Amazon SageMaker
Developer Guide
# Table of Contents

What Is Amazon SageMaker? ................................................................. 1
Are You a First-time User of Amazon SageMaker? .................................... 1

How It Works .................................................................................. 2

Machine Learning with Amazon SageMaker .............................................. 2
How It Works: Next Topic ........................................................................ 4
Explore and Preprocess Data ..................................................................... 4
How It Works: Next Topic ........................................................................ 4
Model Training ..................................................................................... 4
How It Works: Next Topic ........................................................................ 6

Model Deployment ............................................................................... 6
Considerations for Deploying Models on Amazon SageMaker Hosting Services ........................................ 8
How It Works: Next Topic ........................................................................ 8
Validation Models .................................................................................. 9
How It Works: Next Topic ........................................................................ 10

Programming Model ........................................................................... 10
How It Works: Next Topic ........................................................................ 11

Getting Started .................................................................................. 12
Step 1: Setting Up .................................................................................. 12
Step 1.1: Create an Account ..................................................................... 12
Step 1.2: Create an S3 Bucket ................................................................. 13
Step 2: Create an Amazon SageMaker Notebook Instance ....................... 14
Step 2.1: (Optional) Customize a Notebook Instance .................................. 18
Next Step ............................................................................................ 18
Step 3: Train and Deploy a Model .......................................................... 19
Step 3.1: Create a Jupyter Notebook ......................................................... 20
Step 3.2: Download, Explore, and Transform Data ....................................... 21
Step 3.3: Train a Model ........................................................................... 23
Step 3.4: Deploy the Model ..................................................................... 26
Step 3.5: Validate the Model ................................................................... 28
Step 4: Clean up ..................................................................................... 32
Step 5: Additional Considerations .......................................................... 33

Automatic Model Tuning ..................................................................... 34
How Hyperparameter Tuning Works ......................................................... 34
Defining Objective Metrics ....................................................................... 35
Defining Hyperparameter Ranges ............................................................... 36
Example: Hyperparameter Tuning Job ...................................................... 37
Prerequisites ......................................................................................... 37
Create a Notebook .................................................................................. 37
Get the sagemaker boto3 Client ............................................................... 38
Get the Amazon SageMaker Execution Role .............................................. 38
Specify a Bucket and Data Output Location ............................................ 39
Download, Prepare, and Upload Training Data ........................................ 39
Configure and Launch a Hyperparameter Tuning Job ............................... 40
Monitor the Progress of a Hyperparameter Tuning Job ............................ 43
Clean up ............................................................................................... 45
Design Considerations .......................................................................... 45
Choosing the Number of Hyperparameters .............................................. 46
Choosing Hyperparameter Ranges ............................................................ 46
Use Logarithmic Scales for Hyperparameters ........................................... 46
Choosing the Best Degree of Parallelism .................................................. 46
Running Training Jobs on Multiple Instances ........................................... 46

