of the cost of using a GPU instance for your endpoint. EI allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance.

Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type, and then add that model as a production variant to an endpoint configuration that you use to deploy a hosted endpoint. You can also add an EI accelerator to a Amazon SageMaker notebook instance so that you can test and evaluate inference performance when you are building your models.

Elastic Inference is supported in EI-enabled versions of TensorFlow and MXNet. To use any other deep learning framework, export your model by using ONNX, and then import your model into MXNet. You can then use your model with EI as an MXNet model. For information about importing an ONNX model into MXNet, see [https://mxnet.incubator.apache.org/tutorials/onnx/super_resolution.html](https://mxnet.incubator.apache.org/tutorials/onnx/super_resolution.html).

### Topics

• **How EI Works** (p. 347)
• **Choose an EI Accelerator Type** (p. 347)
• **Use EI in a Amazon SageMaker Notebook Instance** (p. 347)
• **Use EI on a Hosted Endpoint** (p. 347)
• **Frameworks that Support EI** (p. 348)
• **Use EI with Amazon SageMaker Built-in Algorithms** (p. 348)
• **EI Sample Notebooks** (p. 348)
How EI Works

EI accelerators are network attached devices that work along with EC2 instances in your endpoint to accelerate your inference calls. When your model is deployed as an endpoint, ML frameworks use a combination of EC2 instance and accelerator resources to execute inference calls.

The following EI accelerator types are available. You can configure your endpoints or notebook instances with any EI accelerator type.

In the table, the throughput in teraflops (TFLOPS) is listed for both single-precision floating-point (F32) and half-precision floating-point (F16) operations. The memory in GB is also listed.

<table>
<thead>
<tr>
<th>Accelerator Type</th>
<th>F32 Throughput in TFLOPS</th>
<th>F16 Throughput in TFLOPS</th>
<th>Memory in GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ml.eia1.medium</td>
<td>1</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>mld.eia1.large</td>
<td>2</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>mld.eia1.xlarge</td>
<td>4</td>
<td>32</td>
<td>4</td>
</tr>
</tbody>
</table>

Choose an EI Accelerator Type

Consider the following factors when choosing an accelerator type for a hosted model:

- Models, input tensors and batch sizes influence the amount of accelerator memory you need. Start with an accelerator type that provides at least as much memory as the file size of your trained model.
- Demands on CPU compute resources, GPU-based acceleration, and CPU memory vary significantly between different kinds of deep learning models. The latency and throughput requirements of the application also determine the amount of compute and acceleration you need. Thoroughly test different configurations of instance types and EI accelerator sizes to make sure you choose the configuration that best fits the performance needs of your application.

Use EI in a Amazon SageMaker Notebook Instance

Typically, you build and test machine learning models in a Amazon SageMaker notebook before you deploy them for production. You can attach EI to your notebook instance when you create the notebook instance. You can set up an endpoint that is hosted locally on the notebook instance by using the local mode supported by TensorFlow and MXNet estimators and models in the Amazon SageMaker Python SDK to test inference performance. For instructions on how to attach EI to a notebook instance and set up a local endpoint for inference, see Attach EI to a Notebook Instance (p. 352).

Use EI on a Hosted Endpoint

When you are ready to deploy your model for production to provide inferences, you create a Amazon SageMaker hosted endpoint. You can attach EI to the instance where your endpoint is hosted to increase its performance at providing inferences. For instructions on how to attach EI to a hosted endpoint instance, see Use EI on Amazon SageMaker Hosted Endpoints (p. 354).
Frameworks that Support EI

EI is designed to be used with AWS enhanced versions of TensorFlow or Apache MXNet machine learning frameworks. These enhanced versions of the frameworks are automatically built into containers when you use the Amazon SageMaker Python SDK, or you can download them as binary files and import them in your own Docker containers. For information about downloading the binary for TensorFlow, see Using TensorFlow Models with Elastic Inference in the Amazon Elastic Compute Cloud User Guide for Linux Instances. For information about downloading the binary for Apache MXNet, see Using MXNet Models with Amazon EI.

To use EI in a hosted endpoint, you can use any of the following, depending on your needs.

- SageMaker Python SDK TensorFlow - if you want to use TensorFlow and you don't need to build a custom container.
- SageMaker Python SDK MXNet - if you want to use MXNet and you don't need to build a custom container.
- The low-level AWS Amazon SageMaker SDK for Python (Boto 3) - if you need to build a custom container.

Typically, you don't need to create a custom container unless your model is very complex and requires extensions to a framework that the Amazon SageMaker pre-built containers do not support.

Use EI with Amazon SageMaker Built-in Algorithms

Currently, the Image Classification Algorithm (p. 106) and Object Detection Algorithm (p. 192) built-in algorithms support EI. For an example that uses the Image Classification algorithm with EI, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/imageclassification_caltech/Image-classification-fulltraining.ipynb.

EI Sample Notebooks

The following Sample notebooks provide examples of using EI in Amazon SageMaker:


Set Up to Use EI

Use the instructions in this topic only if one of the following applies to you:

- You want to use a customized role or permission policy.
You want to use a VPC for your hosted model or notebook instance.

**Note**
If you already have an execution role that has the AmazonSageMakerFullAccess managed policy attached (this is true for any IAM role that you create when you create a notebook instance, training job, or model in the console) and you are not connecting to an EI model or notebook instance in a VPC, you do not need to make any of these changes to use EI in Amazon SageMaker.

**Topics**
- Set Up Required Permissions (p. 349)
- Use a Custom VPC to Connect to EI (p. 351)

**Set Up Required Permissions**
To use EI in Amazon SageMaker, the role that you use to open a notebook instance or create a deployable model must have a policy with the required permissions attached. You can attach the AmazonSageMakerFullAccess managed policy, which contains the required permissions, to the role, or you can add a custom policy that has the required permissions. For information about creating an IAM role, see Creating a Role for an AWS Service (Console) in the AWS Identity and Access Management User Guide. For information about attaching a policy to a role, see Adding and Removing IAM Policies.

Add these permissions specifically for connecting EI in an IAM policy:

```
{
    "Effect": "Allow",
    "Action": [
        "elastic-inference:Connect",
        "ec2:DescribeVpcEndpoints"
    ],
    "Resource": "*"
}
```

The following IAM policy is the complete list of required permissions to use EI in Amazon SageMaker:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elastic-inference:Connect",
                "ec2:DescribeVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        }
    ]
}
```
"ecr:BatchGetImage",
"ecr:BatchCheckLayerAvailability",
"cloudwatch:PutMetricData",
"cloudwatch:PutMetricAlarm",
"cloudwatch:DescribeAlarms",
"cloudwatch:DeleteAlarms",
"ec2:CreateNetworkInterface",
"ec2:CreateNetworkInterfacePermission",
"ec2:DeleteNetworkInterface",
"ec2:DeleteNetworkInterfacePermission",
"ec2:DescribeNetworkInterfaces",
"ec2:DescribeVpcs",
"ec2:DescribeDhcpOptions",
"ec2:DescribeSubnets",
"ec2:DescribeSecurityGroups",
"application-autoscaling:DeleteScalingPolicy",
"application-autoscaling:DeleteScheduledAction",
"application-autoscaling:DeregisterScalableTarget",
"application-autoscaling:DescribeScalableTargets",
"application-autoscaling:DescribeScalingActivities",
"application-autoscaling:DescribeScalingPolicies",
"application-autoscaling:DescribeScheduledActions",
"application-autoscaling:PutScalingPolicy",
"application-autoscaling:PutScheduledAction",
"application-autoscaling:RegisterScalableTarget",
"logs:CreateLogGroup",
"logs:CreateLogStream",
"logs:DescribeLogStreams",
"logs:GetLogEvents",
"logs:PutLogEvents"
],
"Resource": "*",
},
{
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject"
],
"Resource": [
"arn:aws:s3:::*SageMaker*",
"arn:aws:s3:::*Sagemaker*",
"arn:aws:s3:::*sagemaker*
]
},
{
"Effect": "Allow",
"Action": [
"s3:GetObject",
"s3:PutObject",
"s3:DeleteObject"
],
"Resource": ["arn:aws:s3:::SageMaker*",
"arn:aws:s3:::Sagemaker*",
"arn:aws:s3:::sagemaker*"
]
},
{
"Effect": "Allow",
"Action": [
"s3:CreateBucket",
"s3:GetBucketLocation",
"s3:ListBucket",
"s3:ListAllMyBuckets"
],
"Resource": "*"
},
{
"Effect": "Allow",
"Action": [
"s3:GetObject"
],
"Resource": "*",
"Condition": {
"StringEqualsIgnoreCase": {
"s3:ExistingObjectTag/SageMaker": "true"
}
Use a Custom VPC to Connect to EI

To use EI with Amazon SageMaker in a VPC, you need to create and configure two security groups, and set up a PrivateLink VPC interface endpoint. EI uses VPC interface endpoint to communicate with Amazon SageMaker endpoints in your VPC. The security groups you create are used to connect to the VPC interface endpoint.

Set up Security Groups to Connect to EI

To use EI within a VPC, you need to create two security groups:

- A security group to control access to the VPC interface endpoint that you will set up for EI.
- A security group that allows Amazon SageMaker to call into the first security group.

Complete the following steps to configure the two security groups:

1. Create a security group with no outbound connections. You will attach this to the VPC endpoint interface you create in the next section.
2. Create a second security group with no inbound connections, but with an outbound connection to the first security group.
3. Edit the first security group to allow inbound connections only to the second security group and all outbound connections.

For more information about VPC security groups, see Security Groups for Your VPC in the Amazon Virtual Private Cloud User Guide.

Set up a VPC Interface Endpoint to Connect to EI

To use EI with Amazon SageMaker in a custom VPC, you need to set up a VPC interface endpoint (PrivateLink) for the EI service.
• Set up a VPC interface endpoint (PrivateLink) for the EI. Follow the instructions at Creating an Interface Endpoint. In the list of services, choose com.amazonaws.<region>.elastic-inference.runtime. For Security group, make sure you select the first security group you created in the previous section to the endpoint.
• When you set up the interface endpoint, choose all of the Availability Zones where EI is available. EI fails if you do not set up at least two Availability Zones. For information about VPC subnets, see VPCs and Subnets.

Attach EI to a Notebook Instance

To test and evaluate inference performance using EI, you can attach EI to a notebook instance when you create or update a notebook instance. You can then use EI in local mode to host a model at an endpoint hosted on the notebook instance. You should test various sizes of notebook instances and EI accelerators to evaluate the configuration that works best for your use case.

Set Up to Use EI

To use EI locally in a notebook instance, create a notebook instance with an EI instance. To do this:

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker
2. In the navigation pane, choose Notebook instances.
3. Choose Create notebook instance.
4. For Notebook instance name, provide a unique name for your notebook instance.
5. For notebook instance type, choose a CPU instance such as ml.t2.medium.
6. For Elastic Inference (EI), choose an instance from the list, such as ml.eia1.medium.
7. For IAM role, choose an IAM role that has the required permissions to use Amazon SageMaker and EI.
8. (Optional) For VPC - Optional, if you want the notebook instance to use a VPC, choose one from the available list, otherwise leave it as No VPC. If you use a VPC follow the instructions at Use a Custom VPC to Connect to EI (p. 351).
9. (Optional) For Lifecycle configuration - optional, either leave it as No configuration or choose a lifecycle configuration. For more information, see Step 1.1: (Optional) Customize a Notebook Instance (p. 20).
10. (Optional) For Encryption key - optional, Optional) If you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.
11. (Optional) For Volume Size In GB - optional, leave the default value of 5.
12. (Optional) For Tags, add tags to the notebook instance. A tag is a label you assign to help manage your notebook instances. A tag consists of a key and a value both of which you define.
13. Choose Create Notebook Instance.

After you create your notebook instance with EI attached, you can create a Jupyter notebook and set up an EI endpoint that is hosted locally on the notebook instance.

Topics
• Use EI in Local Mode in Amazon SageMaker (p. 352)

Use EI in Local Mode in Amazon SageMaker

To use EI locally in an endpoint hosted on a notebook instance, use local mode with the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet estimators or models. For more
information about local mode support in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview.

Topics
- Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models (p. 353)
- Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models (p. 353)

Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models

To use EI with TensorFlow in local mode, specify local for instance_type and local_sagemaker_notebook for accelerator_type when you call the deploy method of an estimator or a model object. For more information about Amazon SageMaker Python SDK TensorFlow estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

The following code shows how to use local mode with an estimator object. To call the deploy method, you must have previously called the fit method to train a model.

```python
# Deploys the model to a local endpoint
tf_predictor = tf_estimator.deploy(initial_instance_count=1,
instance_type='local',
accelerator_type='local_sagemaker_notebook')
```

The following code shows how to use local mode with an estimator object. To call the deploy method, you must have previously either:

- Trained the model by calling the fit method of an estimator.
- Pass a model artifact when you initialize the model object.

```python
# Deploys the model to a local endpoint
tf_predictor = tf_model.deploy(initial_instance_count=1,
instance_type='local',
accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with TensorFlow, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb

Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models

To use EI with MXNet in local mode, specify local for instance_type and local_sagemaker_notebook for accelerator_type when you call the deploy method of an estimator or a model object. For more information about Amazon SageMaker Python SDK MXNet estimators and models, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

The following code shows how to use local mode with an estimator object. You must have previously called the fit method of the estimator to train the model.

```python
# Deploys the model to a local endpoint
mxnet_predictor = mxnet_estimator.deploy(initial_instance_count=1,
instance_type='local',
```
accelerator_type='local_sagemaker_notebook')

For a complete example of using EI in local mode with MXNet, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb.

Use EI on Amazon SageMaker Hosted Endpoints

To use Elastic Inference (EI) in Amazon SageMaker with a hosted endpoint for real-time inference, specify an EI accelerator when you create the deployable model to be hosted at that endpoint. You can do this in one of the following ways:

- Use the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet and the Amazon SageMaker pre-built containers for TensorFlow and MXNet
- Build your own container, and use the low-level Amazon SageMaker API (Boto 3). You will need to import the EI-enabled version of either TensorFlow or MXNet from the provided Amazon S3 locations into your container, and use one of those versions to write your training script.
- Use either the Image Classification Algorithm (p. 106) or Object Detection Algorithm (p. 192) built-in algorithms, and use Boto 3 to run your training job and create your deployable model and hosted endpoint.

Topics
- Use EI with an Amazon SageMaker TensorFlow Container (p. 354)
- Use EI with an Amazon SageMaker MXNet Container (p. 355)
- Use EI with Your Own Container (p. 355)

Use EI with an Amazon SageMaker TensorFlow Container

To use TensorFlow with EI in Amazon SageMaker, you need to call the deploy method of either the Estimator or Model objects. You then specify an accelerator type using the accelerator_type input argument. For information on using TensorFlow in the Amazon SageMaker Python SDK, see: https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

Use an Estimator Object

To use an estimator object with EI, include the accelerator_type input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```python
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                             instance_type='ml.m4.xlarge',
                             accelerator_type='ml.eia1.medium')
```

Use a Model Object

To use a model object with EI, include the accelerator_type input argument when you use the deploy method. The estimator returns a predictor object which we call its deploy method as shown in the example code:

```python
# Deploy a model using EI (using the accelerator_type input argument)
```
predictor = model.deploy(initial_instance_count=1,
    instance_type='ml.m4.xlarge',
    accelerator_type='ml.eia1.medium')

### Use EI with an Amazon SageMaker MXNet Container

To use MXNet with EI in Amazon SageMaker, you need to call the `deploy` method of either the `Estimator` or `Model` objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using MXNet in the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

### Use an Estimator Object

To use an estimator object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
    instance_type='ml.m4.xlarge',
    accelerator_type='ml.eia1.medium')
```

### Use a Model Object

To use a model object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```python
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
    instance_type='ml.m4.xlarge',
    accelerator_type='ml.eia1.medium')
```

For a complete example of using EI with MXNet in Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb

### Use EI with Your Own Container

To use EI with a model in a custom container that you build, use the low-level Amazon SageMaker SDK for Python (Boto 3). You need to download and import the AWS EI-enabled versions of TensorFlow or Apache MXNet machine learning frameworks, and write your training script using those frameworks. For information about downloading the binary for TensorFlow, see Using TensorFlow Models with Elastic Inference in the Amazon Elastic Compute Cloud User Guide for Linux Instances. For information about downloading the binary for Apache MXNet, see Using MXNet Models with Amazon EI.

### Import the EI Version of TensorFlow or MXNet into Your Docker Container

To use EI with your own container, you need to import either the Amazon EI TensorFlow Serving library or the Amazon EI Apache MXNet library into your container. The EI-enabled versions of TensorFlow and MXNet are currently available as binary files stored in Amazon S3 locations. For information about downloading the binary for TensorFlow, see Using TensorFlow Models with Elastic Inference in the Amazon Elastic Compute Cloud User Guide for Linux Instances. For information about downloading the binary for Apache MXNet, see Using MXNet Models with Amazon EI.
Automatically Scale Amazon SageMaker Models

For more information about building a model container to use in Amazon SageMaker, see Using Your Own Inference Code with Amazon SageMaker Hosting Services (p. 261).

Create an EI Endpoint with Boto 3

To create an endpoint by using Boto 3, you first create an endpoint configuration. The endpoint configuration specifies one or more models (called production variants) that you want to host at the endpoint. To attach EI to one or more of the production variants hosted at the endpoint, you specify one of the EI instance types as the AcceleratorType field for that ProductionVariant. You then pass that endpoint configuration when you create the endpoint.

Create an Endpoint Configuration

To use EI, you need to specify an accelerator type in the endpoint configuration:

```python
# Create Endpoint Configuration
from time import gmtime, strftime
def create_endpoint_config_response = sagemaker.create_endpoint_config(EndpointConfigName = "MyEIEndpointConfig", ProductionVariants=[
    'InstanceType':'ml.m4.xlarge',
    'InitialInstanceCount':1,
    'ModelName':model_name,
    'VariantName':'AllTraffic',
    'AcceleratorType':'ml.eia1.medium']
)
print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

Create an Endpoint

After you create an endpoint configuration with an accelerator type, you can proceed to create an endpoint.

```python
endpoint_response = sagemaker.create_endpoint("MyEIEndpointConfig")
```

After the endpoint is created you can invoke it using the invoke_endpoint method in a boto3 runtime object as you would any other endpoint.

Automatically Scale Amazon SageMaker Models

Amazon SageMaker supports automatic scaling for production variants. **Automatic scaling** dynamically adjusts the number of instances provisioned for a production variant in response to changes in your workload. When the workload increases, automatic scaling brings more instances online. When the workload decreases, automatic scaling removes unnecessary instances so that you don't pay for provisioned variant instances that you aren't using.

To use automatic scaling for a production variant, you define and apply a scaling policy that uses Amazon CloudWatch metrics and target values that you assign. Automatic scaling uses the policy to adjust the number of instances up or down in response to actual workloads.

You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A **predefined metric** is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Alternatively, you can use either the AWS Command Line Interface (AWS CLI) or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric.
We strongly recommend that you load test your automatic scaling configuration to ensure that it works correctly before using it to manage production traffic.

For information about deploying trained models as endpoints, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29).

**Topics**
- Automatic Scaling Components (p. 357)
- Before You Begin (p. 359)
- Related Topics (p. 359)
- Configure Automatic Scaling for a Production Variant (p. 359)
- Edit a Scaling Policy (p. 366)
- Delete a Scaling Policy (p. 366)
- Load Testing for Production Variant Automatic Scaling (p. 368)
- Best Practices for Configuring Automatic Scaling (p. 369)

**Automatic Scaling Components**

To adjust the number of instances hosting a production variant, Amazon SageMaker automatic scaling uses a scaling policy. Automatic scaling has the following components:

- **Required permissions**—Permissions that are required to perform automatic scaling actions.
- **A service-linked role**—An AWS Identity and Access Management (IAM) role that is linked to a specific AWS service. A service-linked role includes all of the permissions that the service requires to call other AWS services on your behalf. Amazon SageMaker automatic scaling automatically generates this role, `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint`, for you.
- **A target metric**—The Amazon CloudWatch metric that Amazon SageMaker automatic scaling uses to determine when and how much to scale.
- **Minimum and maximum capacity**—The minimum and maximum number of instances to use for scaling the variant.
- **A cool down period**—The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.

**Required Permissions for Automatic Scaling**

The `SagemakerFullAccessPolicy` IAM policy has all of the permissions required to perform automatic scaling actions. For more information about Amazon SageMaker IAM roles, see Amazon SageMaker Roles (p. 415).

If you are using a custom permission policy, you must include the following permissions:

```json
{
  "Effect": "Allow",
  "Action": [
    "sagemaker:DescribeEndpoint",
    "sagemaker:DescribeEndpointConfig",
    "sagemaker:UpdateEndpointWeightsAndCapacities"
  ],
  "Resource": "*"
}
{
  "Action": [
    "application-autoscaling:*"
  ],
```
Service-Linked Role for Automatic Scaling

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other AWS services on your behalf. Automatic scaling uses the AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint service-linked role. For more information, see Service-Linked Roles for Application Auto Scaling in the Application Auto Scaling User Guide.

Target Metric for Automatic Scaling

Amazon SageMaker automatic scaling uses target-tracking scaling policies. You configure the target-tracking scaling policy by specifying a predefined or custom metric and a target value for the metric. For more information, see Target Tracking Scaling Policies.

Amazon CloudWatch alarms trigger the scaling policy, which calculate how to adjust scaling based on the metric and target value that you set. The scaling policy adds or removes endpoint instances as required to keep the metric at, or close to, the specified target value. In addition, a target-tracking scaling policy also adjusts to fluctuations in the metric when a workload changes. The policy minimizes rapid fluctuations in the number of available instances for your variant.

For example, a scaling policy that uses the predefined InvocationsPerInstance metric with a target value of 70 can keep InvocationsPerInstance at, or close to 70.

Minimum and Maximum Capacity for Automatic Scaling

You can specify the maximum number of endpoint instances that Application Auto Scaling manages for the variant. The maximum value must be equal to or greater than the value specified for the minimum number of endpoint instances. Amazon SageMaker automatic scaling does not enforce a limit for this value.

You can also specify the minimum number of instances that Application Auto Scaling manages for the variant. This value must be at least 1, and equal to or less than the value specified for the maximum number of variant instances.
To determine the minimum and maximum number of instances that you need for typical traffic, test your automatic scaling configuration with the expected rate of traffic to your variant.

Cooldown Period for Automatic Scaling

Tune the responsiveness of a target-tracking scaling policy by adding a cooldown period. A cooldown period controls when your variant is scaled in and out by blocking subsequent scale-in or scale-out requests until the period expires. This slows the deletion of variant instances for scale-in requests, and the creation of variant instances for scale-out requests. A cooldown period helps to ensure that it doesn't launch or terminate additional instances before the previous scaling activity takes effect. After automatic scaling dynamically scales using a scaling policy, it waits for the cooldown period to complete before resuming scaling activities.

You configure the cooldown period in your automatic scaling policy. You can specify the following cooldown periods:

- A scale-in activity reduces the number of variant instances. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.
- A scale-out activity increases the number of variant instances. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

If you don't specify a scale-in or a scale-out cooldown period automatic scaling use the default, which is 300 seconds for each.

If instances are being added or removed too quickly when you test your automatic scaling configuration, consider increasing this value. You can see this behavior if the traffic to your variant has a lot of spikes, or if you have multiple automatic scaling policies defined for a variant.

If instances are not being added quickly enough to address increased traffic, consider decreasing this value.

Before You Begin

Before you can use automatically scaled model deployment, create an Amazon SageMaker model deployment. For more information about deploying a model endpoint, see Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services (p. 29).

When automatic scaling adds a new variant instance, it is the same instance class as the one used by the primary instance.

Related Topics

- What Is Application Auto Scaling?

Configure Automatic Scaling for a Production Variant

You can configure automatic scaling for a variant with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Topics

- Configure Automatic Scaling for a Production Variant (Console) (p. 360)
Configure Automatic Scaling for a Production Variant (Console)

To configure automatic scaling for a production variant (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose Endpoints.
3. Choose the endpoint that you want to configure.
4. For Endpoint runtime settings, choose the variant that you want to configure.
5. For Endpoint runtime settings, choose Configure auto scaling.

The Configure variant automatic scaling page appears.

6. For Minimum capacity, type the minimum number of instances that you want the scaling policy to maintain. At least 1 instance is required.
7. For Maximum capacity, type the maximum number of instances that you want the scaling policy to maintain.
8. For the target value, type the average number of invocations per instance per minute for the variant. To determine this value, follow the guidelines in Load Testing (p. 368).

Application Auto Scaling adds or removes instances to keep the metric close to the value that you specify.

9. For Scale-in cool down (seconds) and Scale-out cool down (seconds), type the number seconds for each cool down period. Assuming that the order in the list is based on either most important to less important of first applied to last applied.
10. Select Disable scale in to prevent the scaling policy from deleting variant instances if you want to ensure that your variant scales out to address increased traffic, but are not concerned with removing instances to reduce costs when traffic decreases, disable scale-in activities.

Scale-out activities are always enabled so that the scaling policy can create endpoint instances as needed.
11. Choose Save.

This procedure registers a variant as a scalable target with Application Auto Scaling. When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a burstable performance instance such as T2

Note
Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because they already allow for increased capacity under increased workloads. For information about burstable performance instances, see Amazon EC2 Instance Types.

Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API)

With the AWS CLI or the Application Auto Scaling API, you can configure automatic scaling based on either a predefined or a custom metric.
Register a Production Variant

To define the scaling limits for the variant, register your variant with Application Auto Scaling. Application Auto Scaling dynamically scales the number of variant instances.

To register your variant, you can use either the AWS CLI or the Application Auto Scaling API.

When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant resource exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a Burstable Performance Instance such as T2

**Note**
Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because burstable instances already allow for increased capacity under increased workloads. For information about Burstable Performance Instances, see Amazon EC2 Instance Types.

Register a Production Variant (AWS CLI)

To register your endpoint, use the `register-scalable-target` AWS CLI command with the following parameters:

- `--service-namespace`—Set this value to `sagemaker`.
- `--resource-id`—The resource identifier for the production variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example `endpoint/MyEndpoint/variant/MyVariant`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--min-capacity`—The minimum number of instances that Application Auto Scaling must manage for this endpoint. Set `min-capacity` to at least 1. It must be equal to or less than the value specified for `max-capacity`.
- `--max-capacity`—The maximum number of instances that Application Auto Scaling should manage. Set `max-capacity` to a minimum of 1, it must be equal to or greater than the value specified for `min-capacity`.

**Example**

The following example shows how to register an endpoint variant named `MyVariant` that is dynamically scaled to have one to eight instances:

```bash
aws application-autoscaling register-scalable-target
   --service-namespace sagemaker
   --resource-id endpoint/MyEndpoint/variant/MyVariant
   --scalable-dimension sagemaker:variant:DesiredInstanceCount
   --min-capacity 1
   --max-capacity 8
```

Register a Production Variant (Application Auto Scaling API)

To register your endpoint variant with Application Auto Scaling, use the `RegisterScalableTarget` Application Auto Scaling API action with the following parameters:

- `ServiceNamespace`—Set this value to `sagemaker`. 

361
Configure Automatic Scaling for a Production Variant

- **ResourceId**—The resource identifier for the production variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant, for example endpoint/MyEndPoint/variant/MyVariant.
- **ScalableDimension**—Set this value to sagemaker:variant:DesiredInstanceCount.
- **MinCapacity**—The minimum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or less than the value specified for MaxCapacity.
- **MaxCapacity**—The maximum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or greater than the value specified for MinCapacity.

**Example**

The following example shows how to register an Amazon SageMaker production variant that is dynamically scaled to use one to eight instances:

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndPoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

**Define a Target-Tracking Scaling Policy**

To specify the metrics and target values for a scaling policy, you configure a target-tracking scaling policy. You can use either a predefined metric or a custom metric.

Scaling policy configuration is represented by a JSON block. You save your scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

The following options are available for defining a target-tracking scaling policy configuration.

**Topics**
- Use a Predefined Metric (p. 362)
- Use a Custom Metric (p. 363)
- Add a Cooldown Period (p. 363)
- Disable Scale-in Activity (p. 364)

**Use a Predefined Metric**

To quickly define a target-tracking scaling policy for a variant, use the SageMakerVariantInvocationsPerInstance predefined metric. SageMakerVariantInvocationsPerInstance is the average number of times per minute that each instance for a variant is invoked. We strongly recommend using this metric.
To use a predefined metric in a scaling policy, create a target tracking configuration for your policy. In the target tracking configuration, include a `PredefinedMetricSpecification` for the predefined metric and a `TargetValue` for the target value of that metric.

**Example**

The following example is a typical policy configuration for target-tracking scaling for a variant. In this configuration, we use the `SageMakerVariantInvocationsPerInstance` predefined metric to adjust the number of variant instances so that each instance has an `InvocationsPerInstance` metric of 70.

```json
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    }
}
```

**Use a Custom Metric**

If you need to define a target-tracking scaling policy that meets your custom requirements, define a custom metric. You can define a custom metric based on any production variant metric that changes in proportion to scaling.

Not all Amazon SageMaker metrics work for target tracking. The metric must be a valid utilization metric, and it must describe how busy an instance is. The value of the metric must increase or decrease in inverse proportion to the number of variant instances. That is, the value of the metric should decrease when the number of instances increases.

**Important**

Before deploying automatic scaling in production, you must test automatic scaling with your custom metric.

**Example**

The following example is a target-tracking configuration for a scaling policy. In this configuration, for a variant named `my-variant`, a custom metric adjusts the variant based on an average CPU utilization of 50 percent across all instances.

```json
{
    "TargetValue": 50,
    "CustomizedMetricSpecification": {
        "MetricName": "CPUUtilization",
        "Namespace": "/aws/sagemaker/Endpoints",
        "Dimensions": [
            {"Name": "EndpointName", "Value": "my-endpoint" },
            {"Name": "VariantName","Value": "my-variant"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

**Add a Cooldown Period**

To add a cooldown period for scaling out your variant, specify a value, in seconds, for `ScaleOutCooldown`. Similarly, to add a cooldown period for scaling in your variant, add a value, in seconds, for `ScaleInCooldown`. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see `TargetTrackingScalingPolicyConfiguration` in the Application Auto Scaling API Reference.
Example

The following is an example of a target-tracking policy configuration for a scaling policy. In this configuration, the SageMakerVariantInvocationsPerInstance predefined metric is used to adjust a variant based on an average of 70 across all instances of that variant. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```json
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":{
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",
        "ScaleInCooldown": 600,
        "ScaleOutCooldown": 300
    }
}
```

Disable Scale-in Activity

You can prevent the target-tracking scaling policy configuration from scaling in your variant by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting instances, while still allowing it to create them as needed.

To enable or disable scale-in activity for your variant, specify a Boolean value for DisableScaleIn. For more information about DisableScaleIn, see TargetTrackingScalingPolicyConfiguration in the Application Auto Scaling API Reference.

Example

The following is an example of a target-tracking configuration for a scaling policy. In this configuration, the SageMakerVariantInvocationsPerInstance predefined metric adjusts a variant based on an average of 70 across all instances of that variant. The configuration disables scale-in activity for the scaling policy.

```json
{
    "TargetValue": 70.0,
    "PredefinedMetricSpecification":{
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",
        "DisableScaleIn": true
    }
}
```

Apply a Scaling Policy to a Production Variant

After registering your variant and defining a scaling policy, apply the scaling policy to the registered variant. To apply a scaling policy to a variant, you can use the AWS CLI or the Application Auto Scaling API.

Apply a Scaling Policy to a Production Variant (AWS CLI)

To apply a scaling policy to your variant, use the `put-scaling-policy` AWS CLI command with the following parameters:

- **--policy-name**—The name of the scaling policy.
- **--policy-type**—Set this value to TargetTrackingScaling.
- **--resource-id**—The resource identifier for the variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example endpoint/MyEndpoint/variant/MyVariant.
---service-namespace—Set this value to sagemaker.
---scalable-dimension—Set this value to sagemaker:variant:DesiredInstanceCount.
---target-tracking-scaling-policy-configuration—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses with Application Auto Scaling to apply a target-tracking scaling policy named myscalablepolicy to a variant named myscalablevariant. The policy configuration is saved in a file named config.json.

```
aws application-autoscaling put-scaling-policy \
  --policy-name myscalablepolicy \ 
  --policy-type TargetTrackingScaling \ 
  --resource-id endpoint/MyEndpoint/variant/MyVariant \ 
  --service-namespace sagemaker \ 
  --scalable-dimension sagemaker:variant:DesiredInstanceCount \ 
  --target-tracking-scaling-policy-configuration file://config.json
```

Apply a Scaling Policy to a Production Variant (Application Auto Scaling API)

To apply a scaling policy to a variant with the Application Auto Scaling API, use the `PutScalingPolicy` Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to sagemaker.
- **ResourceId**—The resource identifier for the variant. For this parameter, the resource type is endpoint and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to sagemaker:variant:DesiredInstanceCount.
- **PolicyType**—Set this value to TargetTrackingScaling.
- **TargetTrackingScalingPolicyConfiguration**—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses Application Auto Scaling to apply a target-tracking scaling policy named myscalablepolicy to a variant named myscalablevariant. It uses a policy configuration based on the `SageMakerVariantInvocationsPerInstance` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
   "PolicyName": "myscalablepolicy",
   "ServiceNamespace": "sagemaker",
   "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
   "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
   "PolicyType": "TargetTrackingScaling",
   "TargetTrackingScalingPolicyConfiguration": {
      "PredefinedMetric": "SageMakerVariantInvocationsPerInstance",
      "PredefinedMetricOptions": {
         "IgnoreValuesBelow": 0,
         "IgnoreValuesAbove": 100
      }
   }
}
```
"PolicyType": "TargetTrackingScaling",
"TargetTrackingScalingPolicyConfiguration": {
  "TargetValue": 70.0,
  "PredefinedMetricSpecification": {
    "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
  }
}

Edit a Scaling Policy

You can edit a variant scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Edit a Scaling Policy (Console)

To edit a scaling policy with the AWS Management Console, use the same procedure that you used to Configure Automatic Scaling for a Production Variant (Console) (p. 360).

Edit a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- With the AWS CLI, specify the name of the policy that you want to edit in the --policy-name parameter. Specify new values for the parameters that you want to change.
- With the Application Auto Scaling API, specify the name of the policy that you want to edit in the PolicyName parameter. Specify new values for the parameters that you want to change.

For more information, see Apply a Scaling Policy to a Production Variant (p. 364).

Delete a Scaling Policy

You can delete a scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Delete a Scaling Policy (Console)

To delete an automatic scaling policy for a variant (console)

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. In the navigation pane, choose Endpoints.
3. Choose the endpoint for which you want to delete automatic scaling.
4. For Endpoint runtime settings, choose the variant that you want to configure.
5. Choose Configure auto scaling.
6. Choose Deregister auto scaling.

Delete a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from a variant.
Delete a Scaling Policy (AWS CLI)

To delete a scaling policy from a variant, use the `delete-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

**Example**

The following example deletes a target-tracking scaling policy named `myscalablepolicy` from a variant named `myscalablevariant`.

```
aws application-autoscaling delete-scaling-policy \
  --policy-name myscalablepolicy \n  --resource-id endpoint/MyEndpoint/variant/MyVariant \n  --service-namespace sagemaker \n  --scalable-dimension sagemaker:variant:DesiredInstanceCount
```

Delete a Scaling Policy (Application Auto Scaling API)

To delete a scaling policy from your variant, use the `DeleteScalingPolicy` Application Auto Scaling API action with the following parameters:

- `PolicyName`—The name of the scaling policy.
- `ServiceNamespace`—Set this value to `sagemaker`.
- `ResourceId`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- `ScalableDimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

**Example**

The following example uses the Application Auto Scaling API to delete a target-tracking scaling policy named `myscalablepolicy` from a variant named `myscalablevariant`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```
Load Testing for Production Variant Automatic Scaling

Perform load tests to choose an automatic scaling configuration that works the way you want.

For an example of load testing to optimize automatic scaling for a Amazon SageMaker endpoint, see [Load test and optimize an Amazon SageMaker endpoint using automatic scaling](Load test and optimize an Amazon SageMaker endpoint using automatic scaling).

The following guidelines for load testing assume you are using an automatic scaling policy that uses the predefined target metric `SageMakerVariantInvocationsPerInstance`.

**Topics**

- Determine the Performance Characteristics of a Production Variant (p. 368)
- Calculate the Target `SageMakerVariantInvocationsPerInstance` (p. 368)

Determine the Performance Characteristics of a Production Variant

Perform load testing to find the peak `InvocationsPerInstance` that your variant instance can handle, and the latency of requests, as concurrency increases.

This value depends on the instance type chosen, payloads that clients of your variant typically send, and the performance of any external dependencies your variant has.

**To find the peak requests-per-second (RPS) your variant can handle and latency of requests**

1. Set up an endpoint with your variant using a single instance. For information about how to set up an endpoint, see [Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services](Step 2.4.1: Deploy the Model to Amazon SageMaker Hosting Services) (p. 29).
2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the output of the load testing tool.

   **Note**
   
   You can also monitor requests-per-minute instead of RPS. In that case don't multiply by 60 in the equation to calculate `SageMakerVariantInvocationsPerInstance` shown below.

   When the model latency increases or the proportion of successful transactions decreases, this is the peak RPS that your variant can handle.

Calculate the Target `SageMakerVariantInvocationsPerInstance`

After you find the performance characteristics of the variant, you can determine the maximum RPS we should allow to be sent to an instance. The threshold used for scaling must be less than this maximum value. Use the following equation in combination with load testing to determine the correct value for the `SageMakerVariantInvocationsPerInstance` target metric in your automatic scaling configuration.

\[
\text{SageMakerVariantInvocationsPerInstance} = (\text{MAX\_RPS} \times \text{SAFETY\_FACTOR}) \times 60
\]

Where `MAX\_RPS` is the maximum RPS that you determined previously, and `SAFETY\_FACTOR` is the safety factor that you chose to ensure that your clients don't exceed the maximum RPS. Multiply by 60 to
convert from RPS to invocations-per-minute to match the per-minute CloudWatch metric that Amazon SageMaker uses to implement automatic scaling (you don’t need to do this if you measured requests-per-second instead of requests-per-second).

**Note**
Amazon SageMaker recommends that you start testing with a SAFETY_FACTOR of 0.5. Test your automatic scaling configuration to ensure it operates in the way you expect with your model for both increasing and decreasing customer traffic on your endpoint.

### Best Practices for Configuring Automatic Scaling

When configuring automatic scaling, consider the following general guidelines.

#### Testing Your Automatic Scaling Configuration

It is important that you test your automatic scaling configuration to confirm that it works with your model the way you expect it to.

#### Updating Endpoints Configured for Automatic Scaling

When you update an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If the update would change the instance type for any variant that is a target for automatic scaling, the update fails.

In the AWS Management Console, you see a warning that you must deregister the variant from automatic scaling before you can update it. If you are trying to update the endpoint by calling the UpdateEndpoint (p. 722) API, the call fails. Before you update the endpoint, delete any scaling policies configured for it by calling the DeleteScalingPolicy Application Auto Scaling API action, then call DeregisterScalableTarget to deregister the variant as a scalable target. After you update the endpoint, you can register the variant as a scalable target and attach an automatic scaling policy to the updated variant.

There is one exception. If you change the model for a variant that is configured for automatic scaling, Amazon SageMaker automatic scaling allows the update. This is because changing the model doesn’t typically affect performance enough to change automatic scaling behavior. If you do update a model for a variant configured for automatic scaling, ensure that the change to the model doesn’t significantly affect performance and automatic scaling behavior.

#### Deleting Endpoints Configured for Automatic Scaling

If you delete an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If any are and you have permission to deregister the variant, Application Auto Scaling deregisters those variants as scalable targets without notifying you. If you use a custom permission policy that doesn’t provide permission for the DeleteScalingPolicy and DeregisterScalableTarget actions, you must delete automatic scaling policies and deregister scalable targets and before deleting the endpoint.

**Note**
You, as an IAM user, might not have sufficient permission to delete an endpoint if another IAM user configured automatic scaling for a variant on that endpoint.

### Using Step Scaling Policies

Although Amazon SageMaker automatic scaling supports using Application Auto Scaling step scaling policies, we recommend using target tracking policies, instead. For information about using Application Auto Scaling step scaling policies, see Step Scaling Policies.
Scaling In When There Is No Traffic

If a variant's traffic becomes zero, Amazon SageMaker automatic scaling doesn't scale down. This is because Amazon SageMaker doesn't emit metrics with a value of zero.

As a workaround, do either of the following:

- Send requests to the variant until automatic scaling scales down to the minimum capacity
- Change the policy to reduce the maximum provisioned capacity to match the minimum provisioned capacity

Best Practices for Deploying Amazon SageMaker Models

This topic provides guidance on best practices for deploying machine learning models in Amazon SageMaker.

Topics

- Deploy Multiple Instances Across Availability Zones (p. 370)

Deploy Multiple Instances Across Availability Zones

Create robust endpoints when hosting your model. Amazon SageMaker endpoints can help protect your application from Availability Zone outages and instance failures. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones. For this reason, we strongly recommended that you deploy multiple instances for each production endpoint.

If you are using an Amazon Virtual Private Cloud (VPC), configure the VPC with at least two Subnets, each in a different Availability Zone. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones.

In general, to achieve more reliable performance, use more small Instance Types in different Availability Zones to host your endpoints.
Amazon SageMaker integrates with AWS Marketplace, enabling developers to charge other Amazon SageMaker users for the use of their algorithms and model packages. AWS Marketplace is a curated digital catalog that makes it easy for customers to find, buy, deploy, and manage third-party software and services that customers need to build solutions and run their businesses. AWS Marketplace includes thousands of software listings in popular categories, such as security, networking, storage, machine learning, business intelligence, database, and DevOps. It simplifies software licensing and procurement with flexible pricing options and multiple deployment methods. For information, see AWS Marketplace Documentation.

Topics

- Amazon SageMaker Algorithms (p. 371)
- Amazon SageMaker Model Packages (p. 371)
- Sell Amazon SageMaker Algorithms and Model Packages (p. 372)
- Find and Subscribe to Algorithms and Model Packages on AWS Marketplace (p. 374)
- Use Algorithm and Model Package Resources (p. 271)

Amazon SageMaker Algorithms

An algorithm enables you to perform end-to-end machine learning. It has two logical components: training and inference. Buyers can use the training component to create training jobs in Amazon SageMaker and build a machine learning model. Amazon SageMaker saves the model artifacts generated by the algorithm during training to an Amazon S3 bucket. For more information, see Train a Model with Amazon SageMaker (p. 4).

Buyers use the inference component with the model artifacts generated during a training job to create a deployable model in their Amazon SageMaker account. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 7).

Amazon SageMaker Model Packages

Buyers use a model package to build a deployable model in Amazon SageMaker. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see Deploy a Model in Amazon SageMaker (p. 7). As a seller, you can build your model artifacts by training in Amazon SageMaker, or you can use your own model artifacts from a model that you trained outside of Amazon SageMaker. You can charge buyers for inference.
Sell Amazon SageMaker Algorithms and Model Packages

Selling Amazon SageMaker algorithms and model packages is a three-step process:

1. Develop your algorithm or model, and package it in a Docker container. For information, see Develop Algorithms and Models in Amazon SageMaker (p. 372).

2. Create an algorithm or model package resource in Amazon SageMaker. For information, see Create Algorithm and Model Package Resources (p. 265).

3. Register as a seller on AWS Marketplace and list your algorithm or model package on AWS Marketplace. For information about registering as a seller, see Getting Started as a Seller in the User Guide for AWS Marketplace Providers. For information about listing and monetizing your algorithms and model packages, see Listing Algorithms and Model Packages in AWS Marketplace for Machine Learning in the User Guide for AWS Marketplace Providers.

Topics

- Develop Algorithms and Models in Amazon SageMaker (p. 372)
- Create Algorithm and Model Package Resources (p. 265)
- List Your Algorithm or Model Package on AWS Marketplace (p. 374)

Develop Algorithms and Models in Amazon SageMaker

Before you can create algorithm and model package resources to use in Amazon SageMaker or list on AWS Marketplace, you have to develop them and package them in Docker containers.
When algorithms and model packages are created for listing on AWS Marketplace, Amazon SageMaker scans the containers for security vulnerabilities on supported operating systems. Only the following operating system versions are supported:

- Debian: 6.0, 7, 8, 9, 10
- CentOS: 5, 6, 7
- Oracle Linux: 5, 6, 7
- Alpine: 3.3, 3.4, 3.5
- Amazon Linux

Notes

Develop Algorithms in Amazon SageMaker

An algorithm should be packaged as a Docker container and stored in Amazon ECR to use it in Amazon SageMaker. The Docker container contains the training code used to run training jobs and, optionally, the inference code used to get inferences from models trained by using the algorithm.

For information about developing algorithms in Amazon SageMaker and packaging them as containers, see Use Your Own Algorithms with Amazon SageMaker (p. 255). For a complete example of how to create an algorithm container, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit_bring_your_own/scikit_bring_your_own.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the Advanced Functionality section, and is named scikit_bring_your_own.ipynb. For information about using the sample notebooks in a notebook instance, see Use Example Notebooks (p. 47).

Always thoroughly test your algorithms before you create algorithm resources to publish on AWS Marketplace.

Notes when a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

Develop Models in Amazon SageMaker

A deployable model in Amazon SageMaker consists of inference code, model artifacts, an IAM role that is used to access resources, and other information required to deploy the model in Amazon SageMaker. Model artifacts are the results of training a model by using a machine learning algorithm. The inference code must be packaged in a Docker container and stored in Amazon ECR. You can either package the model artifacts in the same container as the inference code, or store them in Amazon S3.

You create a model by running a training job in Amazon SageMaker, or by training a machine learning algorithm outside of Amazon SageMaker. If you run a training job in Amazon SageMaker, the resulting model artifacts are available in the ModelArtifacts field in the response to a call to the DescribeTrainingJob (p. 629) operation. For information about how to develop an Amazon SageMaker model container, see Using Your Own Inference Code (p. 260). For a complete example of how to create
a model container from a model trained outside of Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/xgboost Bring your own model/xgboost Bring your own model.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the Advanced Functionality section, and is named xgboost Bring your own model.ipynb. For information about using the sample notebooks in a notebook instance, see Use Example Notebooks (p. 47).

Always thoroughly test your models before you create model packages to publish on AWS Marketplace.

Note
When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

List Your Algorithm or Model Package on AWS Marketplace

After creating and validating your algorithm or model in Amazon SageMaker, list your product on AWS Marketplace. The listing process makes your products available in the AWS Marketplace and the Amazon SageMaker console.

To list products on AWS Marketplace, you must be a registered seller. To register, use the self-registration process from the AWS Marketplace Management Portal (AMMP). For information, see Getting Started as a Seller in the User Guide for AWS Marketplace Providers. When you start the product listing process from the Amazon SageMaker console, we check your seller registration status. If you have not registered, we direct you to do so.

To start the listing process, do one of the following:

- From the Amazon SageMaker console, choose the product, choose Actions, and choose Publish new ML Marketplace listing. This carries over your product reference, the Amazon Resource Name (ARN), and directs you to the AMMP to create the listing.
- Go to ML listing process, manually enter the Amazon Resource Name (ARN), and start your product listing. This process carries over the product metadata that you entered when creating the product in Amazon SageMaker. For an algorithm listing, the information includes the supported instance types and hyperparameters. In addition, you can enter a product description, promotional information, and support information as you would with other AWS Marketplace products.

Find and Subscribe to Algorithms and Model Packages on AWS Marketplace

With AWS Marketplace, you can browse and search for hundreds of machine learning algorithms and models in a broad range of categories, such as computer vision, natural language processing, speech recognition, text, data, voice, image, video analysis, fraud detection, predictive analysis, and more.
To find algorithms on AWS Marketplace

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Algorithms**, then choose **Find algorithms**.

   This takes you to the AWS Marketplace algorithms page. For information about finding and subscribing to algorithms on AWS Marketplace, see Machine Learning Products in the AWS Marketplace User Guide for AWS Consumers.

To find model packages on AWS Marketplace

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Model packages**, then choose **Find model packages**.

   This takes you to the AWS Marketplace model packages page. For information about finding and subscribing to model packages on AWS Marketplace, see Machine Learning Products in the AWS Marketplace User Guide for AWS Consumers.

Use Algorithms and Model Packages

For information about using algorithms and model packages that you subscribe to in Amazon SageMaker, see Use Algorithm and Model Package Resources (p. 271).

**Note**

When you create a training job, inference endpoint, and batch transform job from an algorithm or model package that you subscribe to on AWS Marketplace, the training and inference containers do not have access to the internet. Because the containers do not have access to the internet, the seller of the algorithm or model package does not have access to your data.
Manage Machine Learning Experiments with Search

Search is in preview release for Amazon SageMaker and is subject to change.

To find, organize, and evaluate experiments, use Amazon SageMaker search. Search can help you to manage your training jobs, models and resources.

Amazon SageMaker search is ideal for:

- Finding, organizing, or evaluating training jobs using properties, hyperparameters, or any other metadata
- Using a search result to rank training jobs and models based on metrics, such as training loss or validation accuracy to find the best performing one
- Tracing the lineage of a model back to the training job and its related resources, such as the training datasets

Topics

- Use Search to Find, Organize, and Evaluate Training Jobs (p. 376)
- Search for Information About Related Resources (p. 379)

Use Search to Find, Organize, and Evaluate Training Jobs

To find a specific training job, model, or resource, use Amazon SageMaker search as you would any search engine. Amazon SageMaker searches for the keyword in all searchable items, which include training jobs, models, hyperparameters, metadata, tags, and URLs. You can use search with tags to help organize your training jobs. To help refine your search results, you can search using multiple search criteria.

To create a model, you need to experiment. When you perform an experiment, you might test a different algorithm, tune a hyperparameter, or use a different dataset. You make changes in steps, then study their effect on the model's performance. Eventually, you need to evaluate all of your models against one or several metrics and choose the best one. You can use a search result to list models and evaluate them.

Find Training Jobs (Console)

To use search (console)

2. In the navigation pane, choose Search.
3. For Resource type, choose Training jobs.
4. To add a parameter, for Parameters, provide the following:
   a. Enter a parameter in the search box and choose a parameter type, for example TrainingJobName.
   b. Choose the conditional operation to use. This sets the condition that values must satisfy to be included in a search result. For numeral values, use operators such as is equals to, lesser than, or or greater than. For text-based values use operators, such as equals to or contains.
   c. Enter the value for the parameter.
5. (optional) To refine your search, add additional search criteria, choose Add parameter and enter the parameter values. When you provide multiple parameters, Amazon SageMaker includes all parameters in the search.
6. Choose Search.

Use Tags to Search for Training Jobs

You can use tags as search criteria. To group training jobs, create tags with descriptive keys and value. For example, create tag keys for: project, owner, customer, and industry.

To use tags in a search

2. In the navigation pane, choose Search.
3. For Tags, enter a tag key and a tag value.
4. To add more tags to the search, choose Add tag and add a another tag key-value pair for each new tag that you want to add.

When you use tags in a search, in the results, the key is a column name and the values are entries in rows.

Evaluate Models Returned by a Search

To evaluate different models, find their metrics with a search. To highlight metrics, adjust the view to show only metrics and important hyperparameters.

To change the viewable metadata, hyperparameters, and metrics:

1. After performing a search, at the search results table, choose the cog icon to show the preferences window.
2. To show or hide a Hyperparameter or Metric, use its toggle switch.
3. To update the view after making changes, choose Update view.

After viewing metrics and important hyperparameters, you can compare and contrast the result. From there, you can choose the best model to host or investigate the models that are performing poorly.

Use Search to Find Training Jobs (API)

To use search, create a search parameter, then use the search function found in smclient of the low-level SDK for Python (Boto 3).

The following example shows how to use search using the API:
import boto3

search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [{
            "Name": "Tags.Project",
            "Operator": "Equals",
            "Value": "Project_Binary_Classifier"
        }],
        "SortBy": "Metrics.train:binary_classification_accuracy",
        "SortOrder": "Descending"
    }
}

smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)

Evaluate Models (API)

To evaluate different models, see the metrics in a search result. To evaluate models using the low-level SDK for Python (Boto 3), create a table and plot it.

The following example shows how to use search to evaluate models and show the results in a table:

import pandas

headers=["Training Job Name", "Training Job Status", "Batch Size", "Binary Classification Accuracy"]
rows=[]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
    rows.append([trainingJob['TrainingJobName'],
    trainingJob['TrainingJobStatus'],
    trainingJob['HyperParameters']['mini_batch_size'],
    metrics[x['MetricName'] for x in
    metrics].index('train:binary_classification_accuracy')]["Value"]
)

df = pandas.DataFrame(data=rows,columns=headers)

from IPython.display import display, HTML
display(HTML(df.to_html()))

Get Suggestions for a Search (API)

To get suggestions for a search, use get_search_suggestions function in the API.

The following code example shows a get_search_suggestions request syntax:

search_suggestion_params={
    "Resource": "TrainingJob",
    "SuggestionQuery": {
        "PropertyNameQuery": {
            "PropertyNameHint": "linear"
        }
    }
}
The following code example shows an expected `get_search_suggestions` response syntax:

```json
{
    'PropertyNameSuggestions': [{'PropertyName': 'hyperparameters.linear_init_method'},
                               {'PropertyName': 'hyperparameters.linear_init_value'},
                               {'PropertyName': 'hyperparameters.linear_init_sigma'},
                               {'PropertyName': 'hyperparameters.linear_lr'},
                               {'PropertyName': 'hyperparameters.linear_wd'}]
}
```

After you get the search suggestions, you can use one of the property names in a search.

### Verify the Contents of Your Training Jobs

You can use Amazon SageMaker search to verify which datasets were used in training, where the holdout datasets were used, and other details about training jobs. For example, if you need to verify that a dataset was used in a training job for an audit or to verify compliance, use Amazon SageMaker search.

To check if a holdout dataset or any other dataset was used in a training job, search for its Amazon S3 URL. Amazon SageMaker uses the dataset as a search parameter and lists the training jobs that used the dataset. If your search result is empty, it proves that the dataset was not used in a training job.

### Search for Information About Related Resources

You can use Amazon SageMaker search to see information about training jobs and the related model or resources. For example, if you find that a hosted model's performance has declined, you can review its training job and the resources used to determine what's causing the problem.

There are three types of investigations:

- You can investigate a model, navigate to review the details for the training job, and then review the job's dataset, algorithm, hyperparameters, and metrics.
- You can investigate a dataset, then inspect the training job and its related settings, investigate the model, and finally investigate the endpoint.
- You can investigate a model's containers. A model might be composed of a single model container or multiple model containers. A model made of multiple model containers is called a **pipeline model**. Each model container is a distinct model that contains links to the training job, dataset, and associated environment variables.

**Note**

In Amazon SageMaker, a pipeline model can have up to three model containers. These pipelines also support the deployment of a linear sequence of pretrained model containers, where the output of one machine learning model provides the input to the next. A pipeline model is useful for solving classification problems, where each supervised model is trained using a specific subset of a dataset that is designated as a class. When generating inference on a pipeline model, you get results from each model container.

### Search for Information About Related Resources (Console)

To see a list of URLs to a model or training job's resources, see the details page for the model or training job. To access the details page, choose **Training jobs** or **Models** in the navigation pane.
Search for Information About Related Resources (API)

To trace a model's lineage, you need to obtain the model's name, then use it to search for training jobs.

The following example shows how to use search to trace a model's lineage using the API:

```python
# Get the name of model deployed at endpoint
descriptor = smclient.describe_endpoint_config(EndpointConfigName=endpointName)
model_name = descriptor['ProductionVariants'][0]['ModelName']

# Get the model's name
model = smclient.describe_model(ModelName=model_name)

# Search the training job by Amazon S3 location of model artifacts
search_params={
    "MaxResults": 1,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "ModelArtifacts.S3ModelArtifacts",
                "Operator": "Equals",
                "Value": model['PrimaryContainer']['ModelDataUrl']
            }
        ]
    }
}
results = smclient.search(**search_params)
```

After you find your training job, you can investigate the resources used to train the model.
Use Machine Learning Frameworks with Amazon SageMaker

The Amazon SageMaker Python SDK provides open source APIs and containers that make it easy to train and deploy models in Amazon SageMaker with several different machine learning and deep learning frameworks. For general information about the Amazon SageMaker Python SDK, see https://github.com/aws/sagemaker-python-sdk. For information about using specific frameworks in Amazon SageMaker, see the following topics:

**Topics**

- Use Apache Spark with Amazon SageMaker (p. 381)
- Use TensorFlow with Amazon SageMaker (p. 390)
- Use Apache MXNet with Amazon SageMaker (p. 390)
- Use Scikit-learn with Amazon SageMaker (p. 390)
- Use PyTorch with Amazon SageMaker (p. 391)
- Use Chainer with Amazon SageMaker (p. 391)

**Use Apache Spark with Amazon SageMaker**

This section provides information for developers who want to use Apache Spark for preprocessing data and Amazon SageMaker for model training and hosting. For information about supported versions of Apache Spark, see https://github.com/aws/sagemaker-spark.

Amazon SageMaker provides an Apache Spark library, in both Python and Scala, that you can use to easily train models in Amazon SageMaker using `org.apache.spark.sql.DataFrame` data frames in your Spark clusters. After model training, you can also host the model using Amazon SageMaker hosting services.

The Amazon SageMaker Spark library, `com.amazonaws.services.sagemaker.sparksdk`, provides the following classes, among others:

- `SageMakerEstimator`—Extends the `org.apache.spark.ml.Estimator` interface. You can use this estimator for model training in Amazon SageMaker.
- `SageMakerModel`—Extends the `org.apache.spark.ml.Model` class. You can use this `SageMakerModel` for model hosting and obtaining inferences in Amazon SageMaker.

**Download the Amazon SageMaker Spark Library**

You have the following options for downloading the Spark library provided by Amazon SageMaker:

- You can download the source code for both PySpark and Scala libraries from GitHub at https://github.com/aws/sagemaker-spark.
For the Python Spark library, you have the following additional options:

- Use pip install:

```
$ pip install sagemaker_pyspark
```

- In a notebook instance, create a new notebook that uses either the Sparkmagic (PySpark) or the Sparkmagic (PySpark3) kernel and connect to a remote Amazon EMR cluster. For more information, see Build Amazon SageMaker Notebooks Backed by Spark in Amazon EMR.

**Note**
The EMR cluster must be configured with an IAM role that has the AmazonSageMakerFullAccess policy attached. For information about configuring roles for an EMR cluster, see Configure IAM Roles for Amazon EMR Permissions to AWS Services in the Amazon EMR Management Guide.

- You can get the Scala library from Maven. Add the Spark library to your project by adding the following dependency to your `pom.xml` file:

```xml
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>sagemaker-spark_2.11</artifactId>
    <version>spark_2.2.0-1.0</version>
</dependency>
```

## Integrate Your Apache Spark Application with Amazon SageMaker

The following is high-level summary of the steps for integrating your Apache Spark application with Amazon SageMaker.

1. **Continue data preprocessing using the Apache Spark library that you are familiar with.** Your dataset remains a DataFrame in your Spark cluster.

   **Note**
   Load your data into a DataFrame and preprocess it so that you have a features column with `org.apache.spark.ml.linalg.Vector` of doubles, and an optional label column with values of `Double` type.

2. **Use the estimator in the Amazon SageMaker Spark library to train your model.** For example, if you choose the k-means algorithm provided by Amazon SageMaker for model training, you call the `KMeansSageMakerEstimator.fit` method.

   Provide your DataFrame as input. The estimator returns a `SageMakerModel` object.

   **Note**
   `SageMakerModel` extends the `org.apache.spark.ml.Model`.

   The `fit` method does the following:

   a. Converts the input DataFrame to the protobuf format by selecting the features and label columns from the input DataFrame and uploading the protobuf data to an Amazon S3 bucket. The protobuf format is efficient for model training in Amazon SageMaker.

   b. Starts model training in Amazon SageMaker by sending an Amazon SageMaker CreateTrainingJob request. After model training has completed, Amazon SageMaker saves the model artifacts to an S3 bucket.
Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark

Topics

- Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark (p. 387)
- Use the SageMakerEstimator in a Spark Pipeline (p. 388)

Amazon SageMaker provides an Apache Spark library (in both Python and Scala) that you can use to integrate your Apache Spark applications with Amazon SageMaker. For example, you might use Apache Spark for data preprocessing and Amazon SageMaker for model training and hosting. For more information, see Use Apache Spark with Amazon SageMaker (p. 381). This section provides example code that uses the Apache Spark Scala library provided by Amazon SageMaker to train a model in Amazon SageMaker using DataFrames in your Spark cluster. The example also hosts the resulting model artifacts using Amazon SageMaker hosting services. Specifically, this example does the following:

- Uses the KMeansSageMakerEstimator to fit (or train) a model on data

Because the example uses the k-means algorithm provided by Amazon SageMaker to train a model, you use the KMeansSageMakerEstimator. You train the model using images of handwritten single-digit numbers (from the MNIST dataset). You provide the images as an input DataFrame. For your convenience, Amazon SageMaker provides this dataset in an S3 bucket.

In response, the estimator returns a SageMakerModel object.

- Obtains inferences using the trained SageMakerModel
To get inferences from a model hosted in Amazon SageMaker, you call the `SageMakerModel.transform` method. You pass a `DataFrame` as input. The method transforms the input `DataFrame` to another `DataFrame` containing inferences obtained from the model.

For a given input image of a handwritten single-digit number, the inference identifies a cluster that the image belongs to. For more information, see K-Means Algorithm (p. 137).

This is the example code:

```scala
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test")

val roleArn = "arn:aws:iam::account-id:role/rolename"

val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(784)

// train
val model = estimator.fit(trainingData)

val transformedData = model.transform(testData)
transformedData.show
```

The code does the following:

- Loads the MNIST dataset from an S3 bucket provided by Amazon SageMaker (awsai-sparksdk-dataset) into a Spark DataFrame (mnistTrainingDataFrame):

```scala
// Get a Spark session.
val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test")
```
```
val roleArn = "arn:aws:iam::account-id:role/rolename"
trainingData.show()
```

The `show` method displays the first 20 rows in the data frame:

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  5.0|(784,[152,153,154...|
|  0.0|(784,[127,128,129...|
|  4.0|(784,[160,161,162...|
|  1.0|(784,[158,159,160...|
|  9.0|(784,[208,209,210...|
|  2.0|(784,[155,156,157...|
|  1.0|(784,[124,125,126...|
|  3.0|(784,[151,152,153...|
|  1.0|(784,[152,153,154...|
|  4.0|(784,[134,135,161...|
|  3.0|(784,[123,124,125...|
|  5.0|(784,[216,217,218...|
|  3.0|(784,[143,144,145...|
|  6.0|(784,[72,73,74,99...|
|  1.0|(784,[151,152,153...|
|  7.0|(784,[211,212,213...|
|  2.0|(784,[151,152,153...|
|  8.0|(784,[159,160,161...|
|  6.0|(784,[100,101,102...|
|  9.0|(784,[209,210,211...|
+-----+--------------------+
```

In each row:
- The `label` column identifies the image's label. For example, if the image of the handwritten number is the digit 5, the label value is 5.
- The `features` column stores a vector (org.apache.spark.ml.linalg.Vector) of Double values. These are the 784 features of the handwritten number. (Each handwritten number is a 28 x 28-pixel image, making 784 features.)