Using Notebook Instances ................................................................... 47
Creating a Notebook Instance ................................................................. 47
Accessing Notebook Instances ................................................................. 47
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Example Notebooks</td>
<td>48</td>
</tr>
<tr>
<td>Set the Notebook Kernel</td>
<td>49</td>
</tr>
<tr>
<td>Using Built-in Algorithms</td>
<td>50</td>
</tr>
<tr>
<td>Common Information</td>
<td>52</td>
</tr>
<tr>
<td>Linear Learner</td>
<td>63</td>
</tr>
<tr>
<td>Factorization Machines</td>
<td>73</td>
</tr>
<tr>
<td>XGBoost Algorithm</td>
<td>81</td>
</tr>
<tr>
<td>Image Classification Algorithm</td>
<td>89</td>
</tr>
<tr>
<td>Sequence to Sequence (seq2seq)</td>
<td>96</td>
</tr>
<tr>
<td>K-Means Algorithm</td>
<td>106</td>
</tr>
<tr>
<td>Principal Component Analysis (PCA)</td>
<td>113</td>
</tr>
<tr>
<td>Latent Dirichlet Allocation (LDA)</td>
<td>116</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Automatically Scaling Amazon SageMaker Models</td>
<td>154</td>
</tr>
<tr>
<td>Related Topics</td>
<td>157</td>
</tr>
<tr>
<td>Configure Automatic Scaling for a Variant</td>
<td>157</td>
</tr>
<tr>
<td>Editing a Scaling Policy</td>
<td>163</td>
</tr>
<tr>
<td>Deleting a Scaling Policy</td>
<td>164</td>
</tr>
<tr>
<td>Using Your Own Algorithms</td>
<td>146</td>
</tr>
<tr>
<td>How Amazon SageMaker Runs Your Training Image</td>
<td>147</td>
</tr>
<tr>
<td>How Amazon SageMaker Provides Training Information</td>
<td>148</td>
</tr>
<tr>
<td>Signalling Algorithm Success and Failure</td>
<td>150</td>
</tr>
<tr>
<td>How Amazon SageMaker Processes Training Output</td>
<td>150</td>
</tr>
<tr>
<td>Next Step</td>
<td>151</td>
</tr>
<tr>
<td>How Amazon SageMaker Runs Your Inference Image</td>
<td>151</td>
</tr>
<tr>
<td>How Amazon SageMaker Loads Your Model Artifacts</td>
<td>152</td>
</tr>
<tr>
<td>How Containers Serve Requests</td>
<td>152</td>
</tr>
<tr>
<td>How Your Container Should Respond to Inference Requests</td>
<td>152</td>
</tr>
<tr>
<td>How Your Container Should Respond to Health Check (Ping) Requests</td>
<td>153</td>
</tr>
<tr>
<td>Example: Using Your Own Algorithms</td>
<td>153</td>
</tr>
<tr>
<td>Inference Formats</td>
<td>121</td>
</tr>
<tr>
<td>Neural Topic Model (NTM)</td>
<td>120</td>
</tr>
<tr>
<td>Input/Output Interface</td>
<td>121</td>
</tr>
<tr>
<td>EC2 Instance Recommendation</td>
<td>122</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>122</td>
</tr>
<tr>
<td>Model Tuning</td>
<td>123</td>
</tr>
<tr>
<td>Inference Formats</td>
<td>125</td>
</tr>
<tr>
<td>Random Cut Forest</td>
<td>134</td>
</tr>
<tr>
<td>Input/Output Interface</td>
<td>134</td>
</tr>
<tr>
<td>EC2 Instance Recommendation</td>
<td>135</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>135</td>
</tr>
<tr>
<td>Model Tuning</td>
<td>136</td>
</tr>
<tr>
<td>Inference Formats</td>
<td>137</td>
</tr>
<tr>
<td>BlazingText</td>
<td>137</td>
</tr>
<tr>
<td>Input/Output Interface</td>
<td>138</td>
</tr>
<tr>
<td>Instance Recommendations</td>
<td>138</td>
</tr>
<tr>
<td>How It Works</td>
<td>139</td>
</tr>
<tr>
<td>Hyperparameters</td>
<td>141</td>
</tr>
<tr>
<td>Model Tuning</td>
<td>142</td>
</tr>
<tr>
<td>Inference Formats</td>
<td>143</td>
</tr>
<tr>
<td>Using Your Own Algorithms</td>
<td>146</td>
</tr>
<tr>
<td>How Amazon SageMaker Runs Your Training Image</td>
<td>147</td>
</tr>
<tr>
<td>How Amazon SageMaker Provides Training Information</td>
<td>148</td>
</tr>
<tr>
<td>Signalling Algorithm Success and Failure</td>
<td>150</td>
</tr>
<tr>
<td>How Amazon SageMaker Processes Training Output</td>
<td>150</td>
</tr>
<tr>
<td>Next Step</td>
<td>151</td>
</tr>
<tr>
<td>How Amazon SageMaker Runs Your Inference Image</td>
<td>151</td>
</tr>
<tr>
<td>How Amazon SageMaker Loads Your Model Artifacts</td>
<td>152</td>
</tr>
<tr>
<td>How Containers Serve Requests</td>
<td>152</td>
</tr>
<tr>
<td>How Your Container Should Respond to Inference Requests</td>
<td>152</td>
</tr>
<tr>
<td>How Your Container Should Respond to Health Check (Ping) Requests</td>
<td>153</td>
</tr>
<tr>
<td>Example: Using Your Own Algorithms</td>
<td>153</td>
</tr>
<tr>
<td>Automatically Scaling Amazon SageMaker Models</td>
<td>154</td>
</tr>
<tr>
<td>Automatic Scaling Components</td>
<td>154</td>
</tr>
<tr>
<td>Required Permissions for Automatic Scaling</td>
<td>155</td>
</tr>
<tr>
<td>Service-Linked Role</td>
<td>155</td>
</tr>
<tr>
<td>Target Metric</td>
<td>156</td>
</tr>
<tr>
<td>Minimum and Maximum Capacity</td>
<td>156</td>
</tr>
<tr>
<td>Cooldown Period</td>
<td>156</td>
</tr>
<tr>
<td>Before You Begin</td>
<td>157</td>
</tr>
<tr>
<td>Related Topics</td>
<td>157</td>
</tr>
<tr>
<td>Configure Automatic Scaling for a Variant</td>
<td>157</td>
</tr>
<tr>
<td>Configure Automatic Scaling for a Variant (Console)</td>
<td>157</td>
</tr>
<tr>
<td>Configure Automatic Scaling for a Variant (AWS CLI or the Application Auto Scaling API)</td>
<td>158</td>
</tr>
<tr>
<td>Editing a Scaling Policy</td>
<td>163</td>
</tr>
<tr>
<td>Editing a Scaling Policy (Console)</td>
<td>163</td>
</tr>
<tr>
<td>Editing a Scaling Policy (AWS CLI or Application Auto Scaling API)</td>
<td>164</td>
</tr>
<tr>
<td>Deleting a Scaling Policy</td>
<td>164</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Notebook Instances Are Enabled with Internet Access by Default</td>
<td>229</td>
</tr>
<tr>
<td>Notebook Instances Provide the Best Experience for a Single User</td>
<td>229</td>
</tr>
<tr>
<td>Saving Updated Sample Notebooks in a New Location</td>
<td>230</td>
</tr>
<tr>
<td>Installing External Libraries and Kernels in Notebook Instances</td>
<td>230</td>
</tr>
<tr>
<td>Protect Training Jobs by Using an Amazon Virtual Private Cloud</td>
<td>232</td>
</tr>
<tr>
<td>Configuring a Training Job for Amazon VPC Access</td>
<td>232</td>
</tr>
<tr>
<td>Configuring Your Private VPC for Amazon SageMaker Training</td>
<td>232</td>
</tr>
<tr>
<td>Protect Models by Using an Amazon Virtual Private Cloud</td>
<td>234</td>
</tr>
<tr>
<td>Configuring a Model for Amazon VPC Access</td>
<td>235</td>
</tr>
<tr>
<td>Configuring Your Private VPC for Amazon SageMaker Hosting</td>
<td>235</td>
</tr>
<tr>
<td>Supported Versions</td>
<td>237</td>
</tr>
<tr>
<td>API Reference</td>
<td>238</td>
</tr>
<tr>
<td>Actions</td>
<td>238</td>
</tr>
<tr>
<td>Amazon SageMaker Service</td>
<td>239</td>
</tr>
<tr>
<td>Amazon SageMaker Runtime</td>
<td>346</td>
</tr>
<tr>
<td>Data Types</td>
<td>349</td>
</tr>
<tr>
<td>Amazon SageMaker Service</td>
<td>350</td>
</tr>
<tr>
<td>Amazon SageMaker Runtime</td>
<td>400</td>
</tr>
<tr>
<td>Common Errors</td>
<td>400</td>
</tr>
<tr>
<td>Common Parameters</td>
<td>402</td>
</tr>
<tr>
<td>Document History</td>
<td>405</td>
</tr>
<tr>
<td>AWS Glossary</td>
<td>407</td>
</tr>
</tbody>
</table>
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.

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 It 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 Getting Started (p. 12)** – 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 3: Train a Model with a Built-in Algorithm and Deploy It (p. 19).

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 TensorFlow or Apache MXNet scripts to train models. For an example, see TensorFlow Example 1: Using the tf.estimator (p. 172) and Apache MXNet Example 1: Using the Module API (p. 184).
   - **Use Amazon SageMaker directly from Apache Spark** – For information, see Using Apache Spark with Amazon SageMaker (p. 193).
   - **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 Using Your Own Algorithms with Amazon SageMaker (p. 146) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.

5. **See the API Reference (p. 238)** – This section describes the Amazon SageMaker API operations.
How It 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)
- Training a Model with Amazon SageMaker (p. 4)
- Deploying a Model on Amazon SageMaker Hosting Services (p. 6)
- Validating Machine Learning Models (p. 9)
- The Amazon SageMaker Programming Model (p. 10)

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:
   
   • **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 Using Built-in Algorithms with Amazon SageMaker (p. 50).

   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 Training a Model with Amazon SageMaker (p. 4).

   • **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 Deploying a Model on Amazon SageMaker Hosting Services (p. 6).

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 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 22).

To preprocess data, 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 Using Notebook Instances (p. 47)

Training 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 Algorithms Provided by Amazon SageMaker: Common Parameters (p. 52).

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 Using Built-in Algorithms with Amazon SageMaker (p. 50). To try an exercise that uses an algorithm provided by Amazon SageMaker, see Getting Started (p. 12).

- **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 Using Apache Spark with Amazon SageMaker (p. 193).

- **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 Using TensorFlow with Amazon SageMaker (p. 168) and Using Apache MXNet with Amazon SageMaker (p. 178).

- **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 Using Your Own Algorithms with Amazon SageMaker (p. 146).

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. 265) API.

**How It Works: Next Topic**

Deploying a Model on Amazon SageMaker Hosting Services (p. 6)

**Deploying 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. 253)` 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. 246)` 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. 243)` API. To get inferences from
the model, client applications send requests to the Amazon SageMaker Runtime HTTPS endpoint. For more information about the API, see the InvokeEndpoint (p. 347) API.

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.

**Considerations 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. 55).

- 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. 385).

- You can configure a ProductionVariant to use Application Auto Scaling. For information about configuring automatic scaling, see Automatically Scaling Amazon SageMaker Models (p. 154).

- 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. 339) and UpdateEndpointWeightsAndCapacities (p. 341).

- 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.

**How It Works: Next Topic**

Validating Machine Learning Models (p. 9)
Validating Machine Learning Models

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.
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. 238).