- Creates an Amazon SageMaker estimator (KMeansSageMakerEstimator)

The `fit` method of this estimator uses the k-means algorithm provided by Amazon SageMaker to train models using an input DataFrame. In response, it returns a SageMakerModel object that you can use to get inferences.

```
val estimator = new KMeansSageMakerEstimator(
    sagemakerRole = IAMRole(roleArn),
    trainingInstanceType = "ml.p2.xlarge",
    trainingInstanceCount = 1,
    endpointInstanceType = "ml.c4.xlarge",
    endpointInitialInstanceCount = 1)
    .setK(10).setFeatureDim(784)
```

The constructor parameters provide information that is used for training a model and deploying it on Amazon SageMaker:
• **trainingInstanceType** and **trainingInstanceCount**—Identify the type and number of ML compute instances to use for model training.

• **endpointInstanceType**—Identifies the ML compute instance type to use when hosting the model in Amazon SageMaker. By default, one ML compute instance is assumed.

• **endpointInitialInstanceCount**—Identifies the number of ML compute instances initially backing the endpoint hosting the model in Amazon SageMaker.

• **sagemakerRole**—Amazon SageMaker assumes this IAM role to perform tasks on your behalf. For example, for model training, it reads data from S3 and writes training results (model artifacts) to S3.

  **Note**
  This example implicitly creates an Amazon SageMaker client. To create this client, you must provide your credentials. The API uses these credentials to authenticate requests to Amazon SageMaker. For example, it uses the credentials to authenticate requests to create a training job and API calls for deploying the model using Amazon SageMaker hosting services.

  • After the `KMeansSageMakerEstimator` object has been created, you set the following parameters, used in model training:
    • The number of clusters that the k-means algorithm should create during model training. You specify 10 clusters, one for each digit, 0 through 9.
    • Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel image, making 784 features).

• Calls the estimator `fit` method

```scala
// train
val model = estimator.fit(trainingData)
```

You pass the input `DataFrame` as a parameter. The model does all the work of training the model and deploying it to Amazon SageMaker. For more information see, Integrate Your Apache Spark Application with Amazon SageMaker (p. 382). In response, you get a `SageMakerModel` object, which you can use to get inferences from your model deployed in Amazon SageMaker.

You provide only the input `DataFrame`. You don't need to specify the registry path to the k-means algorithm used for model training because the `KMeansSageMakerEstimator` knows it.

• Calls the `SageMakerModel.transform` method to get inferences from the model deployed in Amazon SageMaker.

  The `transform` method takes a `DataFrame` as input, transforms it, and returns another `DataFrame` containing inferences obtained from the model.

```scala
val transformedData = model.transform(testData)
transformedData.show
```

For simplicity, we use the same `DataFrame` as input to the `transform` method that we used for model training in this example. The `transform` method does the following:
• Serializes the features column in the input DataFrame to protobuf and sends it to the Amazon SageMaker endpoint for inference.
• Deserializes the protobuf response into the two additional columns (distance_to_cluster and closest_cluster) in the transformed DataFrame.

The show method gets inferences to the first 20 rows in the input DataFrame:

<table>
<thead>
<tr>
<th>label</th>
<th>features</th>
<th>distance_to_cluster</th>
<th>closest_cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>[784, 152, 153, ...]</td>
<td>1767.897705078125</td>
<td>4.0</td>
</tr>
<tr>
<td>0.0</td>
<td>[784, 127, 128, ...]</td>
<td>1392.157470703125</td>
<td>5.0</td>
</tr>
<tr>
<td>4.0</td>
<td>[784, 160, 161, ...]</td>
<td>1671.5711669921875</td>
<td>9.0</td>
</tr>
<tr>
<td>1.0</td>
<td>[784, 158, 159, ...]</td>
<td>1182.6082763671875</td>
<td>6.0</td>
</tr>
<tr>
<td>9.0</td>
<td>[784, 208, 209, ...]</td>
<td>1390.4002685546875</td>
<td>0.0</td>
</tr>
<tr>
<td>2.0</td>
<td>[784, 155, 156, ...]</td>
<td>1713.988037109375</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>[784, 124, 125, ...]</td>
<td>1246.3016357421875</td>
<td>2.0</td>
</tr>
<tr>
<td>3.0</td>
<td>[784, 151, 152, ...]</td>
<td>1753.229248046875</td>
<td>4.0</td>
</tr>
<tr>
<td>1.0</td>
<td>[784, 152, 153, ...]</td>
<td>978.8394165039062</td>
<td>2.0</td>
</tr>
<tr>
<td>4.0</td>
<td>[784, 134, 135, ...]</td>
<td>1623.176513671875</td>
<td>3.0</td>
</tr>
<tr>
<td>3.0</td>
<td>[784, 123, 124, ...]</td>
<td>1533.863525390625</td>
<td>4.0</td>
</tr>
<tr>
<td>5.0</td>
<td>[784, 216, 217, ...]</td>
<td>1469.357177734375</td>
<td>6.0</td>
</tr>
<tr>
<td>3.0</td>
<td>[784, 143, 144, ...]</td>
<td>1736.765869140625</td>
<td>4.0</td>
</tr>
<tr>
<td>6.0</td>
<td>[784, 72, 73, 74, ...]</td>
<td>1473.69384765625</td>
<td>8.0</td>
</tr>
<tr>
<td>1.0</td>
<td>[784, 151, 152, ...]</td>
<td>944.88720703125</td>
<td>2.0</td>
</tr>
<tr>
<td>7.0</td>
<td>[784, 211, 212, ...]</td>
<td>1285.9071044921875</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0</td>
<td>[784, 151, 152, ...]</td>
<td>1635.0125732421875</td>
<td>1.0</td>
</tr>
<tr>
<td>8.0</td>
<td>[784, 159, 160, ...]</td>
<td>1436.3162841796875</td>
<td>6.0</td>
</tr>
<tr>
<td>6.0</td>
<td>[784, 100, 101, ...]</td>
<td>1499.7366943359375</td>
<td>7.0</td>
</tr>
<tr>
<td>9.0</td>
<td>[784, 209, 210, ...]</td>
<td>1364.6319580078125</td>
<td>6.0</td>
</tr>
</tbody>
</table>

You can interpret the data, as follows:
• A handwritten number with the label 5 belongs to cluster 5 (closest_cluster).
• A handwritten number with the label 0 belongs to cluster 2.
• A handwritten number with the label 4 belongs to cluster 4.
• A handwritten number with the label 1 belongs to cluster 1.

For more information on how to run these examples, see https://github.com/aws/sagemaker-spark/blob/master/README.md on GitHub.

Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark

In Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark (p. 383), you use the kMeansSageMakerEstimator because the example uses the k-means algorithm provided by Amazon SageMaker for model training. You might choose to use your own custom algorithm for model training instead. Assuming that you have already created a Docker image, you can create your own SageMakerEstimator and specify the Amazon Elastic Container Registry path for your custom image.

The following example shows how to create a KMeansSageMakerEstimator from the SageMakerEstimator. In the new estimator, you explicitly specify the Docker registry path to your training and inference code images.

```java
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.SageMakerEstimator
```
import com.amazonaws.services.sagemaker.sparksdk.transformation.serializers.ProtobufRequestRowSerializer
import com.amazonaws.services.sagemaker.sparksdk.transformation.deserializers.KMeansProtobufResponseRowDeserializer

val estimator = new SageMakerEstimator(
  trainingImage = "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  modelImage = "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  requestRowSerializer = new ProtobufRequestRowSerializer(),
  responseRowDeserializer = new KMeansProtobufResponseRowDeserializer(),
  hyperParameters = Map("k" -> "10", "feature_dim" -> "784"),
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1,
  trainingSparkDataFormat = "sagemaker")

In the code, the parameters in the SageMakerEstimator constructor include:

- **trainingImage** —Identifies the Docker registry path to the training image containing your custom code.
- **modelImage** —Identifies the Docker registry path to the image containing inference code.
- **requestRowSerializer** —Implements com.amazonaws.services.sagemaker.sparksdk.transformation.RequestRowSerializer. This parameter serializes rows in the input DataFrame to send them to the model hosted in Amazon SageMaker for inference.
- **responseRowDeserializer** —Implements com.amazonaws.services.sagemaker.sparksdk.transformation.ResponseRowDeserializer. This parameter deserializes responses from the model, hosted in Amazon SageMaker, back into a DataFrame.
- **trainingSparkDataFormat** —Specifies the data format that Spark uses when uploading training data from a DataFrame to S3. For example, "sagemaker" for protobuf format, "csv" for comma-separated values, and "libsvm" for LibSVM format.

You can implement your own RequestRowSerializer and ResponseRowDeserializer to serialize and deserialize rows from a data format that your inference code supports, such as .libsvm or .csv.

**Use the SageMakerEstimator in a Spark Pipeline**

You can use org.apache.spark.ml.Estimators and models, and SageMakerEstimator and SageMakerModel models in org.apache.spark.ml.Pipeline pipelines, as shown in the following example:

```scala
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.PCA
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
```
Example 1: Amazon SageMaker with Apache Spark

```scala
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

// substitute your SageMaker IAM role here
val roleArn = "arn:aws:iam::account-id:role/rolename"

val pcaEstimator = new PCA()
  .setInputCol("features")
  .setOutputCol("projectedFeatures")
  .setK(50)

val kMeansSageMakerEstimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(integTestingRole),
  requestRowSerializer =
    new ProtobufRequestRowSerializer(featuresColumnName = "projectedFeatures"),
  trainingSparkDataFormatOptions = Map("featuresColumnName" -> "projectedFeatures"),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(50)

val pipeline = new Pipeline().setStages(Array(pcaEstimator, kMeansSageMakerEstimator))

// train
val pipelineModel = pipeline.fit(trainingData)
val transformedData = pipelineModel.transform(testData)
transformedData.show()
```

The parameter `trainingSparkDataFormatOptions` configures Spark to serialize to protobuf the "projectedFeatures" column for model training. Additionally, Spark serializes to protobuf the "label" column by default.

Because we want to make inferences using the "projectedFeatures" column, we pass the column name into the `ProtobufRequestRowSerializer`.

The following example shows a transformed DataFrame:

<table>
<thead>
<tr>
<th>label</th>
<th>features</th>
<th>projectedFeatures</th>
<th>distance_to_cluster</th>
<th>closest_cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>784,[152,153,154...</td>
<td>[880.731433034386...</td>
<td>1500.470703125</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>784,[127,128,129...</td>
<td>[1768.51722024166...</td>
<td>1142.18359375</td>
<td>4.0</td>
</tr>
<tr>
<td>4.0</td>
<td>784,[160,161,162...</td>
<td>[704.949236329314...</td>
<td>1386.246826171875</td>
<td>9.0</td>
</tr>
<tr>
<td>1.0</td>
<td>784,[158,159,160...</td>
<td>[-42.328192193771...</td>
<td>1277.07368394875</td>
<td>5.0</td>
</tr>
<tr>
<td>9.0</td>
<td>784,[208,209,210...</td>
<td>[374.04390202833...</td>
<td>1211.00927734375</td>
<td>3.0</td>
</tr>
<tr>
<td>2.0</td>
<td>784,[155,156,157...</td>
<td>[941.26771452885...</td>
<td>1496.15795894875</td>
<td>8.0</td>
</tr>
<tr>
<td>1.0</td>
<td>784,[124,125,126...</td>
<td>[30.2848596410594...</td>
<td>1327.6766357421875</td>
<td>5.0</td>
</tr>
<tr>
<td>3.0</td>
<td>784,[151,152,153...</td>
<td>[1270.14374062052...</td>
<td>1570.7674560546875</td>
<td>0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>784,[152,153,154...</td>
<td>[-112.10792566485...</td>
<td>1037.568359375</td>
<td>5.0</td>
</tr>
<tr>
<td>4.0</td>
<td>784,[134,135,161...</td>
<td>[452.068280676606...</td>
<td>1165.12365725625</td>
<td>3.0</td>
</tr>
<tr>
<td>3.0</td>
<td>784,[123,124,125...</td>
<td>[610.596447285397...</td>
<td>1325.953369140625</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>784,[216,217,218...</td>
<td>[142.959601818422...</td>
<td>1353.4930419921875</td>
<td>5.0</td>
</tr>
<tr>
<td>3.0</td>
<td>784,[143,144,145...</td>
<td>[1036.71862533658...</td>
<td>1460.4315185546875</td>
<td>7.0</td>
</tr>
<tr>
<td>6.0</td>
<td>784,[72,73,74,99...</td>
<td>[996.740157435754...</td>
<td>1159.8631591796875</td>
<td>2.0</td>
</tr>
<tr>
<td>1.0</td>
<td>784,[151,152,153...</td>
<td>[-107.26076167417...</td>
<td>960.96362046875</td>
<td>5.0</td>
</tr>
<tr>
<td>7.0</td>
<td>784,[211,212,213...</td>
<td>[619.771820430940...</td>
<td>1245.13623046875</td>
<td>6.0</td>
</tr>
<tr>
<td>2.0</td>
<td>784,[151,152,153...</td>
<td>[850.152101817161...</td>
<td>1304.437744140625</td>
<td>8.0</td>
</tr>
</tbody>
</table>
Use TensorFlow with Amazon SageMaker


For information about TensorFlow versions supported by the Amazon SageMaker TensorFlow container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst.

For information about how to build and contribute to the Amazon SageMaker TensorFlow container, see the GitHub repository at https://github.com/aws/sagemaker-tensorflow-container.

Use Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom MXNet code. The Amazon SageMaker Python SDK MXNet estimators and models and the Amazon SageMaker open-source MXNet container make writing a MXNet script and running it in Amazon SageMaker easier. For information about writing MXNet training scripts and using MXNet estimators and models with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk#mxnet-sagemaker-estimators.

For information about MXNet versions supported by the Amazon SageMaker MXNet container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst.

For information about how to build and contribute to the Amazon SageMaker MXNet container, see the GitHub repository at https://github.com/aws/sagemaker-mxnet-container.

Use Scikit-learn with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Scikit-learn code. The Amazon SageMaker Python SDK Scikit-learn estimators and models and the Amazon SageMaker open-source Scikit-learn container make writing a Scikit-learn script and running it in Amazon SageMaker easier. For information about writing Scikit-learn training scripts and using Scikit-learn estimators and models with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk#scikit-learn-sagemaker-estimators.
For information about Scikit-learn versions supported by the Amazon SageMaker Scikit-learn container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/sklearn/README.rst.

For information about how to build and contribute to the Amazon SageMaker Scikit-learn container, see the GitHub repository at https://github.com/aws/sagemaker-scikit-learn-container.

Use PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. The Amazon SageMaker Python SDK PyTorch estimators and models and the Amazon SageMaker open-source PyTorch container make writing a PyTorch script and running it in Amazon SageMaker easier. For information about writing PyTorch training scripts and using PyTorch estimators and models with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk#pytorch-sagemaker-estimators.

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst.

For information about how to build and contribute to the Amazon SageMaker PyTorch container, see the GitHub repository at https://github.com/aws/sagemaker-pytorch-container.

Use Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. The Amazon SageMaker Python SDK Chainer estimators and models and the Amazon SageMaker open-source Chainer container make writing a Chainer script and running it in Amazon SageMaker easier. For information about writing Chainer training scripts and using Chainer estimators and models with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk#chainer-sagemaker-estimators.

For information about Chainer versions supported by the Amazon SageMaker Chainer container, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/chainer/README.rst.

For information about how to build and contribute to the Amazon SageMaker Chainer container, see the GitHub repository at https://github.com/aws/sagemaker-chainer-container.
Reinforcement Learning with Amazon SageMaker RL

Reinforcement learning (RL) is a machine learning technique that attempts to learn a strategy, called a policy, that optimizes an objective for an agent acting in an environment. For example, the agent might be a robot, the environment might be a maze, and the goal might be to successfully navigate the maze in the smallest amount of time. In RL, the agent takes an action, observes the state of the environment, and gets a reward based on the value of the current state of the environment. The goal is to maximize the long-term reward that the agent receives as a result of its actions. RL is well-suited for solving problems where an agent can make autonomous decisions.

Topics
- Why is Reinforcement Learning Important? (p. 392)
- Markov Decision Process (MDP) (p. 392)
- Key Features of Amazon SageMaker RL (p. 393)
- Sample RL Workflow Using Amazon SageMaker RL (p. 395)
- RL Environments in Amazon SageMaker (p. 396)
- Distributed Training with Amazon SageMaker RL (p. 397)
- Hyperparameter Tuning with Amazon SageMaker RL (p. 398)

Why is Reinforcement Learning Important?

RL is well-suited for solving large, complex problems. For example, supply chain management, HVAC systems, industrial robotics, game artificial intelligence, dialog systems, and autonomous vehicles. Because RL models learn by a continuous process of receiving rewards and punishments for every action taken by the agent, it is possible to train systems to make decisions under uncertainty and in dynamic environments.

Markov Decision Process (MDP)

RL is based on models called Markov Decision Processes (MDPs). An MDP consists of a series of time steps. Each time step consists of the following:

Environment

Defines the space in which the RL model operates. This can be either a real-world environment or a simulator. For example, if you train a physical autonomous vehicle on a physical road, that would be a real-world environment. If you train a computer program that models an autonomous vehicle driving on a road, that would be a simulator.

State

Specifies all information about the environment and past steps that is relevant to the future. For example, in an RL model in which a robot can move in any direction at any time step, then the
position of the robot at the current time step is the state, because if we know where the robot is, it
isn't necessary to know the steps it took to get there.

Action

What the agent does. For example, the robot takes a step forward.

Reward

A number that represents the value of the state that resulted from the last action that the agent
took. For example, if the goal is for a robot to find treasure, the reward for finding treasure might be
5, and the reward for not finding treasure might be 0. The RL model attempts to find a strategy that
optimizes the cumulative reward over the long term. This strategy is called a policy.

Observation

Information about the state of the environment that is available to the agent at each step. This
might be the entire state, or it might be just a part of the state. For example, the agent in a chess-
playing model would be able to observe the entire state of the board at any step, but a robot in a
maze might only be able to observe a small portion of the maze that it currently occupies.

Typically, training in RL consists of many episodes. An episode consists of all of the time steps in an MDP
from the initial state until the environment reaches the terminal state.

Key Features of Amazon SageMaker RL

To train RL models in Amazon SageMaker RL, use the following components:

- A deep learning (DL) framework. Currently, Amazon SageMaker supports RL in TensorFlow and Apache
  MXNet.
- An RL toolkit. An RL toolkit manages the interaction between the agent and the environment,
  and provides a wide selection of state of the art RL algorithms. Amazon SageMaker supports
  the Intel Coach and Ray RLlib toolkits. For information about Intel Coach, see https://
  nervanasystems.github.io/coach/. For information about Ray RLlib, see https://ray.readthedocs.io/en/
  latest/rllib.html.
- An RL environment. You can use custom environments, open-source environments, or commercial
  environments. For information, see RL Environments in Amazon SageMaker (p. 396).

The following diagram shows the RL components that are supported in Amazon SageMaker RL.
Key Features of Amazon SageMaker RL

- End-to-end examples for classic RL and real-world problems
  - Robotics
  - Industrial Control
  - HVAC
  - Autonomous Vehicles
  - Operations

- AWS Simulation Environments
  - Amazon Sumerian
  - AWS RoboMaker

- Open Source Environments
  - EnergyPlus
  - RoboSchool
  - PyBullet

- OpenAI Gym

- RL Toolkits that provide RL agent algorithm implementations
  - RL-Coch
    - DQN
    - PPO
    - HER
    - Rainbow
    - ...
  - RL-Ray RLLib
    - APEX
    - ES
    - IMPALA

- SageMaker Deep Learning Frameworks
  - TensorFlow
  - MxNet
  - PyTorch

- Training Options
  - Single Machine / Distributed
  - Local / Remote simulation

- SageMaker supported
- Customer BYO
Sample RL Workflow Using Amazon SageMaker RL

The following example describes the steps for developing RL models using Amazon SageMaker RL.

For complete code examples, see the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning.

1. **Formulate the RL problem**—First, formulate the business problem into an RL problem. For example, auto scaling enables services to dynamically increase or decrease capacity depending on conditions that you define. Currently, this requires setting up alarms, scaling policies, and thresholds, and other manual steps. To solve this with RL, we define the components of the Markov Decision Process:
   a. **Objective**—Scale instance capacity so that it matches the desired load profile.
   b. **Environment**—A custom environment that includes the load profile. It generates a simulated load with daily and weekly variations and occasional spikes. The simulated system has a delay between when new resources are requested and when they become available for serving requests.
   c. **State**—The current load, number of failed jobs, and number of active machines
   d. **Action**—Remove, add, or keep the same number of instances.
   e. **Reward**—A positive reward for successful transactions, a high penalty for failing transactions beyond a specified threshold.

2. **Define the RL environment**—The RL environment can be the real world where the RL agent interacts or a simulation of the real world. You can connect open source and custom environments developed using Gym interfaces, and commercial simulation environments such as MATLAB and Simulink.

3. **Define the presets**—The presets configure the RL training jobs and define the hyperparameters for the RL algorithms.

4. **Write the training code**—Write training code as a Python script and pass the script to an Amazon SageMaker training job. In your training code, import the environment files and the preset files, and then define the `main()` function.

5. **Train the RL Model**—Use the Amazon SageMaker RLEstimator in the Amazon SageMaker Python SDK to start an RL training job. If you are using local mode, the training job runs on the notebook instance. When you use Amazon SageMaker for training, you can select GPU or CPU instances. Store the output from the training job in a local directory if you train in local mode, or on Amazon S3 if you use Amazon SageMaker training.

   For information about using the Amazon SageMaker Python SDK for RL, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/rl/README.rst.

   The RLEstimator requires the following information as parameters.
   a. The source directory where the environment, presets, and training code are uploaded.
   b. The path to the training script.
   c. The RL toolkit and deep learning framework you want to use. This automatically resolves to the Amazon ECR path for the RL container.
   d. The training parameters, such as the instance count, job name, and S3 path for output.
   e. Metric definitions that you want to capture in your logs. These can also be visualized in CloudWatch and in Amazon SageMaker notebooks.

6. **Visualize training metrics and output**—After a training job that uses an RL model completes, you can view the metrics you defined in the training jobs in CloudWatch. You can also plot the metrics in a notebook by using the Amazon SageMaker Python SDK analytics library. Visualizing metrics helps you understand how the performance of the model as measured by the reward improves over time.
Note
If you train in local mode, you can't visualize metrics in CloudWatch.

7. **Evaluate the model**—Checkpoinited data from the previously trained models can be passed on for evaluation and inference in the checkpoint channel. In local mode, use the local directory. In Amazon SageMaker training mode, you need to upload the data to s3 first.

8. **Deploy RL models**—Finally, deploy the trained model on an endpoint hosted on Amazon SageMaker or on an Edge device by using AWS IoT Greengrass.

---

**RL Environments in Amazon SageMaker**

Amazon SageMaker RL uses environments to mimic real-world scenarios. Given the current state of the environment and an action taken by the agent or agents, the simulator processes the impact of the action, and returns the next state and a reward. Simulators are useful in cases where it is not safe to train an agent in the real world (for example, flying a drone) or if the RL algorithm takes a long time to converge (for example, when playing chess).

The following diagram shows an example of the interactions with a simulator for a car racing game.

![Simulation Environment Diagram](image)

The simulation environment consists of an agent and a simulator. Here, a convolutional neural network (CNN) consumes images from the simulator and generates actions to control the game controller. With multiple simulations, this environment generates training data of the form $state_t, action, state_{t+1}$, and $reward_{t+1}$. Defining the reward is not trivial and impacts the RL model quality. We want to provide a few examples of reward functions, but would like to make it user-configurable.

**Topics**
- Use OpenAI Gym Interface for Environments in Amazon SageMaker RL (p. 397)
- Use Open Source Environments (p. 397)
- Use Commercial Environments (p. 397)
Use OpenAI Gym Interface for Environments in Amazon SageMaker RL

To use OpenAI Gym environments in Amazon SageMaker RL, use the following API elements. For more information about OpenAI Gym, see https://gym.openai.com/docs/.

- `env.action_space`—Defines the actions the agent can take, specifies whether each action is continuous or discrete, and specifies the minimum and maximum if the action is continuous.
- `env.observation_space`—Defines the observations the agent receives from the environment, as well as minimum and maximum for continuous observations.
- `env.reset()`—Initializes a training episode. The `reset()` function returns the initial state of the environment, and the agent uses the initial state to take its first action. The action is then sent to the `step()` repeatedly until the episode reaches a terminal state. When `step()` returns `done = True`, the episode ends. The RL toolkit re-initializes the environment by calling `reset()`.
- `step()`—Takes the agent action as input and outputs the next state of the environment, the reward, whether the episode has terminated, and an `info` dictionary to communicate debugging information. It is the responsibility of the environment to validate the inputs.
- `env.render()`—Used for environments that have visualization. The RL toolkit calls this function to capture visualizations of the environment after each call to the `step()` function.

Use Open Source Environments

You can use open source environments, such as EnergyPlus and RoboSchool, in Amazon SageMaker RL by building your own container. For more information about EnergyPlus, see https://energyplus.net/. For more information about RoboSchool, see https://github.com/openai/roboschool. The HVAC and RoboSchool examples in the samples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning show how to build a custom container to use with Amazon SageMaker RL:

Use Commercial Environments

You can use commercial environments, such as MATLAB and Simulink, in Amazon SageMaker RL by building your own container. You need to manage your own licenses.

Distributed Training with Amazon SageMaker RL

Amazon SageMaker RL supports multi-core and multi-instance distributed training. Depending on your use case, training and/or environment rollout can be distributed. For example, Amazon SageMaker RL works for the following distributed scenarios:

- Single training instance and multiple rollout instances of the same instance type. For an example, see the Neural Network Compression example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance and multiple rollout instances, where different instance types for training and rollouts. For an example, see the AWS DeepRacer / AWS RoboMaker example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance that uses multiple cores for rollout. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning. This is useful if the simulation environment is light-weight and can run on a single thread.
• Multiple instances for training and rollouts. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.

Hyperparameter Tuning with Amazon SageMaker RL

You can run a hyperparameter tuning job to optimize hyperparameters for Amazon SageMaker RL. The Roboschool example in the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning shows how you can do this with RL Coach. The launcher script shows how you can abstract parameters from the Coach preset file and optimize them.
Authentication and Access Control for Amazon SageMaker

Access to Amazon SageMaker requires credentials. Those credentials must have permissions to access AWS resources, such as an Amazon SageMaker notebook instance or an Amazon Elastic Compute Cloud (Amazon EC2) instance. The following sections provide details on how you can use AWS Identity and Access Management (IAM) and Amazon SageMaker to help secure access to your resources.

- Authentication (p. 399)
- Access Control (p. 400)

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create the in Amazon SageMaker). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don’t use AWS tools, you must sign the request yourself. Amazon SageMaker supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

- **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon SageMaker resources. For example, you must have permissions to create an Amazon SageMaker notebook instance.

The following sections describe how to manage permissions for Amazon SageMaker. We recommend that you read the overview first.

- Overview of Managing Access Permissions to Your Amazon SageMaker Resources (p. 400)
- Using Identity-based Policies (IAM Policies) for Amazon SageMaker (p. 404)
- Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 411)

Overview of Managing Access Permissions to Your Amazon SageMaker Resources

Every AWS resource is owned by an AWS account, and permissions to create or access a resource are governed by permissions policies. An account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles). Some services (such as AWS Lambda) also support attaching permissions policies to resources.

**Note**

An account administrator (or administrator user) is a user with administrator privileges. For more information, see IAM Best Practices in the IAM User Guide.

When granting permissions, you decide who is getting the permissions, the resources they get permissions for, and the specific actions that you want to allow on those resources.

**Topics**

- Amazon SageMaker Resources and Operations (p. 401)
- Understanding Resource Ownership (p. 401)
- Managing Access to Resources (p. 402)
- Specifying Policy Elements: Resources, Actions, Effects, and Principals (p. 403)
• Specifying Conditions in a Policy (p. 404)

Amazon SageMaker Resources and Operations

In Amazon SageMaker, the primary resource is a *notebook instance*. Amazon SageMaker also supports additional resource types: *training jobs*, *models*, *endpoint configurations*, *endpoints*, and *tags*. These additional resources are referred to as *subresources*. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to.

Except for *tags*, these resources and subresources have unique ARNs associated with them, as shown in the following table. *Tags* use the ARN of the resource that they are modifying. For example, when used on a model, the *AddTag* action uses the same ARN as the model resource.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td>arn:aws:sagemaker:region:account-id:endpoint/endpointName</td>
</tr>
<tr>
<td>Endpoint Config</td>
<td>arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName</td>
</tr>
<tr>
<td>Hyperparameter Tuning Job</td>
<td>arn:aws:sagemaker:region:account-id:hyper-parameter-tuning-job/hyperParameterTuningJobName</td>
</tr>
<tr>
<td>Model</td>
<td>arn:aws:sagemaker:region:account-id:model/modelName</td>
</tr>
<tr>
<td>Notebook Instance</td>
<td>arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName</td>
</tr>
<tr>
<td>Notebook Instance Lifecycle Configuration</td>
<td>arn:aws:sagemaker:region:account-id:notebook-instance-lifecycle-config/notebookInstanceLifecycleConfigName</td>
</tr>
<tr>
<td>Training Job</td>
<td>arn:aws:sagemaker:region:account-id:training-job/trainingJobName</td>
</tr>
</tbody>
</table>

Amazon SageMaker provides a set of operations to work with Amazon SageMaker resources. For a list of available operations, see the Amazon SageMaker API Reference (p. 506).

Understanding Resource Ownership

The AWS account owns the resources that are created in the account, regardless of who created the resources. Specifically, the resource owner is the AWS account of the principal entity (that is, the root account, an IAM user, or an IAM role) that authenticates the resource creation request. The following examples illustrate how this works:

• If you use the root account credentials of your AWS account to create a notebook instance, your AWS account is the owner of the resource (in Amazon SageMaker, the resource is a notebook instance).
• If you create an IAM user in your AWS account and grant permissions to create a notebook instance to that user, the user can create a notebook instance. However, your AWS account, to which the user belongs, owns the notebook instance resource.
• If you create an IAM role in your AWS account with permissions to create a notebook instance, anyone who can assume the role can create a notebook instance. Your AWS account, to which the user belongs, owns the notebook instance resource.
Managing Access to Resources

A permissions policy describes who has access to what. The following section explains the options for creating permissions policies.

**Note**
This section discusses using IAM in the context of Amazon SageMaker. It doesn’t provide detailed information about the IAM service. For complete IAM documentation, see What Is IAM? in the IAM User Guide. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference in the IAM User Guide.

Permissions policies attached to an IAM identity are referred to as identity-based policies (IAM policies). Permissions policies attached to a resource are referred to as resource-based policies. Amazon SageMaker supports only identity-based permissions policies.

**Topics**
- Identity-Based Policies (IAM Policies) (p. 402)
- Resource-Based Policies (p. 403)

### Identity-Based Policies (IAM Policies)

You can attach permissions policies to IAM identities. For example, you can do the following:

- **Attach a permissions policy to a user or a group in your account** – To grant a user permissions to create a Amazon SageMaker resource, such as a notebook instance, you can attach a permissions policy to a user or to a group that the user belongs to.

- **Attach a permissions policy to a role (grant cross-account permissions)** – You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in account A can create a role to grant cross-account permissions to another AWS account (for example, account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in account A.
  2. Account A administrator attaches a trust policy to the role identifying account B as the principal who can assume the role.
  3. Account B administrator can then delegate permissions to assume the role to any users in account B. Doing this allows users in account B to create or access resources in account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see Access Management in the IAM User Guide.

Some of the Amazon SageMaker actions (for example, CreateTrainingJob) require the user to pass an IAM role to Amazon SageMaker so that the service can assume that role and its permissions. To pass a role (and its permissions) to an AWS service, a user must have permissions to pass the role to the service. To allow a user to pass a role to an AWS service, you must grant permission for the iam:PassRole action. For more information, see Granting a User Permissions to Pass a Role to an AWS Service in the IAM User Guide.

The following is an example permission policy. You attach it to an IAM user.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
```
For more information about using identity-based policies with Amazon SageMaker, see Using Identity-based Policies (IAM Policies) for Amazon SageMaker (p. 404). For more information about users, groups, roles, and permissions, see Identities (Users, Groups, and Roles) in the IAM User Guide.

Resource-Based Policies

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Amazon SageMaker doesn't support resource-based policies.

Specifying Policy Elements: Resources, Actions, Effects, and Principals

For each Amazon SageMaker resource, the service defines a set of API operations. To grant permissions for these API operations, Amazon SageMaker defines a set of actions that you can specify in a policy. For example, for the Amazon SageMaker notebook instance resource, the following actions are defined: CreateNotebookInstance, DeleteNotebookInstance, and DescribeNotebookInstance. Some API operations can require permissions for more than one action in order to perform the API operation. For more information about resources and API operations, see Amazon SageMaker Resources and Operations (p. 401) and API Reference (p. 506).

The following are the most basic policy elements:

- **Resource** – You use an Amazon Resource Name (ARN) to identify the resource that the identity-based policy applies to. For more information, see Amazon SageMaker Resources and Operations (p. 401).
- **Action** – You use action keywords to identify resource operations that you want to allow or deny. For example, you can use sagemaker:CreateModel to add a model to the notebook instance.
- **Effect** – You specify the effect, either allow or deny, when the user requests the specific action. If you don't explicitly grant access to (allow) a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even if a different policy grants access.
- **Principal** – In identity-based policies (IAM policies), the user that the policy is attached to is the implicit principal. Amazon SageMaker doesn't support resource-based policies.

To learn more about IAM policy syntax and to read policy descriptions, see AWS IAM Policy Reference in the IAM User Guide.

For a list showing all of the Amazon SageMaker API operations and the resources that they apply to, see Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 411).
Specifying Conditions in a Policy

When you grant permissions, you can use the IAM policy language to specify the conditions under which a policy should take effect. For example, you might want a policy to be applied only after a specific date. For more information about specifying conditions in a policy language, see Conditions in the IAM User Guide.

To express conditions, you use predefined condition keys. There are no condition keys specific to Amazon SageMaker. However, there are AWS-wide condition keys that you can use as appropriate. For an example of AWS-wide keys used in an Amazon SageMaker permissions policy, see Permissions Required to Use the Amazon SageMaker Console (p. 405). For a complete list of AWS-wide keys, see Available Keys for Conditions in the IAM User Guide.

Using Identity-based Policies (IAM Policies) for Amazon SageMaker

This topic provides examples of identity-based policies that demonstrate how an account administrator can attach permissions policies to IAM identities (that is, users, groups, and roles) and thereby grant permissions to perform operations on Amazon SageMaker resources.

Important
We recommend that you first review the introductory topics that explain the basic concepts and options available to manage access to your Amazon SageMaker resources. For more information, see Overview of Managing Access Permissions to Your Amazon SageMaker Resources (p. 400).

Topics
• Permissions Required to Use the Amazon SageMaker Console (p. 405)
• Permissions Required to Use the Amazon SageMaker Ground Truth Console (p. 407)
• AWS Managed (Predefined) Policies for Amazon SageMaker (p. 408)
• Control Access to Amazon SageMaker Resources by Using Tags (p. 408)

The following is an example of a basic permissions policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [ {
        "Sid": "AllowCreate-Describe-Delete-Models",
        "Effect": "Allow",
        "Action": [ "sagemaker:CreateModel",
                     "sagemaker:DescribeModel",
                     "sagemaker:DeleteModel" ],
        "Resource": "*"
    } ],
    "Statement": [ {
        "Sid": "AdditionalIamPermission",
        "Effect": "Allow",
        "Action": [ "iam:PassRole" ],
        "Resource": "arn:aws:iam::account-id:role/role-name"
    } ]
}
```
The policy has two statements:

- The first statement grants permission for three Amazon SageMaker actions (`sagemaker:CreateModel`, `sagemaker:DescribeModel`, and `sagemaker:DeleteModel`) within an Amazon SageMaker notebook instance. Using the wildcard character (*) as the resource grants universal permissions for these actions across all AWS Regions and models owned by this account.
- The second statement grants permission for the `iam:PassRole` action, which is needed for the Amazon SageMaker action `sagemaker:CreateModel`, which is allowed by the first statement.

The policy doesn't specify the `Principal` element because in an identity-based policy you don't specify the principal who gets the permission. When you attach the policy to a user, the user is the implicit principal. When you attach a permissions policy to an IAM role, the principal identified in the role's trust policy gets the permissions.

For a table showing all of the Amazon SageMaker API operations and the resources that they apply to, see Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 411).

Permissions Required to Use the Amazon SageMaker Console

The permissions reference table lists the Amazon SageMaker API operations and shows the required permissions for each operation. For more information about Amazon SageMaker API operations, see Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 411).

To use the Amazon SageMaker console, you need to grant permissions for additional actions. Specifically, the console needs permissions that allow the `ec2` actions to display subnets, VPCs, and security groups. Optionally, the console needs permission to create `execution roles` for tasks such as `CreateNotebook`, `CreateTrainingJob`, and `CreateModel`. Grant these permissions with the following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        // SageMaker API's
        {
            "Sid": "SageMakerApis",
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*"
        },
        // Populate customer VPCs, Subnets, and Security Groups for Create forms
        {
            "Sid": "VpcConfigurationForCreateForms",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        // Populate customer KMS keys for Create forms
        {
            "Sid": "KmsKeysForCreateForms",
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:UntagResource"
            ],
            "Resource": "*"
        }
    ]
}
```
"Effect": "Allow",
"Action": [
  "kms:DescribeKey",
  "kms:ListAliases"
],
"Resource": "*"
},
// View Subscriptions in AWS Marketplace for Algorithms and ModelPackages.
{
  "Sid": "AccessAWSMarketplaceSubscriptions",
  "Effect": "Allow",
  "Action": [
    "aws-marketplace:ViewSubscriptions"
  ],
  "Resource": "*"
},
// View and create CodeCommit Repositories
{
  "Effect": "Allow",
  "Action": [
    "codecommit:BatchGetRepositories",
    "codecommit:CreateRepository",
    "codecommit:GetRepository",
    "codecommit:ListRepositories",
    "codecommit:ListBranches",
    "secretsmanager:CreateSecret",
    "secretsmanager:DescribeSecret",
    "secretsmanager:ListSecrets"
  ],
  "Resource": "*
}
},
// List/create execution roles for Create forms
{
  "Sid": "ListAndCreateExecutionRoles",
  "Effect": "Allow",
  "Action": [
    "iam:ListRoles",
    "iam:CreateRole",
    "iam:CreateRole",
    "iam:CreatePolicy",
    "iam:AttachRolePolicy"
  ],
  "Resource": "*
},
// Permissions required for ECR
{
  "Sid": "DescribeECRMetaData",
  "Effect": "Allow",
  "Action": [
    "ecr:Describe*"
  ],
  "Resource": "*"
},
// PassRole permissions as required by CreateNotebookInstance, CreateTrainingJob, and CreateModel.
{
  "Sid": "PassRoleForExecutionRoles",
  "Effect": "Allow",
  "Action": [
    "iam:PassRole"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "iam:PassedToService": "sagemaker.amazonaws.com" 
    }
  }
}
Permissions Required to Use the Amazon SageMaker Ground Truth Console

To use the Amazon SageMaker Ground Truth console, you need to grant permissions for additional resources. Specifically, the console needs permissions for the AWS Marketplace to view subscriptions, Amazon Cognito operations to manage your private workforce, Amazon S3 actions for access to your input and output files, and AWS Lambda actions to list and invoke functions. Grant these permissions with the following permissions policy:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GroundTruthConsole",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:DescribeListings",
                "aws-marketplace:ViewSubscriptions",
                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminDisableUser",
                "cognito-idp:AdminEnableUser",
                "cognito-idp:AdminRemoveUserFromGroup",
                "cognito-idp:CreateGroup",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:CreateUserPoolDomain",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:ListGroups",
                "cognito-idp:ListIdentityProviders",
                "cognito-idp:ListUsers",
                "cognito-idp:ListUsersInGroup",
                "cognito-idp:ListUserPoolClients",
                "cognito-idp:ListUserPools",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:UpdateUserPoolClient",
                "groundtruthlabeling:DescribeConsoleJob",
                "groundtruthlabeling:ListDatasetObjects",
                "groundtruthlabeling:RunFilterOrSampleManifestJob",
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",
                "lambda:InvokeFunction",
                "lambda:ListFunctions",
                "s3:GetObject",
                "s3:GetObject",
                "s3:PutObject",
                "s3:SelectObjectContent"
            ],
            "Resource": "*"
        }
    ]
}
```
AWS Managed (Predefined) Policies for Amazon SageMaker

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. These AWS managed policies grant necessary permissions for common use cases so that you can avoid having to investigate which permissions are needed. For more information, see AWS Managed Policies in the IAM User Guide.

The following AWS managed policies, which you can attach to users in your account, are specific to Amazon SageMaker:

- **AmazonSageMakerReadOnly** – Grants read-only access to Amazon SageMaker resources.
- **AmazonSageMakerFullAccess** – Grants full access to Amazon SageMaker resources and the supported operations. (This does not provide unrestricted S3 access, but supports buckets/objects with specific sagemaker tags.)

The following AWS managed policies can also be attached to users in your account:

- **AdministratorAccess** – Grants all actions for all AWS services and for all resources in the account.
- **DataScientist** – Grants a wide range of permissions to cover most of the use cases (primarily for analytics and business intelligence) encountered by data scientists.

You can review these permissions policies by signing in to the IAM console and searching for them.

You can also create your own custom IAM policies to allow permissions for Amazon SageMaker actions and resources as you need them. You can attach these custom policies to the IAM users or groups that require them.

### Control Access to Amazon SageMaker Resources by Using Tags

Control access to groups of Amazon SageMaker resources by attaching tags to the resources and specifying ResourceTag conditions in IAM policies. For example, suppose you've defined two different IAM groups, named DevTeam1 and DevTeam2, in your AWS account. Suppose also that you've created 10 notebook instances, 5 of which are used for one project, and 5 of which are used for a second project. You want to allow members of DevTeam1 to make API calls on notebook instances used for the first project, and members of DevTeam2 to make API calls on notebook instances used for the second project. You can control access to API calls by completing the following steps:

1. Add a tag with the key `Project` and value `A` to the notebook instances used for the first project. For information about adding tags to Amazon SageMaker resources, see AddTags (p. 510).
2. Add a tag with the key `Project` and value `B` to the notebook instances used for the second project.
3. Create an IAM policy with a ResourceTag condition that denies access to the notebook instances used for the second project, and attach that policy to DevTeam1. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `B`:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "ResourceTag": {
                "Project": "B"
            }
        }
    ]
}
```

408
**Control Access to Amazon SageMaker Resources by Using Tags**

For information about creating IAM policies and attaching them to identities, see [Controlling Access Using Policies](#) in the *AWS Identity and Access Management User Guide*.

4. Create an IAM policy with a `ResourceTag` condition that denies access to the notebook instances used for the first project, and attach that policy to DevTeam2. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `A`:

```json
{
    "Version": "2012-10-17",
    "Statement": [  
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:ResourceTag/Project": "A"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [  
                "sagemaker:CreateTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "*"
        }
    ]
}
```
Require the Presence or Absence of Tags for API Calls

Require the presence or absence of specific tags or specific tag values by using RequestTag condition keys in an IAM policy. For example, if you want to require that every endpoint created by any member of an IAM group to be created with a tag with the key `environment` and value `dev`, create a policy as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": "arn:aws:sagemaker:*:*:endpoint/*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": "arn:aws:sagemaker:*:*:endpoint/*"
    },
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreateEndpoint",
      "Resource": "arn:aws:sagemaker:*:*:endpoint/*"
    },
    {
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/environment": "dev"
        }
      }
    }
  ]
}
```

Use Tags with Hyperparameter Tuning Jobs

You can add tags to a hyperparameter tuning job when you create the tuning job by specifying the tags as the Tags parameter when you call `CreateHyperParameterTuningJob (p. 528)`. If you do this, the tags you specify for the hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches.

If you add tags to a hyperparameter tuning job by calling `AddTags (p. 510)`, the tags you add are also added to any training jobs that the hyperparameter tuning job launches after you call `AddTags`, but are not added to training jobs the hyperparameter tuning jobs launched before you called `AddTags`. Similarly, when you remove tags from a hyperparameter tuning job by calling `DeleteTags (p. 581)`, those tags are not removed from training jobs that the hyperparameter tuning job launched previously. Because of this, the tags associated with training jobs can be out of sync with the tags associated with the hyperparameter tuning job that launched them. If you use tags to control access to a hyperparameter tuning job and the training jobs it launches, you might want to keep the tags in sync. To make sure the tags associated with training jobs stay sync with the tags associated with the hyperparameter tuning job that launched them, first call `ListTrainingJobsForHyperParameterTuningJob (p. 690)` for the hyperparameter tuning job to get a list of the training jobs that the hyperparameter tuning job launched. Then, call `AddTags` or `DeleteTags` for the hyperparameter tuning job and for each of the training jobs in the list of training jobs to add or delete the same set of tags for all of the jobs. The following Python example demonstrates this:
tuning_job_arn =
    smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName='MyTuningJob')
    ['HyperParameterTuningJobArn']
smclient.add_tags(ResourceArn=tuning_job_arn, Tags=[{'Key':'Env', 'Value':'Dev'}])

for training_job in training_jobs:
    time.sleep(1) # Wait for 1 second between calls to avoid being throttled
    smclient.add_tags(ResourceArn=training_job['TrainingJobArn'], Tags=[{'Key': 'Env', 'Value': 'Dev'}])

Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control (p. 400) and writing a permissions policy that you can attach to
an IAM identity (an identity-based policy), use the following as a reference. The each Amazon SageMaker
API operation, the corresponding actions for which you can grant permissions to perform the action, and
the AWS resource for which you can grant the permissions. You specify the actions in the policy's Action
field, and you specify the resource value in the policy's Resource field.

Note
Except for the ListTags API, resource-level restrictions are not available on List- calls. Any
user calling a List- API will see all resources of that type in the account.

To express conditions in your Amazon SageMaker policies, you can use AWS-wide condition keys. For a
complete list of AWS-wide keys, see Available Keys in the IAM User Guide.

Amazon SageMaker API and Required Permissions for Actions

API Operation: AddTags (p. 510)

Required Permissions (API Action): sagemaker:AddTags
Resources: *

API Operation: CreateEndpoint (p. 522)

Required Permissions (API Action): sagemaker:CreateEndpoint
Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: CreateEndpointConfig (p. 525)

Required Permissions (API Action): sagemaker:CreateEndpointConfig
Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: CreateModel (p. 538)

Required Permissions (API Action): sagemaker:CreateModel, iam:PassRole
Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: CreateLabelingJob (p. 533)

Required Permissions (API Action): sagemaker:CreateLabelingJob, iam:PassRole
Resources: arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName
API Operation: CreateNotebookInstance (p. 546)


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: CreateTrainingJob (p. 556)

Required Permissions (API Action): sagemaker:CreateTrainingJob, iam:PassRole

Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

API Operation: CreateWorkteam (p. 567)


API Operation: DeleteEndpoint (p. 572)

Required Permissions (API Action): sagemaker:DeleteEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: DeleteEndpointConfig (p. 574)

Required Permissions (API Action): sagemaker:DeleteEndpointConfig

Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: DeleteModel (p. 575)

Required Permissions (API Action): sagemaker:DeleteModel

Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: DeleteNotebookInstance (p. 578)


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: DeleteTags (p. 581)

Required Permissions (API Action): sagemaker:DeleteTags

Resources: *

API Operation: DeleteWorkteam (p. 583)

Required Permissions (API Action): sagemaker:DeleteWorkteam

API Operation: DescribeEndpoint (p. 596)

Required Permissions (API Action): sagemaker:DescribeEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: DescribeEndpointConfig (p. 599)

Required Permissions (API Action): sagemaker:DescribeEndpointConfig

Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: DescribeLabelingJob (p. 607)

Required Permissions (API Action): sagemaker:DescribeLabelingJob

Resources: arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName

API Operation: DescribeModel (p. 612)

Required Permissions (API Action): sagemaker:DescribeModel

Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: DescribeNotebookInstance (p. 619)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: DescribeSubscribedWorkteam (p. 627)

Required Permissions (API Action): sagemaker:DescribeSubscribedWorkteam, aws-marketplace:ViewSubscriptions


API Operation: DescribeTrainingJob (p. 629)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

API Operation: DescribeWorkteam (p. 640)

Required Permissions (API Action): sagemaker:DescribeWorkteam


API Operation: CreatePresignedNotebookInstanceUrl (p. 554)

Required Permissions (API Action): sagemaker:CreatePresignedNotebookInstanceUrl

Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: InvokeEndpoint (p. 735)

Required Permissions (API Action): sagemaker:InvokeEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: ListEndpointConfigs (p. 653)

Required Permissions (API Action): sagemaker:ListEndpointConfigs
Resources: *

**API Operation:** ListEndpoints (p. 656)

Required Permissions (API Action): sagemaker:ListEndpoints

Resources: *

**API Operation:** ListLabelingJobs (p. 663)

Required Permissions (API Action): sagemaker:ListLabelingJobs

Resources: *

**API Operation:** ListLabelingJobsForWorkteam (p. 667)

Required Permissions (API Action): sagemaker:ListLabelingJobsForWorkteam

Resources: *

**API Operation:** ListModels (p. 673)

Required Permissions (API Action): sagemaker:ListModels

Resources: *

**API Operation:** ListNotebookInstances (p. 679)

Required Permissions (API Action): sagemaker:ListNotebookInstances

Resources: *

**API Operation:** ListSubscribedWorkteams (p. 683)


**API Operation:** ListTags (p. 685)

Required Permissions (API Action): sagemaker:ListTags

Resources: *

**API Operation:** ListTrainingJobs (p. 687)

Required Permissions (API Action): sagemaker:ListTrainingJobs

Resources: *

**API Operation:** ListWorkteams (p. 696)

Required Permissions (API Action): sagemaker:ListWorkteams


**API Operation:** StartNotebookInstance (p. 706)


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName
Amazon SageMaker Developer Guide

Amazon SageMaker Roles

As a managed service, Amazon SageMaker performs operations on your behalf on the AWS hardware that is managed by Amazon SageMaker. Amazon SageMaker can perform only operations that the user permits.

An Amazon SageMaker user can grant these permissions with an IAM role (referred to as an execution role). The user passes the role when making these API calls: `CreateNotebookInstance (p. 546), CreateHyperParameterTuningJob (p. 528), CreateTrainingJob (p. 556), and CreateModel (p. 538).`

You attach the following trust policy to the IAM role which grants Amazon SageMaker principal permissions to assume the role, and is the same for all of the execution roles:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "sagemaker.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```
The permissions that you need to grant to the role vary depending on the API that you call. The following sections explain these permissions.

**Note**
Instead of managing permissions by crafting a permission policy, you can use the AWS-managed AmazonSageMakerFullAccess permission policy. The permissions in this policy are fairly well scoped to the actions you need to perform. For more information see Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role (p. 424). If you prefer to create custom policies and manage permissions, see the following topics.

For more information about IAM roles, see IAM Roles in the IAM User Guide.

### Topics

- CreateNotebookInstance API: Execution Role Permissions (p. 416)
- CreateHyperParameterTuningJob API: Execution Role Permissions (p. 419)
- CreateTrainingJob API: Execution Role Permissions (p. 420)
- CreateModel API: Execution Role Permissions (p. 422)
- Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role (p. 424)

### CreateNotebookInstance API: Execution Role Permissions

The permissions that you grant to the execution role for calling the CreateNotebookInstance API depend on what you plan to do with the notebook instance. If you plan to use it to invoke Amazon SageMaker APIs and pass the same role when calling the CreateTrainingJob and CreateModel APIs, attach the following permissions policy to the role:

```json
{
   "Version": "2012-10-17",
   "Statement": [

      {
         "Effect": "Allow",
         "Action": [

            "sagemaker:*",
            "ecr:GetAuthorizationToken",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability",
            "cloudwatch:PutMetricData",
            "cloudwatch:GetMetricData",
            "cloudwatch:GetMetricStatistics",
            "cloudwatch:ListMetrics",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:DescribeLogStreams",
            "logs:PutLogEvents",
            "logs:GetLogEvents",
            "s3:CreateBucket",
            "s3:ListBucket",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject",
            "robomaker:CreateSimulationApplication",
            "robomaker:DescribeSimulationApplication",
            "robomaker:DeleteSimulationApplication",
            "robomaker:CreateSimulationJob"
         ]
      }
   ]
}
```
To tighten the permissions, limit them to specific Amazon S3 and Amazon ECR resources, by replacing
"Resource": "*", as follows:

```
{"Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
     "sagemaker:*",
     "ecr:GetAuthorizationToken",
     "cloudwatch:PutMetricData",
     "logs:CreateLogGroup",
     "logs:CreateLogStream",
     "logs:DescribeLogStreams",
     "logs:PutLogEvents",
     "logs:GetLogEvents"
   ],
   "Resource": "*",
   "Condition": {
     "StringEquals": {
       "iam:PassedToService": "sagemaker.amazonaws.com"
     }
   }
  }
 ]
}
```
If you plan to access other resources, such as Amazon DynamoDB or Amazon Relational Database Service, add the relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- **Scope the `s3:ListBucket` permission to the specific bucket that you specify as `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.**
- **Scope `s3:GetObject`, `s3:PutObject`, and `s3:DeleteObject` permissions as follows:**
  - Scope to the following values that you specify in a `CreateTrainingJob` request:
    - `InputDataConfig.DataSource.S3DataSource.S3Uri`
    - `OutputDataConfig.S3OutputPath`
  - Scope to the following values that you specify in a `CreateModel` request:
    - `PrimaryContainer.ModelDataUrl`
    - `SupplementalContainers.ModelDataUrl`
- **Scope `ecr` permissions as follows:**
  - Scope to the `AlgorithmSpecification.TrainingImage` value that you specify in a `CreateTrainingJob` request.
  - Scope to the `PrimaryContainer.Image` value that you specify in a `CreateModel` request.
The `cloudwatch` and `logs` actions are applicable for `*` resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

CreateHyperParameterTuningJob API: Execution Role Permissions

For an execution role that you can pass in a CreateHyperParameterTuningJob API request, you can attach the following permission policy to the role:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of specifying `"Resource": "*"`, you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
```

419
"Effect": "Allow",
"Action": [
    "s3:GetObject",
    "s3:PutObject"
],
"Resource": [
    "arn:aws:s3:::inputbucket/object",
    "arn:aws:s3:::outputbucket/path"
]
},
{
"Effect": "Allow",
"Action": [
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage"
],
"Resource": "arn:aws:ecr:::repository/my-repo"
}
]
}

If the training container associated with the hyperparameter tuning job needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the s3:ListBucket permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the s3:GetObject and s3:PutObject permissions to the following objects that you specify in the input and output data configuration in a `CreateHyperParameterTuningJob` request:
  
  - `InputDataConfig.DataSource.S3DataSource.S3Uri`
  - `OutputDataConfig.S3OutputPath`
- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateHyperParameterTuningJob` request.

The cloudwatch and logs actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your hyperparameter tuning job, add the following permissions:

```json
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ],
    "Resource": *private VPC resource*
}
```

**CreateTrainingJob API: Execution Role Permissions**

For an execution role that you can pass in a `CreateTrainingJob` API request, you can attach the following permission policy to the role:
CreateTrainingJob API: Execution Role Permissions

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

Instead of specifying "Resource": "*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket/object",
        "arn:aws:s3:::outputbucket/path"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ObjectAclPut"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket/object",
        "arn:aws:s3:::outputbucket/path"
      ]
    }
  ]
}
```
If CreateTrainingJob.AlgorithmSpecifications.TrainingImage needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3:ListBucket` permission to a specific bucket that you specify as the InputDataConfig.DataSource.S3DataSource.S3Uri in a CreateTrainingJob request.
- Scope the `s3:GetObject` and `s3:PutObject` permissions to the following objects that you specify in the input and output data configuration in a CreateTrainingJob request:
  - InputDataConfig.DataSource.S3DataSource.S3Uri
  - OutputDataConfig.S3OutputPath
- Scope Amazon ECR permissions to the registry path (AlgorithmSpecification.TrainingImage) that you specify in a CreateTrainingJob request.

The `cloudwatch` and `logs` actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your training job, add the following permissions:

```json
{
  "Effect": "Allow",
  "Action": [
    "ec2:CreateNetworkInterface",
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface",
    "ec2:DeleteNetworkInterfacePermission",
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeVpcs",
    "ec2:DescribeDhcpOptions",
    "ec2:DescribeSubnets",
    "ec2:DescribeSecurityGroups"
}
```

### CreateModel API: Execution Role Permissions

For an execution role that you can pass in a CreateModel API request, you can attach the following permission policy to the role:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
      ]
    }
  ]
}
```
Instead of specifying "Resource": "*", you can scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        
        
    
    ]
}
```

If `CreateModel.PrimaryContainer.Image` need to access other data sources, such as Amazon DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope S3 permissions to objects that you specify in the `PrimaryContainer.ModelDataUrl` in a `CreateModel (p. 538)` request.
• Scope Amazon ECR permissions to a specific registry path that you specify as the PrimaryContainer.Image and SecondaryContainer.Image in a CreateModel request.

The cloudwatch and logs actions are applicable for "*" resources. For more information, see CloudWatch Resources and Operations in the Amazon CloudWatch User Guide.

If you specify a private VPC for your model, add the following permissions:

```
{
  "Effect": "Allow",
  "Action": [ "ec2:CreateNetworkInterface",
  "ec2:CreateNetworkInterfacePermission",
  "ec2:DeleteNetworkInterface",
  "ec2:DeleteNetworkInterfacePermission",
  "ec2:DescribeNetworkInterfaces",
  "ec2:DescribeVpcs",
  "ec2:DescribeDhcpOptions",
  "ec2:DescribeSubnets",
  "ec2:DescribeSecurityGroups"
}
```

### Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role

You can create an execution role one of two ways:

- In the Amazon SageMaker console when you create a notebook instance, training job, or model.
- In the AWS Identity and Access Management (IAM) console. You then specify the role as you follow the notebook instance, training job, and model creation workflows in the Amazon SageMaker console.

Regardless of how you create an execution role, you can attach the AWS-managed permission policy (AmazonSageMakerFullAccess) to the role.

When attaching the AmazonSageMakerFullAccess policy to a role, you must do one of the following to allow Amazon SageMaker to access your S3 bucket:

- Include the string "SageMaker" or "sagemaker" in the name of the bucket where you store training data, or the model artifacts resulting from model training, or both.
- Include the string "SageMaker" or "sagemaker" in the object name of the training data object(s).
- Tag the S3 object with "sagemaker=true". The key and value are case sensitive. For more information, see Object Tagging in the Amazon Simple Storage Service Developer Guide.
- Add a bucket policy that allows access for the execution role. For more information, see Using Bucket Policies and User Policies in the Amazon Simple Storage Service Developer Guide.

You can attach additional policies that specify the resources for which you want to grant permissions for the s3:GetObject, s3:PutObject, and s3:ListBucket actions. In the IAM console, you can attach a customer managed policy or an inline policy to your execution role(s). Alternatively, when you create a role in the Amazon SageMaker console, you can attach a customer managed policy that specifies the S3 buckets. This resulting execution role has the prefix "AmazonSageMaker-ExecutionRole-".
Monitor Amazon SageMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SageMaker and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon SageMaker, report when something is wrong, and take automatic actions when appropriate:

- **Amazon CloudWatch** monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.

- **Amazon CloudWatch Logs** enables you to monitor, store, and access your log files from EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.

- **AWS CloudTrail** captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Monitor Amazon SageMaker with Amazon CloudWatch

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. However, the Amazon CloudWatch console limits the search to metrics that were updated in the last 2 weeks. This limitation ensures that the most current jobs are shown in your namespace. To graph metrics without using a search, specify its exact name in the source view. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

Amazon SageMaker model training jobs and endpoints write CloudWatch metrics and logs. The following tables list the metrics and dimensions for Amazon SageMaker.

### Endpoint Invocation Metrics

The **AWS/SageMaker** namespace includes the following request metrics from calls to `InvokeEndpoint` (p. 735).