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 TensorFlow or Apache MXNet 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 Using Apache MXNet with Amazon SageMaker (p. 178) and Using TensorFlow with Amazon SageMaker (p. 168).

  - **The AWS SDK**—The SDKs provide methods that correspond to the Amazon SageMaker API (see Actions (p. 238)). Use the SDKs to programatically 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.

In Getting Started (p. 12), 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 Getting Started (p. 12). For more information about these libraries, see Amazon SageMaker Libraries (p. 202).
• 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 Using Apache Spark with Amazon SageMaker (p. 193).

How It Works: Next Topic

Getting Started (p. 12)
Getting Started

In this section, you set up an AWS account, 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 It Works (p. 2) before creating your notebook instance. For general information about notebook instances, see Explore and Preprocess Data (p. 4).

Topics
• Step 1: Setting Up (p. 12)
• Step 2: Create an Amazon SageMaker Notebook Instance (p. 14)
• Step 3: Train a Model with a Built-in Algorithm and Deploy It (p. 19)
• Step 4: Clean up (p. 32)
• Step 5: Additional Considerations: Integrating Amazon SageMaker Endpoints into Internet-facing Applications (p. 33)

Step 1: Setting Up

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

Topics
• Step 1.1: Create an AWS Account and an Administrator User (p. 12)
• Step 1.2: Create an S3 Bucket (p. 13)

Step 1.1: Create an AWS Account and an Administrator User

Before you use Amazon SageMaker for the first time, complete the following tasks:

Topics
• Step 1.1.1: Create an AWS Account (p. 12)
• Step 1.1.2: Create an IAM Administrator User and Sign In (p. 13)

Step 1.1.1: Create 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.
Step 1.2: Create an S3 Bucket

2. Follow the online instructions.

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

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

Step 1.1.2: Create an IAM Administrator User and Sign In

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. 203).

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 1.2: Create an S3 Bucket (p. 13)

Step 1.2: Create an S3 Bucket

In exercises where you create a model training job, you save the following in an Amazon S3 bucket:

- 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.
Step 2: Create an Amazon SageMaker Notebook Instance

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.

Next Step

Step 2: Create an Amazon SageMaker Notebook Instance (p. 14)

Step 2: 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. 221).

iii. Choose **Create role**.

Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmmSS`. 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 the 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. 213).

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 Enabled with Internet Access by Default (p. 229).

f. (Optional) To use shell scripts that run when you create or start the instance, specify a lifecycle configuration. For information, see Step 2.1: (Optional) Customize a Notebook Instance (p. 18)

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. 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. 256) API.

4. When the status of the notebook instance is InService, choose Open next to its name to open the Jupyter dashboard.
Step 2.1: (Optional) Customize a Notebook Instance

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

**Next Step**

You are now ready to train your first model. For step-by-step instructions, see Step 3: Train a Model with a Built-in Algorithm and Deploy It (p. 19).
Next Step

You are now ready to train your first model. For step-by-step instructions, see Step 3: Train a Model with a Built-in Algorithm and Deploy It (p. 19).

Step 3: 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. 106).

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.
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. 10).

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 notebooks, open and run the example notebook that Amazon SageMaker provides in the following notebook instance:

  sample-notebooks/sagemaker-python-sdk/1P_kmeans_highlevel
  sample-notebooks/sagemaker-python-sdk/1P_kmeans_lowlevel
Step 3.1: 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](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. 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 Step 1: Setting Up (p. 12), and run the code. The get_execution_role function retrieves the IAM role you created at the time of creating your notebook instance.

```python
from sagemaker import get_execution_role
role = get_execution_role()
bucket = 'bucket-name'
```

Next Step

Step 3.2: Download, Explore, and Transform the Training Data (p. 21)
Step 3.2: Download, Explore, and Transform the Training 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 3.2.1: Download the MNIST Dataset (p. 21)
- Step 3.2.2: Explore the Training Dataset (p. 21)
- Step 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 22)

Step 3.2.1: Download the MNIST Dataset

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

```python
%%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.

Next Step

Step 3.2.2: Explore the Training Dataset (p. 21)

Step 3.2.2: 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 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 22)

Step 3.2.3: Transform the Training Dataset and Upload It to 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

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 3.3: Train a Model (p. 23)

**Step 3.3: Train a Model**

To start model training, you send a request to the CreateTrainingJob (p. 265) 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 3.3.1: Choose the Training Algorithm (p. 23)
- Step 3.3.2: Create a Training Job (p. 23)

**Step 3.3.1: 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 Using Built-in Algorithms with Amazon SageMaker (p. 50).

Next Step

Step 3.3.2: Create a Training Job (p. 23)

**Step 3.3.2: Create a Training Job**

To train a model, Amazon SageMaker provides the CreateTrainingJob (p. 265) 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 Algorithms Provided by Amazon SageMaker: Common Parameters (p. 52). 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. 110).
- 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 provides 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_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. 110).
     - `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 Training 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. 265) 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)

images = {'us-west-2': '174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest',
          'us-east-1': '382416733822.dkr.ecr.us-east-1.amazonaws.com/kmeans:latest',
          'us-east-2': '404615174143.dkr.ecr.us-east-2.amazonaws.com/kmeans:latest',
          'eu-west-1': '438346466558.dkr.ecr.eu-west-1.amazonaws.com/kmeans:latest'}
image = images[boto3.Session().region_name]

output_location = 's3://{}/kmeans_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"
        }
    ]
}
sagemaker = boto3.client('sagemaker')
sagemaker.create_training_job(**create_training_params)
status = sagemaker.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
print(status)
try:
    sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=job_name)
finally:
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 3.4: Deploy the Model to Amazon SageMaker Hosting Services (p. 26)**

**Step 3.4: 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.**
Step 3.4: Deploy the Model

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.

   %%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.

   from time import gmtime, strftime

   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
   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('Training failed with the following error: {}'.format(message))
        raise Exception('Endpoint creation did not succeed')

Next Step

Step 3.5: Validate the Model (p. 28)

Step 3.5: 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. 347) API.

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

- **Use the 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:
• Use the 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):
    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()
```

The result is a set of clusters that group similar images. The following visualization shows four of the clusters that the model returned:

![Visualization of four clusters](image-url)
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. 110).
- 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 (p. 63).
- 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. 89).
- Use a custom algorithm—Consider using a custom neural network algorithm built on Apache MXNet or TensorFlow. For more information, see Using Apache MXNet with Amazon SageMaker (p. 178) and Using TensorFlow with Amazon SageMaker (p. 168).

Next Step

Step 4: Clean up (p. 32)

Step 4: 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 endpoint configuration.
   - The model.
   - 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 5: Additional Considerations: 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.
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. 81) model. You don't know which values of the \( \text{eta} \), \( \text{alpha} \), \( \text{min\_child\_weight} \), and \( \text{max\_depth} \) hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values to search. 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 then deploy the trained model that training job created.

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. 19).

Topics

- How Hyperparameter Tuning Works (p. 34)
- Defining Objective Metrics (p. 35)
- Defining Hyperparameter Ranges (p. 36)
- Example: Hyperparameter Tuning Job (p. 37)
- Design Considerations (p. 45)

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 candidates 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.

### Defining 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 Using Built-in Algorithms with Amazon SageMaker (p. 50).

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 Logging Amazon SageMaker with Amazon CloudWatch (p. 226).

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 (p. 249) 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. We want it to capture the number, which is the metric. The expression \[0-9\].+ inside the parentheses tells the regex to look for one or more occurrences of any digit or period. The double backslash is an escape character that means "look for a literal period." (Normally, a regex interprets a period to mean match any single character.)

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 (p. 249) operation.

### Defining 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 Design Considerations (p. 45).

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 (p. 249) 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, but each value of a categorical hyperparameter range counts against that limit. Hyperparameter ranges have the following structure:

```json
"ParameterRanges": {
  "CategoricalParameterRanges": [
    {
      "Name": "tree_method",
      "Values": ["auto", "exact", "approx", "hist"]
    },
    "ContinuousParameterRanges": [
      {
        "Name": "eta",
        "MaxValue": "0.5",
        "MinValue": "0"
      }
    ]
  }
}
```
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. 81) 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. 12)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training (p. 13)
- A running Amazon SageMaker notebook instance (p. 14)

Topics

- Create a Notebook (p. 37)
- Get the sagemaker boto3 Client (p. 38)
- Get the Amazon SageMaker Execution Role (p. 38)
- Specify a Bucket and Data Output Location (p. 39)
- Download, Prepare, and Upload Training Data (p. 39)
- Configure and Launch a Hyperparameter Tuning Job (p. 40)
- Monitor the Progress of a Hyperparameter Tuning Job (p. 43)
- Clean up (p. 45)

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:

3. To create a notebook, choose **Files**, **New**, and **conda_python3**.
4. Name the notebook.

**Next Step**

Get the sagemaker boto3 Client (p. 38)

**Get the sagemaker boto3 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
from time import gmtime, strftime
import os

region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')
```

**Next Step**

Get the Amazon SageMaker Execution Role (p. 38)

**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
```
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 `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 1.2: Create an S3 Bucket (p. 13).

`prefix` is the path within the bucket where Amazon SageMaker stores the output from training jobs.

```python
bucket = 'sagemaker-MyBucket'  # Replace with the name of your S3 bucket
prefix = 'sagemaker/DEMO-automatic-model-tuning-xgboost-dm'
```

Next Step

Download, Prepare, and Upload Training Data (p. 39)

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 `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:

```bash
!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
```

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:

```python
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.drop(['duration', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)

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)

boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('validation.csv')
```

Next Step

Configure and Launch a Hyperparameter Tuning Job (p. 40)

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. 40)
- Configure the Training Jobs (p. 41)
- Launch the Hyperparameter Tuning Job (p. 43)
- Next Step (p. 43)

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 the CreateHyperParameterTuningJob (p. 249) call.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune.
- 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. 81) built-in algorithm. The objective
metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```json
import json

tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            {
                "MaxValue": "1",
                "MinValue": "0",
                "Name": "eta"
            },
            {
                "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 (p. 81) built-in algorithm.

```python
containers = {'us-west-2': '433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:latest',
             'us-east-1': '811284229777.dkr.ecr.us-east-1.amazonaws.com/xgboost:latest',
             'us-east-2': '825641698319.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest',
             'eu-west-1': '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest'}

training_image = containers[region]
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
    }
}
```

Launch the Hyperparameter Tuning Job

Now you can launch the hyperparameter tuning job by calling the CreateHyperParameterTuningJob (p. 249) API. Pass the name and JSON objects that you created in previous steps as the values of the parameters.

```python
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. 43)

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 Hyperparameter Tuning Job Status (p. 43)
- View the Status of the Training Jobs (p. 44)
- View the Best Training Job (p. 44)

View the Hyperparameter Tuning Job Status

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:
Monitor the Progress of a Hyperparameter Tuning Job

- 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**.

![Best training job summary](image)

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 (p. 45)**

**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/ and delete the notebook instance. 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 to store 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/.

**Design Considerations**

Hyperparameter optimization is not a fully-automated process. To improve optimization, use the following guidelines when you create hyperparameters.
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.

Use 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 Degree of Parallelism

Running more hyperparameter tuning jobs in parallel 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.
Using 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, and to write code to train models, deploy models to Amazon SageMaker hosting, and to test or validate your models.

Topics
- Creating a Notebook Instance (p. 47)
- Accessing Notebook Instances (p. 47)
- Using Example Notebooks (p. 48)
- Set the Notebook Kernel (p. 49)

Creating a Notebook Instance

To create a notebook instance, use either the Amazon SageMaker console or the CreateNotebookInstance (p. 256) API. For an example of using the Amazon SageMaker console to create a notebook instance, see Step 2: Create an Amazon SageMaker Notebook Instance (p. 14).

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. 229).
- Launches an ML compute instance—Amazon SageMaker launches an ML compute instance in an Amazon SageMaker VPC. It 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 a 5-GB 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. There is also 20 GB of instance storage available in the /tmp directory of each notebook instance. This is not persistent storage. When the instance is stopped or restarted, anything in the /tmp directory is deleted.
- Copies example Jupyter notebooks—These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

Accessing Notebook Instances

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

- Use the console.

Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
The console displays a list of notebook instances in your account. To open a notebook instance, choose the **Open** action for the instance.

The console uses your sign-in credentials to send a `CreatePresignedNotebookInstanceUrl` 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` 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](http://jupyter.org/documentation.html).

### Using 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, which 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](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.
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.
Using Built-in Algorithms with Amazon SageMaker

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 (p. 63) and the XGBoost Algorithm (p. 81). 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 (p. 63) or the XGBoost Algorithm (p. 81) 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 (p. 73) 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. 106).

• "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) (p. 113) 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. 89)—Use this algorithm to classify images. It uses example data with answers (referred to as supervised algorithm).

• Sequence2Sequence (p. 96)—This supervised algorithm is commonly used for neural machine translation.

• Latent Dirichlet Allocation (LDA) (p. 116)—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) (p. 121)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.
Algorithms Provided by Amazon SageMaker: Common Information

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

**Topics**
- Algorithms Provided by Amazon SageMaker: Common Parameters (p. 52)
- Algorithms Provided by Amazon SageMaker: Common Data Formats (p. 55)
- Algorithms Provided by Amazon SageMaker: Suggested Instance Types (p. 61)
- Algorithms Provided by Amazon SageMaker: Logs (p. 62)

Algorithms Provided by Amazon SageMaker: Common Parameters

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 Image Registry Path</th>
<th>Training Input Mode</th>
<th>File Type</th>
<th>Instance Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/kmeans:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU (single GPU device on one or more instances)</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>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>Factorization Machines</td>
<td>train and (optionally) test</td>
<td><code>&lt;ecr_path&gt;/factorization-machines:&lt;tag&gt;</code></td>
<td>File or Pipe</td>
<td>recordIO-protobuf</td>
<td>CPU (GPU for dense data)</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>Linear Learner</td>
<td>train and (optionally) validation, test, or both</td>
<td>&lt;ecr_path&gt;/linear-learner:&lt;tag&gt;</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>&lt;ecr_path&gt;/ntm:&lt;tag&gt;</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>&lt;ecr_path&gt;/randomcutforest:&lt;tag&gt;</td>
<td>File or Pipe</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU</td>
</tr>
<tr>
<td>Seq2Seq Modeling</td>
<td>train, validation, and vocab</td>
<td>&lt;ecr_path&gt;/seq2seq:&lt;tag&gt;</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>&lt;ecr_path&gt;/xgboost:&lt;tag&gt;</td>
<td>File</td>
<td>CSV or LibSVM</td>
<td>CPU</td>
</tr>
<tr>
<td>Image Classification</td>
<td>train and validation, (optionally) train_lst and validation_lst</td>
<td>&lt;ecr_path&gt;/image-classification:&lt;tag&gt;</td>
<td>File</td>
<td>recordIO or image files (.jpg or .png)</td>
<td>GPU</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>BlazingText</td>
<td>train</td>
<td>&lt;ecr_path&gt;/blazingtext:&lt;tag&gt;</td>
<td>File</td>
<td>Text file (one sentence per line with space-separated tokens)</td>
<td>GPU (single instance only) or 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>.
### Algorithm Name
- k-means, PCA, Factorization Machines, Linear Learner, Neural Topic Model, and Random Cut Forest
- LDA
- XGBoost, Image Classification, Seq2Seq, BlazingText
- DeepAR Forecasting

<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>k-means, PCA, Factorization</td>
<td>us-west-2</td>
<td>174872318107.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td>Machines, Linear Learner, Neural Topic Model, and Random Cut Forest</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>eu-west-1</td>
<td>438346466558.dkr.ecr.eu-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>LDA</td>
<td>eu-west-1</td>
<td>999678624901.dkr.ecr.eu-west-1.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-west-2</td>
<td>266724342769.dkr.ecr.us-west-2.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>XGBoost, Image Classification,</td>
<td>us-west-2</td>
<td>433757028032.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td>Seq2Seq, BlazingText</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>eu-west-1</td>
<td>685385470294.dkr.ecr.eu-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>DeepAR Forecasting</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>eu-west-1</td>
<td>224300973850.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>ap-northeast-1</td>
<td>633355088612.dkr.ecr.ap-northeast-1.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` 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` 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).
Algorithms Provided by Amazon SageMaker: Common Data Formats

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

Topics

• Common Data Formats—Training (p. 55)
• Common Data Formats—Inference (p. 58)

Common Data Formats—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:

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.

For training, it needs 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)

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. 352) 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 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 22).

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 SageMaker's protobuf format.