Metrics are available at a 1-minute frequency.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invocation4XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocation5XXErrors</td>
<td>The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Units: None</td>
<td>Valid statistics: Average, Sum</td>
</tr>
<tr>
<td>Invocations</td>
<td>The number of InvokeEndpoint requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>InvocationsPerInstance</td>
<td>The number of invocations sent to a model, normalized by InstanceCount in each ProductionVariant. 1/numberOfInstances is sent as the value on each request, where numberOfInstances is the number of active instances for the ProductionVariant behind the endpoint at the time of the request. Units: None Valid statistics: Sum</td>
</tr>
<tr>
<td>ModelLatency</td>
<td>The interval of time taken by a model to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<tr>
<td>OverheadLatency</td>
<td>The interval of time added to the time taken to respond to a client request by Amazon SageMaker overheads. This interval is measured from the time Amazon SageMaker receives the request until it returns a response to the client, minus the ModelLatency. Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
</tbody>
</table>

**Dimensions for Endpoint Invocation Metrics**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndpointName,</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
<tr>
<td>VariantName</td>
<td></td>
</tr>
</tbody>
</table>

**Training Job, Batch Transform Job, and Endpoint Instance Metrics**

The /aws/sagemaker/TrainingJobs, /aws/sagemaker/TransformJobs and /aws/sagemaker/Endpoints namespaces include the following metrics for the training jobs and endpoint instances. Metrics are available at a 1-minute frequency.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUUtilization</td>
<td>The percentage of CPU units that are used by the containers on an instance. The value can range between 0 and 100, and is multiplied by the number of CPUs. For example, if there are four CPUs, CPUUtilization can range from 0% to 400%.</td>
</tr>
<tr>
<td></td>
<td>For training jobs, the value is the CPU utilization of the algorithm container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For batch transform jobs, the value is the CPU utilization of the transform container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td>MemoryUtilization</td>
<td>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%.</td>
</tr>
<tr>
<td></td>
<td>For training jobs, the value is the memory utilization of the algorithm container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For batch transform jobs, the value is the memory utilization of the transform container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td>GPUUtilization</td>
<td>The percentage of GPU units that are used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUUtilization can range from 0% to 400%.</td>
</tr>
<tr>
<td></td>
<td>For training jobs, the value is the GPU utilization of the algorithm container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For batch transform jobs, the value is the GPU utilization of the transform container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td>GPUMemoryUtilization</td>
<td>The percentage of GPU memory used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUMemoryUtilization can range from 0% to 400%.</td>
</tr>
<tr>
<td></td>
<td>For training jobs, the value is the GPU memory utilization of the algorithm container on the instance.</td>
</tr>
<tr>
<td></td>
<td>For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance.</td>
</tr>
</tbody>
</table>
Metric | Description
---|---
For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance.
Units: Percent

DiskUtilization | The percentage of disk space used by the containers on an instance uses. This value can range between 0% and 100%. This metric is not supported for batch transform jobs.
For training jobs, the value is the disk space utilization of the algorithm container on the instance.
For endpoint variants, the value is the sum of the disk space utilization of the primary and supplementary containers on the instance.
Units: Percent

**Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics**

**Dimension** | **Description**
---|---
Host | For training jobs, the value for this dimension has the format `[training-job-name]/algo-[instance-number-in-cluster]`. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the `/aws/sagemaker/TrainingJobs` namespace.
For batch transform jobs, the value for this dimension has the format `[transform-job-name]/[instance-id]`. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the `/aws/sagemaker/TransformJobs` namespace.
For endpoints, the value for this dimension has the format `[endpoint-name]/[production-variant-name]/[instance-id]`. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the `/aws/sagemaker/Endpoints` namespace.

**Amazon SageMaker Ground Truth Metrics**

**Metric** | **Description**
---|---
**DatasetObjectsAutoAnnotated** | The number of dataset objects auto-annotated in a labeling job. This metric is only emitted when automated labeling is enabled. To view the labeling job progress, use the Max metric.
Units: None
Valid statistics: Max

**DatasetObjectsHumanAnnotated** | The number of dataset objects annotated by a human in a labeling job. To view the labeling job progress, use the Max metric.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DatasetObjectsLabelingFailed</td>
<td>The number of dataset objects that failed labeling in a labeling job. To view the labeling job progress, use the Max metric.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Max</td>
</tr>
<tr>
<td>JobsFailed</td>
<td>The number of labeling jobs that failed. To get the total number of labeling jobs that failed, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>JobsSucceeded</td>
<td>The number of labeling jobs that succeeded. To get the total number of labeling jobs that succeeded, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>JobsStopped</td>
<td>The number of labeling jobs that were stopped. To get the total number of labeling jobs that were stopped, use the Sum statistic.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Sum, Sample Count</td>
</tr>
<tr>
<td>TotalDatasetObjectsLabeled</td>
<td>The number of dataset objects labeled successfully in a labeling job. To view the labeling job progress, use the Max metric.</td>
</tr>
<tr>
<td></td>
<td>Units: None</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Max</td>
</tr>
</tbody>
</table>

### Dimensions for Dataset Object Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LabelingJobName</td>
<td>Filters dataset object count metrics for a labeling job.</td>
</tr>
</tbody>
</table>

### Log Amazon SageMaker Events with Amazon CloudWatch

To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.

**Logs**
### Log Amazon SageMaker API Calls with AWS CloudTrail

Amazon SageMaker is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. CloudTrail captures all API calls for Amazon SageMaker, with the exception of `InvokeEndpoint` (p. 735), as events. The calls captured include calls from the Amazon SageMaker console and code calls to the Amazon SageMaker API operations.

If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon SageMaker. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

### Amazon SageMaker Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon SageMaker, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon SageMaker, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations

---

<table>
<thead>
<tr>
<th>Log Group Name</th>
<th>Log Stream Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aws/sagemaker/TrainingJobs</td>
<td>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</td>
</tr>
<tr>
<td>/aws/sagemaker/Endpoints/[EndpointName]</td>
<td>[production-variant-name]/[instance-id]</td>
</tr>
<tr>
<td>/aws/sagemaker/NotebookInstances</td>
<td>[notebook-instance-name]/[LifecycleConfigHook]</td>
</tr>
<tr>
<td>/aws/sagemaker/TransformJobs</td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]</td>
</tr>
<tr>
<td></td>
<td>[transform-job-name]/[instance-id]-[epoch_timestamp]/data-log</td>
</tr>
</tbody>
</table>

**Note**

The /aws/sagemaker/NotebookInstances log group is made when you create a Notebook Instance with a Lifecycle configuration. For more information, see Step 1.1: (Optional) Customize a Notebook Instance (p. 20).
• Configuring Amazon SNS Notifications for CloudTrail
• Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon SageMaker actions, with the exception of InvokeEndpoint (p. 735), are logged by CloudTrail and are documented in the Actions (p. 506). For example, calls to the CreateTrainingJob, CreateEndpoint and CreateNotebookInstance actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

• Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
• Whether the request was made with temporary security credentials for a role or federated user.
• Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Operations Performed by Automatic Model Tuning

Amazon SageMaker supports logging non-API service events to your CloudTrail log files for automatic model tuning jobs. These events are related to your tuning jobs but are not the direct result of a customer request to the public AWS API. For example, when you create a hyperparameter tuning job by calling CreateHyperParameterTuningJob (p. 528), Amazon SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call StopHyperParameterTuningJob (p. 710) to stop a hyperparameter tuning job, Amazon SageMaker might stop any of the associated running training jobs. Non-API events for your tuning jobs are logged to CloudTrail to help you improve governance, compliance, and operational and risk auditing of your AWS account.

Log entries that result from non-API service events have an eventType of AwsServiceEvent instead of AwsApiCall.

Understanding Amazon SageMaker Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following examples a log entry for the CreateEndpoint action, which creates an endpoint to deploy a trained model.

```json
{
    "eventVersion":"1.05",
    "userIdentity": {
        "type":"IAMUser",
        "principalId":"AIXDAYQEXAMPLEUMLYNGL",
        "arn":"arn:aws:iam::123456789012:user/intern",
        "accountId":"123456789012",
        "accessKeyId":"ASXIAGXEXAMPLEQULKNXV",
        "userName":"intern"
    },
    "eventTime":"2018-01-02T13:39:06Z",
    "eventSource":"sagemaker.amazonaws.com",
```
The following example is a log entry for the `CreateModel` action, which creates one or more containers to host a previously trained model.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",
        "arn": "arn:aws:iam::123456789012:user/intern",
        "accountId": "123456789012",
        "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
        "userName": "intern"
    },
    "eventTime": "2018-01-02T15:23:46Z",
    "eventSource": "sagemaker.amazonaws.com",
    "eventName": "CreateModel",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "USER_AGENT",
    "requestParameters": {
        "modelName": "ExampleModel",
        "primaryContainer": {
            "image": "174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest"
        },
        "executionRoleArn": "arn:aws:iam::123456789012:role/EXAMPLEARN"
    },
    "responseElements": {
    }
}
```
Notebook Instance Security

Notebook Instance Security

Note the following security considerations for notebook instances.

Topics
- Notebooks Instaces Are Internet-Enabled by Default (p. 433)

Notebook Instances Are Internet-Enabled by Default

Amazon SageMaker notebook instances are internet-enabled. This allows you to download popular packages and notebooks, customize your development environment, and work efficiently. However, if you connect a notebook instance to your VPC, the notebook instance provides an additional avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the computer (in the form of a publicly available notebook or a publicly available source code library) could access your data. If you do not want Amazon SageMaker to provide internet access to your notebook instance, you can disable direct internet access when you specify a VPC for your notebook instance. If you disable direct internet access, the notebook instance won't be able to train or host models unless your VPC has an interface endpoint (PrivateLink) or a NAT gateway and your security groups allow outbound connections. For information about creating a VPC interface endpoint to use PrivateLink for your notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 44). For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide. For information about security groups, see Security Groups for Your VPC.

Notebook Instances Provide the Best Experience for a Single User

An Amazon SageMaker notebook instance is designed to work best for an individual user. It is designed to give data scientists and other users the most power for managing their development environment. A notebook instance user has root access for installing packages and other pertinent software. We recommend that you exercise judgement when granting individuals access to notebook instances that are attached to a VPC that contains sensitive information. For example, you might grant a user access to a notebook instance with an IAM policy, as in the following example:
You can connect directly to the Amazon SageMaker API or to the Amazon SageMaker Runtime through an interface endpoint in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the Amazon SageMaker API or Runtime is conducted entirely and securely within the AWS network.

**Note**  
PrivateLink for Amazon SageMaker is not supported in the us-gov-west-1 region.

The Amazon SageMaker API and Runtime support Amazon Virtual Private Cloud (Amazon VPC) interface endpoints that are powered by AWS PrivateLink. Each VPC endpoint is represented by one or more Elastic Network Interfaces (ENIs) with private IP addresses in your VPC subnets.

The VPC interface endpoint connects your VPC directly to the Amazon SageMaker API or Runtime without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Amazon SageMaker API or Runtime.

You can create an interface endpoint to connect to Amazon SageMaker or to Amazon SageMaker Runtime with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see [Creating an Interface Endpoint](https://docs.aws.amazon.com/sagemaker/latest/dg/creating-vpc-endpoint.html).

*After you have created a VPC endpoint,* you can use the following example CLI commands that use the `--endpoint-url` parameter to specify interface endpoints to the Amazon SageMaker API or Runtime:

```bash
aws sagemaker list-notebook-instances --endpoint-url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
aws sagemaker list-training-jobs --endpoint-url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
aws sagemaker-runtime invoke-endpoint --endpoint-url VPC_Endpoint_ID.runtime.sagemaker.Region.vpce.amazonaws.com \
    --endpoint-name Endpoint_Name \
    --body "Endpoint_Body" \
    --content-type "Content_Type" \
    Output_File
```

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint URL. The Amazon SageMaker API DNS hostname that the CLI and Amazon SageMaker SDK use by default (https://api.sagemaker.Region.amazonaws.com) resolves to your VPC endpoint. Similarly, the Amazon SageMaker Runtime DNS hostname that the CLI and Amazon SageMaker Runtime SDK use by default (https://runtime.sagemaker.Region.amazonaws.com) resolves to your VPC endpoint.
The Amazon SageMaker API and Runtime support VPC endpoints in all AWS Regions where both Amazon VPC and Amazon SageMaker are available. Amazon SageMaker supports making calls to all of its Actions (p. 506) inside your VPC. The result AuthorizedUrl from the CreatePresignedNotebookInstanceUrl (p. 554) is not supported by Private Link. For information about how to enable PrivateLink for the authorized URL that users use to connect to a notebook instance, see Connect to a Notebook Instance Through a VPC Interface Endpoint (p. 44).

To learn more about AWS PrivateLink, see the AWS PrivateLink documentation and visit the AWS Blog. Refer to VPC Pricing for the price of VPC Endpoints. To learn more about VPC and Endpoints, see Amazon VPC.

Protect Training Jobs by Using an Amazon Virtual Private Cloud

Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud by default. However, training containers access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To control access to your data and training containers, we recommend that you create a private VPC and configure it so that they aren’t accessible over the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

You specify your private VPC configuration when you create training jobs by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your training containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

Configure a Training Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateTrainingJob (p. 556) API, or provide this information when you create a training job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your training containers. The ENIs provide your training containers with a network connection within your VPC that is not connected to the internet. They also enable your training job to connect to resources in your private VPC.

The following is an example of the VpcConfig parameter that you include in your call to CreateTrainingJob:

```json
VpcConfig: {
   "Subnets": [
      "subnet-0123456789abcdef0",
      "subnet-0123456789abcdef1",
      "subnet-0123456789abcdef2"
   ],
   "SecurityGroupIds": [
      "sg-0123456789abcdef0"
   ]
}
```
Configure Your Private VPC for Amazon SageMaker Training

When configuring the private VPC for your Amazon SageMaker training jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.

Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a training job. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that training containers don’t have access to the internet, they can’t connect to the Amazon S3 buckets that contain your training data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your training containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your training job needs.

```json
{
   "Version": "2008-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:ListBucket",
            "s3:GetBucketLocation",
            "s3:DeleteObject",
            "s3:ListMultipartUploadParts",
            "s3:AbortMultipartUpload"
         ],
         "Resource": "*"
      }
   ]
}
```

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don’t use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your training jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don’t want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```json
{
   "Statement": [
      {
```
Protect Communications Between ML Compute Instances in a Distributed Training Job

By default, Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud (Amazon VPC) to help keep your data secure. You can add another level of security to protect your training containers and data by configuring a private VPC. Distributed ML frameworks and algorithms usually transmit information that is directly related to the model such as weights, not the training dataset. When
performing distributed training, you can further protect data that is transmitted between instances. This can help you to comply with regulatory requirements. To do this, use inter-container traffic encryption. For more information about running training jobs in a private VPC, see Protect Training Jobs by Using an Amazon Virtual Private Cloud (p. 435).

Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. Enabling inter-container traffic encryption doesn't affect training jobs with a single compute instance. However, for training jobs with several compute instances, the effect on training time depends on the amount of communication between compute instances. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

**Using Inter-Container Traffic Encryption**

To enable inter-container traffic encryption:

1. Add the following inbound and outbound rules in the security group for your private VPC:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDP</td>
<td>500</td>
<td>Self Security Group ID</td>
</tr>
<tr>
<td>50</td>
<td>all</td>
<td>Self Security Group ID</td>
</tr>
</tbody>
</table>

2. When you send a request to the `CreateTrainingJob` API, specify `True` for the `EnableInterContainerTrafficEncryption` parameter. You can set the `EnableInterContainerTrafficEncryption` parameter only when configuring a private VPC.

**Protect Endpoints by Using an Amazon Virtual Private Cloud**

Amazon SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access AWS resources—such as the Amazon S3 buckets that you use to store model artifacts—over the internet.

To avoid making your data and model containers accessible over internet, we recommend that you create a private VPC and configure it to control access to them. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide. Using a VPC helps to protect your model containers and data because you can configure the VPC so that it isn't connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your model containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

You specify your VPC configuration when you create a model by specifying subnets and security groups. When you specify your subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) that are associated with your security groups in one of the specified subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

**Configure a Model for Amazon VPC Access**

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the `CreateModel` API or when you create a model in the Amazon SageMaker console.
SageMaker uses this information to create ENIs and then attaches them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. It also enables your model to connect to resources in your private VPC.

**Note**
You must create at least two subnets in different availability zones in your private VPC, even if you have only one hosting instance.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

```json
VpcConfig: {
    "Subnets": [
        "subnet-0123456789abcdef0",
        "subnet-0123456789abcdef1",
        "subnet-0123456789abcdef2"
    ],
    "SecurityGroupIds": [
        "sg-0123456789abcdef0"
    ]
}
```

## Configure Your Private VPC for Amazon SageMaker Hosting

Use the following guidelines to configure a private VPC for your Amazon SageMaker hosting jobs. For information about setting up a VPC, see [Working with VPCs and Subnets](https://docs.aws.amazon.com/vpc/latest/userguide/) in the Amazon VPC User Guide.

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two available private IP addresses for each instance for a hosted model. For more information, see [VPC and Subnet Sizing for IPv4](https://docs.aws.amazon.com/vpc/latest/userguide/) in the Amazon VPC User Guide.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, models that use that VPC can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint for access to the buckets. We recommend that you create a VPC endpoint with a custom policy that restricts access to your S3 buckets to only requests from your VPC. For more information, see [Endpoints for Amazon S3](https://docs.aws.amazon.com/vpc/latest/userguide/) in the Amazon VPC User Guide.

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your model needs.

```json
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "*"
        }
    ]
}
```
Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your models resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

**Note**
The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```json
{
  "Statement": [
    {
      "Sid": "AmazonLinuxAMIRepositoryAccess",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::packages.*.amazonaws.com/*",
        "arn:aws:s3:::repo.*.amazonaws.com/*"
      ]
    }
  ]
}

{
  "Statement": [
    {
      "Sid": "AmazonLinux2AMIRepositoryAccess",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
      ]
    }
  ]
}
```

**Connect Resources Outside Your VPC**

If you configure your VPC so that it doesn't have internet access, models that use that VPC do not have access to resources outside your VPC. If your model needs access to resources on the internet, provide access with one of the following options:

- If your model needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the Amazon VPC User Guide. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

- If your model needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound
Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud

Amazon SageMaker runs batch transform jobs in an Amazon Virtual Private Cloud by default. However, model containers access AWS resources—such as the Amazon S3 buckets where you store your data and model artifacts—over the internet.

To control access to your model containers and data, we recommend creating a private VPC and configuring it so that it is not connected to the internet. For information about creating and configuring a VPC, see Getting Started With Amazon VPC in the Amazon VPC User Guide.

With a VPC, you can monitor network traffic to and from your model containers by using VPC flow logs. For more information, see VPC Flow Logs in the Amazon VPC User Guide.

When you specify subnets and security groups, Amazon SageMaker creates elastic network interfaces (ENIs) and associates them with your security groups in one of the subnets. ENIs allow model containers to connect to resources in your VPC. For information about ENIs, see Elastic Network Interfaces in the Amazon VPC User Guide.

Configure a Batch Transform Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateModel (p. 538) API and provide this model when you create a batch transform job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your batch transform job to connect to resources in your private VPC.

The following is an example of the VpcConfig parameter that you include in your call to CreateModel:

```json
VpcConfig: {
    "Subnets": [
      "subnet-0123456789abcdef0",
      "subnet-0123456789abcdef1",
      "subnet-0123456789abcdef2"
    ],
    "SecurityGroupIds": [
      "sg-0123456789abcdef0"
    ]
}
```

Configure Your Private VPC for Amazon SageMaker Batch Transform

When configuring the private VPC for your Amazon SageMaker batch transform jobs, use the following guidelines. For information about setting up a VPC, see Working with VPCs and Subnets in the Amazon VPC User Guide.
Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a batch transform job. For more information, see VPC and Subnet Sizing for IPv4 in the Amazon VPC User Guide.

Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see Endpoints for Amazon S3.

The following policy allows access to S3 buckets. Edit this policy to allow access only the resources that your batch transform job needs.

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:DeleteObject",
                "s3:ListMultipartUploadParts",
                "s3:AbortMultipartUpload"
            ],
            "Resource": "*"
        }
    ]
}
```

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, http://s3-aws-region.amazonaws.com/MyBucket) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your batch transform jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see Routing for Gateway Endpoints in the Amazon VPC User Guide.

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from them, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
    "Statement": [
        {
            "Sid": "AmazonLinuxAMIRepositoryAccess",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::packages.*.amazonaws.com/**",
                "arn:aws:s3:::repo.*.amazonaws.com/**"
            ]
        }
    ]
}
```
Configure the VPC Security Group

In distributed batch transform, you must allow communication between the different containers in the same batch transform job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see Security Group Rules.

Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn’t have internet access, batch transform jobs that use that VPC do not have access to resources outside your VPC. If your batch transform jobs needs access to resources outside your VPC, provide access with one of the following options:

- If your batch transform job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see VPC Endpoints in the Amazon VPC User Guide. For information about creating an interface VPC endpoint, see Interface VPC Endpoints (AWS PrivateLink) in the Amazon VPC User Guide.
- If your batch transform job needs access to an AWS service that doesn’t support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see Scenario 2: VPC with Public and Private Subnets (NAT) in the Amazon Virtual Private Cloud User Guide.

Training and Inference Containers Run in Internet-Free Mode

Amazon SageMaker training and deployed inference containers are internet-enabled by default. This allows containers to access external services and resources on the public internet as part of your training and inference workloads. However, this offers an avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the container (in the form of a publicly available source code library) could access your data and transfer it to a remote host. If you use an Amazon VPC by specifying a value for the VpcConfig parameter when you call CreateTrainingJob (p. 556), CreateHyperParameterTuningJob (p. 528), or CreateModel (p. 538), you can protect your data and resources by managing security groups and restricting internet access from your VPC. However, this comes at the cost of additional network configuration, and has the risk of configuring your network incorrectly. If you do not want Amazon SageMaker to provide external network access to your training or inference containers, you can enable network isolation when you create your training job or model by setting the value of the EnableNetworkIsolation parameter.
to True when you call CreateTrainingJob (p. 556), CreateHyperParameterTuningJob (p. 528), or CreateModel (p. 538). If you enable network isolation, the containers are not able to make any outbound network calls, even to other AWS services such as Amazon S3. Additionally, no AWS credentials are made available to the container runtime environment. In the case of a training job with multiple instances, network inbound and outbound traffic is limited to the peers of each training container. Amazon SageMaker still performs download and upload operations against Amazon S3 using your Amazon SageMaker Execution Role in isolation from the training or inference container. Network isolation is required for training jobs and models run using resources from AWS Marketplace. Network isolation can be used in conjunction with a VPC. In this scenario, download and upload of customer data and model artifacts will are routed via your VPC subnet. However, the training and inference containers themselves continue to be isolated from the network, and do not have access to any resource within your VPC or on the internet.

Network isolation is not supported by the following managed Amazon SageMaker containers as they require access to Amazon S3:

- TensorFlow
- MXNet
- Chainer
- PyTorch
- Semantic Segmentation Algorithm

Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities

To meet our security requirements, algorithms and model packages listed in AWS Marketplace are scanned for Common Vulnerabilities and Exposures (CVE). CVE is a list of publicly known information about security vulnerability and exposure. The National Vulnerability Database (NVD) provides CVE details such as severity, impact rating, and fix information. Both CVE and NVD are available for public consumption and free for security tools and services to use. For more information, see http://cve.mitre.org/about/faqs.html#what_is_cve.
Amazon SageMaker Ground Truth

To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models. With Ground Truth, you can use workers from either Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce along with machine learning to enable you to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

In order to automate labeling your training dataset, you can optionally use automated data labeling, a Ground Truth process that uses machine learning to decide which data needs to be labeled by humans. Automated data labeling may reduce the labeling time and manual effort required. For more information, see Using Automated Data Labeling (p. 451).

Use either pre-built or custom tools to assign the labeling tasks for your training dataset. A labeling UI template is a webpage that Ground Truth uses to present tasks and instructions to your workers. The Amazon SageMaker console provides built-in templates for labeling data. You can use these templates to get started, or you can build your own tasks and instructions by using our HTML 2.0 components. For more information, see Creating Custom Labeling Workflows (p. 465).

Use the workforce of your choice to label your dataset. You can choose your workforce from:

- The Amazon Mechanical Turk workforce of over 500,000 independent contractors worldwide.
- A private workforce that you create from your employees or contractors for handling data within your organization.
- A vendor company that you can find in the AWS Marketplace that specializes in data labeling services.

For more information, see Managing Your Workforce (p. 461).

You store your datasets in Amazon S3 buckets. The buckets contain three things: The data to be labeled, an input manifest file that Ground Truth uses to read the data files, and an output manifest file. The output file contains the results of the labeling job. For more information, see Using Input and Output Data (p. 452).

Are You a First-time User of Ground Truth?

If you are a first-time user of Ground Truth, we recommend that you do the following:

1. **Read Getting started (p. 446)**—This section walks you through setting up your first Ground Truth labeling job.
2. **Explore other topics**—Depending on your needs, do the following:
   - **Create instruction pages for your labeling jobs**—Create a custom instruction page that makes it easier for your workers to understand the requirements of the job. For more information, see Creating Instruction Pages (p. 459).
   - **Manage your labeling workforce**—Create new work teams and manage your existing workforce. For more information, see Managing Your Workforce (p. 461).
   - **Create a custom UI**—Make it easier for your workers to quickly and correctly label your data by creating a custom UI for them to use. For more information, see Creating Custom Labeling Workflows (p. 465).
3. **See the API Reference (p. 506)**—This section describes operations to automate Ground Truth operations.
Getting started

To get started using Amazon SageMaker Ground Truth, follow the instructions in the following sections. You'll use the console to create a labeling job, assign a public or private workforce, and publish the labeling job to your workforce. You'll also learn how to monitor the progress of a labeling job.

If you want to create a custom labeling job, see Creating Custom Labeling Workflows (p. 465) for instructions.

Before you create a labeling job, you must upload your dataset to an Amazon S3 bucket. For more information, see Using Input and Output Data (p. 452).

Topics

- Step 1: Before You Begin (p. 446)
- Step 2: Create a Labeling Job (p. 447)
- Step 3: Select Workers (p. 447)
- Step 4: Configure the Bounding Box Tool (p. 448)
- Step 5: Monitoring Your Labeling Job (p. 449)

Step 1: Before You Begin

Before you begin using the Amazon SageMaker console to create a labeling job, you must set up the dataset for use. Do this:

1. Save two images at publicly available HTTP URLs. The images are used when creating instructions for completing a labeling task. The images should have an aspect ratio of around 2:1. For this exercise, the content of the images is not important.
2. Create an Amazon S3 bucket to hold the input and output files. The bucket must be in the same region as you are running Ground Truth. Make a note of the bucket name because you will use it during step 2.
3. Place 5–10 PNG images in the bucket.
4. Create a manifest file for the dataset and store it in the S3 bucket. Use these steps:
   b. Add a line similar to the following for each image file in your dataset:

   ```json
   {"source-ref": "s3://bucket/path/imageFile.png"}
   ```

   Add one line for each PNG file in your S3 bucket.
   c. Save the file in the S3 bucket containing your source files. Record the name because you use it in step 2.

   Note
   It is not necessary to store the manifest file in the same bucket as the source file. You use the same bucket in this exercise because it is easier.

   For more information, see Input Data (p. 453).

You also need to assign the following permissions policy to the user that is creating the labeling job:

```json
{
```
Step 2: Create a Labeling Job

In this step you use the console to create a labeling job. You tell Amazon SageMaker Ground Truth the Amazon S3 bucket where the manifest file is stored and configure the parameters for the job. For more information about storing data in an Amazon S3 bucket, see Using Input and Output Data (p. 452).

To create a labeling job

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. From the left navigation, choose Labeling jobs.
3. Choose Create labeling job to start the job creation process.
4. In the Job overview section, provide the following information:
   - **Job name** — Give the labeling job a name that describes the job. This name is shown in your job list. The name must be unique in your account in an AWS region.
   - **Input dataset location** — Enter the S3 location of the manifest file that you created in step 1.
   - **Output dataset location** — the location where your output data is written.
   - **IAM role** — Create or choose an IAM role with the SageMakerFullAccess IAM policy attached.
5. In the Task type section, for the Dataset type field, choose Bounding box as the task type.
6. Choose Next to move on to configuring your labeling job.

Next

Step 3: Select Workers (p. 447)

Step 3: Select Workers

In this step you choose a workforce for labeling your dataset. You can create your own private workforce or you can use the Amazon Mechanical Turk public workforce. If you create a private workforce in this
step you won't be able to import your Amazon Cognito user pool later. For more information, see Managing a Private Workforce (p. 463). Use the public workforce for this exercise instead.

You can create a private workforce to test Amazon SageMaker Ground Truth. Use email addresses to invite the members of your workforce.

**To create a private workforce**

1. In the **Workers** section, choose **Private**.
2. If this is your first time using a private workforce, in the **Email addresses** field, enter up to 100 email addresses. The addresses must be separated by a comma. You should include your own email address so that you are part of the workforce and can see data object labeling tasks.
3. In the **Organization name** field, enter the name of your organization. This information is used to customize the email sent to invite a person to your private workforce.
4. In the **Contact email** field enter an email address that members of the workforce use to report problems with the task.

If you choose to use the public workforce to label the dataset you will be charged for labeling tasks completed on the dataset.

**To use a public workforce**

1. In the **Workers** section, choose **Public**.
2. Choose **The dataset does not contain PII** to acknowledge that the dataset does not contain any personally identifiable information.
3. Choose **The dataset does not contain adult content**. The sample dataset doesn’t contain adult content.
4. Review and accept the statement that the dataset will be viewed by the public workforce.

**Next**

Step 4: Configure the Bounding Box Tool. (p. 448)

**Step 4: Configure the Bounding Box Tool.**

Finally you configure the bounding box tool to give instructions to your workers. You can configure a task title that describes the task and provides high-level instructions for the workers. You can provide both quick instructions that are displayed next to the image to be labeled and full instructions that contain detailed instructions for completing the task. In this example you only provide quick instructions, you can see an example of full instructions by choosing **Full instructions** at the bottom of the section.

**To configure the bounding box tool**

1. In the **Task description** field type in brief instructions for the task. For example:
   
   **Draw a box around any objects in the image.**
   
   Replace **objects** with the name of an object that appears in your images.
2. In the **Labels** field, type a category name for the objects that the worker should draw a bounding box around. For example, if you are asking the worker to draw boxes around football players, you could use "FootballPlayer" in this field.
3. The **Short instructions** section enables you to create instructions that are displayed on screen with the image that your workers are labeling. We suggest that you include an example of a correctly
drawn bounding box and an example of an incorrectly drawn box. To create your own instructions, use these steps:

a. Select the text between **GOOD EXAMPLE** and the image placeholder. Replace it with the following text:

   **Draw the box around the object with a small border.**

b. Select the first image placeholder and delete it.

c. Choose the image button and then enter the HTTPS URL of one of the images that you created in step 1.

d. Select the text between **BAD EXAMPLE** and the image placeholder. Replace it with the following text:

   **Don't make the bounding box too large or cut into the object.**

e. Select the second image placeholder and delete it.

f. Choose the image button and then enter the HTTPS URL of the other image that you created in step 1.

Configuration of your labeling job is complete. Choose **Submit** to start your job.

**Next**

Step 5: Monitoring Your Labeling Job (p. 449)

**Step 5: Monitoring Your Labeling Job**

After creating your labeling job you see a list of the jobs that you have created. You can use this list to monitor that status of your labeling jobs. The list has the following fields:

- **Name**—The name that you assigned the job when you created it.
- **Status**—The completion status of the job. The status can be one of Complete, Failed, In progress, or Stopped.
- **Labeled objects/total**—Shows the total number of objects in the labeling job and how many of them have been labeled.
- **Creation time**—The date and time that you created the job.

You can also clone or stop a job. Select a job and then press:

- **Clone**—Creates a new labeling job with the configuration copied from the selected job. You can clone a job when you want to make a change to the job and run it again. For example, you can clone a job that was sent to a private workforce so that you can send it to the public workforce. Or you can clone a job to rerun it against a new dataset stored in the same location as the original job.
- **Stop**—Stops a running job. You cannot restart a stopped job, though you can clone the job and start the new job later. Labels for any already labeled objects are written to the output file location. For more information see **Output Data** (p. 454).

**Data Labeling**

Amazon SageMaker Ground Truth manages sending your data objects to workers to be labeled. Labeling each data object is a task. Workers complete each task until the entire labeling job is complete. Ground Truth divides the total number of tasks into smaller batches that are sent to workers. A new batch is sent to workers when the previous one is finished.
Ground Truth provides two features that help improve the accuracy of your data labels and reduce the total cost of labeling your data.

The first, *annotation consolidation*, helps improve the accuracy of your data object's labels by combining the results of multiple worker's annotation tasks into one high-fidelity label.

The second, *automated data labeling*, uses machine learning to label portions of your data automatically without having to send them to human workers.

**Topics**
- Batches for Labeling Tasks (p. 450)
- Annotation Consolidation (p. 450)
- Using Automated Data Labeling (p. 451)

**Batches for Labeling Tasks**

Amazon SageMaker Ground Truth sends data objects to your workers in batches. There are one or more tasks for each data object. For each task a worker annotates one of your data objects. A batch provides the following:

- It sets the number of data objects that are available to workers. Once the objects are annotated another batch is sent.
- It breaks the work into smaller chunks to avoid overloading your workforce.
- It provides chunks of data for the iterative training of automated labeling models.

Ground Truth first sends a batch of 10 tasks to your workers. It uses this small batch to set up the labeling job and to make sure that the job is correctly configured.

After the small batch, Ground Truth sends larger batches to your workers. You can configure the batch size when you create the job using the `CreateLabelingJob` (p. 533) operation. When you use the Amazon SageMaker console your job uses 1,000 tasks in each batch.

**Annotation Consolidation**

Annotation consolidation combines the annotations of two or more workers into a single label for your data objects. An *annotation* is the result of a single worker. *Annotation consolidation* combines multiple annotations from different workers to come up with a probabilistic estimate of what the true label should be. The *label* is assigned to each object in the dataset. Each object in the dataset will typically have multiple annotations but only one label or set of labels.

You can decide how many workers should annotate each object in your dataset. More workers can increase the accuracy of your labels but also increases the cost of labeling the data. Amazon SageMaker Ground Truth uses the following defaults in the Amazon SageMaker console. When you use the `CreateLabelingJob` (p. 533) operation you set the number of workers that should annotate each data object using the `NumberOfHumanWorkersPerDataObject` parameter.

- **Text classification**—3 workers
- **Image classification**—3 workers
- **Bounding boxes**—5 workers
- **Semantic segmentation**—3 workers

You can override the default number of workers that label a data object using the console or the `CreateLabelingJob` (p. 533) operation.
Ground Truth provides an annotation consolidation function for each of its predefined labeling tasks: bounding box, image classification, semantic segmentation, and text classification.

- Multi-class annotation consolidation for image and text classification uses a variant of the Expectation Maximization approach to annotations. It estimates parameters for each worker and uses Bayesian inference to estimate the true class based on the class annotations from individual workers.
- Bounding box annotation consolidates bounding boxes from multiple workers by finding the most similar boxes from different workers based on the Jaccard index, or intersection over union, of the boxes and averaging them.
- Semantic segmentation annotation consolidation treats each pixel in a single image as a multi-class classification, and treats the pixel annotations from workers as "votes", with additional information from surrounding pixels incorporated by applying a smoothing function to the image.

**Creating Your Own Annotation Consolidation Function**

There are many possible approaches for writing an annotation consolidation function. The approach that you take depends on the nature of the annotations to consolidate. Broadly, consolidation functions should look at the annotations from workers, measure the similarity between them, and then use some form of probabilistic judgment to determine what the most probable label should be.

**Assessing Similarity**

To assess the similarity between labels, you can use one of the following strategies or you can use one that meets your data labeling needs:

- For label spaces that consist of discrete, mutually exclusive categories, such as multi-class classification, assessing similarity can be straightforward. Discrete labels either match or they don't.
- For label spaces that don't have discrete values, such as bounding box annotations, you instead need to find some broader measure of similarity. In the case of bounding boxes, one such measure is the Jaccard index. This measures the ratio of the intersection of two boxes with the union of the boxes to assess how similar they are. For example, if there are 3 annotators there needs to be some function, based on similarity, to determine which boxes represent the same object and should be consolidated together.

**Assessing the Most Probable Label**

Next you need to make some sort of probabilistic judgement on what the consolidated label should be. In the case of discrete, mutually exclusive categories this can be straightforward. One of the most common ways to do this is to take the results of a majority vote between the annotations. This weights the annotations equally.

More sophisticated approaches, such as the Expectation Maximization method used in the default Ground Truth consolidation function for multi-class annotations, attempt to estimate the accuracy of different annotators and weight their annotations in proportion to the probability of correctness.

For more information about creating an annotation consolidation function, see Step 3: Processing with AWS Lambda (p. 473).

**Using Automated Data Labeling**

Ground Truth can use active learning to automate the labeling of your input data. Active learning is a machine learning technique that identifies data that should be labeled by your workers.

Automated data labeling is optional, you turn it on when you create a labeling job. Automated data labeling incurs Amazon SageMaker training and inference costs, but it can help to reduce the cost and time that it takes to label your dataset compared to having humans label your entire dataset.
You should use automated data labeling only on large datasets. The neural networks used with active learning require a significant amount of data for every new dataset. With larger datasets there is more potential to automatically label the data and therefore reduce the total cost of labeling. We recommend that you use thousands of data objects when using automated data labeling. You must use at least 5,000 data objects.

The potential benefit of automated data labeling also depends on the accuracy that you require. Higher accuracy levels generally reduce the number of data objects that are automatically labeled.

When Amazon SageMaker Ground Truth starts an automated data labeling job, it first selects a random sample of the input data and sends the sample to human workers. When the labeled data are returned, Ground Truth uses this set of data as validation data. It is used to validate the machine learning models that Ground Truth trains for automated data labeling.

Next, Ground Truth runs an Amazon SageMaker batch transform using the validation set. This generates a quality metric that Ground Truth uses to estimate the potential quality of auto-labeling the rest of the unlabeled data.

Ground Truth next runs an Amazon SageMaker batch transform on the unlabeled data in the dataset. Any data where the expected quality of automatically labeling the data is above the requested level of accuracy is considered labeled.

After performing the auto-labeling step, Ground Truth selects a new sample of the most ambiguous unlabeled data points in the dataset and sends those to human workers. Ground Truth uses the existing labeled data and this additional labeled data from human workers to train a new model. The process is repeated until the dataset is fully labeled.

Amazon EC2 Instances Required for Automated Data Labeling

To run automated data labeling, Ground Truth requires the following Amazon EC2 resources for training and batch inference jobs:

<table>
<thead>
<tr>
<th>Automated labeling action</th>
<th>Training instance type</th>
<th>Inference instance type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image classification</td>
<td>ml.p3.2xlarge</td>
<td>ml.c5.xlarge</td>
</tr>
<tr>
<td>Object detection</td>
<td>ml.p3.2xlarge</td>
<td>ml.c5.4large</td>
</tr>
<tr>
<td>Text classification</td>
<td>ml.c5.2xlarge</td>
<td>ml.m4.xlarge</td>
</tr>
</tbody>
</table>

To check the Amazon EC2 service limits for your account, and to request additional resources if necessary, see Amazon EC2 Service Limits.

Using Input and Output Data

The input data that you provide Amazon SageMaker Ground Truth is sent to your workers for labeling. You can choose the data to send to your workers by creating a manifest file that defines the data to label.

The output data is the result of your labeling job. The output data file contains one entry that specifies the label for each object in the input dataset.

Topics
- Input Data (p. 453)
- Output Data (p. 454)
Input Data

The input data are the data objects that you send to your workforce to be labeled. Each object in the input data is described in a manifest file. Each line in the manifest is an entry containing an object to label and may contain labels from previous jobs.

- The input data is stored in an Amazon S3 bucket. The bucket must be in the same region as you are running Amazon SageMaker Ground Truth. You must give access to Amazon SageMaker for the data to be read. In order to read the data, give access to Amazon SageMaker. For more information about Amazon S3 buckets, see Working with Amazon S3 buckets.
- The manifest file must be in the same region the data files but does not need to be in the same location as the data files. It can be in any Amazon S3 bucket accessible to the role that you assigned to Ground Truth when you created the labeling job with either the console or the CreateLabelingJob (p. 533) operation.

The manifest is a UTF-8 encoded file where each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \\n. Since each line must be a valid JSON object, you can’t have unescaped line break characters. For more information about the data format, see JSON Lines.

Each JSON object in the manifest file must contain a key, either source-ref or source. The value of the keys are interpreted as follows:

- source-ref—The source of the object is the S3 object specified in the value. This can be used when the object is a binary object, such as an image, or when you have text in individual files.
- source—The source of the object is the value. This can be used when the object is a text value.

You use the source-ref key for image files for a bounding box or semantic segmentation labeling job. Each image file must be:

- 6 Mb or smaller
- 1920 x 1080 pixels or smaller for semantic segmentation

The following is an example of a manifest file for files stored in an S3 bucket:

```
{"source-ref": "S3 bucket location 1"}
{"source-ref": "S3 bucket location 2"}
...
{"source-ref": "S3 bucket location n"}
```

The following is an example of a manifest file with the input data stored in the manifest:

```
{"source": "Lorem ipsum dolor sit amet"}
{"source": "consectetur adipiscing elit"}
...
{"source": "mollit anim id est laborum"}
```

You can include other key-value pairs in the manifest file. These pairs are passed to the output file unchanged. This is useful when you want to pass information between your applications. For more information, see Output Data (p. 454).

Filtering and Selecting Data (Console)

You can use the Amazon SageMaker console to select a portion of your dataset for labeling. The data must be stored in an Amazon S3 bucket. You have three options:
- Use the full dataset.
- Choose a randomly selected sample of the dataset.
- Specify a subset of the dataset using a query.

**Output Data**

The output from a labeling job is placed in the location that you specified in the console or in the call to the `CreateLabelingJob` operation.

Each line in the output data file is identical to the manifest file with the addition of an attribute and value for the label assigned to the input object. The attribute name for the value is defined in the console or in the call to the `CreateLabelingJob` operation. You can't use `-metadata` in the label attribute name. If you are running a semantic segmentation job, the label attribute must end with `-ref`. For any other type of job, the attribute name can't end with `-ref`.

The output of the labeling job is the value of the key/value pair with the label. The label and the value overwrites any existing JSON data in the input file with the new value.

For example, the following is the output from an image classification labeling job where the input data files were stored in an Amazon S3 bucket and the label attribute name was defined as "sport". In this example the JSON object is formatted for readability, in the actual output file the JSON object is on a single line. For more information about the data format, see JSON Lines.

```json
{
  "source-ref": "S3 bucket location",
}
```
"sport":0,
"sport-metadata":
{
  "class-name": "football",
  "confidence": 0.8,
  "type": "groundtruth/image-classification",
  "job-name": "identify-sport",
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256"
}
}

The value of the label can be any valid JSON. In this case the label's value is the index of the class in the classification list. Other job types, such as bounding box, have more complex values.

Any key-value pair in the input manifest file other than the label attribute is unchanged in the output file. You can use this to pass data to your application.

The output from a labeling job can be used as the input to another labeling job. You can use this when you are chaining together labeling jobs. For example, you can send one labeling job to determine the sport that is being played. Then you send another using the same data to determine if the sport is being played indoors or outdoors. By using the output data from the first job as the manifest for the second job, you can consolidate the results of the two jobs into one output file for easier processing by your applications.

The output data file is written to the output location periodically while the job is in progress. These intermediate files contain one line for each line in the manifest file. If an object is labeled, the label is included, if the object has not been labeled it is written to the intermediate output file identically to the manifest file.

Output Directories

Ground Truth creates several directories in your Amazon S3 output path. These directories contain the results of your labeling job and other artifacts of the job. The top-level directory for a labeling job is given the same name as your labeling job, the output directories are placed beneath it. For example, if you named your labeling job **find-people** you output would be in the following directories:

```
s3://bucket/find-people/activelearning
s3://bucket/find-people/annotations
s3://bucket/find-people/inference
s3://bucket/find-people/manifests
s3://bucket/find-people/training
```

Each directories contain the following output:

**activelearning Directory**

The activelearning directory is only present when you are using automated data labeling. It contains the input and output validation set for automated data labeling, and the input and output folder for automatically labeled data.

**annotations Directory**

The annotations directory contains all of the annotations made by the workforce. These are the responses from individual workers that have not been consolidated into a single label for the data object.

There are three subdirectories in the annotations directory. The first, worker-response contains the responses from individual workers. There may be more than one annotation for each data object in this directory, depending on how many workers you want to annotate each object.
The second, consolidated-annotation contains information required to consolidate the annotations in the current batch into labels for your data objects.

The third, intermediate, contains the output manifest for the current batch with any completed labels. This file is updated as the label for each data object is completed.

**inference Directory**

The inference directory contains the input and output files for the Amazon SageMaker batch transform used while labeling data objects.

**manifest Directory**

The manifest directory contains the output manifest from your labeling job. There are two subdirectories in the manifest directory, output and intermediate. The output directory contains the output manifest file for your labeling job. The file is named output.manifest.

The other directory, intermediate, contains the results of labeling each batch of data objects. The intermediate data is in a directory numbered for each iteration. An iteration directory contains an output.manifest file that contains the results of that iteration and all previous iterations.

**intermediate Directory**

The training directory contains the input and output files used to train the automated data labeling model.

**Confidence Score**

Ground Truth calculates a confidence score for each label. A confidence score is a number between 0 and 1 that indicates how confident Ground Truth is in the label. You can use the confidence score to compare labeled data objects to each other, and to identify the least or most confident labels.

You should not interpret the value of the confidence scores as an absolute value, or compare them across labeling jobs. For example, if all of the confidence scores are between 0.98 and 0.998, you should only compare the data objects with each other and not rely on the high confidence scores.

You should not compare the confidence scores of human-labeled data objects and auto-labeled data objects. The confidence scores for humans are calculated using the annotation consolidation function for the task, the confidence scores for automated labeling are calculated using a model that incorporates object features. The two models generally have different scales and average confidence.

For a bounding box labeling job, Ground Truth calculates a confidence score per box. You can compare confidence scores within one image or across images for the same labeling type (human or auto). You can't compare confidence scores across labeling jobs.

**Output Metadata**

The output from each job contains metadata about the label assigned to data objects. These elements are the same for all jobs with minor variations. The following are the metadata elements:

```
"confidence": 0.93,
"type": "groundtruth/image-classification",
"job-name": "identify-animal-species",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256"
```

The elements have the following meaning:

- **confidence** — The confidence that Ground Truth has that the label is correct. For more information, see Confidence Score (p. 456).
Output Data

- **type** — The type of classification job. For job types, see the descriptions of the individual job types.
- **job-name** — The name assigned to the job when it was created.
- **human-annotated** — Indicates whether the data object was labeled by a human or by automated data labeling. For more information, see Using Automated Data Labeling (p. 451).
- **creation-date** — The date and time that the label was created.

Classification Job Output

The following are sample output from an image classification job and a text classification job. It includes the label that Ground Truth assigned to the data object, the value for the label, and metadata that describes the labeling task.

In addition to the standard metadata elements, the metadata for a classification job includes the text value of the label's class. For more information, see Image Classification Algorithm (p. 106).

```
{
  "source-ref": "S3 bucket location",
  "species": "0",
  "species-metadata":
  {
    "class-name": "dog",
    "confidence": 0.93,
    "type": "groundtruth/image-classification",
    "job-name": "identify-animal-species",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
}
```

```
{
  "source": "a bad day",
  "mood": "1",
  "mood-metadata":
  {
    "class-name": "sad",
    "confidence": 0.8,
    "type": "groundtruth/text-classification",
    "job-name": "label-mood",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
}
```

Bounding Box Job Output

The following is sample output from a bounding box job. For this task, there are three bounding boxes returned. The value of the label contains information about the size of the image, and the location of the bounding boxes.

The **class_id** element is the index of the box's class in the list of available classes for the task. You can see the text of the class in the **class-map** metadata element.

In the metadata there is a separate confidence score for each bounding box. The metadata also includes the **class-map** element that maps the **class_id** to the text value of the class. For more information, see Object Detection Algorithm (p. 192).

```
{
  "source-ref": "S3 bucket location",
}
```
"bounding-box":
{
    "image_size": [{ "width": 500, "height": 400, "depth":3}],
    "annotations":
    [   
        {"class_id": 0, "left": 111, "top": 134,  
            "width": 61, "height": 128},
        {"class_id": 5, "left": 161, "top": 250,  
            "width": 30, "height": 30},
        {"class_id": 5, "left": 20, "top": 20,  
            "width": 30, "height": 30}
    ],
    "bounding-box-metadata":
    {
        "objects":
        [   
            {"confidence": 0.8},
            {"confidence": 0.9},
            {"confidence": 0.9}
        ],
        "class-map":
        {   
            "0": "dog",
            "5": "bone"
        },
        "type": "groundtruth/object_detection",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256",
        "job-name": "identify-dogs-and-toys"
    }
}

For an example notebook, see object_detection_augmented_manifest_training.ipynb

Semantic Segmentation Job Output

The following is the output from a semantic segmentation labeling job. The value of the label for this job is a reference to a PNG file in an S3 bucket.

In addition to the standard elements, the metadata for the label includes a color map that defines which color was used to label the image, the class name associated with the color, and the confidence score for each color. For more information, see Semantic Segmentation Algorithm (p. 225).

{   
    "source-ref": "S3 bucket location",
    "city-streets-ref": "S3 bucket location",
    "city-streets-metadata": {
        "internal-color-map": {
            "5": {
                "class-name": "buildings",
                "confidence": 0.8
            },
            "2": {
                "class-name": "road",
                "confidence": 0.9
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256",
        "job-name": "label-city-streets",
    }
}
After you create an augmented manifest file, you can use it in a training job. For more information, see Provide Dataset Metadata to Training Jobs with an Augmented Manifest File (p. 307).

Creating Instruction Pages

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console or you can create your own. The instructions are shown to the worker on the page where they complete their labeling task.

There are two kinds of instructions:

- **Short instructions**—instructions that are shown on the same webpage where the worker completes their task. These instructions should provide an easy reference to show the worker the correct way to label an object.

- **Full instructions**—instructions that are shown on a dialog box that overlays the page where the worker completes their task. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

Create instructions in the console when you are creating your labeling job. Start with the existing instructions for the task and use the editor to modify them to suit your labeling job.

**Short Instructions**

Short instructions appear on the same webpage that workers use to label your data object. For example, the following is the editing page for a bounding box task. The short instructions panel is on the left.
Keep in mind that a worker will only spend seconds looking at the short instructions. Workers must be able to scan and understand your information quickly. In all cases it should take less time to understand the instructions than it takes to complete the task. Keep these points in mind:

- Your instructions should be clear and simple.
- Pictures are better than words. Create a simple illustration of your task that your workers can immediately understand.
- If you must use words, use short, concise examples.
- Your short instructions are more important than your full instructions.

The Amazon SageMaker Ground Truth console provides an editor so that you can create your short instructions. Replace the placeholder text and images with instructions for your task. Preview the worker’s task page by choosing Preview. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

**Full Instructions**

You can provide additional instructions for your workers in a dialog box that overlays the page where workers label your data objects. Use full instructions to explain more complex tasks and to show workers the proper way to label edge cases or other difficult objects.
You can create full instructions using an editor in the Ground Truth console. As with quick instructions, keep the following in mind:

- Workers will want detailed instruction the first few times that the complete your task. Any information that they must have should be in the quick instructions.
- Pictures are more important than words.
- Text should be concise.
- Full instructions should supplement the short instructions. Don’t repeat information that appears in the short instructions.

The Ground Truth console provides an editor so that you can create your full instructions. Replace the placeholder text and images with instructions for your task. Preview the full instruction page by choosing Preview. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

Managing Your Workforce

A workforce is the group of workers that you have selected to label your dataset. You can choose either a public workforce consisting of the Amazon Mechanical Turk workforce, a vendor-managed workforce, or you can create your own private workforce to label your dataset. Whichever you choose, Amazon SageMaker Ground Truth takes care of sending tasks to workers.

When you use a private workforce, you also create work teams, a group of workers from your workforce that are assigned to specific labeling jobs. You can have multiple work teams and can assign one or more work teams to each labeling job.

Ground Truth uses Amazon Cognito to manage your workforce and work teams. For more information about the permissions required to manage this way, see Permissions Required to Use the Amazon SageMaker Ground Truth Console (p. 407).

Topics
- Using the Public Workforce (p. 461)
- Managing Vendor Workforces (p. 462)
- Managing a Private Workforce (p. 463)

Using the Public Workforce

Use a public workforce when you want the Amazon Mechanical Turk workforce to label your dataset. The public workforce provides the most workers for your labeling job.

You can use the console to choose a public workforce for your labeling job, or you can provide the Amazon Resource Name (ARN) for the public workforce when you use the CreateLabelingJob (p. 533) operation.

The ARN for the public workforce is:


The public workforce is a world-wide resource. Workers are available 24 hours a day, 7 days a week. You typically get the fastest turn-around for your labeling jobs when you use the public workforce.

Adjust the number of workers that annotate each data object based on the complexity of the job and the quality that you need. Ground Truth uses annotation consolidation to improve the quality...
of the labels. More workers can make a difference in the quality of the labels for more complex labeling jobs, but might not make a difference for simpler jobs. For more information, see Annotation Consolidation (p. 450).

Note
Only use the public workforce to label data that is public or has been stripped of any sensitive information. You should not use the public workforce for a dataset that contains personally identifiable information, such as names or email addresses.

To choose the public workforce when you are creating a labeling job using the console, do the following during the Select workers and configure tool step:

To use the public workforce
1. Choose Public from Worker types.
2. Choose The dataset does not contain adult content if your dataset doesn't contain potentially offensive content. This enables workers to opt out if they don't want to work with it.
3. Acknowledge that your data will be viewed by the Amazon Mechanical Turk workforce and that all personally identifiable information (PII) has been removed.
4. Choose Additional configuration to set optional parameters.
5. Optional. Enable automated data labeling to have Ground Truth automatically label some of your dataset. For more information, see Using Automated Data Labeling (p. 451).
6. Optional. Set the number of workers that should see each object in your dataset. Using more workers can increase the quality of your labels but also increases the cost.

Your labeling job will now be sent to the public workforce. You can use the console to continue configuring your labeling job.

Managing Vendor Workforces

You can use a vendor-managed workforce to label your data using Amazon SageMaker Ground Truth. Vendors have extensive experience in providing data labeling services for the purpose of machine learning.

Vendors make their services available via the AWS Marketplace. You can find details of the vendor's services on their detail page, such as the number of workers and the hours that they work. You can use these details to make estimates of how much the labeling job will cost and the amount of time that you can expect the job to take. Once you have chosen a vendor you subscribe to their services using the AWS Marketplace.

A subscription is an agreement between you and the vendor. The agreement spells out the details of the agreement, such as price, schedule, or refund policy. You work directly with the vendor if there are any issues with your labeling job.

You can subscribe to any number of vendors to meet your data annotation needs. When you create a labeling job you can specify that the job be routed to a specific vendor.

Before you send sensitive data to a vendor, check the vendor's security practices on their detail page and review the end user license agreement (EULA) that is part of your subscription agreement.

You must use the console to subscribe to a vendor workforce. Once you have a subscription, you can use the ListSubscribedWorkteams (p. 683) operation to list your subscribed vendors.

To subscribe to a vendor workforce
1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Labeling workforces**, choose **Vendor**, and then choose **Find data labeling services**.
3. The console opens the AWS Marketplace with the data labeling services category selected. You see a list of the data labeling services available.
4. Choose a vendor. The AWS Marketplace shows detailed information about the data labeling service. Use this information to determine if the vendor meets your data labeling requirements.
5. If the vendor meets your requirements, choose **Continue to subscribe**.
6. Review the details of the subscription. If you agree to the terms, choose **Subscribe** to complete your subscription to the service.

Managing a Private Workforce

A **private workforce** is a group of workers that you choose. These can be employees of your company or a group of subject matter experts from your industry. For example, if the task is to label medical images, you could create a private workforce of people knowledgeable about the images in question.

You create **work teams** from your private workforce. If desired, you can assign each work team to a separate labeling job. A single worker can be in more than one work team.

Amazon SageMaker Ground Truth uses Amazon Cognito to define your private workforce and your work teams. Amazon Cognito is a service that you can use to create identities for your workers and authenticate these identities with identity providers. You can use providers such as the following:

- Amazon Cognito identity provider
- Social sign-in providers such as Facebook and Google
- OpenID Connect (OIDC) providers
- Security Assertion Markup Language (SAML) providers such as Active Directory

After your workers are set up, you use Amazon Cognito to manage them. For more information about Amazon Cognito, see [What Is Amazon Cognito?](#).

Your workers are organized into two groups. The **workforce** is the entire set of workers that are available to work on your labeling jobs. The workforce corresponds to an Amazon Cognito user pool. A **work team** is a group of workers within your workforce that you can assign jobs to. The work team corresponds to an Amazon Cognito user group. A worker can be in more than one work team.

**Note**
You can only use one Amazon Cognito user pool as your Ground Truth labeling workforce. If you plan on using an existing Amazon Cognito user pool for your Ground Truth workforce, be sure to import the user pool and create a work team before you create your first labeling job.

For more information, see [Amazon Cognito User Pools](#).

Creating a private workforce

There are three ways that you can create a private workforce:

1. Import an existing Amazon Cognito user pool before you create your first labeling job.
2. Use the Amazon SageMaker console to create a new Amazon Cognito user pool before you create your first labeling job.
3. Create a new Amazon Cognito user pool while you are creating your first labeling job.

If you are using a pre-existing Amazon Cognito user pool for your private workforce, you must import the user pool into Ground Truth and create at least one work team, either by adding work team members in
the Amazon SageMaker console or by importing an Amazon Cognito work group before you create your first labeling job for your private workforce.

After you create or import your private workforce you will see the **Private workforce summary** screen. On this screen you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce.

If you created your workforce using the Amazon SageMaker console, you can manage the members of your workforce in the Amazon SageMaker console or in the Amazon Cognito console. For example, you can use the Amazon SageMaker console to add, delete, and disable workers in the pool. If you imported the workforce from an existing Amazon Cognito user pool, you must use the Amazon Cognito console to manage the workforce.

**Creating a workforce when creating a labeling job**

If you have not created a private workforce when you create your labeling job, you are prompted to create one. Creating the workforce also creates a default work team containing all of the members of the workforce. If you have already created a private workforce, you instead enter the work team that should handle the labeling job.

You provide the following information to create the workforce and work team.

- Up to 100 email addresses of the workforce members. Email addresses are case-sensitive. Your workers must log in using the same case as the address was initially entered. You can add additional workforce members after the job is created.
- The name of your organization. This is used to customize the email sent to the workers.
- A contact email address for workers to report issues related to the task.

When you create the labeling job an email is sent to each worker inviting them to join the workforce. Once the workforce is created, you can add, delete, and disable workers using the Amazon SageMaker console or the Amazon Cognito console.

**Creating a workforce using the console**

You can add up to 100 workers when you create the workforce, and then add additional workers if necessary.

**To create a private workforce using the console**

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose **Labeling workforces** from the left menu.
3. Under **Add workers using Amazon Cognito**, choose **Create a new Amazon Cognito user pool with worker email addresses**.
4. Under **Email addresses**, add the email addresses of your private workforce. You can add up to 100 addresses. Email addresses are case-sensitive. Your workers must log in using the same case as the address was initially entered. If you want to add more addresses, you can update the workforce to include them later.
5. Under **Invitation content**, enter information to customize the email that is sent to your private workforce to invite them to participate. Choose **Preview invitation** to see a preview of the email that is sent to invite your workforce.
6. Choose **Create private workforce** to start creating your private workforce.

After you have created the workforce, Ground Truth automatically creates a work team called **Everyone-in-private-workforce**. You can use this work team to assign a labeling job to your entire workforce.
Creating a Work Team

Use a work team to assign members of your private workforce to a labeling job. When you create your workforce using the Amazon SageMaker Ground Truth console, there is a work team called Everyone-in-private-workforce that you can use if you want to assign your entire workforce to a labeling job. Because an imported Amazon Cognito user pool may contain members that you don't want to include in your work teams, a similar work team is not created for Amazon Cognito user pools.

You have two choices to create a new work team. You can create a user group by using the Amazon Cognito console and then create a work team by importing the user group. You can import more than one user group into each work team. You manage the members of the work team by updating the user group in the Amazon Cognito console.

You can also create a work team in the Ground Truth console and add members from your workforce to the team.

To create a work team using the Ground Truth console

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose Labeling workforces from the left menu.
3. Under Private teams, choose Create private team.
4. Under Team details, give the team a name. The name must be unique in your account in an AWS Region.
5. Under Create a team using Amazon Cognito user groups, choose the method to use to create the group.
6. If you chose Create a team by adding workers to a new Amazon Cognito user group, select the workers to add to the team.
   
   If you chose Create a team by importing existing Amazon Cognito user groups, choose the user groups that are part of the new team.

After you have created a work team, you can choose its name in the list of work teams to see more information about the team. Teams made from more than one imported user group must be edited in the Amazon Cognito console; otherwise you can add and remove workers using the team detail page.

Creating Custom Labeling Workflows

This document will guide you through the process of setting up a workflow with a custom labeling template. For the basics of starting a labeling job, see our Getting started (p. 446) section. When you get to choosing the Task type, choose Custom labeling task, then follow this section's instructions to configure it.

Next

Step 1: Setting up your workforce (p. 465)

Step 1: Setting up your workforce

In this step you use the console to establish which worker type you'll use and make the necessary subselections for the worker type. It assumes you have already completed the steps up to this point in the Getting started (p. 446) section and have chosen the Custom labeling task as the Task type.
To configure your workforce.

1. First choose an option from the **Worker types**. There are three types currently available:
   - **Public** uses an on-demand workforce of independent contractors, powered by Amazon Mechanical Turk. They are paid on a per-task basis.
   - **Private** uses your employees or contractors for handling data that needs to stay within your organization.
   - **Vendor** uses third party vendors that specialize in providing data labeling services, available via the AWS Marketplace.

2. If you choose the **public** option, you'll be asked set the **number of workers per dataset object**. Having more than one worker perform the same task on the same object can help increase the accuracy of your results. The default is three. You can raise or lower that depending on the accuracy you need.

3. If you have chosen the **Public** option, you'll also be asked to set a **price per task**, using a drop-down menu that recommends price points based on how long you believe it will take to complete the task. The recommended method to determine this is to first run a short test of your task with a **private** workforce. That provides a realistic estimate of how long the task takes to complete. You can then select the range your estimate falls within on the **Price per task** menu. If your average time is over 5 minutes, consider breaking your task into smaller units.

Next

Step 2: Creating your custom labeling task template (p. 466)

Step 2: Creating your custom labeling task template

Starting with a base template

To get you started, the **Task type** starts with a drop-down menu listing a number of our more common task types, plus a custom type. Choose one and the code editor area will be filled with a sample template for that task type. If you prefer not to start with a sample, choose **Custom HTML** for a minimal template skeleton.

If you've already created a template, upload the file directly using the **Upload file** button in the upper right of the task setup area or paste your template code into the editor area.

Developing templates locally

While you need to be in the console to test how your template will process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding this code to the top of your HTML file.

Example

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you prefer to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this will not parse your variables. You may want to replace them with sample content while developing locally.
Track your variables

In the process of building the sample below, there will be a step that adds variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you will need to make sure you're aware of the variables it already uses. When you create your pre-processing AWS Lambda script, its output will need to contain values for any of those variables you choose to keep.

A simple sample

All tasks begin and end with the `<crowd-form>` </crowd-form> elements. Like standard HTML <form> elements, all of your form code should go between them.

For a simple tweet-analysis task, use the `<crowd-classifier>` element. It requires the following attributes:

• name - the variable name to use for the result in the form output.
• categories - a JSON formatted array of the possible answers.
• header - a title for the annotation tool

As children of the `<crowd-classifier>` element, you must have three regions.

• `<classification-target>` - the text the worker will classify based on the options specified in the categories attribute above.
• `<full-instructions>` - instructions that are available from the "View full instructions" link in the tool. This can be left blank, but it is recommended that you give good instructions to get better results.
• `<short-instructions>` - a more brief description of the task that appears in the tool's sidebar. This can be left blank, but it is recommended that you give good instructions to get better results.

A simple version of this tool would look like this.

Example of using crowd-classifier

```html
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    sentiment="['positive','negative','neutral','cannot determine']"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      My favorite football team won today!
      Bring on the division finals!
    </classification-target>

    <full-instructions header="Sentiment Analysis Instructions">
      Try to determine the sentiment the author of the tweet is trying to express.
      If none seem to match, choose "cannot determine."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```
You can copy and paste the code into the editor in the console to preview the tool.