```protobuf
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 input channel.

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—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)

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.

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 = {
  "keys": [],
  "values": [],
  "shape": []
}
```

**Simple numeric vector:**

```javascript
// 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:

```javascript
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 ]
    }
  ]
}
```

### Inference Response Deserialization

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

```javascript
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:**

```javascript
let response = {
  "predictions": [
    "closest_cluster": 5,
    "distance_to_cluster": 36.5
  ]
}
```
Multi-record inference:

```javascript
let response = {
    "predictions": [
        // First instance prediction.
        {
            "closest_cluster": 5,
            "distance_to_cluster": 36.5
        },
        // Second instance prediction.
        {
            "closest_cluster": 2,
            "distance_to_cluster": 90.3
        }
    ]
}
```

Multi-record inference with protobuf input:

```javascript
{
    "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' }"
}
```

Common Request Formats for All Algorithms

Most algorithms use several of the following inference request formats.

**JSON**

Content-type: application/json

Dense Format

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

```javascript
let request = {
    "instances": [
        {
            "data": {
                "features": {
                    "values": [ 1.5, 16.0, 14.0, 23.0 ]
                }
            }
        }
    ]
}
```
Algorithms Provided by Amazon SageMaker: Suggested Instance Types

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

- m4.xlarge, m4.4xlarge, and m4.10xlarge
- c4.xlarge, c4.2xlarge, and c4.8xlarge
- p2.xlarge, p2.8xlarge, and p2.16xlarge

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]
        }
      }
    }
  ]
}
```

CSV

Content-type: text/csv; label_size=0

**Note**
CSV support is not available for factorization machines.

RECORDIO

Content-type: application/x-recordio-protobuf

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

- PCA Response Formats (p. 116)
- Linear Learner Response Formats (p. 72)
- NTM Response Formats (p. 125)
- k-means Response Formats (p. 112)
- Factorization Machine Response Formats (p. 80)
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.