**Adding automation with Liquid**

Our custom template system uses Liquid for automation. It is an open source inline markup language. For more information and documentation, visit the Liquid homepage.

The most common use of Liquid will be to parse the data coming from your **pre-labeling task Lambda** and pull out the relevant variables to create the task. In Liquid, the text between single curly braces and percent symbols is an instruction or "tag" that creates control flow. Text between double curly braces is a variable or "object" which outputs its value.

**Variable filters**

In addition to the standard Liquid filters and actions, Ground Truth offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, then specifying a filter name. Filters can be chained in the form of:

```
Example

{{ <content> | <filter> | <filter> }}
```

**Autoescape and explicit escape**

By default, inputs will be HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the escape filter to make it more obvious to someone reading the source of your template that the escaping is being done.

- escape_once
  - escape_once ensures that if you've already escaped your code, it doesn't get re-escaped on top of that. For example, so that & doesn't become &amp;.

- skip_autoescape
  - skip_autoescape is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

  **Use skip_autoescape sparingly**
  - The best practice in templates is to avoid passing in functional code or markup with skip_autoescape unless you are absolutely sure you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a Cross Site Scripting attack.

- to_json
  - to_json will encode what you feed it to JSON (JavaScript Object Notation). If you feed it an object, it will serialize it.

- grant_read_access
  - grant_read_access takes an S3 URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible.

**Example of the filters**

Input
### Example

**Output**

```html
auto-escape: Have you read 'James & the Giant Peach'?
explicit escape: Have you read 'James & the Giant Peach'?
explicit escape_once: Have you read 'James & the Giant Peach'?
skip_autoescape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?
```

### Example of an automated classification template.

To automate the simple text classification sample, replace the tweet text with a variable.

The text classification template is below with automation added. The changes/additions are highlighted in bold.

```html
<!-- Endcode HTML code here -->
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="["positive", "negative", "neutral", "cannot determine"]"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seem to match, choose "other."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>

  </crowd-classifier>
</crowd-form>
```

The tweet text that was in the prior sample is now replaced with an object. The `entry.taskInput` object uses `source` (or another name you specify in your pre-processing Lambda) as the property name for the text and it is inserted directly in the HTML by virtue of being between double curly braces.

### End-to-end demos

You can view the following end-to-end demos which include sample manifests and sample Lambdas:

- **Demo Template: Bounding Box (p. 470)**
Next

Step 3: Processing with AWS Lambda (p. 473)

Demo Template: Bounding Box

After you have selected to use a custom template, you'll reach the Custom labeling task panel. There you can choose from multiple base templates that represent some of the more common tasks and will provide a sample to work from in building your customized labeling task's template.

In this demonstration, you'll work with the BoundingBox template and the AWS Lambda functions needed for processing your data before and after the task.

Example : Base Bounding Box Custom Template

This is the Bounding Box template that is provided.

```html
<crowd-form>
  <crowd-bounding-box
    name="boundingBox"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="{{ task.input.header }}"
    labels="{{ task.input.labels | to_json | escape }}"
  >
    <full-instructions header="Bounding box instructions">
      Sample Full Instructions with examples
    </full-instructions>
    <short-instructions>
      Sample Short Instructions
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

The custom templates use the Liquid template language, and each of the items between double curly braces is a variable. The pre-processing AWS Lambda function should provide an object named taskInput and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

In the base sample, there are three variables: taskObject, header, and labels.

Each of these would be represented in different parts of the bounding box.

- *taskObject* is an HTTP(S) URL or S3 URI for the photo to be annotated. The added | grant_read_access is a filter that will convert an S3 URI to an HTTPS URL with short-lived access to that resource. If you're using an HTTP(S) URL, it's not needed.
- *header* is the text above the photo to be labeled, something like "Draw a box around the bird in the photo."
- *labels* is an array, represented as `[ 'item1', 'item2', ... ]`. These are labels that can be assigned by the worker to the different boxes they draw. You can have one or many.

Each of the variable names comes from the JSON object in the response from your pre-processing Lambda. The names above are merely suggested, Use whatever variable names make sense to you and will promote code readability among your team.

Only use variables when necessary

If a field will not change, you can remove that variable from the template and replace it with that text, otherwise you have to repeat that text as a value in each object in your manifest.
Example: Final Customized Bounding Box Template

To keep things simple, this template will have one variable, one label, and very basic instructions.

```html
<crowd-form>
  <crowd-bounding-box
    name="annotatedResult"
    labels="['bird']"
    src="{{ task.input.sourceRef | grant_read_access }}"
    header="Draw a box around the bird"
  >
    <full-instructions header="Bounding Box Instructions">
      <p>Draw a bounding box around the bird in each image. If there is more than one bird per image, draw a bounding box around the largest one.</p>
      <p>Bounding boxes should be tight around the bird with no more than a couple of pixels of buffer around the edges.</p>
      <p>If there are no birds in the image, check the <strong>Nothing to label</strong> box.</p>
    </full-instructions>
    <short-instructions>
      <p>Draw a bounding box around the bird in each image. If there is more than one bird per image, draw a bounding box around the largest one.</p>
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

One difference to call out is that the variable is `task.input.sourceRef` instead of `task.input.taskObject`. This is to show that variable naming is flexible and should be semantically meaningful to your developers, but it does have to correspond to a field passed from your pre-processing Lambda.

Example: Your manifest file

Your manifest file should key to the variables you're using in your template. You can do some transformation of your manifest data in your pre-processing Lambda, but if you don't need to, you maintain a lower risk of errors and your Lambda will run faster. Here's a sample manifest file for the template.

```json
{
  "source-ref": "<S3 image URI>"
}
```

Example: Your pre-processing Lambda

As part of the job set-up, you'll need to provide the ARN of an AWS Lambda that can be called to process your manifest entries and pass them to the template engine. The AmazonSageMakerFullAccess policy is restricted to invoking AWS Lambda functions with one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-processing and post-processing Lambdas. If you choose to use names without those strings, you must explicitly provide lambda:InvokeFunction permission to the IAM role used for creating the labeling job.

This applies to both your pre-processing and post-processing Lambdas.

When you're using the console, if you have Lambdas that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.
In this very basic sample, where you have only one variable, it's primarily a pass-through function. Here's a sample pre-processing Lambda in Python 3.7.

```python
import json

def lambda_handler(event, context):
    return {
        "taskInput": {
            "sourceRef": event['dataObject']['source-ref']
        }
    }
```

The API contract for the request and response is provided on the page about pre-processing and post-processing Lambdas (p. 473). The JSON object from your manifest will be provided as a child of the event object. In this demonstration, it is a simple passthrough of one variable, but you might want to have a custom header per object, custom examples, etc. The properties inside the taskInput object will be available as variables to your template.

**Example: Your post-processing Lambda**

As part of the job set-up, you'll need to provide the ARN of an AWS Lambda that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer consolidation and scoring as it comes in, you can apply the scoring and/or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

The annotation data will be in a file designated by the s3Uri string in the payload object. To process the annotations as they come in, even for a simple passthrough function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is in Python 2.7.

```python
import json
import boto3
from urlparse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:]).
    filecont = textFile['Body'].read() 
    annotations = json.loads(filecont);

    for dataset in annotations:
        for annotation in dataset['annotations']:
            new_annotation = json.loads(annotation['annotationData']['content'])
            label = {
                'datasetObjectId': dataset['datasetObjectId'],
                'consolidatedAnnotation': {
                    'content': {
                        'event['labelAttributeName']': {
                            'workerId': annotation['workerId'],
                            'boxesInfo': new_annotation,
                            'imageSource': dataset['dataObject']
                        }
                    }
                }
            }
```
consolidated_labels.append(label)
return consolidated_labels

The post-processing Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through. What you’ll send back will be an object meeting the API contract (p. 473).

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named manifests.

For a bounding box task, the output you'll find in the output manifest will look a bit like the demo below. The example has been cleaned up for printing. The actual output will be a single line per record.

Example : JSON in your output manifest

```
{
"source": "<URL>",
"<Labeling job name>": {
  "workerId": "<URL>",
  "imageSource": "<image URL>",
  "boxesInfo": "{"annotatedResult":{"boundingBoxes": [{"height":878, "label": "bird", "left":208, "top":6, "width":809}], "inputImageProperties":{"height":924, "width":1280}}"},
"<Labeling job name>-metadata": {
  "type": "groundTruth/custom",
  "job_name": "<Labeling job name>",
  "human-annotated": "yes"
}
}
```

This should help you create your own custom template.

### Step 3: Processing with AWS Lambda

In this step, you set which AWS Lambda functions to trigger on each dataset object prior to sending it to workers and which function will be used to process the results once the task is submitted. These functions are required.

You will first need to visit the AWS Lambda console or use AWS Lambda's APIs to create your functions. The AmazonSageMakerFullAccess policy is restricted to invoking AWS Lambda functions with one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-processing and post-processing Lambdas. If you choose to use names without those strings, you must explicitly provide lambda:InvokeFunction permission to the IAM role used for creating the labeling job.

Select your lambdas from the Lambda functions section that comes after the code editor for your custom HTML in the Ground Truth console.

If you need an example, there is an end-to-end demo, including Python code for the Lambdas, in the "Demo Template: Bounding Box (p. 470)" document.

### Pre-labeling task Lambda

Before a labeling task is sent to the worker, your AWS Lambda function will be sent a JSON formatted request to provide details.
Example of a Pre-annotation request

```json
{
    "version": "2018-10-16",
    "labelingJobArn": <labelingJobArn>
    "dataObject": {
        "source-ref": "s3://mybucket/myimage.png"
    }
}
```

The `dataObject` will contain the `source` or `source-ref` property specified in the data object from your manifest.

In return, Ground Truth will require a response formatted like this:

**Example of expected return data**

```json
{
    "taskInput": <json object>,
    "humanAnnotationRequired": <boolean> # Optional
}
```

That `<json object>` may be a bit deceiving. It needs to contain all the data your custom form will need. If you're doing a bounding box task where the instructions stay the same all the time, it may just be the HTTP(S) or S3 resource for your image file. If it's a sentiment analysis task and different objects may have different choices, it would be the object reference as a string and the choices as an array of strings.

**The pre-processing Lambda runs first**

Before any tasks are available to workers, your entire manifest will be processed into an intermediate form, using your Lambda. This means you won't be able to change your Lambda part of the way through a labeling job and see that have an impact on the remaining tasks.

**Post-labeling task Lambda**

When the worker has completed the task, Ground Truth will send the results to your Post-labeling task Lambda. This Lambda is generally used for Annotation Consolidation (p. 450). The request object will come in like this:

**Example of a post-labeling task request**

```json
{
    "version": "2018-10-16",
    "labelingJobArn": <labelingJobArn>,
    "labelCategories": [<string>],
    "labelAttributeName": <string>,
    "roleArn": "string",
    "payload": {
        "s3Uri": <string>
    }
}
```

**Post-labeling task Lambda permissions**

The actual annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple passthrough function, you need to assign the necessary permissions to your Lambda to read files from your S3 bucket.
In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

**Example of an annotation data file**

```
[{
  "datasetObjectId": <string>,
  "dataObject": {
    "s3Uri": <string>,
    "content": <string>
  },
  "annotations": [{
    "workerId": <string>,
    "annotationData": {
      "content": <string>,
      "s3Uri": <string>
    }
  }]
}
```

Essentially, all the fields from your form will be in the `content` object. At this point you can start running data consolidation algorithms on the data, using an AWS database service to store results. Or you can pass some processed/optimized results back to Ground Truth for storage in your consolidated annotation manifests in the S3 bucket you specify for output during the configuration of the labeling job.

In return, Ground Truth will require a response formatted like this:

**Example of expected return data**

```
[
  {
    "datasetObjectId": <string>,
    "consolidatedAnnotation": {
      "content": {
        "<labelAttributeName>": {
          # ... label content
        }
      }
    }
  }
]
```

At this point, all the data you're sending to your S3 bucket, other than the `datasetObjectId` will be in the `content` object.

That will result in an entry in your job's consolidation manifest like this:

**Example of label format in output manifest**

```
{} "source-ref"/"source" : "<s3uri or content>",
"<labelAttributeName>" : { 
  # ... label content from you
},
"<labelAttributeName>-metadata": { # This will be added by Ground Truth
  "job_name": "<labelingJobName>",
  "type": "groundTruth/custom",
}
```


Because of the potentially complex nature of a custom template and the data it collects, Ground Truth does not offer further processing of the data or insights into it.

Next

Custom Workflows via the API (p. 476)

Custom Workflows via the API

When you have created your custom UI template (Step 2) and processing Lambda functions (Step 3), you should place the template in an Amazon S3 bucket with a filename format of: `<FileName>.liquid.html`.

Use the CreateLabelingJob (p. 533) action to configure your task. You'll use the location of a custom template (Step 2: Creating your custom labeling task template (p. 466)) stored in a `<filename>.liquid.html` file on S3 as the value for the UiTemplateS3Uri field in the UiConfig (p. 910) object within the HumanTaskConfig (p. 786) object.

For the AWS Lambda tasks described in Step 3: Processing with AWS Lambda (p. 473), the post-processing task's ARN will be used as the value for the AnnotationConsolidationLambdaArn field, and the pre-processing task will be used as the value for the PreHumanTaskLambdaArn.

HTML Elements Reference

Amazon SageMaker Ground Truth provides customers with the ability to design their own custom task templates in HTML. Documentation of how to implement custom templates is available in Creating Custom Labeling Workflows (p. 465). Below is a list of enhanced HTML elements that make building a custom template easier and provide a familiar UI for workers.

Topics

- crowd-bounding-box (p. 477)
- crowd-image-classifier (p. 480)
- crowd-classifier (p. 481)
- crowd-semantic-segmentation (p. 482)
- crowd-alert (p. 484)
- crowd-badge (p. 485)
- crowd-button (p. 485)
- crowd-card (p. 486)
- crowd-checkbox (p. 487)
- crowd-fab (p. 489)
- crowd-form (p. 489)
- crowd-icon-button (p. 490)
- crowd-input (p. 490)
- crowd-instructions (p. 492)
- crowd-keypoint (p. 493)
- crowd-modal (p. 495)
• crowd-radio-button (p. 496)
• crowd-radio-group (p. 497)
• crowd-slider (p. 498)
• crowd-tab (p. 499)
• crowd-tabs (p. 500)
• crowd-text-area (p. 500)
• crowd-toast (p. 502)
• crowd-toggle-button (p. 502)

crowd-bounding-box

A widget for drawing rectangles on an image and assigning a label to the portion of the image that is enclosed in each rectangle.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a rectangle.

name

The name of this widget. It's used as a key for the widget's input in the form output.

current-value

The URL of the image on which to draw bounding boxes.

initial-value

An array of JSON objects, each of which sets a bounding box when the component is loaded. Each JSON object in the array contains the following properties.

• height – The height of the box in pixels.
• label – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the labels attribute of the <crowd-bounding-box> element.
• left – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
• top – Distance of the top-left corner of the box from the top of the image, measured in pixels.
• width – The width of the box in pixels.

Element Hierarchy

This element has the following parent and child elements.

• Parent element: crowd-form (p. 489)
• Child elements: full-instructions (p. 478), short-instructions (p. 478)

Regions
The following regions are required by this element.

full-instructions
General instructions about how to draw bounding boxes.

short-instructions
Important task-specific instructions that are displayed in a prominent place.

Output
The following output is supported by this element.

boundingBoxes
An array of JSON objects, each of which specifies a bounding box that has been created by the worker. Each JSON object in the array contains the following properties.

• height – The height of the box in pixels.
• label – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the labels attribute of the <crowd-bounding-box> element.
• left – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
• top – Distance of the top-left corner of the box from the top of the image, measured in pixels.
• width – The width of the box in pixels.

inputImageProperties
A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

• height – The height, in pixels, of the image.
• width – The width, in pixels, of the image.

Example: Sample Element Outputs
The following are samples of outputs from common use scenarios for this element.

Single Label, Single Box / Multiple Label, Single Box

```
[
    {
        "annotatedResult": {
            "boundingBoxes": [
                {
                    "height": 401,
                    "label": "Dog",
                    "left": 243,
                    "top": 117,
                    "width": 187
                }
            ]
        }
    ]
```
"inputImageProperties": {
  "height": 533,
  "width": 800
}
}
]

Single Label, Multiple Box

[
  {
    "annotatedResult": {
      "boundingBoxes": [
      {
        "height": 401,
        "label": "Dog",
        "left": 243,
        "top": 117,
        "width": 187
      },
      {
        "height": 283,
        "label": "Dog",
        "left": 684,
        "top": 120,
        "width": 116
      }
    ],
    "inputImageProperties": {
      "height": 533,
      "width": 800
    }
  }
}
]

Multiple Label, Multiple Box

[
  {
    "annotatedResult": {
      "boundingBoxes": [
      {
        "height": 395,
        "label": "Dog",
        "left": 241,
        "top": 125,
        "width": 158
      },
      {
        "height": 298,
        "label": "Cat",
        "left": 699,
        "top": 116,
        "width": 101
      }
    ],
    "inputImageProperties": {
      "height": 533,
      "width": 800
    }
  }
}
You could have many labels available, but only the ones that are used appear in the output.

**See Also**

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

**crowd-image-classifier**

A widget for classifying an image, which can be a JPG, PNG, or GIF, with no size limit.

**Attributes**

The following attributes are required by this element.

- **categories**
  A JSON formatted array of strings, each of which is a category that a worker can assign to the image. You should include "other" as a category, so that the worker can provide an answer. You can specify up to 10 categories.

- **header**
  The text to display above the image. This is typically a question or simple instruction for the worker.

- **name**
  The name of this widget. It is used as a key for the widget's input in the form output.

- **src**
  The URL of the image to be classified.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: full-instructions (p. 480), short-instructions (p. 480)

**Regions**

The following regions are required by this element.

- **full-instructions**
  General instructions about how to do image classification.

- **short-instructions**
  Important task-specific instructions that are displayed in a prominent place.
Output

The output of this element is a string that specifies one of the values defined in the `categories` attribute of the `<crowd-image-classifier>` element.

Example: Sample Element Outputs

The following is a sample of output from this element.

```json
[
  {
    "<name>": {
      "label": "<value>"
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

`crowd-classifier`

A widget for classifying non-image content, such as audio, video, or text.

Attributes

The following attributes are supported by this element.

`categories`

A JSON formatted array of strings, each of which is a category that a worker can assign to the text. You should include "other" as a category, otherwise the worker may not be able to provide an answer.

`header`

The text to display above the image. This is typically a question or simple instruction for the worker.

`name`

The name of this widget. It is used as a key for the widget's input in the form output.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: `crowd-form` (p. 489)
- **Child elements**: `classification-target` (p. 482), `full-instructions` (p. 482), `short-instructions` (p. 482)

Regions

The following regions are supported by this element.
classification-target

The content to be classified by the worker. This can be plain text or HTML. Examples of how the HTML can be used include but are not limited to embedding a video or audio player, embedding a PDF, or performing a comparison of two or more images.

full-instructions

General instructions about how to do text classification.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The output of this element is an object using the specified name value as a property name, and a string from the categories as the property's value.

Example: Sample Element Outputs

The following is a sample of output from this element.

```json
[
  {
    "<name>": {
      "label": "<value>"
    }
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-semantic-segmentation

A widget for segmenting an image and assigning a label to each image segment.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to a segment of the image.

name

The name of this widget. It is used as a key for the widget’s input in the form output.
src
The URL of the image that is to be segmented.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: full-instructions (p. 483), short-instructions (p. 483)

Regions
The following regions are supported by this element.

**full-instructions**
General instructions about how to do image segmentation.

**short-instructions**
Important task-specific instructions that are displayed in a prominent place.

Output
The following output is supported by this element.

**labeledImage**
A JSON Object containing a Base64 encoded PNG of the labels.

**labelMappings**
A JSON Object containing objects with named with the segmentation labels.
- **color** – The hexadecimal value of the label's RGB color in the labeledImage PNG.

**inputImageProperties**
A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.
- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs
The following is a sample of output from this element.

```json
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 533,
        "width": 800
      },
      "labelMappings": {
        "<Label 2>": {
```
crowd-alert

A message that alerts the worker to a current situation.

Attributes

The following attributes are supported by this element.

disable

A Boolean switch that, if present, allows the message to be closed by the worker.

type

A string that specifies the type of message to be displayed. The possible values are "info" (the default), "success", "error", and "warning".

visible

A Boolean switch that, if present, prevents the message from being displayed.

Element Hierarchy

This element has the following parent and child elements.

Parent elements: crowd-form (p. 489)
Child elements: none

See Also

For more information, see the following.

• Amazon SageMaker Ground Truth (p. 445)
• HTML Elements Reference (p. 476)
crowd-badge

An icon that floats over the top right corner of another element to which it is attached.

Attributes

The following attributes are supported by this element.

for

A string that specifies the ID of the element to which the badge is attached.

icon

A string that specifies the icon to be displayed in the badge. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

This attribute overrides the label attribute.

label

The text to display in the badge. Three characters or less is recommended because text that is too large will overflow the badge area. An icon can be displayed instead of text by setting the icon attribute.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 489)
- Child elements: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-button

A styled button that represents some action.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

form-action

A switch that either submits its parent crowd-form (p. 489) element, if set to "submit", or resets its parent <crowd-form> element, if set to "reset".

href

The URL to an online resource. Use this property if you need a link styled as a button.
icon
A string that specifies the icon to be displayed next to the button's text. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon. The icon is positioned to either the left or the right of the text, as specified by the icon-align attribute.

icon-align
The left or right position of the icon relative to the button's text. The default is "left".

icon-url
A URL to a custom image for the icon. A custom image can be used in place of a standard icon that is specified by the icon attribute.

loading
A Boolean switch that, if present, displays the button as being in a loading state. This attribute has precedence over the disabled attribute if both attributes are present.

target
When you use the href attribute to make the button act as a hyperlink to a specific URL, the target attribute optionally targets a frame or window where the linked URL should load.

variant
The general style of the button. Use "primary" for primary buttons, "normal" for secondary buttons, "link" for tertiary buttons, or "icon" to display only the icon without text.

wrap-text
A Boolean switch that, if present, prevents the button's text from wrapping.

Element Hierarchy
This element has the following parent and child elements.

- Parent elements: crowd-form (p. 489)
- Child elements: none

See Also
For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-card
A box with an elevated appearance for displaying information.

Attributes
The following attributes are supported by this element.
alt
The text that displays when the mouse hovers over the box.

heading
The text displayed at the top of the box.

image
A URL to an image to be displayed within the box.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

See Also
For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-checkbox
A UI component that can be checked or unchecked allowing a user to select multiple options from a set.

Attributes
The following attributes are supported by this element.

- **checked**: A Boolean switch that, if present, displays the check box as checked.
- **disabled**: A Boolean switch that, if present, displays the check box as disabled and prevents it from being checked.
- **name**: A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.
- **required**: A Boolean switch that, if present, requires the worker to provide input.
- **value**: A string used as the name for the check box state in the output. Defaults to "on" if not specified.

Element Hierarchy
This element has the following parent and child elements.
**Parent elements**: crowd-form (p. 489)

**Child elements**: none

### Output

Provides a JSON object. The `name` string is the object name and the `value` string is the property name for a Boolean value based on the check box state; true if checked, false if not checked.

### Example: Sample Element Outputs

#### Using the same `name` value for multiple boxes.

```html
<!-- INPUT -->
<div><crowd-checkbox name="myformbit" value="Red"> Red </div>
<div><crowd-checkbox name="myformbit" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="myformbit" value="Green"> Green </div>

//Output with "Red" checked
[
  
  "myformbit": {
    "Green": false,
    "Red": true,
    "Yellow": false
  }
]
```

Note that all three color values are properties of a single object.

#### Using different `name` values for each box.

```html
<!-- INPUT -->
<div><crowd-checkbox name="Stop" value="Red"> Red </div>
<div><crowd-checkbox name="Slow" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="Go" value="Green"> Green </div>

//Output with "Red" checked
[
  
  "Stop": {
    "Green": false
  },
  "Slow": {
    "Yellow": false
  },
  "Stop": {
    "Red": true
  }
]
```

### See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
crowd-fab
A floating button with an image in its center.

**Attributes**
The following attributes are supported by this element.

- **disabled**
  A Boolean switch that, if present, displays the floating button as disabled and prevents clicks.

- **icon**
  A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

- **label**
  A string consisting of a single character that can be used instead of an icon. Emojis or multiple characters may result in the button displaying an ellipsis instead.

- **title**
  A string that will display as a tool tip when the mouse hovers over the button.

**Element Hierarchy**
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

**See Also**
For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-form
The form wrapper for all custom tasks. Sets and implements important actions for the proper submission of your form data.

If a crowd-button (p. 485) of type "submit" is not included inside the <crowd-form> element, it will automatically be appended within the <crowd-form> element.

**Element Hierarchy**
This element has the following parent and child elements.

- **Parent elements**: none
- **Child elements**: Any of the UI Template (p. 476) elements
See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-icon-button

A button with an image placed in the center. When the user touches the button, a ripple effect emanates from the center of the button.

Attributes

The following attributes are supported by this element.

alt

The text that displays when the mouse hovers over the box.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source iron-icons set, which is pre-loaded, or the URL to a custom icon.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 489)
- Child elements: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-input

A box that accepts input data.

Cannot be self-closing

Unlike the input element in the HTML standard, this element cannot be self-closed by putting a slash before the ending bracket, e.g. <crowd-input >. It must be followed with a </crowd-input > to close the element.

Attributes

The following attributes are supported by this element.
allowed-pattern
A regular expression that is used with the auto-validate attribute to ignore non-matching characters as the worker types.

auto-focus
When the value is set to true, the browser places focus inside the input area after loading. This way, the worker can start typing without having to select it first.

auto-validate
A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the error-message and allowed-pattern attributes.

disabled
A Boolean switch that, if present, displays the input area as disabled.

error-message
The text to be displayed below the input field, on the left side, if validation fails.

label
A string that is displayed inside a text field.
This text shrinks and rises up above a text field when the worker starts typing in the field or when the value attribute is set.

list

max-length
A maximum number of characters the input will accept. Input beyond this limit is ignored.

min-length
A minimum length for the input in the field.

name
Sets the name of the input to be used in the DOM and the output of the form.

placeholder
A string value that is used as placeholder text, displayed until the worker starts entering data into the input, It is not used as a default value.

read-only
A Boolean switch that, if present, prevents the field from being changed.

required
A Boolean switch that, if present, requires the worker to provide input.

type
Takes a string to set the HTML5 input-type behavior for the input. Examples include file and date.
value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

**Element Hierarchy**

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

**Output**

Provides a `name` string as the property name, and the text that was entered in the field as its value.

**Example: Sample JSON Output**

The values for multiple elements are output in the same object, with their `name` attribute value as their property name. Elements with no input do not appear in the output. For example, let’s use three inputs:

```html
<crowd-input name="tag1" label="Word/phrase 1"></crowd-input>
<crowd-input name="tag2" label="Word/phrase 2"></crowd-input>
<crowd-input name="tag3" label="Word/phrase 3"></crowd-input>
```

This is the output if only two have input:

```json
[
    {
        "tag1": "blue",
        "tag2": "red"
    }
]
```

This means any code built to parse these results should be able to handle the presence or absence of each input in the answers.

**See Also**

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

**crowd-instructions**

An element that displays instructions on three tabbed pages, **Summary**, **Detailed Instructions**, and **Examples**, when the worker clicks on a link or button.

**Attributes**

The following attributes are supported by this element.

- link-text

  The text to display for opening the instructions. The default is **Click for instructions**.
link-type
A string that specifies the type of trigger for the instructions. The possible values are "link" (default) and "button".

Element Hierarchy
This element has the following parent and child elements.
- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

Regions
The following regions are supported by this element.

**detailed-instructions**
Content that provides specific instructions for a task. This appears on the page of the "Detailed Instructions" tab.

**negative-example**
Content that provides examples of inadequate task completion. This appears on the page of the "Examples" tab. More than one example may be provided within this element.

**positive-example**
Content that provides examples of proper task completion. This appears on the page of the "Examples" tab.

**short-summary**
A brief statement that summarizes the task to be completed. This appears on the page of the "Summary" tab. More than one example may be provided within this element.

See Also
For more information, see the following.
- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-keypoint
Generates a tool to select and annotate key points on an image.

Attributes
The following attributes are supported by this element.

**header**
The text to display above the image. This is typically a question or simple instruction for the worker.

**labels**
An array, in JSON format, of strings to be used as keypoint annotation labels.
name
A string used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

src
The source URI of the image to be annotated.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: full-instructions (p. 494), short-instructions (p. 494)

Regions
The following regions are required by this element.

full-instructions
General instructions about how to annotate the image.

short-instructions
Important task-specific instructions that are displayed in a prominent place.

Output
The following output is supported by this element.

inputImageProperties
A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

keypoints
An array of JSON objects containing the coordinates and label of a keypoint. Each object contains the following properties.

- **label** – The assigned label for the keypoint.
- **x** – The X coordinate, in pixels, of the keypoint on the image.
- **y** – The Y coordinate, in pixels, of the keypoint on the image.

Note
X and Y coordinates are based on 0,0 being the top left corner of the image.

Example: Sample Element Outputs
The following is a sample output from using this element.

[ ]

494
You may have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-modal

A small window that pops up on the display when it is opened.

Attributes

The following attributes are supported by this element.

link-text

The text to display for opening the modal. The default is "Click to open modal".
link-type
A string that specifies the type of trigger for the modal. The possible values are "link" (default) and "button".

size
A string that specifies the size of the modal. The possible values are "small" (default), "large", and "fullscreen".

Element Hierarchy
This element has the following parent and child elements.
- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

See Also
For more information, see the following.
- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-radio-button
A button that can be either checked or unchecked. When radio buttons are inside a radio group, exactly one radio button in the group can be checked at any time.

Attributes
The following attributes are supported by this element.

checked
A Boolean switch that, if present, displays the radio button as checked.

disabled
A Boolean switch that, if present, displays the button as disabled and prevents it from being checked.

name
A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

required
A Boolean switch that, if present, requires the worker to provide input.

value
A property name for the element's boolean value. If not specified, it uses "on" as the default, e.g. 

```json
{ "<name>": { "<value>": <true or false> } }
```

Element Hierarchy
This element has the following parent and child elements.
• **Parent elements**: crowd-radio-group (p. 497)
• **Child elements**: none

**Output**

Outputs an object with the following pattern: `{ "<name>": { "<value>": <true or false> } }`. If you use the buttons outside of a crowd-radio-group (p. 497) element, but with the same name string and different value strings, the name object will contain a Boolean value for each value string. To ensure that only one in a group of buttons is selected, make them children of a crowd-radio-group (p. 497) element and use different name values.

**Example Sample output of this element**

```
[
  
  "btn1": {
    "yes": true
  },
  
  "btn2": {
    "no": false
  }
]
```

**See Also**

For more information, see the following.

• *Amazon SageMaker Ground Truth (p. 445)*
• *HTML Elements Reference (p. 476)*

**crowd-radio-group**

A group of radio buttons. Only one radio button within the group can be selected. Choosing one radio button clears any previously chosen radio button within the same group.

**Attributes**

The following attributes are supported by this element.

**allow-empty-selection**

A Boolean switch that, if present, allows no radio button to be checked.

**disabled**

A Boolean switch that, if present, displays the radio group as disabled and prevents buttons in the group from being checked.

**Element Hierarchy**

This element has the following parent and child elements.

• **Parent elements**: crowd-form (p. 489)
• **Child elements**: crowd-radio-button (p. 496)
Output

Outputs an array of objects representing the crowd-radio-button (p. 496) elements within it.

Example Sample of Element Output

```json
[
    {
        "btn1": {
            "yes": true
        },
        "btn2": {
            "no": false
        }
    }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-slider

A bar with a sliding knob that allows a worker to select a value from a range of values by moving the knob. The slider makes it a great choice for settings that reflect intensity levels, such as volume, brightness, or color saturation.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the slider as disabled.

editable

A Boolean switch that, if present, displays an up/down button that can be chosen to select the value.

Selecting the value via the up/down button is an alternative to selecting the value by moving the knob on the slider. The knob on the slider will move synchronously with the up/down button choices.

max

A number that specifies the maximum value on the slider.

min

A number that specifies the minimum value on the slider.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.
pin
A Boolean switch that, if present, displays the current value above the knob as the knob is moved.

required
A Boolean switch that, if present, requires the worker to provide input.

secondary-progress
When used with a crowd-slider-secondary-color CSS attribute, the progress bar is colored to the point represented by the secondary-progress. For example, if this was representing the progress on a streaming video, the value would represent where the viewer was in the video timeline. The secondary-progress value would represent the point on the timeline to which the video had buffered.

step
A number that specifies the difference between selectable values on the slider.

value
A preset that becomes the default if the worker does not provide input.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none

See Also
For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-tab
A component styled to look like a tab with information below.

Attributes
The following attributes are supported by this element.

header
The text appearing on the tab. This is usually some short descriptive name indicative of the information contained below the tab.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-tabs (p. 500)
• **Child elements**: none

## See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

### crowd-tabs

A container for tabbed information.

### Attributes

This element has no attributes.

### Element Hierarchy

This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: crowd-tab (p. 499)

## See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

### crowd-text-area

A field for text input.

### Attributes

The following attributes are supported by this element.

- **allowed-pattern**

  A regular expression that is used with the `auto-validate` attribute to ignore non-matching characters as the worker types.

- **auto-focus**

  A Boolean switch that, if present, puts the cursor in this element on-load so that users can immediately begin typing without having to click inside the element.

- **auto-validate**

  A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the `error-message` and `allowed-pattern` attributes.
char-counter
A Boolean switch that, if present, puts a small text field beneath the lower-right corner of the element, displaying the number of characters inside the element.

disabled
A Boolean switch that, if present, displays the input area as disabled.

error-message
The text to be displayed below the input field, on the left side, if validation fails.

label
A string that is displayed inside a text field.
This text shrinks and rises up above a text field when the worker starts typing in the field or when the value attribute is set.

max-length
An integer that specifies the maximum number of characters allowed by the element. Characters typed or pasted beyond the maximum are ignored.

max-rows
An integer that specifies the maximum number of rows of text that are allowed within a crowd-text-area. Normally the element expands to accommodate new rows. If this is set, after the number of rows exceeds it, content scrolls upward out of view and a scrollbar control appears.

name
A string used to represent the element's data in the output.

placeholder
A string presented to the user as placeholder text. It disappears after the user puts something in the input area.

read-only
A Boolean switch that, when present, disables editing the contents of the element, but allows content to be highlighted and copied.

rows
An integer that specifies the height of the element in rows of text.

value
A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy
This element has the following parent and child elements.

- **Parent elements**: crowd-form (p. 489)
- **Child elements**: none
Output

This element outputs the name as a property name and the element's text contents as the value. Carriage returns in the text are represented as \n.

Example Sample output for this element

```
[
  {
    "textInput1": "This is the text; the text that\nmakes the crowd go wild."
  }
]
```

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-toast

A subtle notification that temporarily appears on the display. Only one crowd-toast is visible.

Attributes

The following attributes are supported by this element.

duration

A number that specifies the duration, in seconds, that the notification appears on the screen.

text

The text to display in the notification.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 489)
- Child elements: none

See Also

For more information, see the following.

- Amazon SageMaker Ground Truth (p. 445)
- HTML Elements Reference (p. 476)

crowd-toggle-button

A button that acts as an ON/OFF switch, toggling a state.
Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the button switched to the ON position.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents toggling.

invalid

When in an off position, a button using this attribute, will display in an alert color. The standard is red, but may be changed in CSS. When toggled on, the button will display in the same color as other buttons in the on position.

name

A string that is used to identify the answer submitted by the worker. This value matches a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A value used in the output as the property name for the element's Boolean state. Defaults to "on" if not provided.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements: crowd-form (p. 489)
- Child elements: none

Output

This element outputs the name as the name of an object, containing the value as a property name and the element's state as Boolean value for the property. If no value for the element is specified, the property name defaults to "on."

Example Sample output for this element

```
[
  {
    "theToggler": {
      "on": true
    }
  }
]
```

See Also

For more information, see the following.
• Amazon SageMaker Ground Truth (p. 445)
• HTML Elements Reference (p. 476)
Limits and Supported Regions

For Amazon SageMaker service limits, see Amazon SageMaker Limits.

For a list of the AWS Regions supporting Amazon SageMaker, see Amazon SageMaker Regions.
API Reference

This section contains the API Reference documentation.

Topics

- Actions (p. 506)
- Data Types (p. 738)

Actions

The following actions are supported by Amazon SageMaker Service:

- AddTags (p. 510)
- CreateAlgorithm (p. 512)
- CreateCodeRepository (p. 517)
- CreateCompilationJob (p. 519)
- CreateEndpoint (p. 522)
- CreateEndpointConfig (p. 525)
- CreateHyperParameterTuningJob (p. 528)
- CreateLabelingJob (p. 533)
- CreateModel (p. 538)
- CreateModelPackage (p. 542)
- CreateNotebookInstance (p. 546)
- CreateNotebookInstanceLifecycleConfig (p. 551)
- CreatePresignedNotebookInstanceUrl (p. 554)
- CreateTrainingJob (p. 556)
- CreateTransformJob (p. 562)
- CreateWorkteam (p. 567)
- DeleteAlgorithm (p. 570)
- DeleteCodeRepository (p. 571)
- DeleteEndpoint (p. 572)
- DeleteEndpointConfig (p. 574)
- DeleteModel (p. 575)
- DeleteModelPackage (p. 576)
- DeleteNotebookInstance (p. 578)
- DeleteNotebookInstanceLifecycleConfig (p. 580)
- DeleteTags (p. 581)
- DeleteWorkteam (p. 583)
- DescribeAlgorithm (p. 585)
- DescribeCodeRepository (p. 590)
- DescribeCompilationJob (p. 592)
- DescribeEndpoint (p. 596)
• DescribeEndpointConfig (p. 599)
• DescribeHyperParameterTuningJob (p. 601)
• DescribeLabelingJob (p. 607)
• DescribeModel (p. 612)
• DescribeModelPackage (p. 615)
• DescribeNotebookInstance (p. 619)
• DescribeNotebookInstanceLifecycleConfig (p. 624)
• DescribeSubscribedWorkteam (p. 627)
• DescribeTrainingJob (p. 629)
• DescribeTransformJob (p. 636)
• DescribeWorkteam (p. 640)
• GetSearchSuggestions (p. 642)
• ListAlgorithms (p. 644)
• ListCodeRepositories (p. 647)
• ListCompilationJobs (p. 650)
• ListEndpointConfigs (p. 653)
• ListEndpoints (p. 656)
• ListHyperParameterTuningJobs (p. 659)
• ListLabelingJobs (p. 663)
• ListLabelingJobsForWorkteam (p. 667)
• ListModelPackages (p. 670)
• ListModels (p. 673)
• ListNotebookInstanceLifecycleConfigs (p. 676)
• ListNotebookInstances (p. 679)
• ListSubscribedWorkteams (p. 683)
• ListTags (p. 685)
• ListTrainingJobs (p. 687)
• ListTrainingJobsForHyperParameterTuningJob (p. 690)
• ListTransformJobs (p. 693)
• ListWorkteams (p. 696)
• RenderUiTemplate (p. 699)
• Search (p. 701)
• StartNotebookInstance (p. 706)
• StopCompilationJob (p. 708)
• StopHyperParameterTuningJob (p. 710)
• StopLabelingJob (p. 712)
• StopNotebookInstance (p. 714)
• StopTrainingJob (p. 716)
• StopTransformJob (p. 718)
• UpdateCodeRepository (p. 720)
• UpdateEndpoint (p. 722)
• UpdateEndpointWeightsAndCapacities (p. 724)
• UpdateNotebookInstance (p. 726)
• UpdateNotebookInstanceLifecycleConfig (p. 730)
• UpdateWorkteam (p. 732)
The following actions are supported by Amazon SageMaker Runtime:

- **InvokeEndpoint** (p. 735)

**Amazon SageMaker Service**

The following actions are supported by Amazon SageMaker Service:

- **AddTags** (p. 510)
- **CreateAlgorithm** (p. 512)
- **CreateCodeRepository** (p. 517)
- **CreateCompilationJob** (p. 519)
- **CreateEndpoint** (p. 522)
- **CreateEndpointConfig** (p. 525)
- **CreateHyperParameterTuningJob** (p. 528)
- **CreateLabelingJob** (p. 533)
- **CreateModel** (p. 538)
- **CreateModelPackage** (p. 542)
- **CreateNotebookInstance** (p. 546)
- **CreateNotebookInstanceLifecycleConfig** (p. 551)
- **CreatePresignedNotebookInstanceUrl** (p. 554)
- **CreateTrainingJob** (p. 556)
- **CreateTransformJob** (p. 562)
- **CreateWorkteam** (p. 567)
- **DeleteAlgorithm** (p. 570)
- **DeleteCodeRepository** (p. 571)
- **DeleteEndpoint** (p. 572)
- **DeleteEndpointConfig** (p. 574)
- **DeleteModel** (p. 575)
- **DeleteModelPackage** (p. 576)
- **DeleteNotebookInstance** (p. 578)
- **DeleteNotebookInstanceLifecycleConfig** (p. 580)
- **DeleteTags** (p. 581)
- **DeleteWorkteam** (p. 583)
- **DescribeAlgorithm** (p. 585)
- **DescribeCodeRepository** (p. 590)
- **DescribeCompilationJob** (p. 592)
- **DescribeEndpoint** (p. 596)
- **DescribeEndpointConfig** (p. 599)
- **DescribeHyperParameterTuningJob** (p. 601)
- **DescribeLabelingJob** (p. 607)
- **DescribeModel** (p. 612)
- **DescribeModelPackage** (p. 615)
- **DescribeNotebookInstance** (p. 619)
- **DescribeNotebookInstanceLifecycleConfig** (p. 624)
- **DescribeSubscribedWorkteam** (p. 627)
- DescribeTrainingJob (p. 629)
- DescribeTransformJob (p. 636)
- DescribeWorkteam (p. 640)
- GetSearchSuggestions (p. 642)
- ListAlgorithms (p. 644)
- ListCodeRepositories (p. 647)
- ListCompilationJobs (p. 650)
- ListEndpointConfigs (p. 653)
- ListEndpoints (p. 656)
- ListHyperParameterTuningJobs (p. 659)
- ListLabelingJobs (p. 663)
- ListLabelingJobsForWorkteam (p. 667)
- ListModelPackages (p. 670)
- ListModels (p. 673)
- ListNotebookInstanceLifecycleConfigs (p. 676)
- ListNotebookInstances (p. 679)
- ListSubscribedWorkteams (p. 683)
- ListTags (p. 685)
- ListTrainingJobs (p. 687)
- ListTrainingJobsForHyperParameterTuningJob (p. 690)
- ListTransformJobs (p. 693)
- ListWorkteams (p. 696)
- RenderUiTemplate (p. 699)
- Search (p. 701)
- StartNotebookInstance (p. 706)
- StopCompilationJob (p. 708)
- StopHyperParameterTuningJob (p. 710)
- StopLabelingJob (p. 712)
- StopNotebookInstance (p. 714)
- StopTrainingJob (p. 716)
- StopTransformJob (p. 718)
- UpdateCodeRepository (p. 720)
- UpdateEndpoint (p. 722)
- UpdateEndpointWeightsAndCapacities (p. 724)
- UpdateNotebookInstance (p. 726)
- UpdateNotebookInstanceLifecycleConfig (p. 730)
- UpdateWorkteam (p. 732)
AddTags
Service: Amazon SageMaker Service

Adds or overwrites one or more tags for the specified Amazon SageMaker resource. You can add tags to notebook instances, training jobs, hyperparameter tuning jobs, models, endpoint configurations, and endpoints.

Each tag consists of a key and an optional value. Tag keys must be unique per resource. For more information about tags, see AWS Tagging Strategies.

Note
Tags that you add to a hyperparameter tuning job by calling this API are also added to any training jobs that the hyperparameter tuning job launches after you call this API, but not to training jobs that the hyperparameter tuning job launched before you called this API. To make sure that the tags associated with a hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches, add the tags when you first create the tuning job by specifying them in the Tags parameter of CreateHyperParameterTuningJob (p. 528)

Request Syntax

```json
{
   "ResourceArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

ResourceArn (p. 510)

The Amazon Resource Name (ARN) of the resource that you want to tag.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Tags (p. 510)

An array of Tag objects. Each tag is a key-value pair. Only the key parameter is required. If you don't specify a value, Amazon SageMaker sets the value to an empty string.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: Yes

Response Syntax

```json
{
}
```
"Tags": [
    {
        "Key": "string",
        "Value": "string"
    }
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Tags (p. 510)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateAlgorithm
Service: Amazon SageMaker Service
Create a machine learning algorithm that you can use in Amazon SageMaker and list in the AWS Marketplace.

Request Syntax

```
{
  "AlgorithmDescription": "string",
  "AlgorithmName": "string",
  "CertifyForMarketplace": boolean,
  "InferenceSpecification": {
    "Containers": [
      {
        "ContainerHostname": "string",
        "Image": "string",
        "ImageDigest": "string",
        "ModelDataUrl": "string",
        "ProductId": "string"
      }
    ],
    "SupportedContentTypes": [ "string" ],
    "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
    "SupportedResponseMIMETypes": [ "string" ],
    "SupportedTransformInstanceTypes": [ "string" ]
  },
  "TrainingSpecification": {
    "MetricDefinitions": [
      {
        "Name": "string",
        "Regex": "string"
      }
    ],
    "SupportedHyperParameters": [
      {
        "DefaultValue": "string",
        "Description": "string",
        "IsRequired": boolean,
        "IsTunable": boolean,
        "Name": "string",
        "Range": {
          "CategoricalParameterRangeSpecification": {
            "Values": [ "string" ]
          },
          "ContinuousParameterRangeSpecification": {
            "MaxValue": "string",
            "MinValue": "string"
          },
          "IntegerParameterRangeSpecification": {
            "MaxValue": "string",
            "MinValue": "string"
          }
        },
        "Type": "string"
      }
    ],
    "SupportedTrainingInstanceTypes": [ "string" ],
    "SupportedTuningJobObjectiveMetrics": [
      {
        "MetricName": "string",
        "Type": "string"
      }
    ]
  }
}
```
"SupportsDistributedTraining": boolean,
"TrainingChannels": [
  {
    "Description": "string",
    "IsRequired": boolean,
    "Name": "string",
    "SupportedCompressionTypes": [ "string" ],
    "SupportedContentTypes": [ "string" ],
    "SupportedInputModes": [ "string" ]
  }
],
"TrainingImage": "string",
"TrainingImageDigest": "string"
},
"ValidationSpecification": {
  "ValidationProfiles": [
    {
      "ProfileName": "string",
      "TrainingJobDefinition": {
        "HyperParameters": {
          "string": "string"
        },
        "InputDataConfig": [ {
          "ChannelName": "string",
          "CompressionType": "string",
          "ContentType": "string",
          "DataSource": { "S3DataSource": { "AttributeNames": [ "string" ],
            "S3DataDistributionType": "string",
            "S3DataType": "string",
            "S3Uri": "string"
          }
        },
        "InputMode": "string",
        "RecordWrapperType": "string",
        "ShuffleConfig": { "Seed": number
      }
    }]
  ],
  "OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number
  },
  "TrainingInputMode": "string"
},
"TransformJobDefinition": {
  "BatchStrategy": "string",
  "Environment": { "string": "string"
  },
  "MaxConcurrentTransforms": number,
  "MaxPayloadInMB": number,
  "TransformInput": { "CompressionType": "string",
  }
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AlgorithmDescription (p. 512)

A description of the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

AlgorithmName (p. 512)

The name of the algorithm.

Type: String


Pattern: ^[a-zA-Z0-9-]+(-*[a-zA-Z0-9-]+)*$

Required: Yes

CertifyForMarketplace (p. 512)

Whether to certify the algorithm so that it can be listed in AWS Marketplace.

Type: Boolean

Required: No
InferenceSpecification (p. 512)

Specifies details about inference jobs that the algorithm runs, including the following:

- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the algorithm supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the algorithm supports for inference.

Type: InferenceSpecification (p. 806) object

Required: No

TrainingSpecification (p. 512)

Specifies details about training jobs run by this algorithm, including the following:

- The Amazon ECR path of the container and the version digest of the algorithm.
- The hyperparameters that the algorithm supports.
- The instance types that the algorithm supports for training.
- Whether the algorithm supports distributed training.
- The metrics that the algorithm emits to Amazon CloudWatch.
- Which metrics that the algorithm emits can be used as the objective metric for hyperparameter tuning jobs.
- The input channels that the algorithm supports for training data. For example, an algorithm might support train, validation, and test channels.

Type: TrainingSpecification (p. 895) object

Required: Yes

ValidationSpecification (p. 512)

Specifies configurations for one or more training jobs and that Amazon SageMaker runs to test the algorithm's training code and, optionally, one or more batch transform jobs that Amazon SageMaker runs to test the algorithm's inference code.

Type: AlgorithmValidationSpecification (p. 751) object

Required: No

Response Syntax

```
{
  "AlgorithmArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmArn (p. 515)

The Amazon Resource Name (ARN) of the new algorithm.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateCodeRepository
Service: Amazon SageMaker Service

Creates a Git repository as a resource in your Amazon SageMaker account. You can associate the repository with notebook instances so that you can use Git source control for the notebooks you create. The Git repository is a resource in your Amazon SageMaker account, so it can be associated with more than one notebook instance, and it persists independently from the lifecycle of any notebook instances it is associated with.

The repository can be hosted either in AWS CodeCommit or in any other Git repository.

Request Syntax

```
{
    "CodeRepositoryName": "string",
    "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 517)**

The name of the Git repository. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String


Pattern: `^[a-zA-Z0-9-]*[a-zA-Z0-9-](([a-zA-Z0-9-]+[a-zA-Z0-9-]*)*`*

Required: Yes

**GitConfig (p. 517)**

Specifies details about the repository, including the URL where the repository is located, the default branch, and credentials to use to access the repository.

Type: GitConfig (p. 784) object

Required: Yes

Response Syntax

```
{
    "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CodeRepositoryArn (p. 517)**

The Amazon Resource Name (ARN) of the new repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateCompilationJob
Service: Amazon SageMaker Service

Starts a model compilation job. After the model has been compiled, Amazon SageMaker saves the resulting model artifacts to an Amazon Simple Storage Service (Amazon S3) bucket that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts with AWS IoT Greengrass. In that case, deploy them as an ML resource.

In the request body, you provide the following:

- A name for the compilation job
- Information about the input model artifacts
- The output location for the compiled model and the device (target) that the model runs on
- The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker assumes to perform the model compilation job

You can also provide a Tag to track the model compilation job's resource use and costs. The response body contains the CompilationJobArn for the compiled job.

To stop a model compilation job, use StopCompilationJob (p. 708). To get information about a particular model compilation job, use DescribeCompilationJob (p. 592). To get information about multiple model compilation jobs, use ListCompilationJobs (p. 650).

Request Syntax

```
{
  "CompilationJobName": "string",
  "InputConfig": {
    "DataInputConfig": "string",
    "Framework": "string",
    "S3Uri": "string"
  },
  "OutputConfig": {
    "S3OutputLocation": "string",
    "TargetDevice": "string"
  },
  "RoleArn": "string",
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number
  }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CompilationJobName (p. 519)

A name for the model compilation job. The name must be unique within the AWS Region and within your AWS account.

Type: String
Pattern: ^[a-zA-Z0-9\-\d\(\_*[a-zA-Z0-9\-\d\)]*\$**$
Required: Yes

**InputConfig (p. 519)**

Provides information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: InputConfig (p. 808) object

Required: Yes

**OutputConfig (p. 519)**

Provides information about the output location for the compiled model and the target device the model runs on.

Type: OutputConfig (p. 850) object

Required: Yes

**RoleArn (p. 519)**

The Amazon Resource Name (ARN) of an IAM role that enables Amazon SageMaker to perform tasks on your behalf.

During model compilation, Amazon SageMaker needs your permission to:
- Read input data from an S3 bucket
- Write model artifacts to an S3 bucket
- Write logs to Amazon CloudWatch Logs
- Publish metrics to Amazon CloudWatch

You grant permissions for all of these tasks to an IAM role. To pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission. For more information, see Amazon SageMaker Roles.

Type: String

Pattern: ^arn:aws[a-zA-Z\-\*]:iam::\d{12}:role/?[a-zA-Z\d-]+,\.*\@\-\_/]+\$**$

Required: Yes

**StoppingCondition (p. 519)**

The duration allowed for model compilation.

Type: StoppingCondition (p. 879) object

Required: Yes

**Response Syntax**

```
{
    "CompilationJobArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CompilationJobArn (p. 520)**

If the action is successful, the service sends back an HTTP 200 response. Amazon SageMaker returns the following data in JSON format:

- **CompilationJobArn**: The Amazon Resource Name (ARN) of the compiled job.
  - Type: String
  - Length Constraints: Maximum length of 256.
  - Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceInUse**

- Resource being accessed is in use.
  - HTTP Status Code: 400

**ResourceLimitExceeded**

- You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
  - HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateEndpoint

Service: Amazon SageMaker Service

Creates an endpoint using the endpoint configuration specified in the request. Amazon SageMaker uses the endpoint to provision resources and deploy models. You create the endpoint configuration with the CreateEndpointConfig API.

**Note**

Use this API only for hosting models using Amazon SageMaker hosting services.

The endpoint name must be unique within an AWS Region in your AWS account.

When it receives the request, Amazon SageMaker creates the endpoint, launches the resources (ML compute instances), and deploys the model(s) on them.

When Amazon SageMaker receives the request, it sets the endpoint status to `Creating`. After it creates the endpoint, it sets the status to `InService`. Amazon SageMaker can then process incoming requests for inferences. To check the status of an endpoint, use the DescribeEndpoint API.

For an example, see Exercise 1: Using the K-Means Algorithm Provided by Amazon SageMaker.

If any of the models hosted at this endpoint get model data from an Amazon S3 location, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provided. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS in an AWS Region in the AWS Identity and Access Management User Guide.

**Request Syntax**

```
{
  "EndpointConfigName": "string",
  "EndpointName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 522)**

The name of an endpoint configuration. For more information, see CreateEndpointConfig.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**EndpointName (p. 522)**

The name of the endpoint. The name must be unique within an AWS Region in your AWS account.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[\-a-zA-Z0-9]*[-a-zA-Z0-9]*
Required: Yes

Tags (p. 522)
An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.
Type: Array of Tag (p. 883) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

Response Syntax

```json
{
  "EndpointArn": "string"
}
```

Response Elements
If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

EndpointArn (p. 523)
The Amazon Resource Name (ARN) of the endpoint.
Type: String

Errors
For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceLimitExceeded
You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateEndpointConfig
Service: Amazon SageMaker Service

Creates an endpoint configuration that Amazon SageMaker hosting services uses to deploy models. In the configuration, you identify one or more models, created using the CreateModel API, to deploy and the resources that you want Amazon SageMaker to provision. Then you call the CreateEndpoint API.

**Note**
Use this API only if you want to use Amazon SageMaker hosting services to deploy models into production.

In the request, you define one or more ProductionVariants, each of which identifies a model. Each ProductionVariant parameter also describes the resources that you want Amazon SageMaker to provision. This includes the number and type of ML compute instances to deploy.

If you are hosting multiple models, you also assign a VariantWeight to specify how much traffic you want to allocate to each model. For example, suppose that you want to host two models, A and B, and you assign traffic weight 2 for model A and 1 for model B. Amazon SageMaker distributes two-thirds of the traffic to Model A, and one-third to model B.

**Request Syntax**

```json
{
  "EndpointConfigName": "string",
  "KmsKeyId": "string",
  "ProductionVariants": [
    {
      "AcceleratorType": "string",
      "InitialInstanceCount": number,
      "InitialVariantWeight": number,
      "InstanceType": "string",
      "ModelName": "string",
      "VariantName": "string"
    }
  ],
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 525)**

The name of the endpoint configuration. You specify this name in a CreateEndpoint request.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9]{0,63}(-*[a-zA-Z0-9]{0,63})* 

Required: Yes
KmsKeyId (p. 525)

The Amazon Resource Name (ARN) of a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance that hosts the endpoint.

Type: String

Length Constraints: Maximum length of 2048.

Required: No

ProductionVariants (p. 525)

An array of ProductionVariant objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 856) objects

Array Members: Minimum number of 1 item.

Required: Yes

Tags (p. 525)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

Response Syntax

```
{
  "EndpointConfigArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointConfigArn (p. 526)

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateHyperParameterTuningJob
Service: Amazon SageMaker Service

Starts a hyperparameter tuning job. A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm you choose and values for hyperparameters within ranges that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by an objective metric that you choose.

Request Syntax

```json
{
   "HyperParameterTuningJobConfig": {
      "HyperParameterTuningJobObjective": {
         "MetricName": "string",
         "Type": "string"
      },
      "ParameterRanges": {
         "CategoricalParameterRanges": [
            {
               "Name": "string",
               "Values": [ "string" ]
            }
         ],
         "ContinuousParameterRanges": [
            {
               "MaxValue": "string",
               "MinValue": "string",
               "Name": "string"
            }
         ],
         "IntegerParameterRanges": [
            {
               "MaxValue": "string",
               "MinValue": "string",
               "Name": "string"
            }
         ]
      },
      "ResourceLimits": {
         "MaxNumberOfTrainingJobs": number,
         "MaxParallelTrainingJobs": number
      },
      "Strategy": "string",
      "TrainingJobEarlyStoppingType": "string"
   },
   "HyperParameterTuningJobName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TrainingJobDefinition": {
      "AlgorithmSpecification": {
         "AlgorithmName": "string",
         "MetricDefinitions": [
            {
               "Name": "string",
               "Regex": "string"
            }
         ],
         "TrainingImage": "string",
         "TrainingInputMode": "string"
      }
   }
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.
HyperParameterTuningJobConfig (p. 528)

The `HyperParameterTuningJobConfig` object that describes the tuning job, including the search strategy, the objective metric used to evaluate training jobs, ranges of parameters to search, and resource limits for the tuning job. For more information, see Automatic Model Tuning (p. 290)

Type: `HyperParameterTuningJobConfig` object

Required: Yes

HyperParameterTuningJobName (p. 528)

The name of the tuning job. This name is the prefix for the names of all training jobs that this tuning job launches. The name must be unique within the same AWS account and AWS Region. The name must have {} to {} characters. Valid characters are a-z, A-Z, 0-9, and : + = @ _ % - (hyphen). The name is not case sensitive.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Tags (p. 528)

An array of key-value pairs. You can use tags to categorize your AWS resources in different ways, for example, by purpose, owner, or environment. For more information, see AWS Tagging Strategies.

Tags that you specify for the tuning job are also added to all training jobs that the tuning job launches.

Type: Array of `Tag` objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingJobDefinition (p. 528)

The `HyperParameterTrainingJobDefinition` object that describes the training jobs that this tuning job launches, including static hyperparameters, input data configuration, output data configuration, resource configuration, and stopping condition.

Type: `HyperParameterTrainingJobDefinition` object

Required: Yes

WarmStartConfig (p. 528)

Specifies the configuration for starting the hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric. If you specify IDENTICAL_DATA_AND_ALGORITHM as the `WarmStartType` value for the warm start configuration, the training job that performs the best in the new tuning job is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

Note

All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.
Type: HyperParameterTuningJobWarmStartConfig (p. 804) object

Required: No

Response Syntax

```json
{
   "HyperParameterTuningJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**HyperParameterTuningJobArn (p. 531)**

The Amazon Resource Name (ARN) of the tuning job. Amazon SageMaker assigns an ARN to a hyperparameter tuning job when you create it.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
CreateLabelingJob
Service: Amazon SageMaker Service

Creates a job that uses workers to label the data objects in your input dataset. You can use the labeled data to train machine learning models.

You can select your workforce from one of three providers:

- A private workforce that you create. It can include employees, contractors, and outside experts. Use a private workforce when you want the data to stay within your organization or when a specific set of skills is required.
- One or more vendors that you select from the AWS Marketplace. Vendors provide expertise in specific areas.
- The Amazon Mechanical Turk workforce. This is the largest workforce, but it should only be used for public data or data that has been stripped of any personally identifiable information.

You can also use automated data labeling to reduce the number of data objects that need to be labeled by a human. Automated data labeling uses active learning to determine if a data object can be labeled by machine or if it needs to be sent to a human worker. For more information, see Using Automated Data Labeling.

The data objects to be labeled are contained in an Amazon S3 bucket. You create a manifest file that describes the location of each object. For more information, see Using Input and Output Data.

The output can be used as the manifest file for another labeling job or as training data for your machine learning models.

Request Syntax

```json
{
    "HumanTaskConfig": {
        "AnnotationConsolidationConfig": {
            "AnnotationConsolidationLambdaArn": "string"
        },
        "MaxConcurrentTaskCount": number,
        "NumberOfHumanWorkersPerDataObject": number,
        "PreHumanTaskLambdaArn": "string",
        "PublicWorkforceTaskPrice": {
            "AmountInUsd": {
                "Cents": number,
                "Dollars": number,
                "TenthFractionsOfACent": number
            }
        },
        "TaskAvailabilityLifetimeInSeconds": number,
        "TaskDescription": "string",
        "TaskKeywords": [ "string" ],
        "TaskTimeLimitInSeconds": number,
        "TaskTitle": "string",
        "UiConfig": {
            "UiTemplateS3Uri": "string"
        },
        "WorkteamArn": "string"
    },
    "InputConfig": {
        "DataAttributes": {
            "ContentClassifiers": [ "string" ]
        },
        "DataSource": {
            "S3DataSource": {
```
"ManifestS3Uri": "string"
}
}
"LabelAttributeName": "string",
"LabelCategoryConfigS3Uri": "string",
"LabelingJobAlgorithmsConfig": {
  "InitialActiveLearningModelArn": "string",
  "LabelingJobAlgorithmSpecificationArn": "string",
  "LabelingJobResourceConfig": {
    "VolumeKmsKeyId": "string"
  }
},
"LabelingJobName": "string",
"OutputConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"RoleArn": "string",
"StoppingConditions": {
  "MaxHumanLabeledObjectCount": number,
  "MaxPercentageOfInputDatasetLabeled": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
]

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**HumanTaskConfig (p. 533)**

Configures the information required for human workers to complete a labeling task.

Type: HumanTaskConfig (p. 786) object

Required: Yes

**InputConfig (p. 533)**

Input data for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.

Type: LabelingJobInputConfig (p. 821) object

Required: Yes

**LabelAttributeName (p. 533)**

The attribute name to use for the label in the output manifest file. This is the key for the key/value pair formed with the label that a worker assigns to the object. The name can't end with "-metadata". If you are running a semantic segmentation labeling job, the attribute name must end with "-ref". If you are running any other kind of labeling job, the attribute name must not end with "-ref".

Type: String

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: Yes

LabelCategoryConfigS3Uri (p. 533)

The S3 URL of the file that defines the categories used to label the data objects.

The file is a JSON structure in the following format:

```json
{
    "document-version": "2018-11-28",
    "labels": [
    {
        "label": "label 1"
    },
    {
        "label": "label 2"
    },
    ...
    {
        "label": "label n"
    }
    ]
}
```

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)(.*)$

Required: No

LabelingJobAlgorithmsConfig (p. 533)

Configures the information required to perform automated data labeling.

Type: LabelingJobAlgorithmsConfig (p. 815) object

Required: No

LabelingJobName (p. 533)

The name of the labeling job. This name is used to identify the job in a list of labeling jobs.

Type: String

Pattern: ^[a-zA-Z0-9]-*[a-zA-Z0-9]*$
Required: Yes

OutputConfig (p. 533)
The location of the output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.
Type: LabelingJobOutputConfig (p. 823) object
Required: Yes

RoleArn (p. 533)
The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete data labeling.
Type: String
Pattern: ^arn:aws[\w\-]*:iam::\d{12}:role/?[\w-]+\@\-\_/]+$
Required: Yes

StoppingConditions (p. 533)
A set of conditions for stopping the labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.
Type: LabelingJobStoppingConditions (p. 826) object
Required: No

Tags (p. 533)
An array of key/value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.
Type: Array of Tag (p. 883) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

Response Syntax

```
{
  "LabelingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobArn (p. 536)
The Amazon Resource Name (ARN) of the labeling job. You use this ARN to identify the labeling job.
Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateModel
Service: Amazon SageMaker Service

Creates a model in Amazon SageMaker. In the request, you name the model and describe a primary container. For the primary container, you specify the docker image containing inference code, artifacts (from prior training), and custom environment map that the inference code uses when you deploy the model for predictions.

Use this API to create a model if you want to use Amazon SageMaker hosting services or run a batch transform job.

To host your model, you create an endpoint configuration with the CreateEndpointConfig API, and then create an endpoint with the CreateEndpoint API. Amazon SageMaker then deploys all of the containers that you defined for the model in the hosting environment.

To run a batch transform using your model, you start a job with the CreateTransformJob API. Amazon SageMaker uses your model and your dataset to get inferences which are then saved to a specified S3 location.

In the CreateModel request, you must define a container with the PrimaryContainer parameter.

In the request, you also provide an IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute hosting instances or for batch transform jobs. In addition, you also use the IAM role to manage permissions the inference code needs. For example, if the inference code access any other AWS resources, you grant necessary permissions via this role.

Request Syntax

```json
{
  "Containers": [
    {
      "ContainerHostname": "string",
      "Environment": {
        "string" : "string"
      },
      "Image": "string",
      "ModelDataUrl": "string",
      "ModelPackageName": "string"
    }
  ],
  "EnableNetworkIsolation": boolean,
  "ExecutionRoleArn": "string",
  "ModelName": "string",
  "PrimaryContainer": {
    "ContainerHostname": "string",
    "Environment": {
      "string" : "string"
    },
    "Image": "string",
    "ModelDataUrl": "string",
    "ModelPackageName": "string"
  },
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "Subnets": [ "string" ]
  }
}```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

Containers (p. 538)

Specifies the containers in the inference pipeline.

Type: Array of ContainerDefinition (p. 765) objects

Array Members: Maximum number of 5 items.

Required: No

EnableNetworkIsolation (p. 538)

Isolates the model container. No inbound or outbound network calls can be made to or from the model container.

Note
The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

ExecutionRoleArn (p. 538)

The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access model artifacts and docker image for deployment on ML compute instances or for batch transform jobs. Deploying on ML compute instances is part of model hosting. For more information, see Amazon SageMaker Roles.

Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-zA-z-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@-/]+$*

Required: Yes

ModelName (p. 538)

The name of the new model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9]{1,63}([-.*][a-zA-Z0-9]){0,63}$

Required: Yes
PrimaryContainer (p. 538)

The location of the primary docker image containing inference code, associated artifacts, and custom environment map that the inference code uses when the model is deployed for predictions.

Type: ContainerDefinition (p. 765) object

Required: No

Tags (p. 538)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

VpcConfig (p. 538)

A VpcConfig (p. 913) object that specifies the VPC that you want your model to connect to. Control access to and from your model container by configuring the VPC. VpcConfig is used in hosting services and in batch transform. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Data in Batch Transform Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

Response Syntax

```json
{
    "ModelArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelArn (p. 540)

The ARN of the model created in Amazon SageMaker.

Type: String


Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateModelPackage
Service: Amazon SageMaker Service

Creates a model package that you can use to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

To create a model package by specifying a Docker container that contains your inference code and the Amazon S3 location of your model artifacts, provide values for `InferenceSpecification`. To create a model from an algorithm resource that you created or subscribed to in AWS Marketplace, provide a value for `SourceAlgorithmSpecification`.

Request Syntax

```
{
  "CertifyForMarketplace": boolean,
  "InferenceSpecification": {
    "Containers": [ {
      "ContainerHostname": "string",
      "Image": "string",
      "ImageDigest": "string",
      "ModelDataUrl": "string",
      "ProductId": "string"
    } ],
    "SupportedContentTypes": [ "string" ],
    "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
    "SupportedResponseMIMETypes": [ "string" ],
    "SupportedTransformInstanceTypes": [ "string" ]
  },
  "ModelPackageDescription": "string",
  "ModelPackageName": "string",
  "SourceAlgorithmSpecification": {
    "SourceAlgorithms": [ { "AlgorithmName": "string",
                         "ModelDataUrl": "string" } ]
  },
  "ValidationSpecification": {
    "ValidationProfiles": [ { "ProfileName": "string",
                             "TransformJobDefinition": { "BatchStrategy": "string",
                                                      "Environment": { "string": "string" }
                             },
                             "MaxConcurrentTransforms": number,
                             "MaxPayloadInMB": number,
                             "TransformInput": { "CompressionType": "string",
                                                  "ContentType": "string",
                                                  "DataSource": { "S3DataSource": { "S3DataType": "string",
                                                     "S3Uri": "string" } } },
                             "SplitType": "string"
                        } ]
  }
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CertifyForMarketplace (p. 542)

Whether to certify the model package for listing on AWS Marketplace.

Type: Boolean

Required: No

InferenceSpecification (p. 542)

Specifies details about inference jobs that can be run with models based on this model package, including the following:

- The Amazon ECR paths of containers that contain the inference code and model artifacts.
- The instance types that the model package supports for transform jobs and real-time endpoints used for inference.
- The input and output content formats that the model package supports for inference.

Type: InferenceSpecification (p. 806) object

Required: No

ModelPackageDescription (p. 542)

A description of the model package.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: \[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

ModelPackageName (p. 542)

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and – (hyphen).
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
Required: Yes

**SourceAlgorithmSpecification (p. 542)**
Details about the algorithm that was used to create the model package.
Type: `SourceAlgorithmSpecification (p. 878)` object
Required: No

**ValidationSpecification (p. 542)**
Specifies configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.
Type: `ModelPackageValidationSpecification (p. 841)` object
Required: No

**Response Syntax**

```json
{
   "ModelPackageArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**ModelPackageArn (p. 544)**
The Amazon Resource Name (ARN) of the new model package.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: `arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:\d{12}:model-package/.*`

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
CreateNotebookInstance
Service: Amazon SageMaker Service

Creates an Amazon SageMaker notebook instance. A notebook instance is a machine learning (ML) compute instance running on a Jupyter notebook.

In a CreateNotebookInstance request, specify the type of ML compute instance that you want to run. Amazon SageMaker launches the instance, installs common libraries that you can use to explore datasets for model training, and attaches an ML storage volume to the notebook instance.

Amazon SageMaker also provides a set of example notebooks. Each notebook demonstrates how to use Amazon SageMaker with a specific algorithm or with a machine learning framework.

After receiving the request, Amazon SageMaker does the following:

1. Creates a network interface in the Amazon SageMaker VPC.
2. (Option) If you specified SubnetId, Amazon SageMaker creates a network interface in your own VPC, which is inferred from the subnet ID that you provide in the input. When creating this network interface, Amazon SageMaker attaches the security group that you specified in the request to the network interface that it creates in your VPC.
3. Launches an EC2 instance of the type specified in the request in the Amazon SageMaker VPC. If you specified SubnetId of your VPC, Amazon SageMaker specifies both network interfaces when launching this instance. This enables inbound traffic from your own VPC to the notebook instance, assuming that the security groups allow it.

After creating the notebook instance, Amazon SageMaker returns its Amazon Resource Name (ARN).

After Amazon SageMaker creates the notebook instance, you can connect to the Jupyter server and work in Jupyter notebooks. For example, you can write code to explore a dataset that you can use for model training, train a model, host models by creating Amazon SageMaker endpoints, and validate hosted models.

For more information, see How It Works.

Request Syntax

```json
{
    "AcceleratorTypes": [ "string" ],
    "AdditionalCodeRepositories": [ "string" ],
    "DefaultCodeRepository": "string",
    "DirectInternetAccess": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LifecycleConfigName": "string",
    "NotebookInstanceName": "string",
    "RoleArn": "string",
    "SecurityGroupIds": [ "string" ],
    "SubnetId": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "VolumeSizeInGB": number
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**AcceleratorTypes (p. 546)**

A list of Elastic Inference (EI) instance types to associate with this notebook instance. Currently, only one instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge

Required: No

**AdditionalCodeRepositories (p. 546)**

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://(\[^/\]+)/(.*)$|^\[a-zA-Z0-9\](-*[a-zA-Z0-9]*)*

Required: No

**DefaultCodeRepository (p. 546)**

A Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String


Pattern: ^https://(\[^/\]+)/(.*)$|^\[a-zA-Z0-9\](-*[a-zA-Z0-9]*)*

Required: No

**DirectInternetAccess (p. 546)**

Sets whether Amazon SageMaker provides internet access to the notebook instance. If you set this to Disabled this notebook instance will be able to access resources only in your VPC, and will not be able to connect to Amazon SageMaker training and endpoint services unless you configure a NAT Gateway in your VPC.

For more information, see Notebook Instances Are Internet-Enabled by Default. You can set the value of this parameter to Disabled only if you set a value for the SubnetId parameter.

Type: String
Valid Values: Enabled | Disabled

Required: No

**InstanceType (p. 546)**

The type of ML compute instance to launch for the notebook instance.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge |
ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge |
| ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge |
| ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge |
| ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge |
| ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
| ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge |
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge |

Required: Yes

**KmsKeyId (p. 546)**

If you provide an AWS KMS key ID, Amazon SageMaker uses it to encrypt data at rest on the ML storage volume that is attached to your notebook instance. The KMS key you provide must be enabled. For information, see [Enabling and Disabling Keys](#) in the *AWS Key Management Service Developer Guide*.

Type: String

Length Constraints: Maximum length of 2048.

Required: No

**LifecycleConfigName (p. 546)**

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see [Step 2.1: (Optional) Customize a Notebook Instance](#).

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9)(-*[a-zA-Z0-9])*%

Required: No

**NotebookInstanceName (p. 546)**

The name of the new notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9)(-*[a-zA-Z0-9])*%

Required: Yes

**RoleArn (p. 546)**

When you send any requests to AWS resources from the notebook instance, Amazon SageMaker assumes this role to perform tasks on your behalf. You must grant this role necessary permissions so Amazon SageMaker can perform these tasks. The policy must allow the Amazon SageMaker service
principal (sagemaker.amazonaws.com) permissions to assume this role. For more information, see Amazon SageMaker Roles.

Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String
Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?\[a-zA-Z_0-9+=,.@\-_/]+$
Required: Yes

SecurityGroupIds (p. 546)
The VPC security group IDs, in the form sg-xxxxxxxx. The security groups must be for the same VPC as specified in the subnet.
Type: Array of strings
Array Members: Maximum number of 5 items.
Length Constraints: Maximum length of 32.
Required: No

SubnetId (p. 546)
The ID of the subnet in a VPC to which you would like to have a connectivity from your ML compute instance.
Type: String
Length Constraints: Maximum length of 32.
Required: No

Tags (p. 546)
A list of tags to associate with the notebook instance. You can add tags later by using the CreateTags API.
Type: Array of Tag (p. 883) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

VolumeSizeInGB (p. 546)
The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB.
Type: Integer
Required: No

Response Syntax
{

"NotebookInstanceArn": "string"
}

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NotebookInstanceArn (p. 549)**

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 915)](#).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateNotebookInstanceLifecycleConfig
Service: Amazon SageMaker Service

Creates a lifecycle configuration that you can associate with a notebook instance. A lifecycle configuration is a collection of shell scripts that run when you create or start a notebook instance.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the $PATH environment variable that is available to both scripts is `/sbin:bin:/usr/sbin:/usr/bin`.


Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```json
{
  "NotebookInstanceLifecycleConfigName": "string",
  "OnCreate": [
    {
      "Content": "string"
    }
  ],
  "OnStart": [
    {
      "Content": "string"
    }
  ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 551)**

- The name of the lifecycle configuration.
  - Type: String
  - Length Constraints: Maximum length of 63.
  - Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`
  - Required: Yes

**OnCreate (p. 551)**

- A shell script that runs only once, when you create a notebook instance. The shell script must be a base64-encoded string.
  - Type: Array of NotebookInstanceLifecycleHook (p. 845) objects
Array Members: Maximum number of 1 item.

Required: No

OnStart (p. 551)

A shell script that runs every time you start a notebook instance, including when you create the notebook instance. The shell script must be a base64-encoded string.

Type: Array of NotebookInstanceLifecycleHook (p. 845) objects

Array Members: Maximum number of 1 item.

Required: No

Response Syntax

```json
{
   "NotebookInstanceLifecycleConfigArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NotebookInstanceLifecycleConfigArn (p. 552)

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreatePresignedNotebookInstanceUrl
Service: Amazon SageMaker Service

Returns a URL that you can use to connect to the Jupyter server from a notebook instance. In the Amazon SageMaker console, when you choose Open next to a notebook instance, Amazon SageMaker opens a new tab showing the Jupyter server home page from the notebook instance. The console uses this API to get the URL and show the page.

You can restrict access to this API and to the URL that it returns to a list of IP addresses that you specify. To restrict access, attach an IAM policy that denies access to this API unless the call comes from an IP address in the specified list to every AWS Identity and Access Management user, group, or role used to access the notebook instance. Use the NotIpAddress condition operator and the aws:SourceIP condition context key to specify the list of IP addresses that you want to have access to the notebook instance. For more information, see Limit Access to a Notebook Instance by IP Address.

Request Syntax

```json
{
    "NotebookInstanceName": "string",
    "SessionExpirationDurationInSeconds": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 554)**

The name of the notebook instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]$

Required: Yes

**SessionExpirationDurationInSeconds (p. 554)**

The duration of the session, in seconds. The default is 12 hours.

Type: Integer


Required: No

Response Syntax

```json
{
    "AuthorizedUrl": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AuthorizedUrl (p. 554)**

A JSON object that contains the URL string.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
CreateTrainingJob
Service: Amazon SageMaker Service

Starts a model training job. After training completes, Amazon SageMaker saves the resulting model artifacts to an Amazon S3 location that you specify.

If you choose to host your model using Amazon SageMaker hosting services, you can use the resulting model artifacts as part of the model. You can also use the artifacts in a deep learning service other than Amazon SageMaker, provided that you know how to use them for inferences.

In the request body, you provide the following:

- **AlgorithmSpecification** - Identifies the training algorithm to use.
- **HyperParameters** - Specify these algorithm-specific parameters to influence the quality of the final model. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.
- **InputDataConfig** - Describes the training dataset and the Amazon S3 location where it is stored.
- **OutputDataConfig** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results of model training.
- **ResourceConfig** - Identifies the resources, ML compute instances, and ML storage volumes to deploy for model training. In distributed training, you specify more than one instance.
- **RoleARN** - The Amazon Resource Number (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during model training. You must grant this role the necessary permissions so that Amazon SageMaker can successfully complete model training.
- **StoppingCondition** - Sets a duration for training. Use this parameter to cap model training costs.

For more information about Amazon SageMaker, see How It Works.

Request Syntax

```json
{
  "AlgorithmSpecification": {
    "AlgorithmName": "string",
    "MetricDefinitions": [
      {
        "Name": "string",
        "Regex": "string"
      }
    ],
    "TrainingImage": "string",
    "TrainingInputMode": "string"
  },
  "EnableInterContainerTrafficEncryption": boolean,
  "EnableNetworkIsolation": boolean,
  "HyperParameters": {
    "string": "string"
  },
  "InputDataConfig": [{
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataTier": "string"
      }
    }
  }
}```
"S3Uri": "string"
}
}),
"InputMode": "string",
"RecordWrapperType": "string",
"ShuffleConfig": {
    "Seed": number
}
}
},
"OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
},
"ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
},
"RoleArn": "string",
"StoppingCondition": {
    "MaxRuntimeInSeconds": number
},
"Tags": [
    {
        "Key": "string",
        "Value": "string"
    }
],
"TrainingJobName": "string",
"VpcConfig": {
    "SecurityGroupIds": [ "string" ],
    "Subnets": [ "string" ]
}
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 556)

The registry path of the Docker image that contains the training algorithm and algorithm-specific metadata, including the input mode. For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about providing your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Type: AlgorithmSpecification (p. 744) object

Required: Yes

EnableInterContainerTrafficEncryption (p. 556)

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean
enableNetworkIsolation (p. 556)

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

**Note**
The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

HyperParameters (p. 556)

Algorithm-specific parameters that influence the quality of the model. You set hyperparameters before you start the learning process. For a list of hyperparameters for each training algorithm provided by Amazon SageMaker, see Algorithms.

You can specify a maximum of 100 hyperparameters. Each hyperparameter is a key-value pair. Each key and value is limited to 256 characters, as specified by the Length Constraint.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

InputDataConfig (p. 556)

An array of Channel objects. Each channel is a named input source. InputDataConfig describes the input data and its location.

Algorithms can accept input data from one or more channels. For example, an algorithm might have two channels of input data, training_data and validation_data. The configuration for each channel provides the S3 location where the input data is stored. It also provides information about the stored data: the MIME type, compression method, and whether the data is wrapped in RecordIO format.

Depending on the input mode that the algorithm supports, Amazon SageMaker either copies input data files from an S3 bucket to a local directory in the Docker container, or makes it available as input streams.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

OutputDataConfig (p. 556)

Specifies the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object

Required: Yes
ResourceConfig (p. 556)

The resources, including the ML compute instances and ML storage volumes, to use for model training.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use ML storage volumes for scratch space. If you want Amazon SageMaker to use the ML storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 866) object

Required: Yes

RoleArn (p. 556)

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker can assume to perform tasks on your behalf.

During model training, Amazon SageMaker needs your permission to read input data from an S3 bucket, download a Docker image that contains training code, write model artifacts to an S3 bucket, write logs to Amazon CloudWatch Logs, and publish metrics to Amazon CloudWatch. You grant permissions for all of these tasks to an IAM role. For more information, see Amazon SageMaker Roles.

Note
To be able to pass this role to Amazon SageMaker, the caller of this API must have the iam:PassRole permission.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@\-_/]+$

Required: Yes

StoppingCondition (p. 556)

Sets a duration for training. Use this parameter to cap model training costs. To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job. This intermediate data is a valid model artifact. You can use it to create a model using the CreateModel API.

Type: StoppingCondition (p. 879) object

Required: Yes

Tags (p. 556)

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No
TrainingJobName (p. 556)

The name of the training job. The name must be unique within an AWS Region in an AWS account.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

VpcConfig (p. 556)

A VpcConfig (p. 913) object that specifies the VPC that you want your training job to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object

Required: No

Response Syntax

```json
{
   "TrainingJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TrainingJobArn (p. 560)

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**CreateTransformJob**

Service: Amazon SageMaker Service

Starts a transform job. A transform job uses a trained model to get inferences on a dataset and saves these results to an Amazon S3 location that you specify.

To perform batch transformations, you create a transform job and use the data that you have readily available.

In the request body, you provide the following:

- **TransformJobName** - Identifies the transform job. The name must be unique within an AWS Region in an AWS account.
- **ModelName** - Identifies the model to use. **ModelName** must be the name of an existing Amazon SageMaker model in the same AWS Region and AWS account. For information on creating a model, see [CreateModel](p. 538).
- **TransformInput** - Describes the dataset to be transformed and the Amazon S3 location where it is stored.
- **TransformOutput** - Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.
- **TransformResources** - Identifies the ML compute instances for the transform job.

For more information about how batch transformation works Amazon SageMaker, see [How It Works](#).

**Request Syntax**

```json
{
  "BatchStrategy": "string",
  "Environment": {
    "string": "string"
  },
  "MaxConcurrentTransforms": number,
  "MaxPayloadInMB": number,
  "ModelName": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ],
  "TransformInput": {
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "SplitType": "string"
  },
  "TransformJobName": "string",
  "TransformOutput": {
    "Accept": "string",
    "AssembleWith": "string",
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "TransformResources": {
    "InstanceCount": number,
    "InstanceType": "string"
  }
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

BatchStrategy (p. 562)

Determines the number of records to include in a mini-batch. If you want to include only one record in a mini-batch, specify SingleRecord. If you want mini-batches to contain a maximum of the number of records specified in the MaxPayloadInMB parameter, specify MultiRecord.

If you set SplitType to Line and BatchStrategy to MultiRecord, a batch transform automatically splits your input data into the specified payload size. There's no need to split the dataset into smaller files or to use larger payload sizes unless the records in your dataset are very large.

Type: String

Valid Values: MultiRecord | SingleRecord

Required: No

Environment (p. 562)

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 10240.

Required: No

MaxConcurrentTransforms (p. 562)

The maximum number of parallel requests that can be sent to an algorithm container on an instance. This is good for algorithms that implement multiple workers on larger instances. The default value is 1. To allow Amazon SageMaker to determine the appropriate number for MaxConcurrentTransforms, do not set the value in the API.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

MaxPayloadInMB (p. 562)

The maximum payload size allowed, in MB. A payload is the data portion of a record (without metadata). The value in MaxPayloadInMB must be greater or equal to the size of a single record. You can approximate the size of a record by dividing the size of your dataset by the number of
records. Then multiply this value by the number of records you want in a mini-batch. We recommend to enter a slightly larger value than this to ensure the records fit within the maximum payload size. The default value is 6 MB.

For cases where the payload might be arbitrarily large and is transmitted using HTTP chunked encoding, set the value to 0. This feature only works in supported algorithms. Currently, Amazon SageMaker built-in algorithms do not support this feature.

Type: Integer
Valid Range: Minimum value of 0.
Required: No

**ModelName (p. 562)**

The name of the model that you want to use for the transform job. `ModelName` must be the name of an existing Amazon SageMaker model within an AWS Region in an AWS account.

Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

**Tags (p. 562)**

(Optional) An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**TransformInput (p. 562)**

Describes the input source and the way the transform job consumes it.

Type: TransformInput (p. 898) object
Required: Yes

**TransformJobName (p. 562)**

The name of the transform job. The name must be unique within an AWS Region in an AWS account.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

**TransformOutput (p. 562)**

Describes the results of the transform job.

Type: TransformOutput (p. 904) object
Required: Yes
TransformResources (p. 562)

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: TransformResources (p. 906) object

Required: Yes

Response Syntax

```json
{
    "TransformJobArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

TransformJobArn (p. 565)

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*`

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceInUse

Resource being accessed is in use.

HTTP Status Code: 400

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
CreateWorkteam
Service: Amazon SageMaker Service

Creates a new work team for labeling your data. A work team is defined by one or more Amazon Cognito user pools. You must first create the user pools before you can create a work team.

You cannot create more than 25 work teams in an account and region.

Request Syntax

```json
{
    "Description": "string",
    "MemberDefinitions": [
        {
            "CognitoMemberDefinition": {
                "ClientId": "string",
                "UserGroup": "string",
                "UserPool": "string"
            }
        }
    ],
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

Description (p. 567)

A description of the work team.

Type: String


Required: Yes

MemberDefinitions (p. 567)

A list of MemberDefinition objects that contains objects that identify the Amazon Cognito user pool that makes up the work team. For more information, see Amazon Cognito User Pools.

All of the CognitoMemberDefinition objects that make up the member definition must have the same ClientId and UserPool values.

Type: Array of MemberDefinition (p. 830) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: Yes
**Tags (p. 567)**

Type: Array of [Tag (p. 883)] objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

**WorkteamName (p. 567)**

The name of the work team. Use this name to identify the work team.

Type: String


Pattern: ^[a-zA-Z0-9\-]*[^a-zA-Z0-9\-]+$

Required: Yes

**Response Syntax**

```json
{
  "WorkteamArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**WorkteamArn (p. 568)**

The Amazon Resource Name (ARN) of the work team. You can use this ARN to identify the work team.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[\-]+sagemaker:[\-]+[0-9]{12}:workteam/.*

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 915)].

**ResourceInUse**

Resource being accessed is in use.

HTTP Status Code: 400

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteAlgorithm
Service: Amazon SageMaker Service
Removes the specified algorithm from your account.

Request Syntax

```json
{
    "AlgorithmName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AlgorithmName (p. 570)

The name of the algorithm to delete.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteCodeRepository
Service: Amazon SageMaker Service

Deletes the specified Git repository from your account.

Request Syntax

```json
{
    "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CodeRepositoryName (p. 571)

The name of the Git repository to delete.

Type: String


Pattern: ^[a-zA-Z0-9\-\[\]{}()*\+\.,;:\[\]@=\^`\-\|~]+\*$

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteEndpoint
Service: Amazon SageMaker Service

Deletes an endpoint. Amazon SageMaker frees up all of the resources that were deployed when the endpoint was created.

Amazon SageMaker retires any custom KMS key grants associated with the endpoint, meaning you don't need to use the RevokeGrant API call.

Request Syntax

```json
{
  "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointName (p. 572)**

The name of the endpoint that you want to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteEndpointConfig
Service: Amazon SageMaker Service

Deletes an endpoint configuration. The `DeleteEndpointConfig` API deletes only the specified configuration. It does not delete endpoints created using the configuration.

**Request Syntax**

```
{
  "EndpointConfigName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 574)**

The name of the endpoint configuration that you want to delete.

- Type: String
- Length Constraints: Maximum length of 63.
- Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`
- Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteModel
Service: Amazon SageMaker Service

Deletes a model. The DeleteModel API deletes only the model entry that was created in Amazon SageMaker when you called the CreateModel API. It does not delete model artifacts, inference code, or the IAM role that you specified when creating the model.

Request Syntax

```json
{
   "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

ModelName (p. 575)

The name of the model to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-20-9](-*[a-zA-20-9])*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteModelPackage
Service: Amazon SageMaker Service

Deletes a model package.

A model package is used to create Amazon SageMaker models or list on AWS Marketplace. Buyers can subscribe to model packages listed on AWS Marketplace to create models in Amazon SageMaker.

Request Syntax

```
{
  "ModelPackageName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**ModelPackageName (p. 576)**

The name of the model package. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*$`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteNotebookInstance
Service: Amazon SageMaker Service

Deletes an Amazon SageMaker notebook instance. Before you can delete a notebook instance, you must call the StopNotebookInstance API.

Important
When you delete a notebook instance, you lose all of your data. Amazon SageMaker removes the ML compute instance, and deletes the ML storage volume and the network interface associated with the notebook instance.

Request Syntax

```
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 578)**

The name of the Amazon SageMaker notebook instance to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9][-][a-zA-Z0-9]*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
DeleteNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Deletes a notebook instance lifecycle configuration.

Request Syntax

```json
{
   "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 580)**

The name of the lifecycle configuration to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**DeleteTags**

Service: Amazon SageMaker Service

Deletes the specified tags from an Amazon SageMaker resource.

To list a resource's tags, use the `ListTags` API.

**Note**

When you call this API to delete tags from a hyperparameter tuning job, the deleted tags are not removed from training jobs that the hyperparameter tuning job launched before you called this API.

**Request Syntax**

```json
{
   "ResourceArn": "string",
   "TagKeys": [ "string" ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**ResourceArn (p. 581)**

The Amazon Resource Name (ARN) of the resource whose tags you want to delete.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

**TagKeys (p. 581)**

An array or one or more tag keys to delete.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 50 items.


Pattern: `^[\p{L}\p{Z}\p{N}\_\-\_/\=\+\@]*$`

Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DeleteWorkteam
Service: Amazon SageMaker Service

Deletes an existing work team. This operation can't be undone.

Request Syntax

```
{
  "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**WorkteamName (p. 583)**

  The name of the work team to delete.

  Type: String


  Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]*$  

  Required: Yes

Response Syntax

```
{
  "Success": boolean
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Success (p. 583)**

  Returns true if the work team was successfully deleted; otherwise, returns false.

  Type: Boolean

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceLimitExceeded**

  You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeAlgorithm
Service: Amazon SageMaker Service

Returns a description of the specified algorithm that is in your account.

Request Syntax

```
{
  "AlgorithmName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**AlgorithmName (p. 585)**

The name of the algorithm to describe.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\//)?([a-zA-Z0-9\-](\[a-zA-Z0-9\-\]){0,62})(?<!-)$

Required: Yes

Response Syntax

```
{
  "AlgorithmArn": "string",
  "AlgorithmDescription": "string",
  "AlgorithmName": "string",
  "AlgorithmStatus": "string",
  "AlgorithmStatusDetails": {
    "ImageScanStatuses": [
      {
        "FailureReason": "string",
        "Name": "string",
        "Status": "string"
      }
    ],
    "ValidationStatuses": [
      {
        "FailureReason": "string",
        "Name": "string",
        "Status": "string"
      }
    ]
  },
  "CertifyForMarketplace": boolean,
  "CreationTime": number,
  "InferenceSpecification": {
    "Containers": [
      {
        "ContainerHostname": "string",
      }
    ]
  }
}
```
"Image": "string",
"ImageDigest": "string",
"ModelDataUrl": "string",
"ProductId": "string"
}
],
"SupportedContentTypes": [ "string" ],
"SupportedRealtimeInferenceInstanceTypes": [ "string" ],
"SupportedResponseMIMETypes": [ "string" ],
"SupportedTransformInstanceTypes": [ "string" ]
},
"ProductId": "string",
"TrainingSpecification": {
"MetricDefinitions": [ {
    "Name": "string",
    "Regex": "string"
  }]
},
"SupportedHyperParameters": [ {
    "DefaultValue": "string",
    "Description": "string",
    "IsRequired": boolean,
    "IsTunable": boolean,
    "Name": "string",
    "Range": { 
      "CategoricalParameterRangeSpecification": {
        "Values": [ "string" ]
      },
      "ContinuousParameterRangeSpecification": { 
        "MaxValue": "string",
        "MinValue": "string"
      },
      "IntegerParameterRangeSpecification": { 
        "MaxValue": "string",
        "MinValue": "string"
      }
    },
    "Type": "string"
  }
],
"SupportedTrainingInstanceTypes": [ "string" ],
"SupportedTuningJobObjectiveMetrics": [ {
    "MetricName": "string",
    "Type": "string"
  }]
},
"SupportsDistributedTraining": boolean,
"TrainingChannels": [ {
    "Description": "string",
    "IsRequired": boolean,
    "Name": "string",
    "SupportedCompressionTypes": [ "string" ],
    "SupportedContentTypes": [ "string" ],
    "SupportedInputModes": [ "string" ]
  }
],
"TrainingImage": "string",
"TrainingImageDigest": "string"
},
"ValidationSpecification": {
"ValidationProfiles": [ 
]
"ProfileName": "string",
"TrainingJobDefinition": {
  "HyperParameters": {
    "string": "string"
  },
  "InputDataConfig": [
    {
      "ChannelName": "string",
      "CompressionType": "string",
      "ContentType": "string",
      "DataSource": {
        "S3DataSource": {
          "AttributeNames": [ "string" ],
          "S3DataDistributionType": "string",
          "S3DataType": "string",
          "S3Uri": "string"
        }
      },
      "InputMode": "string",
      "RecordWrapperType": "string",
      "ShuffleConfig": {
        "Seed": number
      }
    }
  ],
  "OutputDataConfig": {
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "ResourceConfig": {
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": number
  },
  "TrainingInputMode": "string"
},
"TransformJobDefinition": {
  "BatchStrategy": "string",
  "Environment": {
    "string": "string"
  },
  "MaxConcurrentTransforms": number,
  "MaxPayloadInMB": number,
  "TransformInput": {
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "S3DataType": "string",
        "S3Uri": "string"
      }
    }
  },
  "SplitType": "string"
},
"TransformOutput": {
  "Accept": "string",
  "AssembleWith": "string",
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"TransformResources": {
  "InstanceCount": number,
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

AlgorithmArn (p. 585)

The Amazon Resource Name (ARN) of the algorithm.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:algorithm/.*

AlgorithmDescription (p. 585)

A brief summary about the algorithm.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

AlgorithmName (p. 585)

The name of the algorithm being described.

Type: String


Pattern: ^[a-zA-Z0-9-]*(-*[a-zA-Z0-9-])*$

AlgorithmStatus (p. 585)

The current status of the algorithm.

Type: String

Valid Values: Pending | InProgress | Completed | Failed | Deleting

AlgorithmStatusDetails (p. 585)

Details about the current status of the algorithm.

Type: AlgorithmStatusDetails (p. 746) object

CertifyForMarketplace (p. 585)

Whether the algorithm is certified to be listed in AWS Marketplace.

Type: Boolean
CreationTime (p. 585)

A timestamp specifying when the algorithm was created.

Type: Timestamp

InferenceSpecification (p. 585)

Details about inference jobs that the algorithm runs.

Type: InferenceSpecification (p. 806) object

ProductId (p. 585)

The product identifier of the algorithm.

Type: String

TrainingSpecification (p. 585)

Details about training jobs run by this algorithm.

Type: TrainingSpecification (p. 895) object

ValidationSpecification (p. 585)

Details about configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Type: AlgorithmValidationSpecification (p. 751) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeCodeRepository
Service: Amazon SageMaker Service

Gets details about the specified Git repository.

Request Syntax

```
{
   "CodeRepositoryName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 590)**

The name of the Git repository to describe.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

Response Syntax

```
{
   "CodeRepositoryArn": "string",
   "CodeRepositoryName": "string",
   "CreationTime": number,
   "GitConfig": {
      "Branch": "string",
      "RepositoryUrl": "string",
      "SecretArn": "string"
   },
   "LastModifiedTime": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CodeRepositoryArn (p. 590)**

The Amazon Resource Name (ARN) of the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*
CodeRepositoryName (p. 590)
The name of the Git repository.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
CreationTime (p. 590)
The date and time that the repository was created.
Type: Timestamp
GitConfig (p. 590)
Configuration details about the repository, including the URL where the repository is located, the
default branch, and the Amazon Resource Name (ARN) of the AWS Secrets Manager secret that
contains the credentials used to access the repository.
Type: GitConfig (p. 784) object
LastModifiedTime (p. 590)
The date and time that the repository was last changed.
Type: Timestamp

Errors
For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeCompilationJob
Service: Amazon SageMaker Service

Returns information about a model compilation job.

To create a model compilation job, use CreateCompilationJob (p. 519). To get information about multiple model compilation jobs, use ListCompilationJobs (p. 650).

Request Syntax

```json
{
    "CompilationJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CompilationJobName (p. 592)

The name of the model compilation job that you want information about.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
Required: Yes

Response Syntax

```json
{
    "CompilationEndTime": number,
    "CompilationJobArn": "string",
    "CompilationJobName": "string",
    "CompilationJobStatus": "string",
    "CompilationStartTime": number,
    "CreationTime": number,
    "FailureReason": "string",
    "InputConfig": {
        "DataInputConfig": "string",
        "Framework": "string",
        "S3Uri": "string"
    },
    "LastModifiedTime": number,
    "ModelArtifacts": {
        "S3ModelArtifacts": "string"
    },
    "OutputConfig": {
        "S3OutputLocation": "string",
        "TargetDevice": "string"
    },
    "RoleArn": "string",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": number
    }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CompilationEndTime (p. 592)**

The time when the model compilation job on a compilation job instance ended. For a successful or stopped job, this is when the job's model artifacts have finished uploading. For a failed job, this is when Amazon SageMaker detected that the job failed.

Type: Timestamp

**CompilationJobArn (p. 592)**

The Amazon Resource Name (ARN) of an IAM role that Amazon SageMaker assumes to perform the model compilation job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*`

**CompilationJobName (p. 592)**

The name of the model compilation job.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`$

**CompilationJobStatus (p. 592)**

The status of the model compilation job.

Type: String

Valid Values: IN_PROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED

**CompilationStartTime (p. 592)**

The time when the model compilation job started the CompilationJob instances.

You are billed for the time between this timestamp and the timestamp in the `DescribeCompilationJob:CompilationEndTime (p. 593)` field. In Amazon CloudWatch Logs, the start time might be later than this time. That's because it takes time to download the compilation job, which depends on the size of the compilation job container.

Type: Timestamp

**CreationTime (p. 592)**

The time that the model compilation job was created.

Type: Timestamp

**FailureReason (p. 592)**

If a model compilation job failed, the reason it failed.
Type: String
Length Constraints: Maximum length of 1024.

**InputConfig (p. 592)**

Information about the location in Amazon S3 of the input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

Type: InputConfig (p. 808) object

**LastModifiedTime (p. 592)**

The time that the status of the model compilation job was last modified.

Type: Timestamp

**ModelArtifacts (p. 592)**

Information about the location in Amazon S3 that has been configured for storing the model artifacts used in the compilation job.

Type: ModelArtifacts (p. 833) object

**OutputConfig (p. 592)**

Information about the output location for the compiled model and the target device that the model runs on.

Type: OutputConfig (p. 850) object

**RoleArn (p. 592)**

The Amazon Resource Name (ARN) of the model compilation job.

Type: String

Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9+=,.@-_\/]*$`

**StoppingCondition (p. 592)**

The duration allowed for model compilation.

Type: StoppingCondition (p. 879) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeEndpoint
Service: Amazon SageMaker Service

Returns the description of an endpoint.

Request Syntax

```json
{
    "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointName (p. 596)**

The name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Syntax

```json
{
    "CreationTime": number,
    "EndpointArn": "string",
    "EndpointConfigName": "string",
    "EndpointName": "string",
    "EndpointStatus": "string",
    "FailureReason": "string",
    "LastModifiedTime": number,
    "ProductionVariants": [
        {
            "CurrentInstanceCount": number,
            "CurrentWeight": number,
            "DeployedImages": [
                {
                    "ResolutionTime": number,
                    "ResolvedImage": "string",
                    "SpecifiedImage": "string"
                }
            ],
            "DesiredInstanceCount": number,
            "DesiredWeight": number,
            "VariantName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**CreationTime (p. 596)**
A timestamp that shows when the endpoint was created.
Type: Timestamp

**EndpointArn (p. 596)**
The Amazon Resource Name (ARN) of the endpoint.
Type: String

**EndpointConfigName (p. 596)**
The name of the endpoint configuration associated with this endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

**EndpointName (p. 596)**
Name of the endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

**EndpointStatus (p. 596)**
The status of the endpoint.
- **OutOfService**: Endpoint is not available to take incoming requests.
- **Creating**: CreateEndpoint (p. 522) is executing.
- **Updating**: UpdateEndpoint (p. 722) or UpdateEndpointWeightsAndCapacities (p. 724) is executing.
- **SystemUpdating**: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
- **RollingBack**: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an InService status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an UpdateEndpointWeightsAndCapacities (p. 724) call or when the UpdateEndpointWeightsAndCapacities (p. 724) operation is called explicitly.
- **InService**: Endpoint is available to process incoming requests.
- **Deleting**: DeleteEndpoint (p. 572) is executing.
- **Failed**: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 598) for information about the failure. DeleteEndpoint (p. 572) is the only operation that can be performed on a failed endpoint.

Type: String
Valid Values: OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed
FailureReason (p. 596)

If the status of the endpoint is Failed, the reason why it failed.

Type: String

Length Constraints: Maximum length of 1024.

LastModifiedTime (p. 596)

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

ProductionVariants (p. 596)

An array of ProductionVariantSummary (p. 858) objects, one for each model hosted behind this endpoint.

Type: Array of ProductionVariantSummary (p. 858) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeEndpointConfig
Service: Amazon SageMaker Service

Returns the description of an endpoint configuration created using the CreateEndpointConfig API.

**Request Syntax**

```
{
  "EndpointConfigName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 599)**

The name of the endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* Required: Yes

**Response Syntax**

```
{
  "CreationTime": number,
  "EndpointConfigArn": "string",
  "EndpointConfigName": "string",
  "KmsKeyId": "string",
  "ProductionVariants": [
    {
      "AcceleratorType": "string",
      "InitialInstanceCount": number,
      "InitialVariantWeight": number,
      "InstanceType": "string",
      "ModelName": "string",
      "VariantName": "string"
    }
  ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CreationTime (p. 599)**

A timestamp that shows when the endpoint configuration was created.
Type: Timestamp

**EndpointConfigArn (p. 599)**

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


**EndpointConfigName (p. 599)**

Name of the Amazon SageMaker endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

**KmsKeyId (p. 599)**

AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

**ProductionVariants (p. 599)**

An array of *ProductionVariant* objects, one for each model that you want to host at this endpoint.

Type: Array of *ProductionVariant* (p. 856) objects

Array Members: Minimum number of 1 item.

**Errors**

For information about the errors that are common to all actions, see *Common Errors (p. 915).*

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeHyperParameterTuningJob
Service: Amazon SageMaker Service

Gets a description of a hyperparameter tuning job.

Request Syntax

```json
{
  "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 601)

The name of the tuning job to describe.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

Response Syntax

```json
{
  "BestTrainingJob": {
    "CreationTime": number,
    "FailureReason": "string",
    "FinalHyperParameterTuningJobObjectiveMetric": {
      "MetricName": "string",
      "Type": "string",
      "Value": number
    },
    "ObjectiveStatus": "string",
    "TrainingEndTime": number,
    "TrainingJobArn": "string",
    "TrainingJobName": "string",
    "TrainingJobStatus": "string",
    "TrainingStartTime": number,
    "TunedHyperParameters": {
      "string": "string"
    },
    "TuningJobName": "string"
  },
  "CreationTime": number,
  "FailureReason": "string",
  "HyperParameterTuningEndTime": number,
  "HyperParameterTuningJobArn": "string",
  "HyperParameterTuningJobConfig": {
    "HyperParameterTuningJobObjective": {
      "MetricName": "string",
      "Type": "string"
    },
    "ParameterRanges": {
```
"CategoricalParameterRanges": [
    {
        "Name": "string",
        "Values": [ "string" ]
    }
],
"ContinuousParameterRanges": [
    {
        "MaxValue": "string",
        "MinValue": "string",
        "Name": "string"
    }
],
"IntegerParameterRanges": [
    {
        "MaxValue": "string",
        "MinValue": "string",
        "Name": "string"
    }
}
],
"ResourceLimits": {
    "MaxNumberOfTrainingJobs": number,
    "MaxParallelTrainingJobs": number
},
"Strategy": "string",
"TrainingJobEarlyStoppingType": "string"
],
"HyperParameterTuningJobName": "string",
"HyperParameterTuningJobStatus": "string",
"LastModifiedTime": number,
"ObjectiveStatusCounters": {
    "Failed": number,
    "Pending": number,
    "Succeeded": number
},
"OverallBestTrainingJob": {
    "CreationTime": number,
    "FailureReason": "string",
    "FinalHyperParameterTuningJobObjectiveMetric": {
        "MetricName": "string",
        "Type": "string",
        "Value": number
    },
    "ObjectiveStatus": "string",
    "TrainingEndTime": number,
    "TrainingsArn": "string",
    "TrainingJobName": "string",
    "TrainingJobStatus": "string",
    "TrainingStartTime": number,
    "TunedHyperParameters": {
        "string": "string"
    },
    "TuningJobName": "string"
},
"TrainingJobDefinition": {
    "AlgorithmSpecification": {
        "AlgorithmName": "string",
        "MetricDefinitions": [
            {
                "Name": "string",
                "Regex": "string"
            }
        ],
        "TrainingImage": "string",
        "TrainingInputMode": "string"
"EnableInterContainerTrafficEncryption": boolean,
"EnableNetworkIsolation": boolean,
"InputDataConfig": [
  {
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": {
      "S3DataSource": {
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      }
    },
    "InputMode": "string",
    "RecordWrapperType": "string",
    "ShuffleConfig": {
      "Seed": number
    }
  }
],
"OutputDataConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"ResourceConfig": {
  "InstanceCount": number,
  "InstanceType": "string",
  "VolumeKmsKeyId": "string",
  "VolumeSizeInGB": number
},
"RoleArn": "string",
"StaticHyperParameters": {
  "string": "string"
},
"StoppingCondition": {
  "MaxRuntimeInSeconds": number
},
"VpcConfig": {
  "SecurityGroupIds": [ "string" ],
  "Subnets": [ "string" ]
},
"TrainingJobStatusCounters": {
  "Completed": number,
  "InProgress": number,
  "NonRetryableError": number,
  "RetryableError": number,
  "Stopped": number
},
"WarmStartConfig": {
  "ParentHyperParameterTuningJobs": [
    {
      "HyperParameterTuningJobName": "string"
    }
  ],
  "WarmStartType": "string"
}  

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**BestTrainingJob (p. 601)**

A TrainingJobSummary (p. 893) object that describes the training job that completed with the best current HyperParameterTuningJobObjective (p. 801).

Type: HyperParameterTrainingJobSummary (p. 796) object

**CreationTime (p. 601)**

The date and time that the tuning job started.

Type: Timestamp

**FailureReason (p. 601)**

If the tuning job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**HyperParameterTuningEndTime (p. 601)**

The date and time that the tuning job ended.

Type: Timestamp

**HyperParameterTuningJobArn (p. 601)**

The Amazon Resource Name (ARN) of the tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: \[arn:aws[a-zA-Z0-9\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

**HyperParameterTuningJobConfig (p. 601)**

The HyperParameterTuningJobConfig (p. 799) object that specifies the configuration of the tuning job.

Type: HyperParameterTuningJobConfig (p. 799) object

**HyperParameterTuningJobName (p. 601)**

The name of the tuning job.

Type: String


Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9\-]*$

**HyperParameterTuningJobStatus (p. 601)**

The status of the tuning job: InProgress, Completed, Failed, Stopping, or Stopped.

Type: String

Valid Values: Completed | InProgress | Failed | Stopped | Stopping

**LastModifiedTime (p. 601)**

The date and time that the status of the tuning job was modified.
ObjectiveStatusCounters (p. 601)

The ObjectiveStatusCounters (p. 849) object that specifies the number of training jobs, categorized by the status of their final objective metric, that this tuning job launched.

OverallBestTrainingJob (p. 601)

If the hyperparameter tuning job is an warm start tuning job with a WarmStartType of IDENTICAL_DATA_AND_ALGORITHM, this is the TrainingJobSummary (p. 893) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

TrainingJobDefinition (p. 601)

The HyperParameterTrainingJobDefinition (p. 793) object that specifies the definition of the training jobs that this tuning job launches.

TrainingJobStatusCounters (p. 601)

The TrainingJobStatusCounters (p. 891) object that specifies the number of training jobs, categorized by status, that this tuning job launched.

WarmStartConfig (p. 601)

The configuration for starting the hyperparameter parameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeLabelingJob  
Service: Amazon SageMaker Service  

Gets information about a labeling job.  

Request Syntax  

```json
{
  "LabelingJobName": "string"
}
```

Request Parameters  

For information about the parameters that are common to all actions, see Common Parameters (p. 917).  

The request accepts the following data in JSON format.  

**LabelingJobName (p. 607)**  

The name of the labeling job to return information for.  

Type: String  


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`  

Required: Yes  

Response Syntax  

```json
{
  "CreationTime": number,
  "FailureReason": "string",
  "HumanTaskConfig": {
    "AnnotationConsolidationConfig": {
      "AnnotationConsolidationLambdaArn": "string"
    },
    "MaxConcurrentTaskCount": number,
    "NumberOfHumanWorkersPerDataObject": number,
    "PreHumanTaskLambdaArn": "string",
    "PublicWorkforceTaskPrice": {
      "AmountInUsd": {
        "Cents": number,
        "Dollars": number,
        "TenthFractionsOfACent": number
      }
    },
    "TaskAvailabilityLifetimeInSeconds": number,
    "TaskDescription": "string",
    "TaskKeywords": ["string"],
    "TaskTimeLimitInSeconds": number,
    "TaskTitle": "string",
    "UiConfig": {
      "UiTemplateS3Uri": "string"
    },
    "WorkteamArn": "string"
  },
  "InputConfig": {
    "DataAttributes": {
```
"ContentClassifiers": ["string"],
"DataSource": {
  "S3DataSource": {
    "ManifestS3Uri": "string"
  }
},
"JobReferenceCode": "string",
"LabelAttributeName": "string",
"LabelCategoryConfigS3Uri": "string",
"LabelCounters": {
  "FailedNonRetryableError": number,
  "HumanLabeled": number,
  "MachineLabeled": number,
  "TotalLabeled": number,
  "Unlabeled": number
},
"LabelingJobAlgorithmsConfig": {
  "InitialActiveLearningModelArn": "string",
  "LabelingJobAlgorithmSpecificationArn": "string",
  "LabelingJobResourceConfig": {
    "VolumeKmsKeyId": "string"
  }
},
"LabelingJobArn": "string",
"LabelingJobName": "string",
"LabelingJobOutput": {
  "FinalActiveLearningModelArn": "string",
  "OutputDatasetS3Uri": "string"
},
"LabelingJobStatus": "string",
"LastModifiedTime": number,
"OutputConfig": {
  "KmsKeyId": "string",
  "S3OutputPath": "string"
},
"RoleArn": "string",
"StoppingConditions": {
  "MaxHumanLabeledObjectCount": number,
  "MaxPercentageOfInputDatasetLabeled": number
},
"Tags": [
  {
    "Key": "string",
    "Value": "string"
  }
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

CreationTime (p. 607)

The date and time that the labeling job was created.

Type: Timestamp

FailureReason (p. 607)

If the job failed, the reason that it failed.
Type: String
Length Constraints: Maximum length of 1024.

**HumanTaskConfig (p. 607)**

Configuration information required for human workers to complete a labeling task.

Type: HumanTaskConfig (p. 786) object

**InputConfig (p. 607)**

Input configuration information for the labeling job, such as the Amazon S3 location of the data objects and the location of the manifest file that describes the data objects.

Type: LabelingJobInputConfig (p. 821) object

**JobReferenceCode (p. 607)**

A unique identifier for work done as part of a labeling job.

Type: String
Length Constraints: Minimum length of 1.

**LabelAttributeName (p. 607)**

The attribute used as the label in the output manifest file.

Type: String

Pattern: ^[a-zA-Z0-9]( risky[a-zA-Z0-9]*)*

**LabelCategoryConfigS3Uri (p. 607)**

The S3 location of the JSON file that defines the categories used to label data objects.

The file is a JSON structure in the following format:

```json
{
  "document-version": "2018-11-28"
  "labels": [
    {
      "label": "label 1"
    },
    {
      "label": "label 2"
    },
    ...
    {
      "label": "label n"
    }
  ]
}
```
LabelCounters (p. 607)

Provides a breakdown of the number of data objects labeled by humans, the number of objects labeled by machine, the number of objects than couldn't be labeled, and the total number of objects labeled.

Type: LabelCounters (p. 812) object

LabelingJobAlgorithmsConfig (p. 607)

Configuration information for automated data labeling.

Type: LabelingJobAlgorithmsConfig (p. 815) object

LabelingJobArn (p. 607)

The Amazon Resource Name (ARN) of the labeling job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:labeling-job/.*

LabelingJobName (p. 607)

The name assigned to the labeling job when it was created.

Type: String


Pattern: ^[a-zA-Z0-9]-*[^a-zA-Z0-9]*

LabelingJobOutput (p. 607)

The location of the output produced by the labeling job.

Type: LabelingJobOutput (p. 822) object

LabelingJobStatus (p. 607)

The processing status of the labeling job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

LastModifiedTime (p. 607)

The date and time that the labeling job was last updated.

Type: Timestamp

OutputConfig (p. 607)

The location of the job's output data and the AWS Key Management Service key ID for the key used to encrypt the output data, if any.
Type: LabelingJobOutputConfig (p. 823) object

RoleArn (p. 607)

The Amazon Resource Name (ARN) that Amazon SageMaker assumes to perform tasks on your behalf during data labeling.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9+=,.@-_/]+$  

StoppingConditions (p. 607)

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped.

Type: LabelingJobStoppingConditions (p. 826) object

Tags (p. 607)

An array of key/value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeModel
Service: Amazon SageMaker Service

Describes a model that you created using the CreateModel API.

Request Syntax

{
  "ModelName": "string"
}

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

ModelName (p. 612)

The name of the model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]+)*

Required: Yes

Response Syntax

{
  "Containers": [
    {
      "ContainerHostname": "string",
      "Environment": {
        "string": "string"
      },
      "Image": "string",
      "ModelDataUrl": "string",
      "ModelPackageName": "string"
    }
  ],
  "CreationTime": number,
  "EnableNetworkIsolation": boolean,
  "ExecutionRoleArn": "string",
  "ModelArn": "string",
  "ModelName": "string",
  "PrimaryContainer": {
    "ContainerHostname": "string",
    "Environment": {
      "string": "string"
    },
    "Image": "string",
    "ModelDataUrl": "string",
    "ModelPackageName": "string"
  },
  "VpcConfig": {
    "SecurityGroupIds": [ "string" ]
  }
}
"Subnets": [ "string" ]
}

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Containers (p. 612)**

The containers in the inference pipeline.

Type: Array of [ContainerDefinition (p. 765)] objects

Array Members: Maximum number of 5 items.

**CreationTime (p. 612)**

A timestamp that shows when the model was created.

Type: Timestamp

**EnableNetworkIsolation (p. 612)**

If True, no inbound or outbound network calls can be made to or from the model container.

*Note*

The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

**ExecutionRoleArn (p. 612)**

The Amazon Resource Name (ARN) of the IAM role that you specified for the model.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9\-\_\@\-\_\/.\+]+$

**ModelArn (p. 612)**

The Amazon Resource Name (ARN) of the model.

Type: String


**ModelName (p. 612)**

Name of the Amazon SageMaker model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\_]*\[a-zA-Z0-9\-\_\]*$

**PrimaryContainer (p. 612)**

The location of the primary inference code, associated artifacts, and custom environment map that the inference code uses when it is deployed in production.
Type: `ContainerDefinition (p. 765)` object

**VpcConfig (p. 612)**

A `VpcConfig (p. 913)` object that specifies the VPC that this model has access to. For more information, see *Protect Endpoints by Using an Amazon Virtual Private Cloud*

Type: `VpcConfig (p. 913)` object

**Errors**

For information about the errors that are common to all actions, see *Common Errors (p. 915).*

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeModelPackage

Service: Amazon SageMaker Service

Returns a description of the specified model package, which is used to create Amazon SageMaker models or list them on AWS Marketplace.

To create models in Amazon SageMaker, buyers can subscribe to model packages listed on AWS Marketplace.

Request Syntax

```json
{
    "ModelPackageName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**ModelPackageName (p. 615)**

The name of the model package to describe.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\//)?([a-zA-Z0-9]([a-zA-Z0-9-]){0,62})(?!-)$

Required: Yes

Response Syntax

```json
{
    "CertifyForMarketplace": boolean,
    "CreationTime": number,
    "InferenceSpecification": {
        "Containers": [
        {
            "ContainerHostname": "string",
            "Image": "string",
            "ImageDigest": "string",
            "ModelDataUrl": "string",
            "ProductId": "string"
        }
        ],
        "SupportedContentTypes": [ "string" ],
        "SupportedRealtimeInferenceInstanceTypes": [ "string" ],
        "SupportedResponseMIMETypes": [ "string" ],
        "SupportedTransformInstanceTypes": [ "string" ]
    },
    "ModelPackageArn": "string",
    "ModelPackageDescription": "string",
    "ModelPackageName": "string",
    "ModelPackageStatus": "string",
    "ModelPackageStatusDetails": {
        "ImageScanStatuses": [ ]
    }
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

CertifyForMarketplace (p. 615)

Whether the model package is certified for listing on AWS Marketplace.
Type: Boolean

CreationTime (p. 615)

A timestamp specifying when the model package was created.
Type: Timestamp

InferenceSpecification (p. 615)

Details about inference jobs that can be run with models based on this model package.
Type: InferenceSpecification (p. 806) object

ModelPackageArn (p. 615)

The Amazon Resource Name (ARN) of the model package.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:model-package/.*

ModelPackageDescription (p. 615)

A brief summary of the model package.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

ModelPackageName (p. 615)

The name of the model package being described.
Type: String
Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9]$$

ModelPackageStatus (p. 615)

The current status of the model package.
Type: String
Valid Values: Pending | InProgress | Completed | Failed | Deleting

ModelPackageStatusDetails (p. 615)

Details about the current status of the model package.
Type: ModelPackageStatusDetails (p. 836) object

SourceAlgorithmSpecification (p. 615)

Details about the algorithm that was used to create the model package.
Type: `SourceAlgorithmSpecification (p. 878)` object

**ValidationSpecification (p. 615)**

Configurations for one or more transform jobs that Amazon SageMaker runs to test the model package.

Type: `ModelPackageValidationSpecification (p. 841)` object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeNotebookInstance
Service: Amazon SageMaker Service

Returns information about a notebook instance.

Request Syntax

```json
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 619)**

The name of the notebook instance that you want information about.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* [^a-zA-Z0-9]*[^a-zA-Z0-9]*

Required: Yes

Response Syntax

```json
{
   "AcceleratorTypes": [ "string" ],
   "AdditionalCodeRepositories": [ "string" ],
   "CreationTime": number,
   "DefaultCodeRepository": "string",
   "DirectInternetAccess": "string",
   "FailureReason": "string",
   "InstanceType": "string",
   "KmsKeyId": "string",
   "LastModifiedTime": number,
   "NetworkInterfaceId": "string",
   "NotebookInstanceArn": "string",
   "NotebookInstanceLifecycleConfigName": "string",
   "NotebookInstanceName": "string",
   "NotebookInstanceStatus": "string",
   "RoleArn": "string",
   "SecurityGroups": [ "string" ],
   "SubnetId": "string",
   "Url": "string",
   "VolumeSizeInGB": number
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
AcceleratorTypes (p. 619)

A list of the Elastic Inference (EI) instance types associated with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge

AdditionalCodeRepositories (p. 619)

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://([^/]+)/(.*$) | ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)

CreationTime (p. 619)

A timestamp. Use this parameter to return the time when the notebook instance was created

Type: Timestamp

DefaultCodeRepository (p. 619)

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String


Pattern: ^https://([^/]+)/(.*$) | ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)

DirectInternetAccess (p. 619)

Describes whether Amazon SageMaker provides internet access to the notebook instance. If this value is set to Disabled, the notebook instance does not have internet access, and cannot connect to Amazon SageMaker training and endpoint services.