### Algorithms Provided by Amazon SageMaker: Logs

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

### Example 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 schemas
   ```
Linear Learner

Linear models are supervised learning algorithms used for solving either classification or regression problems. As input the model is given labeled examples \((x, y)\). \(x\) is a high dimensional vector and \(y\) is a numeric label. For (binary) classification problems, the algorithm expects the label to be either 0 or 1. For regression problems, \(y\) is a real number. The algorithm learns a linear function, or linear threshold function for classification, mapping 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. This allows you to simultaneously explore different training objectives and choose the best solution from a validation set. It also allows you to explore a large number of models and choose the best, which optimizes either continuous objectives—such as mean square error, cross entropy loss, absolute error, and so on—or discrete objectives suited for classification, such as F1 measure, precision@recall, or accuracy. When compared with solutions providing a solution to only continuous objectives, the implementation provides a significant increase in speed over naive hyperparameter optimization techniques and added convenience.

The linear learner expects a data matrix, with rows representing the observations, and columns the dimensions of the features. It also requires an additional column containing 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. 265). 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 on which you train. Examples include the number of epochs, regularization, and loss type.

Input/Output Interface

Amazon SageMaker linear learner supports three data channels: train, validation, and test. The validation data channel is optional. If you provide validation data, it should be FullyReplicated. The validation loss is logged at every epoch, and a sample of the validation data is used to calibrate and select the best model. If you don’t provide validation data, the final model calibration and selection uses a sample of the training data. The test data channel is also optional. If test data is provided, the algorithm logs contain the test score for the final model.

Linear learner supports both recordIO wrapped protobuf and CSV. For input type x-recordio-protobuf, only Float32 tensors are supported. For input type text/csv, the first column is assumed to be the label, which is the target variable for prediction. Linear learner can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.
For inference, Linear Learner supports the application/json, x-recordio-protobuf, and text/csv formats. For binary classification models, both the score and the predicted label are returned. For regression, just the score is returned.

For more details on training and inference file formats, see example notebooks.

**EC2 Instance Recommendation**

Linear learner can be trained on single- or multi-machine CPU and GPU instances. During our testing, we have not found substantial evidence to multi-GPU to be faster than single GPU, but results vary depending on the specific use case.

**How It Works**

**Note**
We assume that the input data is shuffled. If not, for example if the data is ordered by label, the method fails.

**Step 1: Preprocessing**

If the option is turned on, the algorithm first goes over a small sample of the data to learn its characteristics. For every feature and for the label, you learn the mean value and the standard deviation.

This information is used during training. Based on the configuration, you normalize the data. That is, you shift it to have mean zero and scale it to have unit standard deviation. When the **auto** (default) value is specified to decide the normalization you:

- Shift and scale the label for regression problems, and leave it as is for classification problems
- Always scale the features
- Shift the features only for dense data

**Step 2: Training**

You train using a distributed implementation of stochastic gradient descent. The input allows you to control specifics of the optimization procedure by choosing the exact optimization algorithm, for example, Adam, Adagrad, SGD, and so on, and their parameters, such as momentum, learning rate, and the learning rate schedule. Without specified details, choose a default option that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives: in other words, vary L1 or L2 regularization and try out different optimizer settings.

**Step 3: Validation and Setting the Threshold**

When the training is done, evaluate the different models on a validation set. For regression problems, output the model obtaining the best score on the validation set. When the objective is classification, 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 such a scenario, avoid exploring 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>Number of features in input data. Required. Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>predictor_type</td>
<td>Whether the target variable is binary classification or regression. Required.</td>
</tr>
<tr>
<td></td>
<td>Valid values: binary_classifier or regressor</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Mini batch size for data iterator, consisting of number of observations per</td>
</tr>
<tr>
<td></td>
<td>mini batch. Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</td>
</tr>
<tr>
<td>epochs</td>
<td>Maximum number of passes over the training data. Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>use_bias</td>
<td>Whether the model should include bias (also called an intercept) feature.</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
<tr>
<td>binary_classifier_model</td>
<td>Pick the model with best criteria from the validation dataset for predictor</td>
</tr>
<tr>
<td></td>
<td>type binary_classifier:</td>
</tr>
<tr>
<td></td>
<td>• accuracy: model with highest accuracy.</td>
</tr>
<tr>
<td></td>
<td>• f1: model with highest f1 score.</td>
</tr>
<tr>
<td></td>
<td>• precision_at_target_recall: model with highest precision at a given recall</td>
</tr>
<tr>
<td></td>
<td>target.</td>
</tr>
<tr>
<td></td>
<td>• recall_at_target_precision: model with highest recall at a given precision</td>
</tr>
<tr>
<td></td>
<td>target.</td>
</tr>
<tr>
<td></td>
<td>• cross_entropy_loss: model with lowest cross entropy loss.</td>
</tr>
<tr>
<td></td>
<td>Valid values:accuracy, f1, precision_at_target_recall, recall_at_target_recall,</td>
</tr>
<tr>
<td></td>
<td>cross_entropy_loss</td>
</tr>
<tr>
<td></td>
<td>Default value: accuracy</td>
</tr>
<tr>
<td>target_recall</td>
<td>Target recall. Applicable only if binary_classifier_model_selection_criteria is</td>
</tr>
<tr>
<td></td>
<td>precision_at_target_recall.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Number between 0 and 1.0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>target_precision</td>
<td>Target precision. Only applicable if binary_classifier_model_selection_criteria is recall_at_target_precision. Valid values: Number between 0 and 1.0 Default value: 0.8</td>
</tr>
<tr>
<td>num_models</td>
<td>Description: 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. Valid values: positive integer or auto Default values: auto</td>
</tr>
<tr>
<td>num_calibration_samples</td>
<td>Number of observations to use from the validation dataset for model calibration (finding the best threshold). Valid values: Positive integer or auto Default value: auto</td>
</tr>
</tbody>
</table>
| init_method | Function to use to set the initial model weights.  
- uniform: uniformly between (-scale, +scale)  
- normal: normal, with mean 0 and sigma  
Valid values: uniform or normal Default value: uniform |
<p>| init_scale | Scale. Applies only when init_method is set to uniform. Valid values: positive float Default value: 0.07 |
| init_sigma | Standard deviation. Applies only when init_method is set to normal. Optional. Valid values: positive float Default value: 0.01 |
| init_bias | Initial weight for bias term. Valid values: number Default value: 0 |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **optimizer**  | The optimizer to use. Default setting for *auto* is *adam*.  
|                | Valid values: *sgd, adam, or auto*.  
|                | Default value: *auto* |
| **loss**       | The loss function to apply. The default *auto* is *logistic* for `predictor_type` *binary_classifier* and *squared_loss* for `predictor_type` *regressor*.  
|                | Valid values: *logistic, squared_loss, absolute_loss, hinge_loss, eps_insensitive_squared_loss, eps_insensitive_absolute_loss, quantile_loss, or huber_loss*  
|                | Default value: *auto* |
| **wd**         | L2 regularization parameter. In other words, the weight decay parameter. Use 0 for no L2 regularization.  
|                | Valid values: non-negative float  
|                | Default value: 0.0 |
| **l1**         | L1 regularization parameter. Use 0 for no L1 regularization.  
|                | Valid values: non-negative float  
|                | Default value: 0.0 |
| **momentum**   | Momentum parameter of the *sgd* optimizer.  
|                | Valid values: number between 0 and 1.0  
|                | Default value: 0 |
| **learning_rate** | The default, *auto*, depends on the optimizer chosen.  
|                | Valid values: positive float, *auto*  
|                | Default value: *auto* |
| **beta_1**     | Exponential decay rate for first moment estimates. Applies only when *adam* optimizer.  
|                | Valid values: number between 0 and 1.0  
|                | Default value: 0.9 |
| **beta_2**     | Exponential decay rate for second moment estimates. Only applies for *adam* optimizer. Optional.  
|                | Valid values: Number between 0 and 1.0  
<p>|                | Default value: 0.999 |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 times bias_lr_mult. Optional. Valid values: positive float. Default value: 10</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 times bias_wd_mult. By default there is no regularization on the bias term. Optional. Valid values: positive float. Default value: 0</td>
</tr>
<tr>
<td>use_lr_scheduler</td>
<td>If true, uses a scheduler for the learning rate. Valid values: (true or false). Default value: true</td>
</tr>
<tr>
<td>lr_scheduler_step</td>
<td>The number of steps between decreases of the learning rate. Only applies to learning rate scheduler. Valid values: positive integer. Default value: 100</td>
</tr>
<tr>
<td>lr_scheduler_factor</td>
<td>For every lr_scheduler_step, the learning rate decreases by this quantity. Applies only for learning rate scheduler. Optional Valid values: positive float between 0 and 1. Default value: 0.99</td>
</tr>
<tr>
<td>lr_scheduler_minimum_lr</td>
<td>The learning rate never decreases to a value lower than lr_scheduler_minimum_lr. Applies only for learning rate scheduler. Valid values: positive float. Default values: 0.00001</td>
</tr>
<tr>
<td>normalize_data</td>
<td>Normalizes the features before training to have std_dev of 1. Optional. Valid values: true, false, or auto. Default value: true</td>
</tr>
<tr>
<td>normalize_label</td>
<td>Normalizes label. For regression, the label is normalized, for classification, it is not. If this is set to true during classification, this parameter is ignored. Valid values: true, false, or auto. Default value: auto</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>unbiased_data</td>
<td>Unbiases the features before training so the mean is 0. By default data is unbiased if use_bias is set to true. Valid values: true, false, or auto. Default value: auto</td>
</tr>
<tr>
<td>unbiased_label</td>
<td>Unbiases labels before training so the mean is 0. Only done for regression if use_bias is set to true. Valid values: true, false, or auto. Default value: auto</td>
</tr>
<tr>
<td>num_point_for_scaler</td>
<td>Number of data points to use for calculating normalization or unbiassing of terms. Valid values: positive integer. Default value: 10,000</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. The metric is the binary_classifier_model_selection_criteria if provided, otherwise the metric is the same as loss. The metric is evaluated on the validation data. If no validation data is provided, the metric is always the same as loss and is evaluated on the training data. To disable early stopping, set early_stopping_patience to a value larger than epochs. Valid values: positive integer. Default value: 3</td>
</tr>
<tr>
<td>early_stopping_tolerance</td>
<td>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 will consider the improvement to be zero. Default value: 0.001</td>
</tr>
<tr>
<td>margin</td>
<td>Margin for hinge_loss. Valid values: positive float. Default value: 1.0</td>
</tr>
<tr>
<td>quantile</td>
<td>Quantile for quantile loss. For quantile q, the model will attempt to produce predictions such that true_label &lt; prediction with probability q. Valid values: float between 0 and 1. Default value: 0.5</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
**loss_insensitivity** | Parameter for epsilon insensitive loss type. During training and metric evaluation, any error smaller than this is considered to be zero. 
Valid values: positive float 
Default value: 0.01

**huber_delta** | 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. 
Valid values: positive float 
Default value: 1.0

**positive_example_weight_mult** | Weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If balanced, then a weight will be selected so that errors in classifying negative vs. positive examples have equal impact on the training loss. If auto, the algorithm will attempt to select the weight that optimizes performance. 
Valid values: balanced, auto, or a positive float 
Default value: 1.0

---

**Tuning 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 hyperparameter optimization, the linear learner internal tuning mechanism is turned off automatically, which 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](#).  

**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>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. You can set the loss to other types with the loss hyperparameter.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>
## Metric Name | Description | Optimization Direction
--- | --- | ---
`test:binary_classification_accuracy` | Accuracy of the final model on the test dataset. | Maximize
`test:binary_f_beta` | 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. | Maximize
`test:precision` | 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. | Maximize
`test:recall` | 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. | Maximize
`validation:objective_loss` | Mean value of the objective loss function on the validation dataset every epoch. By default, the loss is logistic loss for binary classification and squared loss for regression. To set loss to other types, use the `loss` hyperparameter. | Minimize
`validation:binary_classification_accuracy` | Accuracy of the final model on the validation dataset. | Maximize
`validation:binary_f_beta` | F_beta score of the final model on the validation dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall. | Maximize
`validation:precision` | 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. | Maximize
`validation:recall` | 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. | Maximize

### Tuning Hyperparameters

You can tune a linear learner model with the following hyperparameters.
### Parameter Table

<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_mult</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-5, MaxValue: 1e5</td>
</tr>
</tbody>
</table>

### Linear Learner Response Formats

#### JSON

**Binary classification**

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

**Regression**

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

#### RECORDIO

**Binary classification**