For more information, see Notebook Instances Are Internet-Enabled by Default.

Type: String

Valid Values: Enabled | Disabled

FailureReason (p. 619)

If status is Failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.
InstanceType (p. 619)

The type of ML compute instance running on the notebook instance.

Type: String

Valid Values:
ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge |
ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge |
| ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge |
| ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge |
| ml.m5.24xlarge | ml.m4.xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge |
| ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
| ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge |
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge

KmsKeyId (p. 619)

The AWS KMS key ID Amazon SageMaker uses to encrypt data when storing it on the ML storage volume attached to the instance.

Type: String

Length Constraints: Maximum length of 2048.

LastModifiedTime (p. 619)

A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.

Type: Timestamp

NetworkInterfaceId (p. 619)

The network interface IDs that Amazon SageMaker created at the time of creating the instance.

Type: String

NotebookInstanceArn (p. 619)

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 619)

Returns the name of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 

NotebookInstanceName (p. 619)

The name of the Amazon SageMaker notebook instance.

Type: String

Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

**NotebookInstanceStatus (p. 619)**  
The status of the notebook instance.  
Type: String  
Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating  

**RoleArn (p. 619)**  
The Amazon Resource Name (ARN) of the IAM role associated with the instance.  
Type: String  
Pattern: ^arn:aws[a-z]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_\/]+$  

**SecurityGroups (p. 619)**  
The IDs of the VPC security groups.  
Type: Array of strings  
Array Members: Maximum number of 5 items.  
Length Constraints: Maximum length of 32.  

**SubnetId (p. 619)**  
The ID of the VPC subnet.  
Type: String  
Length Constraints: Maximum length of 32.  

**Url (p. 619)**  
The URL that you use to connect to the Jupyter notebook that is running in your notebook instance.  
Type: String  

**VolumeSizeInGB (p. 619)**  
The size, in GB, of the ML storage volume attached to the notebook instance.  
Type: Integer  

**Errors**  
For information about the errors that are common to all actions, see Common Errors (p. 915).  

**See Also**  
For more information about using this API in one of the language-specific AWS SDKs, see the following:  
- AWS Command Line Interface  
- AWS SDK for .NET  
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeNotebookInstanceLifecycleConfig
Service: Amazon SageMaker Service

Returns a description of a notebook instance lifecycle configuration.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Request Syntax

```json
{
   "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 624)**

The name of the lifecycle configuration to describe.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 

Required: Yes

Response Syntax

```json
{
   "CreationTime": number,
   "LastModifiedTime": number,
   "NotebookInstanceLifecycleConfigArn": "string",
   "NotebookInstanceLifecycleConfigName": "string",
   "OnCreate": [
      {
         "Content": "string"
      }
   ],
   "OnStart": [
      {
         "Content": "string"
      }
   ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
CreationTime (p. 624)

A timestamp that tells when the lifecycle configuration was created.

Type: Timestamp

LastModifiedTime (p. 624)

A timestamp that tells when the lifecycle configuration was last modified.

Type: Timestamp

NotebookInstanceLifecycleConfigArn (p. 624)

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 624)

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9][-*[a-zA-Z0-9]*$ 

OnCreate (p. 624)

The shell script that runs only once, when you create a notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 845) objects

Array Members: Maximum number of 1 item.

OnStart (p. 624)

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 845) objects

Array Members: Maximum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
DescribeSubscribedWorkteam
Service: Amazon SageMaker Service

Gets information about a work team provided by a vendor. It returns details about the subscription with a vendor in the AWS Marketplace.

Request Syntax

```json
{
  "WorkteamArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**WorkteamArn (p. 627)**

The Amazon Resource Name (ARN) of the subscribed work team to describe.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes

Response Syntax

```json
{
  "SubscribedWorkteam": {
    "ListingId": "string",
    "MarketplaceDescription": "string",
    "MarketplaceTitle": "string",
    "SellerName": "string",
    "WorkteamArn": "string"
  }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**SubscribedWorkteam (p. 627)**

A Workteam instance that contains information about the work team.

Type: SubscribedWorkteam (p. 880) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTrainingJob
Service: Amazon SageMaker Service

Returns information about a training job.

Request Syntax

```
{
  "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**TrainingJobName (p. 629)**

The name of the training job.

Type: String


Pattern: `^[a-zA-Z0-9][-]*[a-zA-Z0-9]$`

Required: Yes

Response Syntax

```
{
  "AlgorithmSpecification": {
    "AlgorithmName": "string",
    "MetricDefinitions": [
      {
        "Name": "string",
        "Regex": "string"
      }
    ],
    "TrainingImage": "string",
    "TrainingInputMode": "string"
  },
  "CreationTime": number,
  "EnableInterContainerTrafficEncryption": boolean,
  "EnableNetworkIsolation": boolean,
  "FailureReason": "string",
  "FinalMetricDataList": [
    {
      "MetricName": "string",
      "Timestamp": number,
      "Value": number
    }
  ],
  "HyperParameters": {
    "string": "string"
  }
}
```
"InputDataConfig": [ 
  { 
    "ChannelName": "string",
    "CompressionType": "string",
    "ContentType": "string",
    "DataSource": { 
      "S3DataSource": { 
        "AttributeNames": [ "string" ],
        "S3DataDistributionType": "string",
        "S3DataType": "string",
        "S3Uri": "string"
      },
      "InputMode": "string",
      "RecordWrapperType": "string",
      "ShuffleConfig": { 
        "Seed": number
      }
    }
  },
  "LabelingJobArn": "string",
  "LastModifiedTime": number,
  "ModelArtifacts": { 
    "S3ModelArtifacts": "string"
  },
  "OutputDataConfig": { 
    "KmsKeyId": "string",
    "S3OutputPath": "string"
  },
  "ResourceConfig": { 
    "InstanceCount": number,
    "InstanceType": "string",
    "VolumeKmsKeyId": "string",
    "VolumeSizeInGB": number
  },
  "RoleArn": "string",
  "SecondaryStatus": "string",
  "SecondaryStatusTransitions": [ 
    { 
      "EndTime": number,
      "StartTime": number,
      "Status": "string",
      "StatusMessage": "string"
    }
  ],
  "StoppingCondition": { 
    "MaxRuntimeInSeconds": number
  },
  "TrainingEndTime": number,
  "TrainingJobArn": "string",
  "TrainingJobName": "string",
  "TrainingJobStatus": "string",
  "TrainingStartTime": number,
  "TuningJobArn": "string",
  "VpcConfig": { 
    "SecurityGroupIds": [ "string" ],
    "Subnets": [ "string" ]
  }
]

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
AlgorithmSpecification (p. 629)

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 744) object

CreationTime (p. 629)

A timestamp that indicates when the training job was created.

Type: Timestamp

EnableInterContainerTrafficEncryption (p. 629)

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

EnableNetworkIsolation (p. 629)

If you want to allow inbound or outbound network calls, except for calls between peers within a training cluster for distributed training, choose True. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note

The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

FailureReason (p. 629)

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

FinalMetricDataList (p. 629)

A collection of MetricData objects that specify the names, values, and dates and times that the training algorithm emitted to Amazon CloudWatch.

Type: Array of MetricData (p. 831) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

HyperParameters (p. 629)

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

InputDataConfig (p. 629)

An array of Channel objects that describes each data input channel.

Type: Array of Channel (p. 756) objects
Array Members: Minimum number of 1 item. Maximum number of 8 items.

**LabelingJobArn (p. 629)**

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`

**LastModifiedTime (p. 629)**

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

**ModelArtifacts (p. 629)**

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: `ModelArtifacts (p. 833)` object

**OutputDataConfig (p. 629)**

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: `OutputDataConfig (p. 851)` object

**ResourceConfig (p. 629)**

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: `ResourceConfig (p. 866)` object

**RoleArn (p. 629)**

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String


Pattern: `^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-ZA-Z_0-9+=,.@\-_\/]+$`

**SecondaryStatus (p. 629)**

Provides detailed information about the state of the training job. For detailed information on the secondary status of the training job, see `StatusMessage` under `SecondaryStatusTransition (p. 874)`.

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

**InProgress**

- **Starting** - Starting the training job.
- **Downloading** - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
- **Training** - Training is in progress.
- **Uploading** - Training is complete and the model artifacts are being uploaded to the S3 location.
Completed
  • Completed - The training job has completed.

Failed
  • Failed - The training job has failed. The reason for the failure is returned in the
    FailureReason field of DescribeTrainingJobResponse.

Stopped
  • MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed
    runtime.
  • Stopped - The training job has stopped.

Stopping
  • Stopping - Stopping the training job.

**Important**
Valid values for SecondaryStatus are subject to change.

We no longer support the following secondary statuses:
  • LaunchingMLInstances
  • PreparingTrainingStack
  • DownloadingTrainingImage

Type: String

Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack |
  Downloading | DownloadingTrainingImage | Training | Uploading | Stopping |
  Stopped | MaxRuntimeExceeded | Completed | Failed

**SecondaryStatusTransitions (p. 629)**
A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 874) objects

**StoppingCondition (p. 629)**
The condition under which to stop the training job.

Type: StoppingCondition (p. 879) object

**TrainingEndTime (p. 629)**
Indicates the time when the training job ends on training instances. You are billed for the time
interval between the value of TrainingStartTime and this time. For successful jobs and stopped
jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when
Amazon SageMaker detects a job failure.

Type: Timestamp

**TrainingJobArn (p. 629)**
The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:training-job/.*

**TrainingJobName (p. 629)**
Name of the model training job.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*$

**TrainingJobStatus (p. 629)**

The status of the training job.

Amazon SageMaker provides the following training job statuses:

- **InProgress** - The training is in progress.
- **Completed** - The training job has completed.
- **Failed** - The training job has failed. To see the reason for the failure, see the `FailureReason` field in the response to a `DescribeTrainingJobResponse` call.
- **Stopping** - The training job is stopping.
- **Stopped** - The training job has stopped.

For more detailed information, see `SecondaryStatus`.

Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped

**TrainingStartTime (p. 629)**

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of `TrainingEndTime`. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

**TuningJobArn (p. 629)**

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

**VpcConfig (p. 629)**

A `VpcConfig` (p. 913) object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: `VpcConfig` (p. 913) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeTransformJob
Service: Amazon SageMaker Service

Returns information about a transform job.

Request Syntax

```json
{
    "TransformJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**TransformJobName (p. 636)**

The name of the transform job that you want to view details of.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

Response Syntax

```json
{
    "BatchStrategy": "string",
    "CreationTime": number,
    "Environment": {
        "string": "string"
    },
    "FailureReason": "string",
    "LabelingJobArn": "string",
    "MaxConcurrentTransforms": number,
    "MaxPayloadInMB": number,
    "ModelName": "string",
    "TransformEndTime": number,
    "TransformInput": {
        "CompressionType": "string",
        "ContentType": "string",
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "string",
                "S3Uri": "string"
            }
        },
        "SplitType": "string"
    },
    "TransformJobArn": "string",
    "TransformJobName": "string",
    "TransformJobStatus": "string",
    "TransformOutput": {
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**BatchStrategy (p. 636)**

If you want to include only one record in a batch, specify `SingleRecord`. If you want batches to contain a maximum of the number of records specified in the `MaxPayloadInMB` parameter, specify `MultiRecord`.

Type: String

Valid Values: `MultiRecord` | `SingleRecord`

**CreationTime (p. 636)**

A timestamp that shows when the transform Job was created.

Type: Timestamp

**Environment (p. 636)**

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: `[a-zA-Z_][a-zA-Z0-9_]*`

Value Length Constraints: Maximum length of 10240.

**FailureReason (p. 636)**

If the transform job failed, the reason that it failed.

Type: String

Length Constraints: Maximum length of 1024.

**LabelingJobArn (p. 636)**

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`
MaxConcurrentTransforms (p. 636)

The maximum number of parallel requests on each instance node that can be launched in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

MaxPayloadInMB (p. 636)

The maximum payload size, in MB, used in the transform job.

Type: Integer

Valid Range: Minimum value of 0.

ModelName (p. 636)

The name of the model used in the transform job.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* TransformEndTime (p. 636)

Indicates when the transform job has been completed, or has stopped or failed. You are billed for the time interval between this time and the value of TransformStartTime.

Type: Timestamp

TransformInput (p. 636)

Describes the dataset to be transformed and the Amazon S3 location where it is stored.

Type: TransformInput (p. 898) object

TransformJobArn (p. 636)

The Amazon Resource Name (ARN) of the transform job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:transform-job/.*

TransformJobName (p. 636)

The name of the transform job.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* TransformJobStatus (p. 636)

The status of the transform job. If the transform job failed, the reason is returned in the FailureReason field.

Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped

**TransformOutput (p. 636)**

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: TransformOutput (p. 904) object

**TransformResources (p. 636)**

Describes the resources, including ML instance types and ML instance count, to use for the transform job.

Type: TransformResources (p. 906) object

**TransformStartTime (p. 636)**

Indicates when the transform job starts on ML instances. You are billed for the time interval between this time and the value of TransformEndTime.

Type: Timestamp

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DescribeWorkteam
Service: Amazon SageMaker Service

Gets information about a specific work team. You can see information such as the create date, the last updated date, membership information, and the work team's Amazon Resource Name (ARN).

Request Syntax

```json
{
   "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**WorkteamName (p. 640)**

The name of the work team to return a description of.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Syntax

```json
{
   "Workteam": {
      "CreateDate": number,
      "Description": "string",
      "LastUpdatedDate": number,
      "MemberDefinitions": [
         {
            "CognitoMemberDefinition": {
               "ClientId": "string",
               "UserGroup": "string",
               "UserPool": "string"
            }
         }
      ],
      "ProductListingIds": [ "string" ],
      "SubDomain": "string",
      "WorkteamArn": "string",
      "WorkteamName": "string"
   }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
Workteam (p. 640)

A Workteam instance that contains information about the work team.

Type: Workteam (p. 914) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
GetSearchSuggestions
Service: Amazon SageMaker Service

An auto-complete API for the search functionality in the Amazon SageMaker console. It returns suggestions of possible matches for the property name to use in Search queries. Provides suggestions for HyperParameters, Tags, and Metrics.

Request Syntax

```json
{
    "Resource": "string",
    "SuggestionQuery": {
        "PropertyNameQuery": {
            "PropertyNameHint": "string"
        }
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

Resource (p. 642)

The name of the Amazon SageMaker resource to Search for. The only valid Resource value is TrainingJob.

Type: String

Valid Values: TrainingJob

Required: Yes

SuggestionQuery (p. 642)

Limits the property names that are included in the response.

Type: SuggestionQuery (p. 882) object

Required: No

Response Syntax

```json
{
    "PropertyNameSuggestions": [
        {
            "PropertyName": "string"
        }
    ]
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
PropertyNameSuggestions (p. 642)

A list of property names for a Resource that match a SuggestionQuery.

Type: Array of PropertyNameSuggestion (p. 861) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListAlgorithms
Service: Amazon SageMaker Service

Lists the machine learning algorithms that have been created.

Request Syntax

```
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 644)**

A filter that returns only algorithms created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 644)**

A filter that returns only algorithms created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 644)**

The maximum number of algorithms to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 644)**

A string in the algorithm name. This filter returns only algorithms whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-]+

Required: No
**NextToken (p. 644)**

If the response to a previous ListAlgorithms request was truncated, the response includes a NextToken. To retrieve the next set of algorithms, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

**SortBy (p. 644)**

The parameter by which to sort the results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

**SortOrder (p. 644)**

The sort order for the results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```json
{
  "AlgorithmSummaryList": [
    {
      "AlgorithmArn": "string",
      "AlgorithmDescription": "string",
      "AlgorithmName": "string",
      "AlgorithmStatus": "string",
      "CreationTime": number
    }
  ],
  "NextToken": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**AlgorithmSummaryList (p. 645)**

> An array of AlgorithmSummary objects, each of which lists an algorithm.

Type: Array of AlgorithmSummary (p. 748) objects

**NextToken (p. 645)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of algorithms, use it in the subsequent request.
Type: String
Length Constraints: Maximum length of 8192.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListCodeRepositories
Service: Amazon SageMaker Service

Gets a list of the Git repositories in your account.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 647)

A filter that returns only Git repositories that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 647)

A filter that returns only Git repositories that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 647)

A filter that returns only Git repositories that were last modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 647)

A filter that returns only Git repositories that were last modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 647)

The maximum number of Git repositories to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 647)**

A string in the Git repositories name. This filter returns only repositories whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-]+
Required: No

**NextToken (p. 647)**

If the result of a ListCodeRepositoriesOutput request was truncated, the response includes a NextToken. To get the next set of Git repositories, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Required: No

**SortBy (p. 647)**

The field to sort results by. The default is Name.
Type: String
Valid Values: Name | CreationTime | LastModifiedTime
Required: No

**SortOrder (p. 647)**

The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**Response Syntax**

```json
{
  "CodeRepositorySummaryList": [
    {
      "CodeRepositoryArn": "string",
      "CodeRepositoryName": "string",
      "CreationTime": number,
      "GitConfig": {
        "Branch": "string",
        "RepositoryUrl": "string",
        "SecretArn": "string"
      },
      "LastModifiedTime": number
    }
  ]
}
```
"NextToken": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CodeRepositorySummaryList (p. 648)**

- Gets a list of summaries of the Git repositories. Each summary specifies the following values for the repository:
  - Name
  - Amazon Resource Name (ARN)
  - Creation time
  - Last modified time
  - Configuration information, including the URL location of the repository and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.

  Type: Array of `CodeRepositorySummary (p. 760)` objects

**NextToken (p. 648)**

- If the result of a `ListCodeRepositoriesOutput` request was truncated, the response includes a NextToken. To get the next set of Git repositories, use the token in the next request.

  Type: String

  Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see [Common Errors (p. 915)](#).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- [AWS Command Line Interface](#)
- [AWS SDK for .NET](#)
- [AWS SDK for C++](#)
- [AWS SDK for Go](#)
- [AWS SDK for Java](#)
- [AWS SDK for JavaScript](#)
- [AWS SDK for PHP V3](#)
- [AWS SDK for Python](#)
- [AWS SDK for Ruby V2](#)
ListCompilationJobs
Service: Amazon SageMaker Service

Lists model compilation jobs that satisfy various filters.

To create a model compilation job, use CreateCompilationJob (p. 519). To get information about a particular model compilation job you have created, use DescribeCompilationJob (p. 592).

Request Syntax

```json
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "LastModifiedTimeAfter": number,
   "LastModifiedTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 650)

A filter that returns the model compilation jobs that were created after a specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 650)

A filter that returns the model compilation jobs that were created before a specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 650)

A filter that returns the model compilation jobs that were modified after a specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 650)

A filter that returns the model compilation jobs that were modified before a specified time.

Type: Timestamp

Required: No
MaxResults (p. 650)

The maximum number of model compilation jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 650)

A filter that returns the model compilation jobs whose name contains a specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-\-]+

Required: No

NextToken (p. 650)

If the result of the previous ListCompilationJobs request was truncated, the response includes a NextToken. To retrieve the next set of model compilation jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

SortBy (p. 650)

The field by which to sort results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

SortOrder (p. 650)

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

StatusEquals (p. 650)

A filter that retrieves model compilation jobs with a specific DescribeCompilationJob:CompilationJobStatus (p. 593) status.

Type: String

Valid Values: INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED

Required: No
Response Syntax

```json
{
   "CompilationJobSummaries": [
      {
         "CompilationEndTime": number,
         "CompilationJobArn": "string",
         "CompilationJobName": "string",
         "CompilationJobStatus": "string",
         "CompilationStartTime": number,
         "CompilationTargetDevice": "string",
         "CreationTime": number,
         "LastModifiedTime": number
      }
   ],
   "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**CompilationJobSummaries (p. 652)**

An array of [CompilationJobSummary (p. 763)] objects, each describing a model compilation job.

Type: Array of CompilationJobSummary (p. 763) objects

**NextToken (p. 652)**

If the response is truncated, Amazon SageMaker returns this NextToken. To retrieve the next set of model compilation jobs, use this token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see [Common Errors (p. 915)].

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListEndpointConfigs
Service: Amazon SageMaker Service
Lists endpoint configurations.

Request Syntax

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 653)**

A filter that returns only endpoint configurations created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 653)**

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 653)**

The maximum number of training jobs to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 653)**

A string in the endpoint configuration name. This filter returns only endpoint configurations whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No
NextToken (p. 653)

If the result of the previous ListEndpointConfig request was truncated, the response includes a NextToken. To retrieve the next set of endpoint configurations, use the token in the next request.

Type: String
Length Constraints: Maximum length of 8192.
Required: No

SortBy (p. 653)

The field to sort results by. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime
Required: No

SortOrder (p. 653)

The sort order for results. The default is Ascending.

Type: String
Valid Values: Ascending | Descending
Required: No

Response Syntax

```
{
  "EndpointConfigs": [
    {
      "CreationTime": number,
      "EndpointConfigArn": "string",
      "EndpointConfigName": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

EndpointConfigs (p. 654)

An array of endpoint configurations.

Type: Array of EndpointConfigSummary (p. 777) objects

NextToken (p. 654)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of endpoint configurations, use it in the subsequent request.

Type: String
Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListEndpoints
Service: Amazon SageMaker Service

Lists endpoints.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 656)

A filter that returns only endpoints that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 656)

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 656)

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 656)

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

MaxResults (p. 656)

The maximum number of endpoints to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 656)**
A string in endpoint names. This filter returns only endpoints whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-]+
Required: No

**NextToken (p. 656)**
If the result of a ListEndpoints request was truncated, the response includes a NextToken. To retrieve the next set of endpoints, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Required: No

**SortBy (p. 656)**
Sorts the list of results. The default is CreationTime.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

**SortOrder (p. 656)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 656)**
A filter that returns only endpoints with the specified status.
Type: String
Valid Values: OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed
Required: No

**Response Syntax**

```json
{
  "Endpoints": [
    
  ]
}
```
"CreationTime": number,
"EndpointArn": "string",
"EndpointName": "string",
"EndpointStatus": "string",
"LastModifiedTime": number
} },
"NextToken": "string"
}

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**Endpoints (p. 657)**

An array or endpoint objects.

Type: Array of `EndpointSummary (p. 778)` objects

**NextToken (p. 657)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of training jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListHyperParameterTuningJobs
Service: Amazon SageMaker Service

Gets a list of HyperParameterTuningJobSummary (p. 802) objects that describe the hyperparameter tuning jobs launched in your account.

Request Syntax

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 659)

A filter that returns only tuning jobs that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 659)

A filter that returns only tuning jobs that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 659)

A filter that returns only tuning jobs that were modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 659)

A filter that returns only tuning jobs that were modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 659)

The maximum number of tuning jobs to return. The default value is 10.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 659)**
A string in the tuning job name. This filter returns only tuning jobs whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9-\-]+
Required: No

**NextToken (p. 659)**
If the result of the previous ListHyperParameterTuningJobs request was truncated, the response includes a NextToken. To retrieve the next set of tuning jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Required: No

**SortBy (p. 659)**
The field to sort results by. The default is Name.
Type: String
Valid Values: Name | Status | CreationTime
Required: No

**SortOrder (p. 659)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 659)**
A filter that returns only tuning jobs with the specified status.
Type: String
Valid Values: Completed | InProgress | Failed | Stopped | Stopping
Required: No

**Response Syntax**
```json
{
}
```
"HyperParameterTuningJobSummaries": [  
  {  
    "CreationTime": number,
    "HyperParameterTuningEndTime": number,
    "HyperParameterTuningJobArn": "string",
    "HyperParameterTuningJobName": "string",
    "HyperParameterTuningJobStatus": "string",
    "LastModifiedTime": number,
    "ObjectiveStatusCounters": {  
      "Failed": number,
      "Pending": number,
      "Succeeded": number
    },
    "ResourceLimits": {  
      "MaxNumberOfTrainingJobs": number,
      "MaxParallelTrainingJobs": number
    },
    "Strategy": "string",
    "TrainingJobStatusCounters": {  
      "Completed": number,
      "InProgress": number,
      "NonRetryableError": number,
      "RetryableError": number,
      "Stopped": number
    }
  },
  
  ],
  "NextToken": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

HyperParameterTuningJobSummaries (p. 660)

A list of HyperParameterTuningJobSummary (p. 802) objects that describe the tuning jobs that the ListHyperParameterTuningJobs request returned.

Type: Array of HyperParameterTuningJobSummary (p. 802) objects

NextToken (p. 660)

If the result of this ListHyperParameterTuningJobs request was truncated, the response includes a NextToken. To retrieve the next set of tuning jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListLabelingJobs
Service: Amazon SageMaker Service

Gets a list of labeling jobs.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 663)**

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 663)**

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 663)**

A filter that returns only labeling jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 663)**

A filter that returns only labeling jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 663)**

The maximum number of labeling jobs to return in each page of the response.

Type: Integer
NameContains (p. 663)

A string in the labeling job name. This filter returns only labeling jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9\-]+

Required: No

NextToken (p. 663)

If the result of the previous ListLabelingJobs request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

SortBy (p. 663)

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

SortOrder (p. 663)

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

StatusEquals (p. 663)

A filter that retrieves only labeling jobs with a specific status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

Response Syntax

{
    "LabelingJobSummaryList": [
        {
            "AnnotationConsolidationLambdaArn": "string",
        }
    ]
}
"CreationTime": number,
"FailureReason": "string",
"InputConfig": {
    "DataAttributes": {
        "ContentClassifiers": [ "string" ]
    },
    "DataSource": {
        "S3DataSource": {
            "ManifestS3Uri": "string"
        }
    }
},
"LabelCounters": {
    "FailedNonRetryableError": number,
    "HumanLabeled": number,
    "MachineLabeled": number,
    "TotalLabeled": number,
    "Unlabeled": number
},
"LabelingJobArn": "string",
"LabelingJobName": "string",
"LabelingJobOutput": {
    "FinalActiveLearningModelArn": "string",
    "OutputDatasetS3Uri": "string"
},
"LabelingJobStatus": "string",
"LastModifiedTime": number,
"PreHumanTaskLambdaArn": "string",
"WorkteamArn": "string"
],
"NextToken": "string"
}

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

LabelingJobSummaryList (p. 664)

An array of LabelingJobSummary objects, each describing a labeling job.

Type: Array of LabelingJobSummary (p. 827) objects

NextToken (p. 664)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
**ListLabelingJobsForWorkteam**  
Service: Amazon SageMaker Service

Gets a list of labeling jobs assigned to a specified work team.

**Request Syntax**

```json
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "JobReferenceCodeContains": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "WorkteamArn": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 667)**

A filter that returns only labeling jobs created after the specified time (timestamp).

Type: Timestamp  
Required: No

**CreationTimeBefore (p. 667)**

A filter that returns only labeling jobs created before the specified time (timestamp).

Type: Timestamp  
Required: No

**JobReferenceCodeContains (p. 667)**

A filter the limits jobs to only the ones whose job reference code contains the specified string.

Type: String  
Length Constraints: Minimum length of 1. Maximum length of 255.  
Required: No

**MaxResults (p. 667)**

The maximum number of labeling jobs to return in each page of the response.

Type: Integer  
Valid Range: Minimum value of 1. Maximum value of 100.  
Required: No

**NextToken (p. 667)**

If the result of the previous ListLabelingJobsForWorkteam request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Required: No

**SortBy (p. 667)**

The field to sort results by. The default is CreationTime.
Type: String
Valid Values: CreationTime
Required: No

**SortOrder (p. 667)**

The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**WorkteamArn (p. 667)**

The Amazon Resource Name (ARN) of the work team for which you want to see labeling jobs for.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-\*:sagemaker:[a-z0-9\-\*][0-9]{12}:workteam/.*
Required: Yes

**Response Syntax**

```json
{
  "LabelingJobSummaryList": [
    {
      "CreationTime": number,
      "JobReferenceCode": "string",
      "LabelCounters": {
        "HumanLabeled": number,
        "PendingHuman": number,
        "Total": number
      },
      "LabelingJobName": "string",
      "WorkRequesterAccountId": "string"
    }
  ],
  "NextToken": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
LabelingJobSummaryList (p. 668)

An array of LabelingJobSummary objects, each describing a labeling job.

Type: Array of LabelingJobForWorkteamSummary (p. 819) objects

NextToken (p. 668)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of labeling jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListModelPackages
Service: Amazon SageMaker Service

Lists the model packages that have been created.

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 670)**

A filter that returns only model packages created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 670)**

A filter that returns only model packages created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 670)**

The maximum number of model packages to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 670)**

A string in the model package name. This filter returns only model packages whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-\-]+

Required: No
NextToken (p. 670)

If the response to a previous ListModelPackages request was truncated, the response includes a NextToken. To retrieve the next set of model packages, use the token in the next request.

Type: String
Length Constraints: Maximum length of 8192.
Required: No

SortBy (p. 670)

The parameter by which to sort the results. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime
Required: No

SortOrder (p. 670)

The sort order for the results. The default is Ascending.

Type: String
Valid Values: Ascending | Descending
Required: No

Response Syntax

```json
{
  "ModelPackageSummaryList": [
    {
      "CreationTime": number,
      "ModelPackageArn": "string",
      "ModelPackageDescription": "string",
      "ModelPackageName": "string",
      "ModelPackageStatus": "string"
    }
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

ModelPackageSummaryList (p. 671)

An array of ModelPackageSummary objects, each of which lists a model package.

Type: Array of ModelPackageSummary (p. 838) objects

NextToken (p. 671)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of model packages, use it in the subsequent request.
Type: String
Length Constraints: Maximum length of 8192.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListModels
Service: Amazon SageMaker Service

Lists models created with the CreateModel API.

Request Syntax

```
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 673)

A filter that returns only models created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 673)

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 673)

The maximum number of models to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 673)

A string in the training job name. This filter returns only models in the training job whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No
NextToken (p. 673)

If the response to a previous ListModels request was truncated, the response includes a NextToken. To retrieve the next set of models, use the token in the next request.

Type: String
Length Constraints: Maximum length of 8192.
Required: No

SortBy (p. 673)

Sorts the list of results. The default is CreationTime.

Type: String
Valid Values: Name  |  CreationTime
Required: No

SortOrder (p. 673)

The sort order for results. The default is Ascending.

Type: String
Valid Values: Ascending  |  Descending
Required: No

Response Syntax

```json
{
  "Models": [
    
    "CreationTime": number,
    "ModelArn": "string",
    "ModelName": "string"
  
  
  ],
  "NextToken": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Models (p. 674)

An array of ModelSummary objects, each of which lists a model.

Type: Array of ModelSummary (p. 842) objects

NextToken (p. 674)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of models, use it in the subsequent request.

Type: String
Length Constraints: Maximum length of 8192.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListNotebookInstanceLifecycleConfigs
Service: Amazon SageMaker Service

Lists notebook instance lifestyle configurations created with the
CreateNotebookInstanceLifecycleConfig (p. 551) API.

Request Syntax

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common
Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 676)**

A filter that returns only lifecycle configurations that were created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 676)**

A filter that returns only lifecycle configurations that were created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 676)**

A filter that returns only lifecycle configurations that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 676)**

A filter that returns only lifecycle configurations that were modified before the specified time (timestamp).

Type: Timestamp

Required: No
MaxResults (p. 676)

The maximum number of lifecycle configurations to return in the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 676)

A string in the lifecycle configuration name. This filter returns only lifecycle configurations whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No

NextToken (p. 676)

If the result of a ListNotebookInstanceLifecycleConfigs request was truncated, the response includes a NextToken. To get the next set of lifecycle configurations, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

SortBy (p. 676)

Sorts the list of results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | LastModifiedTime

Required: No

SortOrder (p. 676)

The sort order for results.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "NotebookInstanceLifecycleConfigs": [
        {
            "CreationTime": number,
            "LastModifiedTime": number,
            ...
        }...
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 677)**

If the response is truncated, Amazon SageMaker returns this token. To get the next set of lifecycle configurations, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

**NotebookInstanceLifecycleConfigs (p. 677)**

An array of NotebookInstanceLifecycleConfiguration objects, each listing a lifecycle configuration.

Type: Array of NotebookInstanceLifecycleConfigSummary (p. 844) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListNotebookInstances
Service: Amazon SageMaker Service

Returns a list of the Amazon SageMaker notebook instances in the requester's account in an AWS Region.

Request Syntax

```
{
   "AdditionalCodeRepositoryEquals": "string",
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "DefaultCodeRepositoryContains": "string",
   "LastModifiedTimeAfter": number,
   "LastModifiedTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "NotebookInstanceLifecycleConfigNameContains": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

AdditionalCodeRepositoryEquals (p. 679)

A filter that returns only notebook instances with associated with the specified git repository.

Type: String


Pattern: ^https://([/]+)/?(.*)$|^[a-zA-Z0-9](-*[a-zA-Z0-9])*$\

Required: No

CreationTimeAfter (p. 679)

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 679)

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

DefaultCodeRepositoryContains (p. 679)

A string in the name or URL of a Git repository associated with this notebook instance. This filter returns only notebook instances associated with a git repository with a name that contains the specified string.
Type: String

Length Constraints: Maximum length of 1024.

Pattern: [a-zA-Z0-9-]+

Required: No

**LastModifiedTimeAfter (p. 679)**

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 679)**

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 679)**

The maximum number of notebook instances to return.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 679)**

A string in the notebook instances' name. This filter returns only notebook instances whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No

**NextToken (p. 679)**

If the previous call to the ListNotebookInstances is truncated, the response includes a NextToken. You can use this token in your subsequent ListNotebookInstances request to fetch the next set of notebook instances.

**Note**

You might specify a filter or a sort order in your request. When response is truncated, you must use the same values for the filter and sort order in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No
**NotebookInstanceLifecycleConfigNameContains (p. 679)**

A string in the name of a notebook instances lifecycle configuration associated with this notebook instance. This filter returns only notebook instances associated with a lifecycle configuration with a name that contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 

Required: No

**SortBy (p. 679)**

The field to sort results by. The default is Name.

Type: String

Valid Values: Name | CreationTime | Status 

Required: No

**SortOrder (p. 679)**

The sort order for results.

Type: String

Valid Values: Ascending | Descending 

Required: No

**StatusEquals (p. 679)**

A filter that returns only notebook instances with the specified status.

Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating 

Required: No

**Response Syntax**

```json
{
  "NextToken": "string",
  "NotebookInstances": [
    {
      "AdditionalCodeRepositories": [ "string" ],
      "CreationTime": number,
      "DefaultCodeRepository": "string",
      "InstanceType": "string",
      "LastModifiedTime": number,
      "NotebookInstanceArn": "string",
      "NotebookInstanceLifecycleConfigName": "string",
      "NotebookInstanceName": "string",
      "NotebookInstanceStatus": "string",
      "Url": "string"
    }
  ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 681)**

If the response to the previous `ListNotebookInstances` request was truncated, Amazon SageMaker returns this token. To retrieve the next set of notebook instances, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

**NotebookInstances (p. 681)**

An array of `NotebookInstanceSummary` objects, one for each notebook instance.

Type: Array of `NotebookInstanceSummary` objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListSubscribedWorkteams
Service: Amazon SageMaker Service

Gets a list of the work teams that you are subscribed to in the AWS Marketplace. The list may be empty if no work team satisfies the filter specified in the NameContains parameter.

Request Syntax

```
{
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

MaxResults (p. 683)

The maximum number of work teams to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 683)

A string in the work team name. This filter returns only work teams whose name contains the specified string.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

NextToken (p. 683)

If the result of the previous ListSubscribedWorkteams request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

Response Syntax

```
{
    "NextToken": "string",
    "SubscribedWorkteams": [  
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**NextToken (p. 683)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

**SubscribedWorkteams (p. 683)**

An array of Workteam objects, each describing a work team.

Type: Array of SubscribedWorkteam (p. 880) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTags
Service: Amazon SageMaker Service

Returns the tags for the specified Amazon SageMaker resource.

Request Syntax

```json
{
    "MaxResults": number,
    "NextToken": "string",
    "ResourceArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

MaxResults (p. 685)

- Maximum number of tags to return.
  - Type: Integer
  - Valid Range: Minimum value of 50.
  - Required: No

NextToken (p. 685)

- If the response to the previous ListTags request is truncated, Amazon SageMaker returns this token. To retrieve the next set of tags, use it in the subsequent request.
  - Type: String
  - Length Constraints: Maximum length of 8192.
  - Required: No

ResourceArn (p. 685)

- The Amazon Resource Name (ARN) of the resource whose tags you want to retrieve.
  - Type: String
  - Length Constraints: Maximum length of 256.
  - Required: Yes

Response Syntax

```json
{
    "NextToken": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 685)**

If response is truncated, Amazon SageMaker includes a token in the response. You can use this token in your subsequent request to fetch next set of tokens.

- Type: String
- Length Constraints: Maximum length of 8192.

**Tags (p. 685)**

An array of `Tag` objects, each with a tag key and a value.

- Type: Array of `Tag` (p. 883) objects
- Array Members: Minimum number of 0 items. Maximum number of 50 items.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrainingJobs
Service: Amazon SageMaker Service

Lists training jobs.

Request Syntax

```json
{
    "CreationTimeAfter": number,
    "CreationTimeBefore": number,
    "LastModifiedTimeAfter": number,
    "LastModifiedTimeBefore": number,
    "MaxResults": number,
    "NameContains": "string",
    "NextToken": "string",
    "SortBy": "string",
    "SortOrder": "string",
    "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 687)

A filter that returns only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 687)

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 687)

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 687)

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 687)

The maximum number of training jobs to return in the response.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

NameContains (p. 687)
A string in the training job name. This filter returns only training jobs whose name contains the specified string.
Type: String
Length Constraints: Maximum length of 63.
Pattern: [a-zA-Z0-9\-\_]+
Required: No

NextToken (p. 687)
If the result of the previous ListTrainingJobs request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.
Type: String
Length Constraints: Maximum length of 8192.
Required: No

SortBy (p. 687)
The field to sort results by. The default is CreationTime.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

SortOrder (p. 687)
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

StatusEquals (p. 687)
A filter that retrieves only training jobs with a specific status.
Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "TrainingJobSummaries": [
        
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 688)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of training jobs, use it in the subsequent request.

Type: String

Length Constraints: Maximum length of 8192.

**TrainingJobSummaries (p. 688)**

An array of `TrainingJobSummary` objects, each listing a training job.

Type: Array of `TrainingJobSummary (p. 893)` objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTrainingJobsForHyperParameterTuningJob

Service: Amazon SageMaker Service

Gets a list of TrainingJobSummary (p. 893) objects that describe the training jobs that a hyperparameter tuning job launched.

Request Syntax

```
{
   "HyperParameterTuningJobName": "string",
   "MaxResults": number,
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 690)

The name of the tuning job whose training jobs you want to list.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

MaxResults (p. 690)

The maximum number of training jobs to return. The default value is 10.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NextToken (p. 690)

If the result of the previous ListTrainingJobsForHyperParameterTuningJob request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

SortBy (p. 690)

The field to sort results by. The default is Name.
If the value of this field is `FinalObjectiveMetricValue`, any training jobs that did not return an objective metric are not listed.

**Type:** String

**Valid Values:** Name | CreationTime | Status | FinalObjectiveMetricValue

**Required:** No

**SortOrder (p. 690)**

The sort order for results. The default is Ascending.

**Type:** String

**Valid Values:** Ascending | Descending

**Required:** No

**StatusEquals (p. 690)**

A filter that returns only training jobs with the specified status.

**Type:** String

**Valid Values:** InProgress | Completed | Failed | Stopping | Stopped

**Required:** No

**Response Syntax**

```json
{
    "NextToken": "string",
    "TrainingJobSummaries": [
    {
        "CreationTime": number,
        "FailureReason": "string",
        "FinalHyperParameterTuningJobObjectiveMetric": { 
            "MetricName": "string",
            "Type": "string",
            "Value": number
        },
        "ObjectiveStatus": "string",
        "TrainingEndTime": number,
        "TrainingJobArn": "string",
        "TrainingJobName": "string",
        "TrainingJobStatus": "string",
        "TrainingStartTime": number,
        "TunedHyperParameters": { 
            "string" : "string"
        },
        "TuningJobName": "string"
    }
    ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.
NextToken (p. 691)

If the result of this ListTrainingJobsForHyperParameterTuningJob request was truncated, the response includes a NextToken. To retrieve the next set of training jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

TrainingJobSummaries (p. 691)

A list of TrainingJobSummary (p. 893) objects that describe the training jobs that the ListTrainingJobsForHyperParameterTuningJob request returned.

Type: Array of HyperParameterTrainingJobSummary (p. 796) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTransformJobs
Service: Amazon SageMaker Service

Lists transform jobs.

Request Syntax

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "LastModifiedTimeAfter": number,
  "LastModifiedTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string",
  "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 693)**

A filter that returns only transform jobs created after the specified time.

Type: Timestamp

Required: No

**CreationTimeBefore (p. 693)**

A filter that returns only transform jobs created before the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 693)**

A filter that returns only transform jobs modified after the specified time.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 693)**

A filter that returns only transform jobs modified before the specified time.

Type: Timestamp

Required: No

**MaxResults (p. 693)**

The maximum number of transform jobs to return in the response. The default value is 10.

Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 693)

A string in the transform job name. This filter returns only transform jobs whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: \[a-zA-Z0-9\-\]+

Required: No

NextToken (p. 693)

If the result of the previous ListTransformJobs request was truncated, the response includes a NextToken. To retrieve the next set of transform jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

SortBy (p. 693)

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

SortOrder (p. 693)

The sort order for results. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

StatusEquals (p. 693)

A filter that retrieves only transform jobs with a specific status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "TransformJobSummaries": [
        {
        }
    ]
}
```
"CreationTime": number,
"FailureReason": "string",
"LastModifiedTime": number,
"TransformEndTime": number,
"TransformJobArn": "string",
"TransformJobName": "string",
"TransformJobStatus": "string"
}
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 694)

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of transform jobs, use it in the next request.

Type: String

Length Constraints: Maximum length of 8192.

TransformJobSummaries (p. 694)

An array of TransformJobSummary objects.

Type: Array of TransformJobSummary (p. 902) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**ListWorkteams**  
**Service:** Amazon SageMaker Service

Gets a list of work teams that you have defined in a region. The list may be empty if no work team satisfies the filter specified in the `NameContains` parameter.

**Request Syntax**

```json
{
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**MaxResults (p. 696)**

The maximum number of work teams to return in each page of the response.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NameContains (p. 696)**

A string in the work team's name. This filter returns only work teams whose name contains the specified string.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

**NextToken (p. 696)**

If the result of the previous `ListWorkteams` request was truncated, the response includes a NextToken. To retrieve the next set of labeling jobs, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

**SortBy (p. 696)**

The field to sort results by. The default is `CreationTime`.  

---

696
Type: String
Valid Values: Name | CreateDate
Required: No

**SortOrder (p. 696)**

The sort order for results. The default is Ascending.

Type: String
Valid Values: Ascending | Descending
Required: No

**Response Syntax**

```json
{
   "NextToken": "string",
   "Workteams": [
      {
         "CreateDate": number,
         "Description": "string",
         "LastUpdatedDate": number,
         "MemberDefinitions": [
            {
               "CognitoMemberDefinition": {
                  "ClientId": "string",
                  "UserGroup": "string",
                  "UserPool": "string"
               }
            }
         ],
         "ProductListingIds": [ "string" ],
         "SubDomain": "string",
         "WorkteamArn": "string",
         "WorkteamName": "string"
      }
   ]
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**NextToken (p. 697)**

If the response is truncated, Amazon SageMaker returns this token. To retrieve the next set of work teams, use it in the subsequent request.

Type: String
Length Constraints: Maximum length of 8192.

**Workteams (p. 697)**

An array of Workteam objects, each describing a work team.

Type: Array of Workteam (p. 914) objects
Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
RenderUiTemplate
Service: Amazon SageMaker Service

Renders the UI template so that you can preview the worker's experience.

Request Syntax

```
{
    "RoleArn": "string",
    "Task": {
        "Input": "string"
    },
    "UiTemplate": {
        "Content": "string"
    }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**RoleArn (p. 699)**

The Amazon Resource Name (ARN) that has access to the S3 objects that are used by the template.

Type: String


Pattern: ^arn:aws[a-zA-Z\-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@-_/]+$

Required: Yes

**Task (p. 699)**

A `RenderableTask` object containing a representative task to render.

Type: `RenderableTask` (p. 864) object

Required: Yes

**UiTemplate (p. 699)**

A `Template` object containing the worker UI template to render.

Type: `UiTemplate` (p. 911) object

Required: Yes

Response Syntax

```
{
    "Errors": [
        {
            "Code": "string",
            "Message": "string"
        }
    ]
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

Errors (p. 699)

A list of one or more RenderingError objects if any were encountered while rendering the template. If there were no errors, the list is empty.

Type: Array of RenderingError (p. 865) objects

RenderedContent (p. 699)

A Liquid template that renders the HTML for the worker UI.

Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
Search
Service: Amazon SageMaker Service

Finds Amazon SageMaker resources that match a search query. Matching resource objects are returned as a list of SearchResult objects in the response. You can sort the search results by any resource property in an ascending or descending order.

You can query against the following value types: numerical, text, Booleans, and timestamps.

Request Syntax

```
{
  "MaxResults": number,
  "NextToken": "string",
  "Resource": "string",
  "SearchExpression": {
    "Filters": [{
      "Name": "string",
      "Operator": "string",
      "Value": "string"
    }
  ],
  "NestedFilters": [{
    "Filters": [{
      "Name": "string",
      "Operator": "string",
      "Value": "string"
    }
  ],
  "NestedPropertyName": "string"
  }
},
"Operator": "string",
"SubExpressions": [
  "SearchExpression"
],
"SortBy": "string",
"SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

MaxResults (p. 701)

The maximum number of results to return in a SearchResponse.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No
NextToken (p. 701)

If more than MaxResults resource objects match the specified SearchExpression, the SearchResponse includes a NextToken. The NextToken can be passed to the next SearchRequest to continue retrieving results for the specified SearchExpression and Sort parameters.

Type: String

Length Constraints: Maximum length of 8192.

Required: No

Resource (p. 701)

The name of the Amazon SageMaker resource to search for. Currently, the only valid Resource value is TrainingJob.

Type: String

Valid Values: TrainingJob

Required: Yes

SearchExpression (p. 701)

A Boolean conditional statement. Resource objects must satisfy this condition to be included in search results. You must provide at least one subexpression, filter, or nested filter. The maximum number of recursive SubExpressions, NestedFilters, and Filters that can be included in a SearchExpression object is 50.

Type: SearchExpression (p. 871) object

Required: No

SortBy (p. 701)

The name of the resource property used to sort the SearchResults. The default is LastModifiedTime.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Required: No

SortOrder (p. 701)

How SearchResults are ordered. Valid values are Ascending or Descending. The default is Descending.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```json
{
    "NextToken": "string",
    "Results": [
        
```
"TrainingJob": {  
  "AlgorithmSpecification": {  
    "AlgorithmName": "string",  
    "MetricDefinitions": [  
      {  
        "Name": "string",  
        "Regex": "string"  
      }  
    ],  
    "TrainingImage": "string",  
    "TrainingInputMode": "string"  
  },  
  "CreationTime": number,  
  "EnableNetworkIsolation": boolean,  
  "FailureReason": "string",  
  "FinalMetricDataList": [  
    {  
      "MetricName": "string",  
      "Timestamp": number,  
      "Value": number  
    }  
  ],  
  "HyperParameters": {  
    "string" : "string"  
  },  
  "InputDataConfig": [  
    {  
      "ChannelName": "string",  
      "CompressionType": "string",  
      "ContentType": "string",  
      "DataSource": {  
        "$S3DataSource": {  
          "AttributeNames": [ "string" ],  
          "$S3DataDistributionType": "string",  
          "$S3DataType": "string",  
          "$S3Uri": "string"  
        }  
      },  
      "InputMode": "string",  
      "RecordWrapperType": "string",  
      "ShuffleConfig": {  
        "Seed": number  
      }  
    }  
  ],  
  "LabelingJobArn": "string",  
  "LastModifiedTime": number,  
  "ModelArtifacts": {  
    "$S3ModelArtifacts": "string"  
  },  
  "OutputDataConfig": {  
    "KmsKeyId": "string",  
    "$S3OutputPath": "string"  
  },  
  "ResourceConfig": {  
    "InstanceCount": number,  
    "InstanceType": "string",  
    "VolumeKmsKeyId": "string",  
    "VolumeSizeInGB": number  
  },  
  "RoleArn": "string",  
  "SecondaryStatus": "string",  
  "SecondaryStatusTransitions": [  
    {  
      "EndTime": number,  
      "StartTime": number,  
      "Status": "string"  
    }  
  ]  
}
"Status": "string",
"StatusMessage": "string"
},
"StoppingCondition": {
"MaxRuntimeInSeconds": number
},
"Tags": [
{
"Key": "string",
"Value": "string"
}
],
"TrainingEndTime": number,
"TrainingJobArn": "string",
"TrainingJobName": "string",
"TrainingJobStatus": "string",
"TrainingStart_time": number,
"TuningJobArn": "string",
"VpcConfig": {
"SecurityGroupIds": [ "string" ],
"Subnets": [ "string" ]
}
]
]

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

NextToken (p. 702)

If the result of the previous Search request was truncated, the response includes a NextToken. To retrieve the next set of results, use the token in the next request.

Type: String

Length Constraints: Maximum length of 8192.

Results (p. 702)

A list of SearchResult objects.

Type: Array of SearchRecord (p. 873) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StartNotebookInstance
Service: Amazon SageMaker Service

Launches an ML compute instance with the latest version of the libraries and attaches your ML storage volume. After configuring the notebook instance, Amazon SageMaker sets the notebook instance status to InService. A notebook instance’s status must be InService before you can connect to your Jupyter notebook.

Request Syntax

```
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 706)**

The name of the notebook instance to start.

- Type: String
- Length Constraints: Maximum length of 63.
- Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9].*
- Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
**StopCompilationJob**  
Service: Amazon SageMaker Service

Stops a model compilation job.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal. This gracefully shuts the job down. If the job hasn't stopped, it sends the SIGKILL signal.

When it receives a `StopCompilationJob` request, Amazon SageMaker changes the `CompilationJobSummary:CompilationJobStatus (p. 763)` of the job to `Stopping`. After Amazon SageMaker stops the job, it sets the `CompilationJobSummary:CompilationJobStatus (p. 763)` to `Stopped`.

**Request Syntax**

```json
{
    "CompilationJobName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters (p. 917)].

The request accepts the following data in JSON format.

**CompilationJobName (p. 708)**

The name of the model compilation job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 915)].

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StopHyperParameterTuningJob
Service: Amazon SageMaker Service

Stops a running hyperparameter tuning job and all running training jobs that the tuning job launched.

All model artifacts output from the training jobs are stored in Amazon Simple Storage Service (Amazon S3). All data that the training jobs write to Amazon CloudWatch Logs are still available in CloudWatch. After the tuning job moves to the Stopped state, it releases all reserved resources for the tuning job.

Request Syntax

```
{
    "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**HyperParameterTuningJobName (p. 710)**

The name of the tuning job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceNotFoundException**

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**StopLabelingJob**
Service: Amazon SageMaker Service

Stops a running labeling job. A job that is stopped cannot be restarted. Any results obtained before the job is stopped are placed in the Amazon S3 output bucket.

**Request Syntax**

```
{
    "LabelingJobName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**LabelingJobName** (p. 712)

The name of the labeling job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceNotFound**

Resource being access is not found.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StopNotebookInstance
Service: Amazon SageMaker Service

Terminates the ML compute instance. Before terminating the instance, Amazon SageMaker disconnects the ML storage volume from it. Amazon SageMaker preserves the ML storage volume.

To access data on the ML storage volume for a notebook instance that has been terminated, call the StartNotebookInstance API. StartNotebookInstance launches another ML compute instance, configures it, and attaches the preserved ML storage volume so you can continue your work.

Request Syntax

```json
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 714)**

The name of the notebook instance to terminate.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^(a-zA-Z0-9)(-*[a-zA-Z0-9])*  

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
StopTrainingJob
Service: Amazon SageMaker Service

Stops a training job. To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts, so the results of the training is not lost.

Training algorithms provided by Amazon SageMaker save the intermediate results of a model training job. This intermediate data is a valid model artifact. You can use the model artifacts that are saved when Amazon SageMaker stops a training job to create a model.

When it receives a StopTrainingJob request, Amazon SageMaker changes the status of the job to Stopping. After Amazon SageMaker stops the job, it sets the status to Stopped.

Request Syntax

```
{
  "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

TrainingJobName (p. 716)

The name of the training job to stop.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`  

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFound

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS Command Line Interface
• AWS SDK for .NET
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for JavaScript
• AWS SDK for PHP V3
• AWS SDK for Python
• AWS SDK for Ruby V2
StopTransformJob
Service: Amazon SageMaker Service

Stops a transform job.

When Amazon SageMaker receives a StopTransformJob request, the status of the job changes to Stopping. After Amazon SageMaker stops the job, the status is set to Stopped. When you stop a transform job before it is completed, Amazon SageMaker doesn't store the job's output in Amazon S3.

Request Syntax

```
{
  "TransformJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

TransformJobName (p. 718)

The name of the transform job to stop.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceNotFoundException

Resource being access is not found.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateCodeRepository
Service: Amazon SageMaker Service

Updates the specified Git repository with the specified values.

Request Syntax

```json
{
   "CodeRepositoryName": "string",
   "GitConfig": {
      "SecretArn": "string"
   }
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**CodeRepositoryName (p. 720)**

The name of the Git repository to update.

Type: String


Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*$`

Required: Yes

**GitConfig (p. 720)**

The configuration of the git repository, including the URL and the Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the repository. The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```json
{"username": UserName, "password": Password}
```

Type: GitConfigForUpdate (p. 785) object

Required: No

Response Syntax

```json
{
   "CodeRepositoryArn": "string"
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**CodeRepositoryArn (p. 720)**

The ARN of the Git repository.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:code-repository/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**UpdateEndpoint**  
Service: Amazon SageMaker Service

Deploys the new EndpointConfig specified in the request, switches to using newly created endpoint, and then deletes resources provisioned for the endpoint using the previous EndpointConfig (there is no availability loss).

When Amazon SageMaker receives the request, it sets the endpoint status to **Updating**. After updating the endpoint, it sets the status to **InService**. To check the status of an endpoint, use the DescribeEndpoint API.

**Note**  
You cannot update an endpoint with the current EndpointConfig. To update an endpoint, you must create a new EndpointConfig.

**Request Syntax**

```json
{
    "EndpointConfigName": "string",
    "EndpointName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 722)**

The name of the new endpoint configuration.

Type: String  
Length Constraints: Maximum length of 63.  
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

**EndpointName (p. 722)**

The name of the endpoint whose configuration you want to update.

Type: String  
Length Constraints: Maximum length of 63.  
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

**Response Syntax**

```json
{
    "EndpointArn": "string"
}
```
Response Elements

If the action is successful, the service sends back an HTTP 200 response. The following data is returned in JSON format by the service.

**EndpointArn (p. 722)**

- The Amazon Resource Name (ARN) of the endpoint.
- Type: String

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

**ResourceLimitExceeded**

- You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.
- HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateEndpointWeightsAndCapacities

Service: Amazon SageMaker Service

Updates variant weight of one or more variants associated with an existing endpoint, or capacity of one variant associated with an existing endpoint. When it receives the request, Amazon SageMaker sets the endpoint status to `Updating`. After updating the endpoint, it sets the status to `InService`. To check the status of an endpoint, use the `DescribeEndpoint` API.

**Request Syntax**

```json
{
   "DesiredWeightsAndCapacities": [
      {
         "DesiredInstanceCount": number,
         "DesiredWeight": number,
         "VariantName": "string"
      }
   ],
   "EndpointName": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**DesiredWeightsAndCapacities (p. 724)**

An object that provides new capacity and weight values for a variant.

Type: Array of DesiredWeightAndCapacity (p. 776) objects

Array Members: Minimum number of 1 item.

Required: Yes

**EndpointName (p. 724)**

The name of an existing Amazon SageMaker endpoint.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**Response Syntax**

```json
{
   "EndpointArn": "string"
}
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.
The following data is returned in JSON format by the service.

**EndpointArn (p. 724)**

The Amazon Resource Name (ARN) of the updated endpoint.

Type: String


**Errors**

For information about the errors that are common to all actions, see [Common Errors (p. 915)](https://docs.aws.amazon.com/sagemaker/latest/dg/errors.html).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateNotebookInstance
Service: Amazon SageMaker Service

Updates a notebook instance. NotebookInstance updates include upgrading or downgrading the ML compute instance used for your notebook instance to accommodate changes in your workload requirements. You can also update the VPC security groups.

Request Syntax

```json
{
   "AcceleratorTypes": [ "string" ],
   "AdditionalCodeRepositories": [ "string" ],
   "DefaultCodeRepository": "string",
   "DisassociateAcceleratorTypes": boolean,
   "DisassociateAdditionalCodeRepositories": boolean,
   "DisassociateDefaultCodeRepository": boolean,
   "DisassociateLifecycleConfig": boolean,
   "InstanceType": "string",
   "LifecycleConfigName": "string",
   "NotebookInstanceName": "string",
   "RoleArn": "string",
   "VolumeSizeInGB": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**AcceleratorTypes (p. 726)**

A list of the Elastic Inference (EI) instance types to associate with this notebook instance. Currently only one EI instance type can be associated with a notebook instance. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: Array of strings

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge

Required: No

**AdditionalCodeRepositories (p. 726)**

An array of up to three Git repositories to associate with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://([^/]+)/?(.*)$|^\[a-zA-Z0-9\](-*[a-zA-Z0-9])*  

Required: No
DefaultCodeRepository (p. 726)

The Git repository to associate with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String
Pattern: ^https://([^/]+)/?(.*)$|^^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*
Required: No

DisassociateAcceleratorTypes (p. 726)

A list of the Elastic Inference (EI) instance types to remove from this notebook instance.

Type: Boolean
Required: No

DisassociateAdditionalCodeRepositories (p. 726)

A list of names or URLs of the default Git repositories to remove from this notebook instance.

Type: Boolean
Required: No

DisassociateDefaultCodeRepository (p. 726)

The name or URL of the default Git repository to remove from this notebook instance.

Type: Boolean
Required: No

DisassociateLifecycleConfig (p. 726)

Set to true to remove the notebook instance lifecycle configuration currently associated with the notebook instance.

Type: Boolean
Required: No

InstanceType (p. 726)

The Amazon ML compute instance type.

Type: String
Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.16xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge | ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.xlarge | ml.p3.8xlarge | ml.p3.16xlarge
**LifecycleConfigName (p. 726)**

The name of a lifecycle configuration to associate with the notebook instance. For information about lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9\-\[\]_]{1,63}$`

Required: No

**NotebookInstanceName (p. 726)**

The name of the notebook instance to update.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9\-\[\]_]{1,63}$`

Required: Yes

**RoleArn (p. 726)**

The Amazon Resource Name (ARN) of the IAM role that Amazon SageMaker can assume to access the notebook instance. For more information, see Amazon SageMaker Roles.

Note

To be able to pass this role to Amazon SageMaker, the caller of this API must have the `iam:PassRole` permission.

Type: String


Pattern: `^arn:aws[a-zA-Z\-]*:iam::\d{12}:role/?.*\[a-zA-Z0-9+=,.@\-_/\]+$`

Required: No

**VolumeSizeInGB (p. 726)**

The size, in GB, of the ML storage volume to attach to the notebook instance. The default value is 5 GB.

Type: Integer


Required: No

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 915).
ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
**UpdateNotebookInstanceLifecycleConfig**

Service: Amazon SageMaker Service

Updates a notebook instance lifecycle configuration created with the `CreateNotebookInstanceLifecycleConfig` API.

**Request Syntax**

```json
{
    "NotebookInstanceLifecycleConfigName": "string",
    "OnCreate": [
        {
            "Content": "string"
        }
    ],
    "OnStart": [
        {
            "Content": "string"
        }
    ]
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters](#).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 730)**

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**OnCreate (p. 730)**

The shell script that runs only once, when you create a notebook instance

Type: Array of `NotebookInstanceLifecycleHook` objects

Array Members: Maximum number of 1 item.

Required: No

**OnStart (p. 730)**

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of `NotebookInstanceLifecycleHook` objects

Array Members: Maximum number of 1 item.

Required: No
Response Elements

If the action is successful, the service sends back an HTTP 200 response with an empty HTTP body.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 915).

ResourceLimitExceeded

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
UpdateWorkteam
Service: Amazon SageMaker Service

Updates an existing work team with new member definitions or description.

Request Syntax

```json
{
   "Description": "string",
   "MemberDefinitions": [
      { "CognitoMemberDefinition": {
         "ClientId": "string",
         "UserGroup": "string",
         "UserPool": "string"
      }
   ],
   "WorkteamName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 917).

The request accepts the following data in JSON format.

**Description (p. 732)**

An updated description for the work team.

Type: String


Required: No

**MemberDefinitions (p. 732)**

A list of MemberDefinition objects that contain the updated work team members.

Type: Array of MemberDefinition (p. 830) objects

Array Members: Minimum number of 1 item. Maximum number of 10 items.