```json
[  
    Record = {  
        features = {},  
        label = {  
            'score': {  
                keys: [],  
                values: [0.4] # float32  
            },  
            'predicted_label': {
```
Factorization Machines

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

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 example notebooks for more details on training and inference file formats.

### EC2 Instance Recommendation

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.

Topics
- How Factorization Machines Work (p. 74)
- Factorization Machines Hyperparameters (p. 75)
- Tuning a Factorization Machines Model (p. 78)
- Factorization Machine Response Formats (p. 80)

How Factorization Machines Work

The prediction task for a factorization machine model is to estimate a function ŷ from a feature set x 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 (xₙ, yₙ) 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:

\[
\hat{y} = w_0 + \sum_i w_i x_i + \sum_i \sum_{j>i} <v_i, v_j> x_i x_j
\]

The three terms in this equation correspond respectively to the three components of the model:
- The w₀ term represents the global bias.
- The wᵢ linear terms model the strength of the iᵗʰ variable.
- The <vᵢ, vⱼ> factorization terms model the pairwise interaction between the iᵗʰ and jᵗʰ 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>
</table>
| feature_dim        | Dimension of the input feature space. This could be very high with sparse input. Required.  
Valid values: Positive integer. Suggested value range: [10000,10000000]  
Default value: - |
| num_factors        | Dimensionality of factorization. Required.  
Valid values: Positive integer. Suggested value range: [2,1000]  
Default value: - |
| predictor_type     | Type of predictor. Required.  
Valid values: String: `binary_classifier` or `regressor`  
Default value: - |
| mini_batch_size    | Size of mini-batch used for training.  
Valid values: positive integer  
Default value: 1000 |
| epochs             | Number of training epochs to run.  
Valid values: positive integer  
Default value: 1 |
| clip_gradient      | Optimizer parameter. Clip the gradient by projecting onto the box [-clip_gradient, +clip_gradient].  
Valid values: float  
Default value: - |
| eps                | Optimizer parameter. Small value to avoid division by 0.  
Valid values: float  
Default value: - |
| rescale_grad       | Optimizer parameter. If set, multiplies the gradient with rescale_grad before updating. Often choose to be 1.0/batch_size.  
Valid values: float  
Default value: - |
<p>| bias_lr            | Learning rate for the bias term. |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| linear_lr | Learning rate for linear terms.  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.001 |
| factors_lr | Learning rate for factorization terms.  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.0001 |
| bias_wd | Weight decay for the bias term.  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.01 |
| linear_wd | Weight decay for linear terms.  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.001 |
| factors_wd | Weight decay for factorization terms.  
Valid values: Non-negative float. Suggested value range: [1e-8, 512].  
Default value: 0.00001 |
| bias_init_method | Initialization method for the bias term.  
- *normal*: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by `bias_init_sigma`.  
- *uniform*: Initializes weights with random values uniformly sampled from a range specified by `[−bias_init_scale, +bias_init_scale]`.  
- *constant*: Initializes the weights to a scalar value specified by `bias_init_value`.  
Valid values: *uniform*, *normal*, or *constant*  
Default value: *normal* |
### Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias_init_scale</td>
<td>Range for initialization of the bias term. Takes effect if bias_init_method is set to <em>uniform</em>. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: -</td>
</tr>
<tr>
<td>bias_init_sigma</td>
<td>Standard deviation for initialization of the bias term. Takes effect if bias_init_method is set to <em>normal</em>. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01</td>
</tr>
<tr>
<td>bias_init_value</td>
<td>Initial value of the bias term. Takes effect if bias_init_method is set to <em>constant</em>. Valid values: Float. Suggested value range: [1e-8, 512]. Default value: -</td>
</tr>
<tr>
<td>linear_init_method</td>
<td>Initialization method for linear terms. Valid values: <em>uniform, normal, or constant</em>. Default value: <em>normal</em></td>
</tr>
<tr>
<td>linear_init_scale</td>
<td>Range for initialization of linear terms. Takes effect if linear_init_method is set to <em>uniform</em>. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: -</td>
</tr>
<tr>
<td>linear_init_sigma</td>
<td>Standard deviation for initialization of linear terms. Takes effect if linear_init_method is set to <em>normal</em>. Valid values: Non-negative float. Suggested value range: [1e-8, 512]. Default value: 0.01</td>
</tr>
</tbody>
</table>
## Parameter Name | Description
---|---
linear_init_value | Initial value of linear terms. Takes effect if linear_init_method is set to constant.

Valid values: Float. Suggested value range: [1e-8, 512].
Default value: -

factors_init_method | Initialization method for factorization terms.

- **normal**: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by factors_init_sigma.
- **uniform**: Initializes weights with random values uniformly sampled from a range specified by [-factors_init_scale, +factors_init_scale].
- **constant**: Initializes the weights to a scalar value specified by factors_init_value.

Valid values: uniform, normal, or constant.
Default value: normal

factors_init_scale | Range for initialization of factorization terms. Takes effect if factors_init_method is set to uniform.

Valid values: Non-negative float. Suggested value range: [1e-8, 512].
Default value: -

factors_init_sigma | Standard deviation for initialization of factorization terms. Takes effect if factors_init_method is set to normal.

Valid values: Non-negative float. Suggested value range: [1e-8, 512].
Default value: 0.001

factors_init_value | Initial value of factorization terms. Takes effect if factors_init_method is set to constant.

Valid values: Float. Suggested value range: [1e-8, 512].
Default value: -

---

### Tuning 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. 34).
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 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. The initialization methods themselves are not tunable. You must set the method before the optimization procedure. For example, if the initialization method is uniform, then only the scale parameters are tunable. 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>mini_batch_size</td>
<td>IntegerParameterRange</td>
<td>MinValue: 100, MaxValue: 10000</td>
<td>None</td>
</tr>
<tr>
<td>epoch</td>
<td>IntegerParameterRange</td>
<td>MinValue: 1, MaxValue: 1000</td>
<td>None</td>
</tr>
<tr>
<td>bias_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>linear_lr</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>factors_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>
</tbody>
</table>
### Inference Formats

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<tr>
<td>factors_wd</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-8, MaxValue: 512</td>
<td>None</td>
</tr>
<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>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>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>
</tbody>
</table>

### Factorization Machine Response Formats

#### JSON

**Binary classification**

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

**Regression**

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

Binary classification