Required: No

**WorkteamName (p. 732)**

The name of the work team to update.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes
Response Syntax

```json
{
    "Workteam": {
        "CreateDate": number,
        "Description": "string",
        "LastUpdatedDate": number,
        "MemberDefinitions": [
            {
                "CognitoMemberDefinition": {
                    "ClientId": "string",
                    "UserGroup": "string",
                    "UserPool": "string"
                }
            }
        ],
        "ProductListingIds": [ "string" ],
        "SubDomain": "string",
        "WorkteamArn": "string",
        "WorkteamName": "string"
    }
}
```

Response Elements

If the action is successful, the service sends back an HTTP 200 response.

The following data is returned in JSON format by the service.

**Workteam** (p. 733)

A *Workteam* object that describes the updated work team.

Type: *Workteam* (p. 914) object

**Errors**

For information about the errors that are common to all actions, see *Common Errors* (p. 915).

**ResourceLimitExceeded**

You have exceeded an Amazon SageMaker resource limit. For example, you might have too many training jobs created.

HTTP Status Code: 400

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
Amazon SageMaker Runtime

The following actions are supported by Amazon SageMaker Runtime:

- InvokeEndpoint (p. 735)
InvokeEndpoint

Service: Amazon SageMaker Runtime

After you deploy a model into production using Amazon SageMaker hosting services, your client applications use this API to get inferences from the model hosted at the specified endpoint.

For an overview of Amazon SageMaker, see How It Works.

Amazon SageMaker strips all POST headers except those supported by the API. Amazon SageMaker might add additional headers. You should not rely on the behavior of headers outside those enumerated in the request syntax.

Calls to InvokeEndpoint are authenticated by using AWS Signature Version 4. For information, see Authenticating Requests (AWS Signature Version 4) in the Amazon S3 API Reference.

A customer’s model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

Note

Endpoints are scoped to an individual account, and are not public. The URL does not contain the account ID, but Amazon SageMaker determines the account ID from the authentication token that is supplied by the caller.

Request Syntax

```
POST /endpoints/EndpointName/invocations HTTP/1.1
Content-Type: ContentType
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes

Body
```

URI Request Parameters

The request requires the following URI parameters.

Accept (p. 735)

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

ContentType (p. 735)

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

CustomAttributes (p. 735)

Provides additional information about a request for an inference submitted to a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to provide an ID that you can use to track a request or to provide other metadata that a service endpoint was programmed to process. The value must consist of no more than 1024 visible US-ASCII characters as specified in Section 3.3.6. Field Value Components of
the Hypertext Transfer Protocol (HTTP/1.1). This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

**EndpointName (p. 735)**

The name of the endpoint that you specified when you created the endpoint using the CreateEndpoint API.

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

**Request Body**

The request accepts the following binary data.

**Body (p. 735)**

Provides input data, in the format specified in the `ContentType` request header. Amazon SageMaker passes all of the data in the body to the model.

For information about the format of the request body, see [Common Data Formats—Inference](#).

Length Constraints: Maximum length of 5242880.

**Response Syntax**

```
HTTP/1.1 200
Content-Type: ContentType
x-Amzn-Invoked-Production-Variant: InvokedProductionVariant
x-Amzn-SageMaker-Custom-Attributes: CustomAttributes

Body
```

**Response Elements**

If the action is successful, the service sends back an HTTP 200 response.

The response returns the following HTTP headers.

**ContentType (p. 736)**

The MIME type of the inference returned in the response body.

Length Constraints: Maximum length of 1024.

Pattern: \p{ASCII}*

**CustomAttributes (p. 736)**

Provides additional information in the response about the inference returned by a model hosted at an Amazon SageMaker endpoint. The information is an opaque value that is forwarded verbatim. You could use this value, for example, to return an ID received in the CustomAttributes header of a request or other metadata that a service endpoint was programmed to produce. The value must consist of no more than 1024 visible US-ASCII characters as specified in [Section 3.3.6. Field Value](#).
Components of the Hypertext Transfer Protocol (HTTP/1.1). This feature is currently supported in the AWS SDKs but not in the Amazon SageMaker Python SDK.

Length Constraints: Maximum length of 1024.
Pattern: \p{ASCII}*  
InvokedProductionVariant (p. 736)  
Identifies the production variant that was invoked.
Length Constraints: Maximum length of 1024.
Pattern: \p{ASCII}*

The response returns the following as the HTTP body.

Body (p. 736)
Includes the inference provided by the model.
For information about the format of the response body, see Common Data Formats—Inference.
Length Constraints: Maximum length of 5242880.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 915).

InternalFailure
An internal failure occurred.
HTTP Status Code: 500

ModelError
Model (owned by the customer in the container) returned an error 500.
HTTP Status Code: 424

ServiceUnavailable
The service is unavailable. Try your call again.
HTTP Status Code: 503

ValidationError
Inspect your request and try again.
HTTP Status Code: 400

Example
Pass a trace ID in the CustomAttribute of a request and return it in the CustomAttribute of the response.

In this example a trace ID is passed to the service endpoint in the CustomAttributes header of the request and then retrieved and returned in the CustomAttributes header of the response.
Sample Request

```python
import boto3
client = boto3.client('sagemaker-runtime')
custom_attributes = "c000b4f9-df62-4c85-a0bf-7c525f9104a4"  # An example of a trace ID.
endpoint_name = "..."  # Your endpoint name.
content_type = "..."  # The MIME type of the input data in the request body.
accept = "..."  # The desired MIME type of the inference in the response.
payload = "..."  # Payload for inference.
```

Sample Response

```python
response = client.invoke_endpoint(
    EndpointName=endpoint_name,
   CustomAttributes=custom_attributes,
   ContentType=content_type,
   Accept=accept,
    Body=payload
)
print(response['CustomAttributes'])  # If model receives and updates the custom_attributes header
# by adding "Trace id: " in front of custom_attributes in the request,
# custom_attributes in response becomes "Trace ID: c000b4f9-
df62-4c85-a0bf-7c525f9104a4"
```

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2

Data Types

The following data types are supported by Amazon SageMaker Service:

- AlgorithmSpecification (p. 744)
- AlgorithmStatusDetails (p. 746)
• AlgorithmStatusItem (p. 747)
• AlgorithmSummary (p. 748)
• AlgorithmValidationProfile (p. 750)
• AlgorithmValidationSpecification (p. 751)
• AnnotationConsolidationConfig (p. 752)
• CategoricalParameterRange (p. 754)
• CategoricalParameterRangeSpecification (p. 755)
• Channel (p. 756)
• ChannelSpecification (p. 758)
• CodeRepositorySummary (p. 760)
• CognitoMemberDefinition (p. 762)
• CompilationJobSummary (p. 763)
• ContainerDefinition (p. 765)
• ContinuousParameterRange (p. 767)
• ContinuousParameterRangeSpecification (p. 768)
• DataSource (p. 769)
• DeployedImage (p. 770)
• DescribeTrainingJobResponse (p. 771)
• DesiredWeightAndCapacity (p. 776)
• EndpointConfigSummary (p. 777)
• EndpointSummary (p. 778)
• Filter (p. 780)
• FinalHyperParameterTuningJobObjectiveMetric (p. 783)
• GitConfig (p. 784)
• GitConfigForUpdate (p. 785)
• HumanTaskConfig (p. 786)
• HyperParameterAlgorithmSpecification (p. 789)
• HyperParameterSpecification (p. 791)
• HyperParameterTrainingJobDefinition (p. 793)
• HyperParameterTrainingJobSummary (p. 796)
• HyperParameterTuningJobConfig (p. 799)
• HyperParameterTuningJobObjective (p. 801)
• HyperParameterTuningJobSummary (p. 802)
• HyperParameterTuningJobWarmStartConfig (p. 804)
• InferenceSpecification (p. 806)
• InputConfig (p. 808)
• IntegerParameterRange (p. 810)
• IntegerParameterRangeSpecification (p. 811)
• LabelCounters (p. 812)
• LabelCountersForWorkteam (p. 814)
• LabelingJobAlgorithmsConfig (p. 815)
• LabelingJobDataAttributes (p. 817)
• LabelingJobDataSource (p. 818)
• LabelingJobForWorkteamSummary (p. 819)
• LabelingJobInputConfig (p. 821)
• LabelingJobOutput (p. 822)
• LabelingJobOutputConfig (p. 823)
• LabelingJobResourceConfig (p. 824)
• LabelingJobS3DataSource (p. 825)
• LabelingJobStoppingConditions (p. 826)
• LabelingJobSummary (p. 827)
• MemberDefinition (p. 830)
• MetricData (p. 831)
• MetricDefinition (p. 832)
• ModelArtifacts (p. 833)
• ModelPackageContainerDefinition (p. 834)
• ModelPackageStatusDetails (p. 836)
• ModelPackageStatusItem (p. 837)
• ModelPackageSummary (p. 838)
• ModelPackageValidationProfile (p. 840)
• ModelPackageValidationSpecification (p. 841)
• ModelSummary (p. 842)
• NestedFilters (p. 843)
• NotebookInstanceLifecycleConfigSummary (p. 844)
• NotebookInstanceLifecycleHook (p. 845)
• NotebookInstanceSummary (p. 846)
• ObjectiveStatusCounters (p. 849)
• OutputConfig (p. 850)
• OutputDataConfig (p. 851)
• ParameterRange (p. 853)
• ParameterRanges (p. 854)
• ParentHyperParameterTuningJob (p. 855)
• ProductionVariant (p. 856)
• ProductionVariantSummary (p. 858)
• PropertyNameQuery (p. 860)
• PropertyNameSuggestion (p. 861)
• PublicWorkforceTaskPrice (p. 862)
• RenderableTask (p. 864)
• RenderingError (p. 865)
• ResourceConfig (p. 866)
• ResourceLimits (p. 868)
• S3DataSource (p. 869)
• SearchExpression (p. 871)
• SearchRecord (p. 873)
• SecondaryStatusTransition (p. 874)
• ShuffleConfig (p. 876)
• SourceAlgorithm (p. 877)
• SourceAlgorithmSpecification (p. 878)
• StoppingCondition (p. 879)
• SubscribedWorkteam (p. 880)
• SuggestionQuery (p. 882)
• Tag (p. 883)
The following data types are supported by Amazon SageMaker Runtime:

Amazon SageMaker Service

The following data types are supported by Amazon SageMaker Service:

- AlgorithmSpecification (p. 744)
- AlgorithmStatusDetails (p. 746)
- AlgorithmStatusItem (p. 747)
- AlgorithmSummary (p. 748)
- AlgorithmValidationProfile (p. 750)
- AlgorithmValidationSpecification (p. 751)
- AnnotationConsolidationConfig (p. 752)
- CategoricalParameterRange (p. 754)
- CategoricalParameterRangeSpecification (p. 755)
- Channel (p. 756)
- ChannelSpecification (p. 758)
- CodeRepositorySummary (p. 760)
- CognitoMemberDefinition (p. 762)
- CompilationJobSummary (p. 763)
- ContainerDefinition (p. 765)
- ContinuousParameterRange (p. 767)
- ContinuousParameterRangeSpecification (p. 768)
- DataSource (p. 769)
- DeployedImage (p. 770)
- DescribeTrainingJobResponse (p. 771)
- DesiredWeightAndCapacity (p. 776)
- EndpointConfigSummary (p. 777)
• ObjectiveStatusCounters (p. 849)
• OutputConfig (p. 850)
• OutputDataConfig (p. 851)
• ParameterRange (p. 853)
• ParameterRanges (p. 854)
• ParentHyperParameterTuningJob (p. 855)
• ProductionVariant (p. 856)
• ProductionVariantSummary (p. 858)
• PropertyNameQuery (p. 860)
• PropertyNameSuggestion (p. 861)
• PublicWorkforceTaskPrice (p. 862)
• RenderableTask (p. 864)
• RenderingError (p. 865)
• ResourceConfig (p. 866)
• ResourceLimits (p. 868)
• S3DataSource (p. 869)
• SearchExpression (p. 871)
• SearchRecord (p. 873)
• SecondaryStatusTransition (p. 874)
• ShuffleConfig (p. 876)
• SourceAlgorithm (p. 877)
• SourceAlgorithmSpecification (p. 878)
• StoppingCondition (p. 879)
• SubscribedWorkteam (p. 880)
• SuggestionQuery (p. 882)
• Tag (p. 883)
• TrainingJob (p. 884)
• TrainingJobDefinition (p. 889)
• TrainingJobStatusCounters (p. 891)
• TrainingJobSummary (p. 893)
• TrainingSpecification (p. 895)
• TransformDataSource (p. 897)
• TransformInput (p. 898)
• TransformJobDefinition (p. 900)
• TransformJobSummary (p. 902)
• TransformOutput (p. 904)
• TransformResources (p. 906)
• TransformS3DataSource (p. 908)
• UiConfig (p. 910)
• UiTemplate (p. 911)
• USD (p. 912)
• VpcConfig (p. 913)
• Workteam (p. 914)
AlgorithmSpecification
Service: Amazon SageMaker Service

Specifies the training algorithm to use in a CreateTrainingJob request.

For more information about algorithms provided by Amazon SageMaker, see Algorithms. For information about using your own algorithms, see Using Your Own Algorithms with Amazon SageMaker.

Contents

AlgorithmName

The name of the algorithm resource to use for the training job. This must be an algorithm resource that you created or subscribe to on AWS Marketplace. If you specify a value for this parameter, you can't specify a value for TrainingImage.

Type: String


Pattern: (arn:aws[\w\-]*:sagemaker:[\w\-]*:[0-9]{12}:\[\w\-]*\//)?([\w\-]{0,62})(?!-)$

Required: No

MetricDefinitions

A list of metric definition objects. Each object specifies the metric name and regular expressions used to parse algorithm logs. Amazon SageMaker publishes each metric to Amazon CloudWatch.

Type: Array of MetricDefinition (p. 832) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters.

Type: String

Length Constraints: Maximum length of 255.

Required: No

TrainingInputMode

The input mode that the algorithm supports. For the input modes that Amazon SageMaker algorithms support, see Algorithms. If an algorithm supports the File input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the Pipe input mode, Amazon SageMaker streams data directly from S3 to the container.

In File mode, make sure you provision ML storage volume with sufficient capacity to accommodate the data download from S3. In addition to the training data, the ML storage volume also stores the output model. The algorithm container use ML storage volume to also store intermediate information, if any.

For distributed algorithms using File mode, training data is distributed uniformly, and your training duration is predictable if the input data objects size is approximately same. Amazon SageMaker does
not split the files any further for model training. If the object sizes are skewed, training won't be optimal as the data distribution is also skewed where one host in a training cluster is overloaded, thus becoming bottleneck in training.

Type: String

Valid Values: Pipe | File

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmStatusDetails
Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the algorithm.

Contents

ImageScanStatuses
The status of the scan of the algorithm's Docker image container.
Type: Array of AlgorithmStatusItem (p. 747) objects
Required: No

ValidationStatuses
The status of algorithm validation.
Type: Array of AlgorithmStatusItem (p. 747) objects
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmStatusItem
Service: Amazon SageMaker Service

Represents the overall status of an algorithm.

Contents

FailureReason

if the overall status is Failed, the reason for the failure.

Type: String
Required: No

Name

The name of the algorithm for which the overall status is being reported.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
Required: Yes

Status

The current status.

Type: String
Valid Values: NotStarted | InProgress | Completed | Failed
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmSummary
Service: Amazon SageMaker Service

Provides summary information about an algorithm.

Contents

AlgorithmArn
The Amazon Resource Name (ARN) of the algorithm.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z-]*:sagemaker:[a-z0-9-]*:[0-9]{12}:algorithm/.*
Required: Yes

AlgorithmDescription
A brief description of the algorithm.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: \[\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}\]*
Required: No

AlgorithmException
The name of the algorithm that is described by the summary.
Type: String
Pattern: ^[a-zA-Z0-9](\-[a-zA-Z0-9]*)*$
Required: Yes

CreateTime
A timestamp that shows when the algorithm was created.
Type: Timestamp
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
**AlgorithmValidationProfile**

Service: Amazon SageMaker Service

Defines a training job and a batch transform job that Amazon SageMaker runs to validate your algorithm.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

**Contents**

**ProfileName**

The name of the profile for the algorithm. The name must have 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

**TrainingJobDefinition**

The `TrainingJobDefinition` object that describes the training job that Amazon SageMaker runs to validate your algorithm.

Type: `TrainingJobDefinition (p. 889)` object

Required: Yes

**TransformJobDefinition**

The `TransformJobDefinition` object that describes the transform job that Amazon SageMaker runs to validate your algorithm.

Type: `TransformJobDefinition (p. 900)` object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AlgorithmValidationSpecification
Service: Amazon SageMaker Service

Specifies configurations for one or more training jobs that Amazon SageMaker runs to test the algorithm.

Contents

ValidationProfiles

An array of AlgorithmValidationProfile objects, each of which specifies a training job and batch transform job that Amazon SageMaker runs to validate your algorithm.

Type: Array of AlgorithmValidationProfile (p. 750) objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole

The IAM roles that Amazon SageMaker uses to run the training jobs.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z0-9=+.@\-_/]+$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
AnnotationConsolidationConfig
Service: Amazon SageMaker Service

Configures how labels are consolidated across human workers.

Contents

AnnotationConsolidationLambdaArn

The Amazon Resource Name (ARN) of a Lambda function implements the logic for annotation consolidation.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

- **Bounding box** - Finds the most similar boxes from different workers based on the Jaccard index of the boxes.

  arn:aws:lambda:us-east-1:432418664414:function:ACS-BoundingBox
  arn:aws:lambda:us-east-2:266458841044:function:ACS-BoundingBox
  arn:aws:lambda:us-west-2:081040173940:function:ACS-BoundingBox
  arn:aws:lambda:eu-west-1:568282634449:function:ACS-BoundingBox
  arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-BoundingBox

- **Image classification** - Uses a variant of the Expectation Maximization approach to estimate the true class of an image based on annotations from individual workers.

  arn:aws:lambda:us-east-1:432418664414:function:ACS-ImageMultiClass
  arn:aws:lambda:eu-west-1:568282634449:function:ACS-ImageMultiClass
  arn:aws:lambda:ap-northeast-1:477331159723:function:ACS-ImageMultiClass

- **Semantic segmentation** - Treats each pixel in an image as a multi-class classification and treats pixel annotations from workers as "votes" for the correct label.

  arn:aws:lambda:eu-west-1:568282634449:function:ACS-SemanticSegmentation

- **Text classification** - Uses a variant of the Expectation Maximization approach to estimate the true class of text based on annotations from individual workers.

  arn:aws:lambda:us-east-1:432418664414:function:ACS-TextMultiClass
For more information, see *Annotation Consolidation*.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: `arn:aws[a-z\-]*:lambda:[a-z]{2}-[a-z]+\-\d{1}:\d{12}:function:[a-zA-Z0-9-\_\-\.]\+(\$LATEST|[a-zA-Z0-9-\_]+)\?`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**CategoricalParameterRange**  
Service: Amazon SageMaker Service

A list of categorical hyperparameters to tune.

**Contents**

**Name**

The name of the categorical hyperparameter to tune.

Type: String  
Length Constraints: Maximum length of 256.  
Required: Yes

**Values**

A list of the categories for the hyperparameter.

Type: Array of strings  
Array Members: Minimum number of 1 item. Maximum number of 20 items.  
Length Constraints: Maximum length of 256.  
Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CategoricalParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for a categorical hyperparameter.

Contents

Values

The allowed categories for the hyperparameter.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Length Constraints: Maximum length of 256.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**Channel**
Service: Amazon SageMaker Service

A channel is a named input source that training algorithms can consume.

**Contents**

**ChannelName**
The name of the channel.
Type: String

Length Constraints: Minimum length of 1. Maximum length of 64.

Pattern: [A-Za-z0-9.\-_]+

Required: Yes

**CompressionType**
If training data is compressed, the compression type. The default value is None. CompressionType is used only in Pipe input mode. In File mode, leave this field unset or set it to None.

Type: String

Valid Values: None | Gzip

Required: No

**ContentType**
The MIME type of the data.
Type: String

Length Constraints: Maximum length of 256.

Required: No

**DataSource**
The location of the channel data.

Type: DataSource (p. 769) object

Required: Yes

**InputMode**
(Optional) The input mode to use for the data channel in a training job. If you don't set a value for InputMode, Amazon SageMaker uses the value set for TrainingInputMode. Use this parameter to override the TrainingInputMode setting in a AlgorithmSpecification (p. 744) request when you have a channel that needs a different input mode from the training job's general setting. To download the data from Amazon Simple Storage Service (Amazon S3) to the provisioned ML storage volume, and mount the directory to a Docker volume, use File input mode. To stream data directly from Amazon S3 to the container, choose Pipe input mode.

To use a model for incremental training, choose File input model.

Type: String

Valid Values: Pipe | File
RecordWrapperType

Specify RecordIO as the value when input data is in raw format but the training algorithm requires the RecordIO format. In this case, Amazon SageMaker wraps each individual S3 object in a RecordIO record. If the input data is already in RecordIO format, you don't need to set this attribute. For more information, see Create a Dataset Using RecordIO.

In File mode, leave this field unset or set it to None.

Type: String

Valid Values: None | RecordIO

Required: No

ShuffleConfig

A configuration for a shuffle option for input data in a channel. If you use S3Prefix for S3DataType, this shuffles the results of the S3 key prefix matches. If you use ManifestFile, the order of the S3 object references in the ManifestFile is shuffled. If you use AugmentedManifestFile, the order of the JSON lines in the AugmentedManifestFile is shuffled. The shuffling order is determined using the Seed value.

For Pipe input mode, shuffling is done at the start of every epoch. With large datasets this ensures that the order of the training data is different for each epoch, it helps reduce bias and possible overfitting. In a multi-node training job when ShuffleConfig is combined with S3DataDistributionType of ShardedByS3Key, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

Type: ShuffleConfig (p. 876) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ChannelSpecification
Service: Amazon SageMaker Service
Defines a named input source, called a channel, to be used by an algorithm.

Contents

Description
A brief description of the channel.
Type: String
Length Constraints: Maximum length of 1024.
Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*
Required: No

IsRequired
Indicates whether the channel is required by the algorithm.
Type: Boolean
Required: No

Name
The name of the channel.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 64.
Pattern: [A-Za-z0-9\.-_]+
Required: Yes

SupportedCompressionTypes
The allowed compression types, if data compression is used.
Type: Array of strings
Valid Values: None | Gzip
Required: No

SupportedContentTypes
The supported MIME types for the data.
Type: Array of strings
Length Constraints: Maximum length of 256.
Required: Yes

SupportedInputModes
The allowed input mode, either FILE or PIPE.
In FILE mode, Amazon SageMaker copies the data from the input source onto the local Amazon Elastic Block Store (Amazon EBS) volumes before starting your training algorithm. This is the most commonly used input mode.
In PIPE mode, Amazon SageMaker streams input data from the source directly to your algorithm without using the EBS volume.

Type: Array of strings

Array Members: Minimum number of 1 item.

Valid Values: Pipe | File

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
## CodeRepositorySummary

Service: Amazon SageMaker Service

Specifies summary information about a Git repository.

### Contents

#### CodeRepositoryArn

The Amazon Resource Name (ARN) of the Git repository.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:code-repository/.*`

Required: Yes

#### CodeRepositoryName

The name of the Git repository.

Type: String


Pattern: `^[a-zA-Z0-9\-\(\)\[\]\{\}\*\+\-\\.]$`

Required: Yes

#### CreationTime

The date and time that the Git repository was created.

Type: Timestamp

Required: Yes

#### GitConfig

Configuration details for the Git repository, including the URL where it is located and the ARN of the AWS Secrets Manager secret that contains the credentials used to access the repository.

Type: `GitConfig` object

Required: No

#### LastModifiedTime

The date and time that the Git repository was last modified.

Type: Timestamp

Required: Yes

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
CognitoMemberDefinition
Service: Amazon SageMaker Service

Identifies a Amazon Cognito user group. A user group can be used in on or more work teams.

Contents

ClientId

An identifier for an application client. You must create the app client ID using Amazon Cognito.

Type: String


Pattern: [ \w+ ]+

Required: Yes

UserGroup

An identifier for a user group.

Type: String


Pattern: [ \p{L}\p{M}\p{S}\p{N}\p{P} ]+

Required: Yes

UserPool

An identifier for a user pool. The user pool must be in the same region as the service that you are calling.

Type: String


Pattern: [ \w-]+[0-9a-zA-Z]+

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
CompilationJobSummary
Service: Amazon SageMaker Service

A summary of a model compilation job.

Contents

CompilationEndTime

The time when the model compilation job completed.
Type: Timestamp
Required: No

CompilationJobArn

The Amazon Resource Name (ARN) of the model compilation job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:compilation-job/.*
Required: Yes

CompilationJobName

The name of the model compilation job that you want a summary for.
Type: String
Pattern: ^[a-zA-Z0-9\-]*[a-zA-Z0-9]*$
Required: Yes

CompilationJobStatus

The status of the model compilation job.
Type: String
Valid Values: INPROGRESS | COMPLETED | FAILED | STARTING | STOPPING | STOPPED
Required: Yes

CompilationStartTime

The time when the model compilation job started.
Type: Timestamp
Required: No

CompilationTargetDevice

The type of device that the model will run on after compilation has completed.
Type: String
Valid Values: ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | jetson_tx1 | jetson_tx2 | rasp3b | deep lens
Required: Yes

**CreationTime**

The time when the model compilation job was created.

Type: Timestamp

Required: Yes

**LastModifiedTime**

The time when the model compilation job was last modified.

Type: Timestamp

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContainerDefinition
Service: Amazon SageMaker Service

Describes the container, as part of model definition.

Contents

ContainerHostname

This parameter is ignored.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: No

Environment

The environment variables to set in the Docker container. Each key and value in the Environment string to string map can have length of up to 1024. We support up to 16 entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 1024.

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored. If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both registry/repository[:tag] and registry/repository[@digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker

Type: String

Length Constraints: Maximum length of 255.

Pattern: [:S]+

Required: No

ModelDataUrl

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

If you provide a value for this parameter, Amazon SageMaker uses AWS Security Token Service to download model artifacts from the S3 path you provide. AWS STS is activated in your IAM user account by default. If you previously deactivated AWS STS for a region, you need to reactivate AWS STS for that region. For more information, see Activating and Deactivating AWS STS in an AWS Region in the AWS Identity and Access Management User Guide.

Type: String
Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/*(.*)$

Required: No

**ModelPackageName**

The name of the model package to use to create the model.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-\/]*/)?([a-zA-Z0-9](\[a-zA-Z0-9-\]){0,62})(?<!-)$

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContinuousParameterRange
Service: Amazon SageMaker Service

A list of continuous hyperparameters to tune.

Contents

MaxValue

The maximum value for the hyperparameter. The tuning job uses floating-point values between MinValue value and this value for tuning.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

MinValue

The minimum value for the hyperparameter. The tuning job uses floating-point values between this value and MaxValue for tuning.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Name

The name of the continuous hyperparameter to tune.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ContinuousParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for a continuous hyperparameter.

Contents

MaxValue

The maximum floating-point value allowed.

Type: String
Length Constraints: Maximum length of 256.
Required: Yes

MinValue

The minimum floating-point value allowed.

Type: String
Length Constraints: Maximum length of 256.
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DataSource
Service: Amazon SageMaker Service

Describes the location of the channel data.

Contents

S3DataSource

The S3 location of the data source that is associated with a channel.

Type: S3DataSource (p. 869) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
### DeployedImage

**Service:** Amazon SageMaker Service

Gets the Amazon EC2 Container Registry path of the docker image of the model that is hosted in this `ProductionVariant` (p. 856).

If you used the `registry/repository[:tag]` form to specify the image path of the primary container when you created the model hosted in this `ProductionVariant`, the path resolves to a path of the form `registry/repository[@digest]`. A digest is a hash value that identifies a specific version of an image. For information about Amazon ECR paths, see Pulling an Image in the Amazon ECR User Guide.

#### Contents

**ResolutionTime**

The date and time when the image path for the model resolved to the `ResolvedImage`

Type: `Timestamp`

Required: No

**ResolvedImage**

The specific digest path of the image hosted in this `ProductionVariant`.

Type: `String`

Length Constraints: Maximum length of 255.

Pattern: `[^\S]+`

Required: No

**SpecifiedImage**

The image path you specified when you created the model.

Type: `String`

Length Constraints: Maximum length of 255.

Pattern: `[^\S]+`

Required: No

### See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DescribeTrainingJobResponse
Service: Amazon SageMaker Service

Contents

AlgorithmSpecification
Information about the algorithm used for training, and algorithm metadata.
Type: AlgorithmSpecification (p. 744) object
Required: Yes

CreationTime
A timestamp that indicates when the training job was created.
Type: Timestamp
Required: Yes

EnableInterContainerTrafficEncryption
To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.
Type: Boolean
Required: No

EnableNetworkIsolation
If you want to allow inbound or outbound network calls, except for calls between peers within a training cluster for distributed training, choose True. If you enable network isolation for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note
The Semantic Segmentation built-in algorithm does not support network isolation.
Type: Boolean
Required: No

FailureReason
If the training job failed, the reason it failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

FinalMetricDataList
A collection of MetricData objects that specify the names, values, and dates and times that the training algorithm emitted to Amazon CloudWatch.
Type: Array of MetricData (p. 831) objects
Array Members: Minimum number of 0 items. Maximum number of 20 items.
**Required: No**

**HyperParameters**

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

**Required: No**

**InputDataConfig**

An array of Channel objects that describes each data input channel.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

**Required: No**

**LabelingJobArn**

The Amazon Resource Name (ARN) of the Amazon SageMaker Ground Truth labeling job that created the transform or training job.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-\:]*:0-9\{12\}:labeling-job/.*

**Required: No**

**LastModifiedTime**

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

**Required: No**

**ModelArtifacts**

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 833) object

**Required: Yes**

**OutputDataConfig**

The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.

Type: OutputDataConfig (p. 851) object

**Required: No**

**ResourceConfig**

Resources, including ML compute instances and ML storage volumes, that are configured for model training.
Type: *ResourceConfig (p. 866)* object

Required: Yes

**RoleArn**

The AWS Identity and Access Management (IAM) role configured for the training job.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$

Required: No

**SecondaryStatus**

Provides detailed information about the state of the training job. For detailed information on the secondary status of the training job, see *StatusMessage* under *SecondaryStatusTransition (p. 874).*

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

**InProgress**

- **Starting** - Starting the training job.
- **Downloading** - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
- **Training** - Training is in progress.
- **Uploading** - Training is complete and the model artifacts are being uploaded to the S3 location.

**Completed**

- **Completed** - The training job has completed.

**Failed**

- **Failed** - The training job has failed. The reason for the failure is returned in the *FailureReason* field of *DescribeTrainingJobResponse*.

**Stopped**

- **MaxRuntimeExceeded** - The job stopped because it exceeded the maximum allowed runtime.
- **Stopped** - The training job has stopped.

**Stopping**

- **Stopping** - Stopping the training job.

**Important**

Valid values for *SecondaryStatus* are subject to change.

We no longer support the following secondary statuses:

- **LaunchingMLInstances**
- **PreparingTrainingStack**
- **DownloadingTrainingImage**

Type: String

Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed

Required: Yes
SecondaryStatusTransitions

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 874) objects

Required: No

StoppingCondition

The condition under which to stop the training job.

Type: StoppingCondition (p. 879) object

Required: Yes

TrainingEndTime

Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of TrainingStartime and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Required: Yes

TrainingJobName

Name of the model training job.

Type: String


Pattern: ^[a-zA-Z0-9\-]*\-*[a-zA-Z0-9\-]*

Required: Yes

TrainingJobStatus

The status of the training job.

Amazon SageMaker provides the following training job statuses:

- **InProgress** - The training is in progress.
- **Completed** - The training job has completed.
- **Failed** - The training job has failed. To see the reason for the failure, see the FailureReason field in the response to a DescribeTrainingJobResponse call.
- **Stopping** - The training job is stopping.
- **Stopped** - The training job has stopped.

For more detailed information, see SecondaryStatus.
Type: String

Valid Values:InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**TrainingStartTime**

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of `TrainingEndTime`. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

Required: No

**TuningJobArn**

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

Required: No

**VpcConfig**

A `VpcConfig (p. 913)` object that specifies the VPC that this training job has access to. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: `VpcConfig (p. 913)` object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
DesiredWeightAndCapacity
Service: Amazon SageMaker Service

Specifies weight and capacity values for a production variant.

Contents

DesiredInstanceCount
  The variant’s capacity.
  Type: Integer
  Valid Range: Minimum value of 1.
  Required: No

DesiredWeight
  The variant’s weight.
  Type: Float
  Valid Range: Minimum value of 0.
  Required: No

VariantName
  The name of the variant to update.
  Type: String
  Length Constraints: Maximum length of 63.
  Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 
  Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**EndpointConfigSummary**

Service: Amazon SageMaker Service

Provides summary information for an endpoint configuration.

**Contents**

**CreationTime**

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp

Required: Yes

**EndpointConfigArn**

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


Required: Yes

**EndpointConfigName**

The name of the endpoint configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9](-*[a-zA-Z0-9])*`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
EndpointSummary
Service: Amazon SageMaker Service

Provides summary information for an endpoint.

Contents

CreationTime
A timestamp that shows when the endpoint was created.
Type: Timestamp
Required: Yes

EndpointArn
The Amazon Resource Name (ARN) of the endpoint.
Type: String
Required: Yes

EndpointName
The name of the endpoint.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])* 
Required: Yes

EndpointStatus
The status of the endpoint.
• OutOfService: Endpoint is not available to take incoming requests.
• Creating: CreateEndpoint (p. 522) is executing.
• Updating: UpdateEndpoint (p. 722) or UpdateEndpointWeightsAndCapacities (p. 724) is executing.
• SystemUpdating: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
• RollingBack: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an InService status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an UpdateEndpointWeightsAndCapacities (p. 724) call or when the UpdateEndpointWeightsAndCapacities (p. 724) operation is called explicitly.
• InService: Endpoint is available to process incoming requests.
• Deleting: DeleteEndpoint (p. 572) is executing.
• Failed: Endpoint could not be created, updated, or re-scaled. Use DescribeEndpoint:FailureReason (p. 598) for information about the failure. DeleteEndpoint (p. 572) is the only operation that can be performed on a failed endpoint.

To get a list of endpoints with a specified status, use the ListEndpoints:StatusEquals (p. 657) filter.
Type: String

Valid Values: OutOfService | Creating | Updating | SystemUpdating | RollingBack | InService | Deleting | Failed

Required: Yes

_LastModifiedTime_

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Filter
Service: Amazon SageMaker Service

A conditional statement for a search expression that includes a Boolean operator, a resource property, and a value.

If you don’t specify an Operator and a Value, the filter searches for only the specified property. For example, defining a Filter for the FailureReason for the TrainingJob Resource searches for training job objects that have a value in the FailureReason field.

If you specify a Value, but not an Operator, Amazon SageMaker uses the equals operator as the default.

In search, there are several property types:

Metrics

To define a metric filter, enter a value using the form "Metrics.<name>", where <name> is a metric name. For example, the following filter searches for training jobs with an "accuracy" metric greater than "0.9":

```
{
    "Name": "Metrics.accuracy",
    "Operator": "GREATER_THAN",
    "Value": "0.9"
}
```

HyperParameters

To define a hyperparameter filter, enter a value with the form "HyperParameters.<name>". Decimal hyperparameter values are treated as a decimal in a comparison if the specified Value is also a decimal value. If the specified Value is an integer, the decimal hyperparameter values are treated as integers. For example, the following filter is satisfied by training jobs with a "learning_rate" hyperparameter that is less than "0.5":

```
{
    "Name": "HyperParameters.learning_rate",
    "Operator": "LESS_THAN",
    "Value": "0.5"
}
```

Tags

To define a tag filter, enter a value with the form "Tags.<key>".

Contents

Name

A property name. For example, TrainingJobName. For the list of valid property names returned in a search result for each supported resource, see TrainingJob (p. 884) properties. You must specify a valid property name for the resource.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.

Required: Yes

**Operator**

A Boolean binary operator that is used to evaluate the filter. The operator field contains one of the following values:

* Equals
  
  The specified resource in `Name` equals the specified `Value`.

* NotEquals
  
  The specified resource in `Name` does not equal the specified `Value`.

* GreaterThan
  
  The specified resource in `Name` is greater than the specified `Value`. Not supported for text-based properties.

* GreaterThanOrEqualTo
  
  The specified resource in `Name` is greater than or equal to the specified `Value`. Not supported for text-based properties.

* LessThan
  
  The specified resource in `Name` is less than the specified `Value`. Not supported for text-based properties.

* LessThanOrEqualTo
  
  The specified resource in `Name` is less than or equal to the specified `Value`. Not supported for text-based properties.

* Contains
  
  Only supported for text-based properties. The word-list of the property contains the specified `Value`.

If you have specified a filter `Value`, the default is `Equals`.

Type: String

Valid Values: Equals | NotEquals | GreaterThan | GreaterThanOrEqualTo | LessThan | LessThanOrEqualTo | Contains

Required: No

**Value**

A value used with `Resource` and `Operator` to determine if objects satisfy the filter's condition. For numerical properties, `Value` must be an integer or floating-point decimal. For timestamp properties, `Value` must be an ISO 8601 date-time string of the following format: `YYYY-mm-dd'T'HH:MM:SS`.

Type: String


Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
FinalHyperParameterTuningJobObjectiveMetric
Service: Amazon SageMaker Service

Shows the final value for the objective metric for a training job that was launched by a hyperparameter tuning job. You define the objective metric in the HyperParameterTuningJobObjective parameter of HyperParameterTuningJobConfig (p. 799).

Contents

MetricName

The name of the objective metric.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Required: Yes

Type

Whether to minimize or maximize the objective metric. Valid values are Minimize and Maximize.
Type: String
Valid Values: Maximize | Minimize
Required: No

Value

The value of the objective metric.
Type: Float
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**GitConfig**
Service: Amazon SageMaker Service

Specifies configuration details for a Git repository in your AWS account.

**Contents**

**Branch**
The default branch for the Git repository.
Type: String
Required: No

**RepositoryUrl**
The URL where the Git repository is located.
Type: String
Pattern: ^https://([^/]+)/?(.*)$
Required: Yes

**SecretArn**
The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of AWSCURRENT and must be in the following format:

```
{"username": UserName, "password": Password}
```
Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:secretsmanager:a-z0-9\-]*:[0-9]{12}:secret:.*
Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**GitConfigForUpdate**

Service: Amazon SageMaker Service

Specifies configuration details for a Git repository when the repository is updated.

**Contents**

**SecretArn**

The Amazon Resource Name (ARN) of the AWS Secrets Manager secret that contains the credentials used to access the git repository. The secret must have a staging label of `AWSCURRENT` and must be in the following format:

`{"username": UserName, "password": Password}`

Type: String

Length Constraints: Minimum length of 1. Maximum length of 2048.

Pattern: `arn:aws[a-z-]*:secretsmanager:[a-z0-9-]*:[0-9]{12}:secret:.`

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HumanTaskConfig
Service: Amazon SageMaker Service

Information required for human workers to complete a labeling task.

Contents

AnnotationConsolidationConfig

Configures how labels are consolidated across human workers.

Type: AnnotationConsolidationConfig (p. 752) object

Required: Yes

MaxConcurrentTaskCount

Defines the maximum number of data objects that can be labeled by human workers at the same time. Each object may have more than one worker at one time.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 1000.

Required: No

NumberOfHumanWorkersPerDataObject

The number of human workers that will label an object.

Type: Integer


Required: Yes

PreHumanTaskLambdaArn

The Amazon Resource Name (ARN) of a Lambda function that is run before a data object is sent to a human worker. Use this function to provide input to a custom labeling job.

For the built-in bounding box, image classification, semantic segmentation, and text classification task types, Amazon SageMaker Ground Truth provides the following Lambda functions:

**US East (Northern Virginia) (us-east-1):**
- arn:aws:lambda:us-east-1:432418664414:function:PRE-BoundingBox
- arn:aws:lambda:us-east-1:432418664414:function:PRE-TextMultiClass

**US East (Ohio) (us-east-2):**

**US West (Oregon) (us-west-2):**
• arn:aws:lambda:us-west-2:081040173940:function:PRE-TextMultiClass

EU (Ireland) (eu-west-1):
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-BoundingBox
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-ImageMultiClass
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-SemanticSegmentation
• arn:aws:lambda:eu-west-1:568282634449:function:PRE-TextMultiClass

Asia Pacific (Tokyo (ap-northeast-1):
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-BoundingBox
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-ImageMultiClass
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-SemanticSegmentation
• arn:aws:lambda:ap-northeast-1:477331159723:function:PRE-TextMultiClass

Type: String
Length Constraints: Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:lambda:[a-z]{2}([a-z][-\d{1}]:\d{12}:function:[a-zA-Z0-9-\_\-]+(:($LATEST|[a-zA-Z0-9-\_\-]+))?

Required: Yes

PublicWorkforceTaskPrice

The price that you pay for each task performed by a public worker.
Type: PublicWorkforceTaskPrice (p. 862) object
Required: No

TaskAvailabilityLifetimeInSeconds

The length of time that a task remains available for labelling by human workers.
Type: Integer
Required: No

TaskDescription

A description of the task for your human workers.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Required: Yes

TaskKeywords

Keywords used to describe the task so that workers on Amazon Mechanical Turk can discover the task.
Type: Array of strings
Array Members: Minimum number of 1 item. Maximum number of 5 items.

Pattern: ^[A-Za-z0-9]+( [A-Za-z0-9]+)*$

Required: No

**TaskTimeLimitInSeconds**

The amount of time that a worker has to complete a task.

Type: Integer


Required: Yes

**TaskTitle**

A title for the task for your human workers.

Type: String


Pattern: ^[	
\r -\uD7FF\uE000-\uFFFD]*$

Required: Yes

**UiConfig**

Information about the user interface that workers use to complete the labeling task.

Type: **UiConfig** (p. 910) object

Required: Yes

**WorkteamArn**

The Amazon Resource Name (ARN) of the work team assigned to complete the tasks.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterAlgorithmSpecification
Service: Amazon SageMaker Service

Specifies which training algorithm to use for training jobs that a hyperparameter tuning job launches and the metrics to monitor.

Contents

AlgorithmName

The name of the resource algorithm to use for the hyperparameter tuning job. If you specify a value for this parameter, do not specify a value for TrainingImage.

Type: String
Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*[/]?([a-zA-Z0-9\-A-Za-z0-9\-]{0,62})(?!-)$

Required: No

MetricDefinitions

An array of MetricDefinition (p. 832) objects that specify the metrics that the algorithm emits.

Type: Array of MetricDefinition (p. 832) objects
Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about Docker registry paths for built-in algorithms, see Algorithms Provided by Amazon SageMaker: Common Parameters.

Type: String
Length Constraints: Maximum length of 255.

Required: No

TrainingInputMode

The input mode that the algorithm supports: File or Pipe. In File input mode, Amazon SageMaker downloads the training data from Amazon S3 to the storage volume that is attached to the training instance and mounts the directory to the Docker volume for the training container. In Pipe input mode, Amazon SageMaker streams data directly from Amazon S3 to the container.

If you specify File mode, make sure that you provision the storage volume that is attached to the training instance with enough capacity to accommodate the training data downloaded from Amazon S3, the model artifacts, and intermediate information.

For more information about input modes, see Algorithms.

Type: String
Valid Values: Pipe | File
Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterSpecification

Service: Amazon SageMaker Service

Defines a hyperparameter to be used by an algorithm.

Contents

DefaultValue

The default value for this hyperparameter. If a default value is specified, a hyperparameter cannot be required.

Type: String

Length Constraints: Maximum length of 256.

Required: No

Description

A brief description of the hyperparameter.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*

Required: No

IsRequired

Indicates whether this hyperparameter is required.

Type: Boolean

Required: No

IsTunable

Indicates whether this hyperparameter is tunable in a hyperparameter tuning job.

Type: Boolean

Required: No

Name

The name of this hyperparameter. The name must be unique.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Range

The allowed range for this hyperparameter.

Type: ParameterRange (p. 853) object

Required: No
Type

The type of this hyperparameter. The valid types are Integer, Continuous, Categorical, and FreeText.

Type: String

Valid Values: Integer | Continuous | Categorical | FreeText

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTrainingJobDefinition
Service: Amazon SageMaker Service

Defines the training jobs launched by a hyperparameter tuning job.

Contents

AlgorithmSpecification

The HyperParameterAlgorithmSpecification (p. 789) object that specifies the resource algorithm to use for the training jobs that the tuning job launches.

Type: HyperParameterAlgorithmSpecification (p. 789) object

Required: Yes

EnableInterContainerTrafficEncryption

To encrypt all communications between ML compute instances in distributed training, choose True. Encryption provides greater security for distributed training, but training might take longer. How long it takes depends on the amount of communication between compute instances, especially if you use a deep learning algorithm in distributed training.

Type: Boolean

Required: No

EnableNetworkIsolation

Isolates the training container. No inbound or outbound network calls can be made, except for calls between peers within a training cluster for distributed training. If network isolation is used for training jobs that are configured to use a VPC, Amazon SageMaker downloads and uploads customer data and model artifacts through the specified VPC, but the training container does not have network access.

Note

The Semantic Segmentation built-in algorithm does not support network isolation.

Type: Boolean

Required: No

InputDataConfig

An array of Channel (p. 756) objects that specify the input for the training jobs that the tuning job launches.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: No

OutputDataConfig

Specifies the path to the Amazon S3 bucket where you store model artifacts from the training jobs that the tuning job launches.

Type: OutputDataConfig (p. 851) object

Required: Yes
ResourceConfig

The resources, including the compute instances and storage volumes, to use for the training jobs that the tuning job launches.

Storage volumes store model artifacts and incremental states. Training algorithms might also use storage volumes for scratch space. If you want Amazon SageMaker to use the storage volume to store the training data, choose File as the TrainingInputMode in the algorithm specification. For distributed training algorithms, specify an instance count greater than 1.

Type: ResourceConfig (p. 866) object

Required: Yes

RoleArn

The Amazon Resource Name (ARN) of the IAM role associated with the training jobs that the tuning job launches.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z_0-9+=,.@\-_/]+$ 

Required: Yes

StaticHyperParameters

Specifies the values of hyperparameters that do not change for the tuning job.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

StoppingCondition

Sets a maximum duration for the training jobs that the tuning job launches. Use this parameter to limit model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal. This delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

When Amazon SageMaker terminates a job because the stopping condition has been met, training algorithms provided by Amazon SageMaker save the intermediate results of the job.

Type: StoppingCondition (p. 879) object

Required: Yes

VpcConfig

The VpcConfig (p. 913) object that specifies the VPC that you want the training jobs that this hyperparameter tuning job launches to connect to. Control access to and from your training container by configuring the VPC. For more information, see Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Type: VpcConfig (p. 913) object
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTrainingJobSummary

Service: Amazon SageMaker Service

Specifies summary information about a training job.

Contents

**CreationTime**

The date and time that the training job was created.

Type: Timestamp

Required: Yes

**FailureReason**

The reason that the training job failed.

Type: String

Length Constraints: Maximum length of 1024.

Required: No

**FinalHyperParameterTuningJobObjectiveMetric**

The **FinalHyperParameterTuningJobObjectiveMetric** (p. 783) object that specifies the value of the objective metric of the tuning job that launched this training job.

Type: **FinalHyperParameterTuningJobObjectiveMetric** (p. 783) object

Required: No

**ObjectiveStatus**

The status of the objective metric for the training job:

- **Succeeded**: The final objective metric for the training job was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.
- **Pending**: The training job is in progress and evaluation of its final objective metric is pending.
- **Failed**: The final objective metric for the training job was not evaluated, and was not used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.

Type: String

Valid Values: Succeeded | Pending | Failed

Required: No

**TrainingEndTime**

Specifies the time when the training job ends on training instances. You are billed for the time interval between the value of **TrainingStartTime** and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

**TrainingJobArn**

The Amazon Resource Name (ARN) of the training job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*
Required: Yes

**TrainingJobName**

The name of the training job.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*\nRequired: Yes

**TrainingJobStatus**

The status of the training job.

Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: Yes

**TrainingStartTime**

The date and time that the training job started.

Type: Timestamp
Required: No

**TunedHyperParameters**

A list of the hyperparameters for which you specified ranges to search.

Type: String to string map
Key Length Constraints: Maximum length of 256.
Value Length Constraints: Maximum length of 256.
Required: Yes

**TuningJobName**

The HyperParameter tuning job that launched the training job.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*\nRequired: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
HyperParameterTuningJobConfig

Service: Amazon SageMaker Service

Configures a hyperparameter tuning job.

Contents

HyperParameterTuningJobObjective

The HyperParameterTuningJobObjective (p. 801) object that specifies the objective metric for this tuning job.

Type: HyperParameterTuningJobObjective (p. 801) object

Required: Yes

ParameterRanges

The ParameterRanges (p. 854) object that specifies the ranges of hyperparameters that this tuning job searches.

Type: ParameterRanges (p. 854) object

Required: Yes

ResourceLimits

The ResourceLimits (p. 868) object that specifies the maximum number of training jobs and parallel training jobs for this tuning job.

Type: ResourceLimits (p. 868) object

Required: Yes

Strategy

Specifies the search strategy for hyperparameters. Currently, the only valid value is Bayesian.

Type: String

Valid Values: Bayesian

Required: Yes

TrainingJobEarlyStoppingType

Specifies whether to use early stopping for training jobs launched by the hyperparameter tuning job. This can be one of the following values (the default value is OFF):

OFF

Training jobs launched by the hyperparameter tuning job do not use early stopping.

AUTO

Amazon SageMaker stops training jobs launched by the hyperparameter tuning job when they are unlikely to perform better than previously completed training jobs. For more information, see Stop Training Jobs Early.

Type: String

Valid Values: Off | Auto

Required: No
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobObjective
Service: Amazon SageMaker Service

Defines the objective metric for a hyperparameter tuning job. Hyperparameter tuning uses the value of this metric to evaluate the training jobs it launches, and returns the training job that results in either the highest or lowest value for this metric, depending on the value you specify for the Type parameter.

Contents

MetricName
The name of the metric to use for the objective metric.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Required: Yes

Type
Whether to minimize or maximize the objective metric.
Type: String
Valid Values: Maximize | Minimize
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobSummary

Service: Amazon SageMaker Service

Provides summary information about a hyperparameter tuning job.

Contents

CreationTime

The date and time that the tuning job was created.

Type: Timestamp

Required: Yes

HyperParameterTuningEndTime

The date and time that the tuning job ended.

Type: Timestamp

Required: No

HyperParameterTuningJobArn

The Amazon Resource Name (ARN) of the tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*

Required: Yes

HyperParameterTuningJobName

The name of the tuning job.

Type: String


Pattern: ^[a-zA-Z0-9\-]*([a-zA-Z0-9\-]*)*

Required: Yes

HyperParameterTuningJobStatus

The status of the tuning job.

Type: String

Valid Values: Completed | InProgress | Failed | Stopped | Stopping

Required: Yes

LastModifiedTime

The date and time that the tuning job was modified.

Type: Timestamp

Required: No
**ObjectiveStatusCounters**

The `ObjectiveStatusCounters (p. 849)` object that specifies the numbers of training jobs, categorized by objective metric status, that this tuning job launched.

Type: `ObjectiveStatusCounters (p. 849)` object

Required: Yes

**ResourceLimits**

The `ResourceLimits (p. 868)` object that specifies the maximum number of training jobs and parallel training jobs allowed for this tuning job.

Type: `ResourceLimits (p. 868)` object

Required: No

**Strategy**

Specifies the search strategy hyperparameter tuning uses to choose which hyperparameters to use for each iteration. Currently, the only valid value is Bayesian.

Type: String

Valid Values: Bayesian

Required: Yes

**TrainingJobStatusCounters**

The `TrainingJobStatusCounters (p. 891)` object that specifies the numbers of training jobs, categorized by status, that this tuning job launched.

Type: `TrainingJobStatusCounters (p. 891)` object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
HyperParameterTuningJobWarmStartConfig

Service: Amazon SageMaker Service

Specifies the configuration for a hyperparameter tuning job that uses one or more previous hyperparameter tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job.

All training jobs launched by the new hyperparameter tuning job are evaluated by using the objective metric, and the training job that performs the best is compared to the best training jobs from the parent tuning jobs. From these, the training job that performs the best as measured by the objective metric is returned as the overall best training job.

**Note**

All training jobs launched by parent hyperparameter tuning jobs and the new hyperparameter tuning jobs count against the limit of training jobs for the tuning job.

Contents

ParentHyperParameterTuningJobs

An array of hyperparameter tuning jobs that are used as the starting point for the new hyperparameter tuning job. For more information about warm starting a hyperparameter tuning job, see Using a Previous Hyperparameter Tuning Job as a Starting Point.

Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Type: Array of ParentHyperParameterTuningJob (p. 855) objects

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Required: Yes

WarmStartType

Specifies one of the following:

**IDENTICAL_DATA_AND_ALGORITHM**

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

**TRANSFER_LEARNING**

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. The training image can also be a different version from the version used in the parent hyperparameter tuning job. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The objective metric for the new tuning job must be the same as for all parent jobs.

Type: String

Valid Values: IdenticalDataAndAlgorithm | TransferLearning
Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
InferenceSpecification
Service: Amazon SageMaker Service

Defines how to perform inference generation after a training job is run.

Contents

Containers
The Amazon ECR registry path of the Docker image that contains the inference code.

Type: Array of ModelPackageContainerDefinition (p. 834) objects

Array Members: Fixed number of 1 item.

Required: Yes

SupportedContentTypes
The supported MIME types for the input data.

Type: Array of strings

Length Constraints: Maximum length of 256.

Required: Yes

SupportedRealtimeInferenceInstanceTypes
A list of the instance types that are used to generate inferences in real-time.

Type: Array of strings

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge
| ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge |
| ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge |
| ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.large | ml.c4.xlarge |
| ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge |
| ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
| ml.c5.18xlarge

Required: Yes

SupportedResponseMIMETypes
The supported MIME types for the output data.

Type: Array of strings

Length Constraints: Maximum length of 1024.

Required: Yes

SupportedTransformInstanceTypes
A list of the instance types on which a transformation job can be run or on which an endpoint can be deployed.

Type: Array of strings

Array Members: Minimum number of 1 item.
Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.8xlarge | ml.c5.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.m5.9xlarge | ml.m5.18xlarge | ml.m5.xlarge

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
### InputConfig

Service: Amazon SageMaker Service

Contains information about the location of input model artifacts, the name and shape of the expected data inputs, and the framework in which the model was trained.

#### Contents

**DataInputConfig**

Specifies the name and shape of the expected data inputs for your trained model with a JSON dictionary form. The data inputs are **InputConfig:Framework** specific.

- **TensorFlow**: You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, 
      ```json
      {"input": [1, 1024, 1024, 3]}
      ```
    - If using the CLI, 
      ```json
      {"input": [1, 1024, 1024, 3]}
      ```
  - Examples for two inputs:
    - If using the console, 
      ```json
      {"input1": [1, 128, 128, 1], "input2": [1, 128, 128, 1]}
      ```
    - If using the CLI, 
      ```json
      {"input1": [1, 128, 128, 1], "input2": [1, 128, 128, 1]}
      ```

- **MXNET/ONNX**: You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
  - Examples for one input:
    - If using the console, 
      ```json
      {"data": [1, 3, 1024, 1024]}
      ```
    - If using the CLI, 
      ```json
      {"data": [1, 3, 1024, 1024]}
      ```
  - Examples for two inputs:
    - If using the console, 
      ```json
      {"var1": [1, 1, 28, 28], "var2": [1, 1, 28, 28]}
      ```
    - If using the CLI, 
      ```json
      {"var1": [1, 1, 28, 28], "var2": [1, 1, 28, 28]}
      ```

- **PyTorch**: You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
  - Examples for one input in dictionary format:
    - If using the console, 
      ```json
      {"input0": [1, 3, 224, 224]}
      ```
    - If using the CLI, 
      ```json
      {"input0": [1, 3, 224, 224]}
      ```
  - Example for one input in list format: 
    ```json
    [1, 3, 224, 224]
    ```
  - Examples for two inputs in dictionary format:
    - If using the console, 
      ```json
      {"input0": [1, 3, 224, 224], "input1": [1, 3, 224, 224]}
      ```
    - If using the CLI, 
      ```json
      {"input0": [1, 3, 224, 224], "input1": [1, 3, 224, 224]}
      ```
  - Example for two inputs in list format: 
    ```json
    [[1, 3, 224, 224], [1, 3, 224, 224]]
    ```

- **XGBOOST**: input data name and shape are not needed.

**Type**: String

**Length Constraints**: Minimum length of 1. Maximum length of 1024.

**Required**: Yes

#### Framework

Identifies the framework in which the model was trained. For example: **TENSORFLOW**.
**Type:** String

**Valid Values:** TENSORFLOW | MXNET | ONNX | PYTORCH | XGBOOST

**Required:** Yes

**S3Uri**

The S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

**Type:** String

**Length Constraints:** Maximum length of 1024.

**Pattern:** ^https://[^/]+/(.*)$

**Required:** Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
IntegerParameterRange
Service: Amazon SageMaker Service

For a hyperparameter of the integer type, specifies the range that a hyperparameter tuning job searches.

Contents

MaxValue
The maximum value of the hyperparameter to search.
Type: String
Length Constraints: Maximum length of 256.
Required: Yes

MinValue
The minimum value of the hyperparameter to search.
Type: String
Length Constraints: Maximum length of 256.
Required: Yes

Name
The name of the hyperparameter to search.
Type: String
Length Constraints: Maximum length of 256.
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
IntegerParameterRangeSpecification
Service: Amazon SageMaker Service

Defines the possible values for an integer hyperparameter.

Contents

MaxValue

The maximum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

MinValue

The minimum integer value allowed.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**LabelCounters**  
Service: Amazon SageMaker Service  
Provides a breakdown of the number of objects labeled.

**Contents**

**FailedNonRetryableError**  
The total number of objects that could not be labeled due to an error.  
Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**HumanLabeled**  
The total number of objects labeled by a human worker.  
Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**MachineLabeled**  
The total number of objects labeled by automated data labeling.  
Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**TotalLabeled**  
The total number of objects labeled.  
Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**Unlabeled**  
The total number of objects not yet labeled.  
Type: Integer  
Valid Range: Minimum value of 0.  
Required: No

**See Also**
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelCountersForWorkteam
Service: Amazon SageMaker Service

Provides counts for human-labeled tasks in the labeling job.

Contents

HumanLabeled
The total number of data objects labeled by a human worker.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

PendingHuman
The total number of data objects that need to be labeled by a human worker.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

Total
The total number of tasks in the labeling job.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobAlgorithmsConfig

Service: Amazon SageMaker Service

Provides configuration information for auto-labeling of your data objects. A LabelingJobAlgorithmsConfig object must be supplied in order to use auto-labeling.

Contents

InitialActiveLearningModelArn

At the end of an auto-label job Amazon SageMaker Ground Truth sends the Amazon Resource Name (ARN) of the final model used for auto-labeling. You can use this model as the starting point for subsequent similar jobs by providing the ARN of the model here.

Type: String


Required: No

LabelingJobAlgorithmSpecificationArn

Specifies the Amazon Resource Name (ARN) of the algorithm used for auto-labeling. You must select one of the following ARNs:

- **Image classification**
  

- **Text classification**
  

- **Object detection**
  

Type: String

Length Constraints: Maximum length of 2048.

Required: Yes

LabelingJobResourceConfig

Provides configuration information for a labeling job.

Type: LabelingJobResourceConfig (p. 824) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobDataAttributes
Service: Amazon SageMaker Service

Attributes of the data specified by the customer. Use these to describe the data to be labeled.

Contents

ContentClassifiers

Declares that your content is free of personally identifiable information or adult content. Amazon SageMaker may restrict the Amazon Mechanical Turk workers that can view your task based on this information.

Type: Array of strings
Array Members: Maximum number of 256 items.
Valid Values: FreeOfPersonallyIdentifiableInformation | FreeOfAdultContent
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobDataSource
Service: Amazon SageMaker Service

Provides information about the location of input data.

Contents

S3DataSource

The Amazon S3 location of the input data objects.

Type: LabelingJobS3DataSource (p. 825) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobForWorkteamSummary
Service: Amazon SageMaker Service
Provides summary information for a work team.

Contents

CreationTime
The date and time that the labeling job was created.
Type: Timestamp
Required: Yes

JobReferenceCode
A unique identifier for a labeling job. You can use this to refer to a specific labeling job.
Type: String
Length Constraints: Minimum length of 1.
Required: Yes

LabelCounters
Provides information about the progress of a labeling job.
Type: LabelCountersForWorkteam (p. 814) object
Required: No

LabelingJobName
The name of the labeling job that the work team is assigned to.
Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]+)*
Required: No

WorkRequesterAccountId
Type: String
Pattern: ^\d+$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobInputConfig
Service: Amazon SageMaker Service

Input configuration information for a labeling job.

Contents

DataAttributes

Attributes of the data specified by the customer.

Type: LabelingJobDataAttributes (p. 817) object

Required: No

DataSource

The location of the input data.

Type: LabelingJobDataSource (p. 818) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobOutput
Service: Amazon SageMaker Service

Specifies the location of the output produced by the labeling job.

Contents

FinalActiveLearningModelArn

The Amazon Resource Name (ARN) for the most recent Amazon SageMaker model trained as part of automated data labeling.

Type: String
Required: No

OutputDatasetS3Uri

The Amazon S3 bucket location of the manifest file for labeled data.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3)://([^/]+)/?([^/]*)$
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobOutputConfig
Service: Amazon SageMaker Service

Output configuration information for a labeling job.

Contents

KmsKeyId
The AWS Key Management Service ID of the key used to encrypt the output data, if any.

Type: String
Length Constraints: Maximum length of 2048.
Required: No

S3OutputPath
The Amazon S3 location to write output data.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^(https|s3):/([^/]+)/?$(/.*)$
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobResourceConfig
Service: Amazon SageMaker Service

Provides configuration information for labeling jobs.

Contents

VolumeKmsKeyId

The AWS Key Management Service key ID for the key used to encrypt the output data, if any.

Type: String

Length Constraints: Maximum length of 2048.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobS3DataSource
Service: Amazon SageMaker Service

The Amazon S3 location of the input data objects.

Contents

ManifestS3Uri

The Amazon S3 location of the manifest file that describes the input data objects.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3):///([^/]+)/*(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
LabelingJobStoppingConditions
Service: Amazon SageMaker Service

A set of conditions for stopping a labeling job. If any of the conditions are met, the job is automatically stopped. You can use these conditions to control the cost of data labeling.

Contents

MaxHumanLabeledObjectCount

- The maximum number of objects that can be labeled by human workers.
- Type: Integer
- Valid Range: Minimum value of 1.
- Required: No

MaxPercentageOfInputDatasetLabeled

- The maximum number of input data objects that should be labeled.
- Type: Integer
- Valid Range: Minimum value of 1. Maximum value of 100.
- Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**LabelingJobSummary**
Service: Amazon SageMaker Service

Provides summary information about a labeling job.

**Contents**

**AnnotationConsolidationLambdaArn**

The Amazon Resource Name (ARN) of the Lambda function used to consolidate the annotations from individual workers into a label for a data object. For more information, see [Annotation Consolidation](#).

Type: String
Length Constraints: Maximum length of 2048.

**CreationTime**

The date and time that the job was created (timestamp).

Type: Timestamp
Required: Yes

**FailureReason**

If the **LabelingJobStatus** field is Failed, this field contains a description of the error.

Type: String
Length Constraints: Maximum length of 1024.

**InputConfig**

Input configuration for the labeling job.

Type: **LabelingJobInputConfig** (p. 821) object
Required: No

**LabelCounters**

Counts showing the progress of the labeling job.

Type: **LabelCounters** (p. 812) object
Required: Yes

**LabelingJobArn**

The Amazon Resource Name (ARN) assigned to the labeling job when it was created.

Type: String
Length Constraints: Maximum length of 2048.

**Pattern**: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*`
Required: Yes

**LabelingJobName**

The name of the labeling job.

Type: String


Pattern: ^[a-zA-Z0-9\-\*\[a-zA-Z0-9\-\*\]]*

Required: Yes

**LabelingJobOutput**

The location of the output produced by the labeling job.

Type: LabelingJobOutput (p. 822) object

Required: No

**LabelingJobStatus**

The current status of the labeling job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**LastModifiedTime**

The date and time that the job was last modified (timestamp).

Type: Timestamp

Required: Yes

**PreHumanTaskLambdaArn**

The Amazon Resource Name (ARN) of a Lambda function. The function is run before each data object is sent to a worker.

Type: String

Length Constraints: Maximum length of 2048.

Pattern: arn:aws[a-z\-]*:lambda:[a-z]{2}[-][a-z]+--\d{1}:\d{12}:function:[a-zA-Z0-9-\-\*\.\-]*\$LATEST|\ [[a-zA-Z0-9-\*\.\-]+)?

Required: Yes

**WorkteamArn**

The Amazon Resource Name (ARN) of the work team assigned to the job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:workteam/.*

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**MemberDefinition**  
*Service: Amazon SageMaker Service*

Defines the Amazon Cognito user group that is part of a work team.

**Contents**

**CognitoMemberDefinition**

The Amazon Cognito user group that is part of the work team.

Type: *CognitoMemberDefinition (p. 762)* object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MetricData
Service: Amazon SageMaker Service

The name, value, and date and time of a metric that was emitted to Amazon CloudWatch.

Contents

MetricName

The name of the metric.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Required: No

Timestamp

The date and time that the algorithm emitted the metric.

Type: Timestamp

Required: No

Value

The value of the metric.

Type: Float

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
MetricDefinition
Service: Amazon SageMaker Service

Specifies a metric that the training algorithm writes to stderr or stdout. Amazon SageMaker hyperparameter tuning captures all defined metrics. You specify one metric that a hyperparameter tuning job uses as its objective metric to choose the best training job.

Contents

Name
The name of the metric.
Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Required: Yes

Regex
A regular expression that searches the output of a training job and gets the value of the metric. For more information about using regular expressions to define metrics, see Defining Objective Metrics.
Type: String
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ModelArtifacts**
Service: Amazon SageMaker Service

Provides information about the location that is configured for storing model artifacts.

**Contents**

**S3ModelArtifacts**

The path of the S3 object that contains the model artifacts. For example, `s3://bucket-name/keynameprefix/model.tar.gz`.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: `^(https|s3)://([^/]+)/?([^/]+)$`

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageContainerDefinition

Service: Amazon SageMaker Service

Describes the Docker container for the model package.

Contents

ContainerHostname

The DNS host name for the Docker container.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9\-\*][a-zA-Z0-9\-\*]+$ 

Required: No

Image

The Amazon EC2 Container Registry (Amazon ECR) path where inference code is stored.

If you are using your own custom algorithm instead of an algorithm provided by Amazon SageMaker, the inference code must meet Amazon SageMaker requirements. Amazon SageMaker supports both registry/repository[:tag] and registry/repository[@digest] image path formats. For more information, see Using Your Own Algorithms with Amazon SageMaker.

Type: String

Length Constraints: Maximum length of 255.

Pattern: [\S]+ 

Required: Yes

ImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: ^[Ss][Hh][Aa]256:[0-9a-fA-F]{64}$

Required: No

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*$)

Required: No

ProductId

The AWS Marketplace product ID of the model package.
Type: String
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageStatusDetails

Service: Amazon SageMaker Service

Specifies the validation and image scan statuses of the model package.

Contents

ImageScanStatuses

The status of the scan of the Docker image container for the model package.

Type: Array of ModelPackageStatusItem (p. 837) objects

Required: No

ValidationStatuses

The validation status of the model package.

Type: Array of ModelPackageStatusItem (p. 837) objects

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageStatusItem

Service: Amazon SageMaker Service

Represents the overall status of a model package.

Contents

FailureReason

if the overall status is Failed, the reason for the failure.

Type: String
Required: No

Name

The name of the model package for which the overall status is being reported.

Type: String
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$
Required: Yes

Status

The current status.

Type: String
Valid Values: NotStarted | InProgress | Completed | Failed
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageSummary
Service: Amazon SageMaker Service

Provides summary information about a model package.

Contents

CreationTime
A timestamp that shows when the model package was created.

Type: Timestamp
Required: Yes

ModelPackageArn
The Amazon Resource Name (ARN) of the model package.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:model-package/.*
Required: Yes

ModelPackageDescription
A brief description of the model package.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: [\p{L}\p{M}\p{Z}\p{S}\p{N}\p{P}]*
Required: No

ModelPackageName
The name of the model package.

Type: String
Pattern: ^[a-zA-Z0-9-]*[^a-zA-Z0-9 ]*$
Required: Yes

ModelPackageStatus
The overall status of the model package.

Type: String
Valid Values: Pending | InProgress | Completed | Failed | Deleting
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
ModelPackageValidationProfile

Service: Amazon SageMaker Service

Contains data, such as the inputs and targeted instance types that are used in the process of validating the model package.

The data provided in the validation profile is made available to your buyers on AWS Marketplace.

Contents

ProfileName

The name of the profile for the model package.

Type: String


Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$

Required: Yes

TransformJobDefinition

The TransformJobDefinition object that describes the transform job used for the validation of the model package.

Type: TransformJobDefinition (p. 900) object

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelPackageValidationSpecification
Service: Amazon SageMaker Service

Specifies batch transform jobs that Amazon SageMaker runs to validate your model package.

Contents

ValidationProfiles
An array of ModelPackageValidationProfile objects, each of which specifies a batch transform job that Amazon SageMaker runs to validate your model package.

Type: Array of ModelPackageValidationProfile (p. 840) objects

Array Members: Fixed number of 1 item.

Required: Yes

ValidationRole
The IAM roles to be used for the validation of the model package.

Type: String


Pattern: ^arn:aws[a-zA-Z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9+=,.@\-_/]+$^

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ModelSummary
Service: Amazon SageMaker Service
Provides summary information about a model.

Contents

CreationTime
A timestamp that indicates when the model was created.
Type: Timestamp
Required: Yes

ModelArn
The Amazon Resource Name (ARN) of the model.
Type: String
Required: Yes

ModelName
The name of the model that you want a summary for.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**NestedFilters**

Service: Amazon SageMaker Service

Defines a list of NestedFilters objects. To satisfy the conditions specified in the NestedFilters call, a resource must satisfy the conditions of all of the filters.

For example, you could define a NestedFilters using the training job's InputDataConfig property to filter on Channel objects.

A NestedFilters object contains multiple filters. For example, to find all training jobs whose name contains `train` and that have `cat/data` in their S3Uri (specified in InputDataConfig), you need to create a NestedFilters object that specifies the InputDataConfig property with the following Filter objects:

- `{Name:"InputDataConfig.ChannelName", "Operator":"EQUALS", "Value":"train"}`,
- `{Name:"InputDataConfig.DataSource.S3DataSource.S3Uri", "Operator":"CONTAINS", "Value":"cat/data"}`

**Contents**

**Filters**

A list of filters. Each filter acts on a property. Filters must contain at least one Filters value. For example, a NestedFilters call might include a filter on the PropertyName parameter of the InputDataConfig property: InputDataConfig.DataSource.S3DataSource.S3Uri.

Type: Array of Filter (p. 780) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: Yes

**NestedPropertyName**

The name of the property to use in the nested filters. The value must match a listed property name, such as InputDataConfig.

Type: String

Length Constraints: Minimum length of 1. Maximum length of 255.

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceLifecycleConfigSummary

Service: Amazon SageMaker Service

Provides a summary of a notebook instance lifecycle configuration.

Contents

**CreationTime**

A timestamp that tells when the lifecycle configuration was created.

Type: Timestamp

Required: No

**LastModifiedTime**

A timestamp that tells when the lifecycle configuration was last modified.

Type: Timestamp

Required: No

**NotebookInstanceLifecycleConfigArn**

The Amazon Resource Name (ARN) of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

**NotebookInstanceLifecycleConfigName**

The name of the lifecycle configuration.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9-]*[a-zA-Z0-9-]*$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceLifecycleHook
Service: Amazon SageMaker Service

Contains the notebook instance lifecycle configuration script.

Each lifecycle configuration script has a limit of 16384 characters.

The value of the $PATH environment variable that is available to both scripts is /sbin:bin:/usr/sbin:/usr/bin.

View CloudWatch Logs for notebook instance lifecycle configurations in log group /aws/sagemaker/NotebookInstances in log stream [notebook-instance-name]/[LifecycleConfigHook].

Lifecycle configuration scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Contents

Content

A base64-encoded string that contains a shell script for a notebook instance lifecycle configuration.

Type: String


Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
NotebookInstanceSummary
Service: Amazon SageMaker Service

Provides summary information for an Amazon SageMaker notebook instance.

Contents

AdditionalCodeRepositories

An array of up to three Git repositories associated with the notebook instance. These can be either the names of Git repositories stored as resources in your account, or the URL of Git repositories in AWS CodeCommit or in any other Git repository. These repositories are cloned at the same level as the default repository of your notebook instance. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: Array of strings

Array Members: Maximum number of 3 items.


Pattern: ^https://([^/]+)/(.*$)|^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: No

CreationTime

A timestamp that shows when the notebook instance was created.

Type: Timestamp

Required: No

DefaultCodeRepository

The Git repository associated with the notebook instance as its default code repository. This can be either the name of a Git repository stored as a resource in your account, or the URL of a Git repository in AWS CodeCommit or in any other Git repository. When you open a notebook instance, it opens in the directory that contains this repository. For more information, see Associating Git Repositories with Amazon SageMaker Notebook Instances.

Type: String


Pattern: ^https://([^/]+)/(.*$)|^[a-zA-Z0-9](-*[a-zA-Z0-9])*

Required: No

InstanceType

The type of ML compute instance that the notebook instance is running on.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.t3.medium | ml.t3.large | ml.t3.xlarge | ml.t3.2xlarge | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.c5d.xlarge | ml.c5d.2xlarge | ml.c5d.4xlarge
| ml.c5d.9xlarge | ml.c5d.18xlarge | ml.p2.xlarge | ml.p2.8xlarge |
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge |

Required: No

**LastModifiedTime**

A timestamp that shows when the notebook instance was last modified.

Type: Timestamp

Required: No

**NotebookInstanceArn**

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

**NotebookInstanceLifecycleConfigName**

The name of a notebook instance lifecycle configuration associated with this notebook instance.

For information about notebook instance lifestyle configurations, see Step 2.1: (Optional) Customize a Notebook Instance.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: No

**NotebookInstanceName**

The name of the notebook instance that you want a summary for.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  

Required: Yes

**NotebookInstanceStatus**

The status of the notebook instance.

Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting | Updating  

Required: No

**Url**

The URL that you use to connect to the Jupyter instance running in your notebook instance.

Type: String
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ObjectiveStatusCounters
Service: Amazon SageMaker Service

Specifies the number of training jobs that this hyperparameter tuning job launched, categorized by the status of their objective metric. The objective metric status shows whether the final objective metric for the training job has been evaluated by the tuning job and used in the hyperparameter tuning process.

Contents

Failed
The number of training jobs whose final objective metric was not evaluated and used in the hyperparameter tuning process. This typically occurs when the training job failed or did not emit an objective metric.

Type: Integer
Valid Range: Minimum value of 0.
Required: No

Pending
The number of training jobs that are in progress and pending evaluation of their final objective metric.

Type: Integer
Valid Range: Minimum value of 0.
Required: No

Succeeded
The number of training jobs whose final objective metric was evaluated by the hyperparameter tuning job and used in the hyperparameter tuning process.

Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
OutputConfig
Service: Amazon SageMaker Service

Contains information about the output location for the compiled model and the device (target) that the model runs on.

Contents

S3OutputLocation

Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, s3://bucket-name/key-name-prefix.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^((https|s3):/(/[^/]+/)?([^/]+))$

Required: Yes

TargetDevice

Identifies the device that you want to run your model on after it has been compiled. For example: ml_c5.

Type: String

Valid Values: ml_m4 | ml_m5 | ml_c4 | ml_c5 | ml_p2 | ml_p3 | jetson_tx1 | jetson_tx2 | rasp3b | deaplens

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
OutputDataConfig
Service: Amazon SageMaker Service

Provides information about how to store model training results (model artifacts).

Contents

KmsKeyId
The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The KmsKeyId can be any of the following formats:

- // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"
- // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
- // KMS Key Alias
  "alias/ExampleAlias"
- // Amazon Resource Name (ARN) of a KMS Key Alias

If you don't provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role's account. For more information, see KMS-Managed Encryption Keys in the Amazon Simple Storage Service Developer Guide.

The KMS key policy must grant permission to the IAM role that you specify in your CreateTransformJob request. For more information, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Type: String

Length Constraints: Maximum length of 2048.

Required: No

S3OutputPath
Identifies the S3 path where you want Amazon SageMaker to store the model artifacts. For example, s3://bucket-name/key-name-prefix.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([/]+)?(.*$)

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
ParameterRange
Service: Amazon SageMaker Service

Defines the possible values for categorical, continuous, and integer hyperparameters to be used by an algorithm.

Contents

CategoricalParameterRangeSpecification

A CategoricalParameterRangeSpecification object that defines the possible values for a categorical hyperparameter.

Type: CategoricalParameterRangeSpecification (p. 755) object

Required: No

ContinuousParameterRangeSpecification

A ContinuousParameterRangeSpecification object that defines the possible values for a continuous hyperparameter.

Type: ContinuousParameterRangeSpecification (p. 768) object

Required: No

IntegerParameterRangeSpecification

A IntegerParameterRangeSpecification object that defines the possible values for an integer hyperparameter.

Type: IntegerParameterRangeSpecification (p. 811) object

Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ParameterRanges
Service: Amazon SageMaker Service

Specifies ranges of integer, continuous, and categorical hyperparameters that a hyperparameter tuning job searches. The hyperparameter tuning job launches training jobs with hyperparameter values within these ranges to find the combination of values that result in the training job with the best performance as measured by the objective metric of the hyperparameter tuning job.

**Note**
You can specify a maximum of 20 hyperparameters that a hyperparameter tuning job can search over. Every possible value of a categorical parameter range counts against this limit.

**Contents**

CategoricalParameterRanges

The array of CategoricalParameterRange (p. 754) objects that specify ranges of categorical hyperparameters that a hyperparameter tuning job searches.

Type: Array of CategoricalParameterRange (p. 754) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

ContinuousParameterRanges

The array of ContinuousParameterRange (p. 767) objects that specify ranges of continuous hyperparameters that a hyperparameter tuning job searches.

Type: Array of ContinuousParameterRange (p. 767) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

IntegerParameterRanges

The array of IntegerParameterRange (p. 810) objects that specify ranges of integer hyperparameters that a hyperparameter tuning job searches.

Type: Array of IntegerParameterRange (p. 810) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ParentHyperParameterTuningJob
Service: Amazon SageMaker Service

A previously completed or stopped hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Contents

HyperParameterTuningJobName

The name of the hyperparameter tuning job to be used as a starting point for a new hyperparameter tuning job.

Type: String


Pattern: ^[a-zA-Z0-9](-[a-zA-Z0-9])*$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ProductionVariant**

Service: Amazon SageMaker Service

Identifies a model that you want to host and the resources to deploy for hosting it. If you are deploying multiple models, tell Amazon SageMaker how to distribute traffic among the models by specifying variant weights.

**Contents**

**AcceleratorType**

The size of the Elastic Inference (EI) instance to use for the production variant. EI instances provide on-demand GPU computing for inference. For more information, see Using Elastic Inference in Amazon SageMaker. For more information, see Using Elastic Inference in Amazon SageMaker.

Type: String

Valid Values: ml.eia1.medium | ml.eia1.large | ml.eia1.xlarge

Required: No

**InitialInstanceCount**

Number of instances to launch initially.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

**InitialVariantWeight**

Determines initial traffic distribution among all of the models that you specify in the endpoint configuration. The traffic to a production variant is determined by the ratio of the VariantWeight to the sum of all VariantWeight values across all ProductionVariants. If unspecified, it defaults to 1.0.

Type: Float

Valid Range: Minimum value of 0.

Required: No

**InstanceType**

The ML compute instance type.

Type: String

Valid Values: ml.t2.medium | ml.t2.large | ml.t2.xlarge | ml.t2.2xlarge | ml.m4.large | ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.c4.large | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.8xlarge | ml.c4.16xlarge | ml.c4.32xlarge | ml.p2.large | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.large | ml.p3.xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.p3.24xlarge | ml.c5.large | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge

Required: Yes
ModelName

The name of the model that you want to host. This is the name that you specified when creating the model.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](\-*[a-zA-Z0-9])* 

Required: Yes

VariantName

The name of the production variant.

Type: String

Length Constraints: Maximum length of 63.

Pattern: ^[a-zA-Z0-9](\-*[a-zA-Z0-9])* 

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ProductionVariantSummary
Service: Amazon SageMaker Service

Describes weight and capacities for a production variant associated with an endpoint. If you sent a request to the UpdateEndpointWeightsAndCapacities API and the endpoint status is Updating, you get different desired and current values.

Contents

CurrentInstanceCount

The number of instances associated with the variant.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

CurrentWeight

The weight associated with the variant.

Type: Float

Valid Range: Minimum value of 0.

Required: No

DeployedImages

An array of DeployedImage objects that specify the Amazon EC2 Container Registry paths of the inference images deployed on instances of this ProductionVariant.

Type: Array of DeployedImage (p. 770) objects

Required: No

DesiredInstanceCount

The number of instances requested in the UpdateEndpointWeightsAndCapacities request.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

DesiredWeight

The requested weight, as specified in the UpdateEndpointWeightsAndCapacities request.

Type: Float

Valid Range: Minimum value of 0.

Required: No

VariantName

The name of the variant.

Type: String

Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9]*)*

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**PropertyNameQuery**
Service: Amazon SageMaker Service

A type of `SuggestionQuery`. A suggestion query for retrieving property names that match the specified hint.

**Contents**

**PropertyNameHint**

Text that is part of a property's name. The property names of hyperparameter, metric, and tag key names that begin with the specified text in the `PropertyNameHint`.

Type: String

Length Constraints: Minimum length of 0. Maximum length of 100.

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
PropertyNameSuggestion
Service: Amazon SageMaker Service

A property name returned from a GetSearchSuggestions call that specifies a value in the PropertyNameQuery field.

Contents

PropertyName

A suggested property name based on what you entered in the search textbox in the Amazon SageMaker console.

Type: String
Length Constraints: Minimum length of 1. Maximum length of 255.
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
PublicWorkforceTaskPrice
Service: Amazon SageMaker Service

Defines the amount of money paid to an Amazon Mechanical Turk worker for each task performed.

Use one of the following prices for bounding box tasks. Prices are in US dollars.

- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for image classification, text classification, and custom tasks. Prices are in US dollars.

- 0.012
- 0.024
- 0.036
- 0.048
- 0.060
- 0.072
- 0.120
- 0.240
- 0.360
- 0.480
- 0.600
- 0.720
- 0.840
- 0.960
- 1.080
- 1.200

Use one of the following prices for semantic segmentation tasks. Prices are in US dollars.

- 0.840
- 0.960
- 1.080
- 1.200
Contents

**AmountInUsd**

Defines the amount of money paid to a worker in United States dollars.

Type: [USD (p. 912)](p. 912) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**RenderableTask**

Service: Amazon SageMaker Service

Contains input values for a task.

**Contents**

**Input**

A JSON object that contains values for the variables defined in the template. It is made available to the template under the substitution variable `task.input`. For example, if you define a variable `task.input.text` in your template, you can supply the variable in the JSON object as "text": "sample text".

Type: String


Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
RenderingError
Service: Amazon SageMaker Service

A description of an error that occurred while rendering the template.

Contents

Code
A unique identifier for a specific class of errors.
Type: String
Required: Yes

Message
A human-readable message describing the error.
Type: String
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ResourceConfig
Service: Amazon SageMaker Service

Describes the resources, including ML compute instances and ML storage volumes, to use for model training.

Contents

InstanceCount
The number of ML compute instances to use. For distributed training, provide a value greater than 1.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType
The ML compute instance type.

Type: String

Valid Values:
- ml.m4.xlarge
- ml.m4.2xlarge
- ml.m4.4xlarge
- ml.m4.10xlarge
- ml.m4.16xlarge
- ml.m5.large
- ml.m5.xlarge
- ml.m5.2xlarge
- ml.m5.4xlarge
- ml.m5.12xlarge
- ml.m5.24xlarge
- ml.c4.xlarge
- ml.c4.2xlarge
- ml.c4.4xlarge
- ml.c4.8xlarge
- ml.p2.xlarge
- ml.p2.8xlarge
- ml.p2.16xlarge
- ml.p3.2xlarge
- ml.p3.8xlarge
- ml.p3.16xlarge
- ml.c5.xlarge
- ml.c5.2xlarge
- ml.c5.4xlarge
- ml.c5.9xlarge
- ml.c5.18xlarge

Required: Yes

VolumeKmsKeyId
The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the training job. The VolumeKmsKeyId can be any of the following formats:

- // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"

- // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

Type: String

Length Constraints: Maximum length of 2048.

Required: No

VolumeSizeInGB
The size of the ML storage volume that you want to provision.

ML storage volumes store model artifacts and incremental states. Training algorithms might also use the ML storage volume for scratch space. If you want to store the training data in the ML storage volume, choose File as the TrainingInputMode in the algorithm specification.

You must specify sufficient ML storage for your scenario.
Note
Amazon SageMaker supports only the General Purpose SSD (gp2) ML storage volume type.

Type: Integer
Valid Range: Minimum value of 1.
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**ResourceLimits**
Service: Amazon SageMaker Service

Specifies the maximum number of training jobs and parallel training jobs that a hyperparameter tuning job can launch.

**Contents**

**MaxNumberOfTrainingJobs**

The maximum number of training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

**MaxParallelTrainingJobs**

The maximum number of concurrent training jobs that a hyperparameter tuning job can launch.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
S3DataSource
Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

AttributeNames
A list of one or more attribute names to use that are found in a specified augmented manifest file.

Type: Array of strings

Array Members: Maximum number of 16 items.

Length Constraints: Minimum length of 1. Maximum length of 256.

Required: No

S3DataDistributionType
If you want Amazon SageMaker to replicate the entire dataset on each ML compute instance that is launched for model training, specify FullyReplicated.

If you want Amazon SageMaker to replicate a subset of data on each ML compute instance that is launched for model training, specify ShardedByS3Key. If there are n ML compute instances launched for a training job, each instance gets approximately 1/n of the number of S3 objects. In this case, model training on each machine uses only the subset of training data.

Don't choose more ML compute instances for training than available S3 objects. If you do, some nodes won't get any data and you will pay for nodes that aren't getting any training data. This applies in both File and Pipe modes. Keep this in mind when developing algorithms.

In distributed training, where you use multiple ML compute EC2 instances, you might choose ShardedByS3Key. If the algorithm requires copying training data to the ML storage volume (when TrainingInputMode is set to File), this copies 1/n of the number of objects.

Type: String

Valid Values: FullyReplicated | ShardedByS3Key

Required: No

S3DataType
If you choose S3Prefix, S3Uri identifies a key name prefix. Amazon SageMaker uses all objects that match the specified key name prefix for model training.

If you choose ManifestFile, S3Uri identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for model training.

If you choose AugmentedManifestFile, S3Uri identifies an object that is an augmented manifest file in JSON lines format. This file contains the data you want to use for model training. AugmentedManifestFile can only be used if the Channel's input mode is Pipe.

Type: String

Valid Values: ManifestFile | S3Prefix | AugmentedManifestFile

Required: Yes
S3Uri

Depending on the value specified for the S3DataType, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: s3://bucketname/exampleprefix.
- A manifest might look like this: s3://bucketname/example.manifest

The manifest is an S3 object which is a JSON file with the following format:

```
[
    "prefix": "s3://customer_bucket/some/prefix/",
    "relative/path/to/custdata-1",
    "relative/path/custdata-2",
    ...
]
```

The preceding JSON matches the following s3Uris:

```
s3://customer_bucket/some/prefix/relative/path/to/custdata-1
s3://customer_bucket/some/prefix/relative/path/custdata-2
...
```

The complete set of s3uris in this manifest is the input data for the channel for this datasource. The object that each s3uris points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/?(.*)$

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**SearchExpression**
Service: Amazon SageMaker Service

A multi-expression that searches for the specified resource or resources in a search. All resource objects that satisfy the expression's condition are included in the search results. You must specify at least one subexpression, filter, or nested filter. A `SearchExpression` can contain up to twenty elements.

A `SearchExpression` contains the following components:

- A list of `Filter` objects. Each filter defines a simple Boolean expression comprised of a resource property name, Boolean operator, and value.
- A list of `NestedFilter` objects. Each nested filter defines a list of Boolean expressions using a list of resource properties. A nested filter is satisfied if a single object in the list satisfies all Boolean expressions.
- A list of `SearchExpression` objects. A search expression object can be nested in a list of search expression objects.
- A Boolean operator: `And` or `Or`.

**Contents**

**Filters**

A list of filter objects.

Type: Array of `Filter` (p. 780) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

**NestedFilters**

A list of nested filter objects.

Type: Array of `NestedFilters` (p. 843) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.

Required: No

**Operator**

A Boolean operator used to evaluate the search expression. If you want every conditional statement in all lists to be satisfied for the entire search expression to be true, specify `And`. If only a single conditional statement needs to be true for the entire search expression to be true, specify `Or`. The default value is `And`.

Type: String

Valid Values: `And` | `Or`

Required: No

**SubExpressions**

A list of search expression objects.

Type: Array of `SearchExpression` (p. 871) objects

Array Members: Minimum number of 1 item. Maximum number of 20 items.
Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SearchRecord
Service: Amazon SageMaker Service

An individual search result record that contains a single resource object.

Contents

TrainingJob

A TrainingJob object that is returned as part of a Search request.

Type: TrainingJob (p. 884) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SecondaryStatusTransition
Service: Amazon SageMaker Service

An array element of DescribeTrainingJob:SecondaryStatusTransitions (p. 633). It provides additional details about a status that the training job has transitioned through. A training job can be in one of several states, for example, starting, downloading, training, or uploading. Within each state, there are a number of intermediate states. For example, within the starting state, Amazon SageMaker could be starting the training job or launching the ML instances. These transitional states are referred to as the job's secondary status.

Contents

EndTime
A timestamp that shows when the training job transitioned out of this secondary status state into another secondary status state or when the training job has ended.

Type: Timestamp
Required: No

StartTime
A timestamp that shows when the training job transitioned to the current secondary status state.

Type: Timestamp
Required: Yes

Status
Contains a secondary status information from a training job.

Status might be one of the following secondary statuses:

InProgress
• Starting - Starting the training job.
• Downloading - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
• Training - Training is in progress.
• Uploading - Training is complete and the model artifacts are being uploaded to the S3 location.

Completed
• Completed - The training job has completed.

Failed
• Failed - The training job has failed. The reason for the failure is returned in the FailureReason field of DescribeTrainingJobResponse.

Stopped
• MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed runtime.
• Stopped - The training job has stopped.

Stopping
• Stopping - Stopping the training job.

We no longer support the following secondary statuses:
• LaunchingMLInstances
• PreparingTrainingStack
• **DownloadingTrainingImage**

  **Type:** String

  **Valid Values:** Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed

  **Required:** Yes

**StatusMessage**

A detailed description of the progress within a secondary status.

Amazon SageMaker provides secondary statuses and status messages that apply to each of them:

**Starting**

- Starting the training job.
- Launching requested ML instances.
- Insufficient capacity error from EC2 while launching instances, retrying!
- Launched instance was unhealthy, replacing it!
- Preparing the instances for training.

**Training**

- Downloading the training image.
- Training image download completed. Training in progress.

**Important**

Status messages are subject to change. Therefore, we recommend not including them in code that programmatically initiates actions. For examples, don't use status messages in if statements.

To have an overview of your training job's progress, view **TrainingJobStatus** and **SecondaryStatus** in `DescribeTrainingJobResponse` and **StatusMessage** together. For example, at the start of a training job, you might see the following:

- **TrainingJobStatus** - InProgress
- **SecondaryStatus** - Training
- **StatusMessage** - Downloading the training image

**Type:** String

**Required:** No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
ShuffleConfig
Service: Amazon SageMaker Service

A configuration for a shuffle option for input data in a channel. If you use S3Prefix for S3DataType, the results of the S3 key prefix matches are shuffled. If you use ManifestFile, the order of the S3 object references in the ManifestFile is shuffled. If you use AugmentedManifestFile, the order of the JSON lines in the AugmentedManifestFile is shuffled. The shuffling order is determined using the Seed value.

For Pipe input mode, shuffling is done at the start of every epoch. With large datasets, this ensures that the order of the training data is different for each epoch, and it helps reduce bias and possible overfitting. In a multi-node training job when ShuffleConfig is combined with S3DataDistributionType of ShardedByS3Key, the data is shuffled across nodes so that the content sent to a particular node on the first epoch might be sent to a different node on the second epoch.

Contents

Seed
Determines the shuffling order in ShuffleConfig value.

Type: Long
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SourceAlgorithm
Service: Amazon SageMaker Service

Specifies an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Contents

AlgorithmName

The name of an algorithm that was used to create the model package. The algorithm must be either an algorithm resource in your Amazon SageMaker account or an algorithm in AWS Marketplace that you are subscribed to.

Type: String


Pattern: (arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:[a-z\-]*\//)?([a-zA-Z0-9\[\[a-zA-Z0-9\-\]\]\){0,62}\(\?<!-\)#

Required: Yes

ModelDataUrl

The Amazon S3 path where the model artifacts, which result from model training, are stored. This path must point to a single gzip compressed tar archive (.tar.gz suffix).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3://(^[^/]+)?(/.*))$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SourceAlgorithmSpecification
Service: Amazon SageMaker Service

A list of algorithms that were used to create a model package.

Contents

SourceAlgorithms

A list of the algorithms that were used to create a model package.

Type: Array of SourceAlgorithm (p. 877) objects

Array Members: Fixed number of 1 item.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
StoppingCondition
Service: Amazon SageMaker Service

Specifies how long model training can run. When model training reaches the limit, Amazon SageMaker ends the training job. Use this API to cap model training cost.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts, so the results of training is not lost.

Training algorithms provided by Amazon SageMaker automatically saves the intermediate results of a model training job (it is best effort case, as model might not be ready to save as some stages, for example training just started). This intermediate data is a valid model artifact. You can use it to create a model (CreateModel).

Contents

MaxRuntimeInSeconds

The maximum length of time, in seconds, that the training job can run. If model training does not complete during this time, Amazon SageMaker ends the job. If value is not specified, default value is 1 day. Maximum value is 28 days.

Type: Integer

Valid Range: Minimum value of 1.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SubscribedWorkteam
Service: Amazon SageMaker Service

Describes a work team of a vendor that does the labelling job.

Contents

ListingId
Type: String
Required: No

MarketplaceDescription
The description of the vendor from the Amazon Marketplace.
Type: String
Required: No

MarketplaceTitle
The title of the service provided by the vendor in the Amazon Marketplace.
Type: String
Required: No

SellerName
The name of the vendor in the Amazon Marketplace.
Type: String
Required: No

WorkteamArn
The Amazon Resource Name (ARN) of the vendor that you have subscribed.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*
Required: Yes

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
SuggestionQuery
Service: Amazon SageMaker Service

Limits the property names that are included in the response.

Contents

PropertyNameQuery

A type of SuggestionQuery. Defines a property name hint. Only property names that match the specified hint are included in the response.

Type: PropertyNameQuery (p. 860) object

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Tag
Service: Amazon SageMaker Service

Describes a tag.

Contents

Key

The tag key.
Type: String
Pattern: ^([\p{L}\p{Z}\p{N}_.:/=+-@]*)$
Required: Yes

Value

The tag value.
Type: String
Length Constraints: Minimum length of 0. Maximum length of 256.
Pattern: ^([\p{L}\p{Z}\p{N}_.:/=+-@]*)$
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJob
Service: Amazon SageMaker Service
Contains information about a training job.

Contents

AlgorithmSpecification
Information about the algorithm used for training, and algorithm metadata.
Type: AlgorithmSpecification (p. 744) object
Required: No

CreationTime
A timestamp that indicates when the training job was created.
Type: Timestamp
Required: No

EnableNetworkIsolation
If the TrainingJob was created with network isolation, the value is set to true. If network isolation is enabled, nodes can't communicate beyond the VPC they run in.
Type: Boolean
Required: No

FailureReason
If the training job failed, the reason it failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

FinalMetricDataList
A list of final metric values that are set when the training job completes. Used only if the training job was configured to use metrics.
Type: Array of MetricData (p. 831) objects
Array Members: Minimum number of 0 items. Maximum number of 20 items.
Required: No

HyperParameters
Algorithm-specific parameters.
Type: String to string map
Key Length Constraints: Maximum length of 256.
Value Length Constraints: Maximum length of 256.
Required: No
InputDataConfig
An array of Channel objects that describes each data input channel.
Type: Array of Channel (p. 756) objects
Array Members: Minimum number of 1 item. Maximum number of 8 items.
Required: No

LabelingJobArn
The Amazon Resource Name (ARN) of the labeling job.
Type: String
Length Constraints: Maximum length of 2048.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:labeling-job/.*
Required: No

LastModifiedTime
A timestamp that indicates when the status of the training job was last modified.
Type: Timestamp
Required: No

ModelArtifacts
Information about the Amazon S3 location that is configured for storing model artifacts.
Type: ModelArtifacts (p. 833) object
Required: No

OutputDataConfig
The S3 path where model artifacts that you configured when creating the job are stored. Amazon SageMaker creates subfolders for model artifacts.
Type: OutputDataConfig (p. 851) object
Required: No

ResourceConfig
Resources, including ML compute instances and ML storage volumes, that are configured for model training.
Type: ResourceConfig (p. 866) object
Required: No

RoleArn
The AWS Identity and Access Management (IAM) role configured for the training job.
Type: String
Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z0-9\-_@=\.,]+$/
Required: No

SecondaryStatus

Provides detailed information about the state of the training job. For detailed information about the secondary status of the training job, see StatusMessage under SecondaryStatusTransition (p. 874).

Amazon SageMaker provides primary statuses and secondary statuses that apply to each of them:

InProgress
- Starting - Starting the training job.
- Downloading - An optional stage for algorithms that support File training input mode. It indicates that data is being downloaded to the ML storage volumes.
- Training - Training is in progress.
- Uploading - Training is complete and the model artifacts are being uploaded to the S3 location.

Completed
- Completed - The training job has completed.

Failed
- Failed - The training job has failed. The reason for the failure is returned in the FailureReason field of DescribeTrainingJobResponse.

Stopped
- MaxRuntimeExceeded - The job stopped because it exceeded the maximum allowed runtime.
- Stopped - The training job has stopped.

Stopping
- Stopping - Stopping the training job.

Important
Valid values for SecondaryStatus are subject to change.

We no longer support the following secondary statuses:
- LaunchingMLInstances
- PreparingTrainingStack
- DownloadingTrainingImage

Type: String

Valid Values: Starting | LaunchingMLInstances | PreparingTrainingStack | Downloading | DownloadingTrainingImage | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed

Required: No

SecondaryStatusTransitions

A history of all of the secondary statuses that the training job has transitioned through.

Type: Array of SecondaryStatusTransition (p. 874) objects

Required: No

StoppingCondition

The condition under which to stop the training job.

Type: StoppingCondition (p. 879) object

Required: No
Tags

An array of key-value pairs. For more information, see Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 883) objects

Array Members: Minimum number of 0 items. Maximum number of 50 items.

Required: No

TrainingEndTime

Indicates the time when the training job ends on training instances. You are billed for the time interval between the value of TrainingStartTime and this time. For successful jobs and stopped jobs, this is the time after model artifacts are uploaded. For failed jobs, this is the time when Amazon SageMaker detects a job failure.

Type: Timestamp

Required: No

TrainingJobArn

The Amazon Resource Name (ARN) of the training job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:training-job/.*

Required: No

TrainingJobName

The name of the training job.

Type: String


Pattern: ^[a-zA-Z0-9\-]*\([a-zA-Z0-9\-]([a-zA-Z0-9\-])*$

Required: No

TrainingJobStatus

The status of the training job.

Training job statuses are:
- InProgress - The training is in progress.
- Completed - The training job has completed.
- Failed - The training job has failed. To see the reason for the failure, see the FailureReason field in the response to a DescribeTrainingJobResponse call.
- Stopping - The training job is stopping.
- Stopped - The training job has stopped.

For more detailed information, see SecondaryStatus.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: No

**TrainingStartTime**

Indicates the time when the training job starts on training instances. You are billed for the time interval between this time and the value of **TrainingEndTime**. The start time in CloudWatch Logs might be later than this time. The difference is due to the time it takes to download the training data and to the size of the training container.

Type: Timestamp

Required: No

**TuningJobArn**

The Amazon Resource Name (ARN) of the associated hyperparameter tuning job if the training job was launched by a hyperparameter tuning job.

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:hyper-parameter-tuning-job/.*`

Required: No

**VpcConfig**

A [VpcConfig](p. 913) object that specifies the VPC that this training job has access to. For more information, see [Protect Training Jobs by Using an Amazon Virtual Private Cloud](p. 913).

Type: [VpcConfig](p. 913) object

Required: No

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobDefinition
Service: Amazon SageMaker Service

Defines the input needed to run a training job using the algorithm.

Contents

HyperParameters

The hyperparameters used for the training job.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

Required: No

InputDataConfig

An array of Channel objects, each of which specifies an input source.

Type: Array of Channel (p. 756) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: Yes

OutputDataConfig

the path to the S3 bucket where you want to store model artifacts. Amazon SageMaker creates subfolders for the artifacts.

Type: OutputDataConfig (p. 851) object

Required: Yes

ResourceConfig

The resources, including the ML compute instances and ML storage volumes, to use for model training.

Type: ResourceConfig (p. 866) object

Required: Yes

StoppingCondition

Sets a duration for training. Use this parameter to cap model training costs.

To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts.

Type: StoppingCondition (p. 879) object

Required: Yes

TrainingInputMode

The input mode used by the algorithm for the training job. For the input modes that Amazon SageMaker algorithms support, see Algorithms.
If an algorithm supports the File input mode, Amazon SageMaker downloads the training data from S3 to the provisioned ML storage Volume, and mounts the directory to docker volume for training container. If an algorithm supports the Pipe input mode, Amazon SageMaker streams data directly from S3 to the container.

Type: String

Valid Values: Pipe | File

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobStatusCounters
Service: Amazon SageMaker Service

The numbers of training jobs launched by a hyperparameter tuning job, categorized by status.

Contents

Completed
The number of completed training jobs launched by the hyperparameter tuning job.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

InProgress
The number of in-progress training jobs launched by a hyperparameter tuning job.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

NonRetryableError
The number of training jobs that failed and can't be retried. A failed training job can't be retried if it failed because a client error occurred.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

RetryableError
The number of training jobs that failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

Stopped
The number of training jobs launched by a hyperparameter tuning job that were manually stopped.
Type: Integer
Valid Range: Minimum value of 0.
Required: No

See Also
For more information about using this API in one of the language-specific AWS SDKs, see the following:
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingJobSummary
Service: Amazon SageMaker Service

Provides summary information about a training job.

Contents

CreationTime
A timestamp that shows when the training job was created.
Type: Timestamp
Required: Yes

LastModifiedTime
Timestamp when the training job was last modified.
Type: Timestamp
Required: No

TrainingEndTime
A timestamp that shows when the training job ended. This field is set only if the training job has one of the terminal statuses (Completed, Failed, or Stopped).
Type: Timestamp
Required: No

TrainingJobArn
The Amazon Resource Name (ARN) of the training job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-zA-Z\-]*:sagemaker:[a-zA-Z0-9\-]*:[0-9]{12}:training-job/.*
Required: Yes

TrainingJobName
The name of the training job that you want a summary for.
Type: String
Pattern: ^[a-zA-Z0-9\-]([-\[a-zA-Z0-9\-]*[a-zA-Z0-9\-]]*)
Required: Yes

TrainingJobStatus
The status of the training job.
Type: String
Valid Values: InProgress | Completed | Failed | Stopping | Stopped
Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TrainingSpecification
Service: Amazon SageMaker Service

Defines how the algorithm is used for a training job.

Contents

MetricDefinitions

A list of MetricDefinition objects, which are used for parsing metrics generated by the algorithm.

Type: Array of MetricDefinition (p. 832) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

SupportedHyperParameters

A list of the HyperParameterSpecification objects, that define the supported hyperparameters. This is required if the algorithm supports automatic model tuning.

Type: Array of HyperParameterSpecification (p. 791) objects

Array Members: Minimum number of 0 items. Maximum number of 100 items.

Required: No

SupportedTrainingInstanceTypes

A list of the instance types that this algorithm can use for training.

Type: Array of strings

Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge |
| ml.m4.16xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge |
| ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge | ml.m5.xlarge |
| ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge |
| ml.p2.16xlarge | ml.p2.32xlarge | ml.p2.8xlarge |
| ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
| ml.c5.18xlarge

Required: Yes

SupportedTuningJobObjectiveMetrics

A list of the metrics that the algorithm emits that can be used as the objective metric in a hyperparameter tuning job.

Type: Array of HyperParameterTuningJobObjective (p. 801) objects

Required: No

SupportsDistributedTraining

Indicates whether the algorithm supports distributed training. If set to false, buyers can't request more than one instance during training.

Type: Boolean

Required: No
TrainingChannels

A list of ChannelSpecification objects, which specify the input sources to be used by the algorithm.

Type: Array of ChannelSpecification (p. 758) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.

Required: Yes

TrainingImage

The Amazon ECR registry path of the Docker image that contains the training algorithm.

Type: String

Length Constraints: Maximum length of 255.

Pattern: \S+

Required: Yes

TrainingImageDigest

An MD5 hash of the training algorithm that identifies the Docker image used for training.

Type: String

Length Constraints: Maximum length of 72.

Pattern: ^[Ss][Hh][Aa]256:[0-9a-fA-F]{64}$

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TransformDataSource**

Service: Amazon SageMaker Service

Describes the location of the channel data.

**Contents**

**S3DataSource**

The S3 location of the data source that is associated with a channel.

Type: TransformS3DataSource (p. 908) object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformInput
Service: Amazon SageMaker Service

Describes the input source of a transform job and the way the transform job consumes it.

Contents

CompressionType

If your transform data is compressed, specify the compression type. Amazon SageMaker automatically decompresses the data for the transform job accordingly. The default value is None.

Type: String

Valid Values: None | Gzip

Required: No

ContentType

The multipurpose internet mail extension (MIME) type of the data. Amazon SageMaker uses the MIME type with each http call to transfer data to the transform job.

Type: String

Length Constraints: Maximum length of 256.

Required: No

DataSource

Describes the location of the channel data, which is, the S3 location of the input data that the model can consume.

Type: TransformDataSource (p. 897) object

Required: Yes

SplitType

The method to use to split the transform job's data files into smaller batches. Splitting is necessary when the total size of each object is too large to fit in a single request. You can also use data splitting to improve performance by processing multiple concurrent mini-batches. The default value for SplitType is None, which indicates that input data files are not split, and request payloads contain the entire contents of an input object. Set the value of this parameter to Line to split records on a newline character boundary. SplitType also supports a number of record-oriented binary data formats.

When splitting is enabled, the size of a mini-batch depends on the values of the BatchStrategy and MaxPayloadInMB parameters. When the value of BatchStrategy is MultiRecord, Amazon SageMaker sends the maximum number of records in each request, up to the MaxPayloadInMB limit. If the value of BatchStrategy is SingleRecord, Amazon SageMaker sends individual records in each request.

Note

Some data formats represent a record as a binary payload wrapped with extra padding bytes. When splitting is applied to a binary data format, padding is removed if the value of BatchStrategy is set to SingleRecord. Padding is not removed if the value of BatchStrategy is set to MultiRecord.

For more information about the RecordIO data format, see Data Format in the MXNet documentation. For more information about the TFRecord format, see Consuming TFRecord data in the TensorFlow documentation.
Type: String

Valid Values: None | Line | RecordIO | TFRecord

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**TransformJobDefinition**
Service: Amazon SageMaker Service

Defines the input needed to run a transform job using the inference specification specified in the algorithm.

**Contents**

**BatchStrategy**

A string that determines the number of records included in a single mini-batch.

*SingleRecord* means only one record is used per mini-batch. *MultiRecord* means a mini-batch is set to contain as many records that can fit within the MaxPayloadInMB limit.

Type: String

Valid Values: MultiRecord | SingleRecord

Required: No

**Environment**

The environment variables to set in the Docker container. We support up to 16 key and values entries in the map.

Type: String to string map

Key Length Constraints: Maximum length of 1024.

Key Pattern: [a-zA-Z_][a-zA-Z0-9_]*

Value Length Constraints: Maximum length of 10240.

Required: No

**MaxConcurrentTransforms**

The maximum number of parallel requests that can be sent to each instance in a transform job. The default value is 1.

Type: Integer

Valid Range: Minimum value of 0.

Required: No

**MaxPayloadInMB**

The maximum payload size allowed, in MB. A payload is the data portion of a record (without metadata).

Type: Integer

Valid Range: Minimum value of 0.

Required: No

**TransformInput**

A description of the input source and the way the transform job consumes it.

Type: TransformInput (p. 898) object
Required: Yes

**TransformOutput**

Identifies the Amazon S3 location where you want Amazon SageMaker to save the results from the transform job.

Type: `TransformOutput (p. 904)` object

Required: Yes

**TransformResources**

Identifies the ML compute instances for the transform job.

Type: `TransformResources (p. 906)` object

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformJobSummary
Service: Amazon SageMaker Service

Provides a summary of a transform job. Multiple TransformJobSummary objects are returned as a list after in response to a ListTransformJobs (p. 693) call.

Contents

CreationTime
A timestamp that shows when the transform Job was created.
Type: Timestamp
Required: Yes

FailureReason
If the transform job failed, the reason it failed.
Type: String
Length Constraints: Maximum length of 1024.
Required: No

LastModifiedTime
Indicates when the transform job was last modified.
Type: Timestamp
Required: No

TransformEndTime
Indicates when the transform job ends on compute instances. For successful jobs and stopped jobs, this is the exact time recorded after the results are uploaded. For failed jobs, this is when Amazon SageMaker detected that the job failed.
Type: Timestamp
Required: No

TransformJobArn
The Amazon Resource Name (ARN) of the transform job.
Type: String
Length Constraints: Maximum length of 256.
Pattern: arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:transform-job/.*
Required: Yes

TransformJobName
The name of the transform job.
Type: String
Pattern: ^[a-zA-Z0-9\-](-*[a-zA-Z0-9])*
Required: Yes

**TransformJobStatus**

The status of the transform job.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: Yes

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformOutput
Service: Amazon SageMaker Service

Describes the results of a transform job output.

Contents

Accept
The MIME type used to specify the output data. Amazon SageMaker uses the MIME type with each http call to transfer data from the transform job.

Type: String
Length Constraints: Maximum length of 256.
Required: No

AssembleWith
Defines how to assemble the results of the transform job as a single S3 object. Choose a format that is most convenient to you. To concatenate the results in binary format, specify None. To add a newline character at the end of every transformed record, specify Line.

Type: String
Valid Values: None | Line
Required: No

KmsKeyId
The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt the model artifacts at rest using Amazon S3 server-side encryption. The KmsKeyId can be any of the following formats:

- // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"
- // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
- // KMS Key Alias
  "alias/ExampleAlias"
- // Amazon Resource Name (ARN) of a KMS Key Alias

If you don’t provide a KMS key ID, Amazon SageMaker uses the default KMS key for Amazon S3 for your role’s account. For more information, see KMS-Managed Encryption Keys in the Amazon Simple Storage Service Developer Guide.

The KMS key policy must grant permission to the IAM role that you specify in your CreateTransformJob request. For more information, see Using Key Policies in AWS KMS in the AWS Key Management Service Developer Guide.

Type: String
Length Constraints: Maximum length of 2048.
**S3OutputPath**

The Amazon S3 path where you want Amazon SageMaker to store the results of the transform job. For example, s3://bucket-name/key-name-prefix.

For every S3 object used as input for the transform job, the transformed data is stored in a corresponding subfolder in the location under the output prefix. For example, for the input data s3://bucket-name/input-name-prefix/dataset01/data.csv the transformed data is stored at s3://bucket-name/key-name-prefix/dataset01/. This is based on the original name, as a series of .part files (.part0001, part0002, etc.).

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+)/(.*$)

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
TransformResources
Service: Amazon SageMaker Service

Describes the resources, including ML instance types and ML instance count, to use for transform job.

Contents

InstanceCount

The number of ML compute instances to use in the transform job. For distributed transform, provide a value greater than 1. The default value is 1.

Type: Integer

Valid Range: Minimum value of 1.

Required: Yes

InstanceType

The ML compute instance type for the transform job. For using built-in algorithms to transform moderately sized datasets, ml.m4.xlarge or ml.m5.large should suffice. There is no default value for InstanceType.

Type: String

Valid Values: ml.m4.xlarge | ml.m4.2xlarge | ml.m4.4xlarge | ml.m4.10xlarge | ml.m4.16xlarge | ml.c4.xlarge | ml.c4.2xlarge | ml.c4.4xlarge | ml.c4.8xlarge | ml.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge | ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge | ml.m5.large | ml.m5.xlarge | ml.m5.2xlarge | ml.m5.4xlarge | ml.m5.12xlarge | ml.m5.24xlarge

Required: Yes

VolumeKmsKeyId

The AWS Key Management Service (AWS KMS) key that Amazon SageMaker uses to encrypt data on the storage volume attached to the ML compute instance(s) that run the batch transform job. The VolumeKmsKeyId can be any of the following formats:

• // KMS Key ID
  "1234abcd-12ab-34cd-56ef-1234567890ab"

• // Amazon Resource Name (ARN) of a KMS Key
  "arn:aws:kms:us-west-2:11112223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

Type: String

Length Constraints: Maximum length of 2048.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

• AWS SDK for C++
• AWS SDK for Go
• AWS SDK for Java
• AWS SDK for Ruby V2
TransformS3DataSource
Service: Amazon SageMaker Service

Describes the S3 data source.

Contents

S3DataType

If you choose S3Prefix, S3Uri identifies a key name prefix. Amazon SageMaker uses all objects with the specified key name prefix for batch transform.

If you choose ManifestFile, S3Uri identifies an object that is a manifest file containing a list of object keys that you want Amazon SageMaker to use for batch transform.

Type: String

Valid Values: ManifestFile | S3Prefix | AugmentedManifestFile

Required: Yes

S3Uri

Depending on the value specified for the S3DataType, identifies either a key name prefix or a manifest. For example:

- A key name prefix might look like this: s3://bucketname/exampleprefix.
- A manifest might look like this: s3://bucketname/example.manifest

The manifest is an S3 object which is a JSON file with the following format:

```
[
  {"prefix": "s3://customer_bucket/some/prefix/"},
  "relative/path/to/custdata-1",
  "relative/path/custdata-2",
  ...
]
```

The preceding JSON matches the following S3Uris:

s3://customer_bucket/some/prefix/relative/path/to/custdata-1
s3://customer_bucket/some/prefix/relative/path/custdata-2
...

The complete set of S3Uris in this manifest constitutes the input data for the channel for this datasource. The object that each S3Uri points to must be readable by the IAM role that Amazon SageMaker uses to perform tasks on your behalf.

Type: String

Length Constraints: Maximum length of 1024.

Pattern: ^(https|s3)://([^/]+//)?(.*).

Required: Yes
See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UiConfig
Service: Amazon SageMaker Service

Provided configuration information for the worker UI for a labeling job.

Contents

UiTemplateS3Uri

The Amazon S3 bucket location of the UI template. For more information about the contents of a UI template, see Creating Your Custom Labeling Task Template.

Type: String
Length Constraints: Maximum length of 1024.
Pattern: ^https?://([^/]*)/(.*)$
Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
UiTemplate
Service: Amazon SageMaker Service

The Liquid template for the worker user interface.

Contents

Content

The content of the Liquid template for the worker user interface.

Type: String


Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
USD
Service: Amazon SageMaker Service

Represents an amount of money in United States dollars/

Contents

Cents

The fractional portion, in cents, of the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 99.

Required: No

Dollars

The whole number of dollars in the amount.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 1.

Required: No

TenthFractionsOfACent

Fractions of a cent, in tenths.

Type: Integer

Valid Range: Minimum value of 0. Maximum value of 9.

Required: No

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
VpcConfig
Service: Amazon SageMaker Service

Specifies a VPC that your training jobs and hosted models have access to. Control access to and from your training and model containers by configuring the VPC. For more information, see Protect Endpoints by Using an Amazon Virtual Private Cloud and Protect Training Jobs by Using an Amazon Virtual Private Cloud.

Contents

SecurityGroupIds

The VPC security group IDs, in the form sg-xxxxxxxx. Specify the security groups for the VPC that is specified in the Subnets field.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 5 items.

Length Constraints: Maximum length of 32.

Required: Yes

Subnets

The ID of the subnets in the VPC to which you want to connect your training job or model.

Type: Array of strings

Array Members: Minimum number of 1 item. Maximum number of 16 items.

Length Constraints: Maximum length of 32.

Required: Yes

See Also

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
Workteam
Service: Amazon SageMaker Service
Provides details about a labeling work team.

Contents
CreateDate
The date and time that the work team was created (timestamp).
Type: Timestamp
Required: No
Description
A description of the work team.
Type: String
Required: Yes
LastUpdatedDate
The date and time that the work team was last updated (timestamp).
Type: Timestamp
Required: No
MemberDefinitions
The Amazon Cognito user groups that make up the work team.
Type: Array of MemberDefinition (p. 830) objects
Array Members: Minimum number of 1 item. Maximum number of 10 items.
Required: Yes
ProductListingIds
The Amazon Marketplace identifier for a vendor’s work team.
Type: Array of strings
Required: No
SubDomain
The URI of the labeling job’s user interface. Workers open this URI to start labeling your data objects.
Type: String
Required: No
WorkteamArn
The Amazon Resource Name (ARN) that identifies the work team.
Type: String
Length Constraints: Maximum length of 256.

Pattern: `arn:aws[a-z\-]*:sagemaker:[a-z0-9\-]*:[0-9]{12}:workteam/.*`

Required: Yes

**WorkteamName**

The name of the work team.

Type: String


Pattern: `^[a-zA-Z0-9-][a-zA-Z0-9\-]*[a-zA-Z0-9-]*$`

Required: Yes

---

**See Also**

For more information about using this API in one of the language-specific AWS SDKs, see the following:

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2

---

**Amazon SageMaker Runtime**

Currently Amazon SageMaker Runtime does not support any data types.

---

**Common Errors**

This section lists the errors common to the API actions of all AWS services. For errors specific to an API action for this service, see the topic for that API action.

**AccessDeniedException**

You do not have sufficient access to perform this action.

HTTP Status Code: 400

**IncompleteSignature**

The request signature does not conform to AWS standards.

HTTP Status Code: 400

**InternalFailure**

The request processing has failed because of an unknown error, exception or failure.

HTTP Status Code: 500

**InvalidAction**

The action or operation requested is invalid. Verify that the action is typed correctly.
HTTP Status Code: 400

**InvalidClientTokenId**

The X.509 certificate or AWS access key ID provided does not exist in our records.

HTTP Status Code: 403

**InvalidParameterCombination**

Parameters that must not be used together were used together.

HTTP Status Code: 400

**InvalidParameterValue**

An invalid or out-of-range value was supplied for the input parameter.

HTTP Status Code: 400

**InvalidQueryParameter**

The AWS query string is malformed or does not adhere to AWS standards.

HTTP Status Code: 400

**MalformedQueryString**

The query string contains a syntax error.

HTTP Status Code: 404

**MissingAction**

The request is missing an action or a required parameter.

HTTP Status Code: 400

**MissingAuthenticationToken**

The request must contain either a valid (registered) AWS access key ID or X.509 certificate.

HTTP Status Code: 403

**MissingParameter**

A required parameter for the specified action is not supplied.

HTTP Status Code: 400

**OptInRequired**

The AWS access key ID needs a subscription for the service.

HTTP Status Code: 403

**RequestExpired**

The request reached the service more than 15 minutes after the date stamp on the request or more than 15 minutes after the request expiration date (such as for pre-signed URLs), or the date stamp on the request is more than 15 minutes in the future.

HTTP Status Code: 400

**ServiceUnavailable**

The request has failed due to a temporary failure of the server.

HTTP Status Code: 503
ThrottlingException

The request was denied due to request throttling.

HTTP Status Code: 400

ValidationError

The input fails to satisfy the constraints specified by an AWS service.

HTTP Status Code: 400

Common Parameters

The following list contains the parameters that all actions use for signing Signature Version 4 requests with a query string. Any action-specific parameters are listed in the topic for that action. For more information about Signature Version 4, see Signature Version 4 Signing Process in the Amazon Web Services General Reference.

Action

The action to be performed.

Type: string

Required: Yes

Version

The API version that the request is written for, expressed in the format YYYY-MM-DD.

Type: string

Required: Yes

X-Amz-Algorithm

The hash algorithm that you used to create the request signature.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Valid Values: AWS4-HMAC-SHA256

Required: Conditional

X-Amz-Credential

The credential scope value, which is a string that includes your access key, the date, the region you are targeting, the service you are requesting, and a termination string ("aws4_request"). The value is expressed in the following format: access_key/YYYYMMDD/region/service/aws4_request.

For more information, see Task 2: Create a String to Sign for Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string
Required: Conditional

X-Amz-Date

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: 20120325T120000Z.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the Amazon Web Services General Reference.

Type: string

Required: Conditional

X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the IAM User Guide.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional
## Document History for Amazon SageMaker

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring notebook instances</td>
<td>You can use shell scripts to configure notebook instances when you create or start them. For more information, see Customize a Notebook Instance.</td>
<td>May 1, 2018</td>
</tr>
<tr>
<td>Disable direct internet access</td>
<td>You can now disable direct internet access for notebook instances. For more information, see Notebook Instances Are Enabled with Internet Access by Default.</td>
<td>March 15, 2018</td>
</tr>
<tr>
<td>Application Auto Scaling support</td>
<td>Amazon SageMaker now supports Application Auto Scaling for production variants. For information, see Automatically Scaling Amazon SageMaker SageMaker Models</td>
<td>February 28, 2018</td>
</tr>
<tr>
<td>TensorFlow 1.5 and MXNet 1.0 support (p. 919)</td>
<td>Amazon SageMaker Deep Learning containers now support TensorFlow 1.5 and Apache MXNet 1.0.</td>
<td>February 27, 2018</td>
</tr>
<tr>
<td>BlazingText algorithm</td>
<td>Amazon SageMaker now supports the BlazingText algorithm.</td>
<td>January 18, 2018</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td><strong>KMS encryption support for training and hosting</strong></td>
<td>Amazon SageMaker now supports KMS encryption for hosting instances and training model artifacts at rest. You can specify an AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to a hosting endpoint by using the <code>KmsKeyId</code> request parameter in a call to <code>CreateEndpointConfig</code>. You can specify an AWS KMS key that Amazon SageMaker uses to encrypt training model artifacts at rest by setting the <code>KmsKeyId</code> field of the <code>OutputDataConfig</code> object you use to configure your training job.</td>
<td>January 17, 2018</td>
</tr>
<tr>
<td><strong>CloudTrail support</strong></td>
<td>Amazon SageMaker now supports logging with AWS CloudTrail.</td>
<td>January 11, 2018</td>
</tr>
<tr>
<td><strong>DeepAR Forecasting algorithm</strong></td>
<td>Amazon SageMaker now supports the DeepAR algorithm for time series forecasting.</td>
<td>January 8, 2018</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.