```
[  
  Record = {  
    features = {},  
    label = {  
      'score': {  
        keys: [ ],  
        values: [0.4] # float32  
      },  
      'predicted_label': {  
        keys: [ ],  
        values: [0.0] # float32  
      }  
    }  
  }  
]
```

Regression

```
[  
  Record = {  
    features = {},  
    label = {  
      'score': {  
        keys: [ ],  
        values: [0.4] # float32  
      }  
    }  
  }  
]
```

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.

Input/Output Interface

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.

Amazon SageMaker’s 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 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.

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 instance importance use Instance Weight Support
• Amazon SageMaker XGBoost allows customers to differentiate the importance of instances
by assigning each instance a weight value. For text/libsvm input, customers can assign
weight values to 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

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.

Topics
• How XGBoost Works (p. 82)
• XGBoost Hyperparameters (p. 83)
• Tuning a XGBoost Model (p. 88)

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. For more detail about hyperparameter configurations, see here.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_round</td>
<td>The number of rounds to run the training. Required. Valid values: integer Default value: -</td>
</tr>
<tr>
<td>num_class</td>
<td>Required if objective is set to multi:softmax or multi:softprob. Valid values: integer Default value: -</td>
</tr>
<tr>
<td>booster</td>
<td>Which booster to use. The gtree and dart values use a tree-based model, while glinear uses a linear function. Valid values: String. One of gtree, glinear, or dart. Default value: gtree</td>
</tr>
<tr>
<td>silent</td>
<td>0 means print running messages, 1 means silent mode. Valid values: 0 or 1 Default value: 0</td>
</tr>
<tr>
<td>nthread</td>
<td>Number of parallel threads used to run xgboost. Valid values: integer Default value: Maximum number of threads.</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. Valid values: Float. Range: [0,1].</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.  
Valid values: Float. Range: \([0, \infty)\).  
Default value: 0 |
| max_depth      | 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 \(\text{grow\_policy}=\text{depth-wise}\).  
Valid values: Integer. Range: \([0, \infty)\)  
Default value: 6 |
| min_child_weight | 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 \(\text{min\_child\_weight}\), 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.  
Valid values: Float. Range: \([0, \infty)\).  
Default value: 1 |
| max_delta_step | Maximum delta step allowed for each tree's weight estimation.  
Valid inputs: 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.  
Valid values: Integer. Range: \([0, \infty)\).  
Default value: 0 |
| subsample      | Subsample ratio of the training instance. Setting it to 0.5 means that \(\text{XGBoost}\) randomly collects half of the data instances to grow trees. This prevents overfitting.  
Valid values: Float. Range: \([0, 1]\).  
Default value: 1 |
| colsample_bytree | Subsample ratio of columns when constructing each tree.  
Valid values: Float. Range: \([0, 1]\).  
Default value: 1 |
| colsample_bylevel | Subsample ratio of columns for each split, in each level.  
Valid values: Float. Range: \([0, 1]\).  
Default value: 1 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **lambda**     | L2 regularization term on weights. Increasing this value makes models more conservative.  
Valid values: float  
Default value: 1 |
| **alpha**      | L1 regularization term on weights. Increasing this value makes models more conservative.  
Valid values: float  
Default value: 1 |
| **tree_method**| The tree construction algorithm used in XGBoost.  
Valid values: One of *auto*, *exact*, *approx*, or *hist*.  
Default value: *auto* |
| **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.  
Valid values: Float, Range: [0, 1].  
Default value: 0.03 |
| **scale_pos_weight** | Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: $\frac{\text{sum(negative cases)}}{\text{sum(positive cases)}}$.  
Valid values: float  
Default value: 1 |
| **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.  
Valid values: comma-separated string.  
Default value: `grow_colmaker`, prune |
| **refresh_leaf** | This is a parameter of the 'refresh' updater plugin. When set to `true`, tree leaves and tree node stats are updated. When set to `false`, only tree node stats are updated.  
Valid values: 0/1  
Default value: 1 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process_type</td>
<td>The type of boosting process to run.</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>grow_policy</td>
<td>Controls the way that new nodes are added to the tree.</td>
</tr>
<tr>
<td></td>
<td>Currently supported only if tree_method is set to hist.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either depthwise or lossguide.</td>
</tr>
<tr>
<td></td>
<td>Default value: depthwise</td>
</tr>
<tr>
<td>max_leaves</td>
<td>Maximum number of nodes to be added. Relevant only if grow_policy is set to</td>
</tr>
<tr>
<td></td>
<td>lossguide.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>max_bin</td>
<td>Maximum number of discrete bins to bucket continuous features.</td>
</tr>
<tr>
<td></td>
<td>Used only if tree_method is set to hist.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 256</td>
</tr>
<tr>
<td>sample_type</td>
<td>Type of sampling algorithm.</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>normalize_type</td>
<td>Type of normalization algorithm.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either tree or forest.</td>
</tr>
<tr>
<td></td>
<td>Default value: tree</td>
</tr>
<tr>
<td>rate_drop</td>
<td>Dropout rate (a fraction of previous trees to drop during the dropout).</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>one_drop</td>
<td>When this flag is enabled, at least one tree is always dropped during the</td>
</tr>
<tr>
<td></td>
<td>dropout.</td>
</tr>
<tr>
<td></td>
<td>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>skip_drop</td>
<td>Probability of skipping the dropout procedure during a boosting iteration.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</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>lambda_bias</td>
<td>L2 regularization term on bias.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: [0.0, 1.0].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>tweedie_variance_power</td>
<td>Parameter that controls the variance of the Tweedie distribution.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range: (1, 2).</td>
</tr>
<tr>
<td></td>
<td>Default value: 1.5</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></td>
<td>Valid values: string</td>
</tr>
<tr>
<td></td>
<td>Default value: reg:linear</td>
</tr>
<tr>
<td>base_score</td>
<td>The initial prediction score of all instances, global bias.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.5</td>
</tr>
<tr>
<td>eval_metric</td>
<td>Evaluation metrics for validation data. A default metric is assigned according to the objective (rmse for regression, error for classification, and map for ranking). For a list of valid inputs, see XGBoost Parameters.</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>seed</td>
<td>Random number seed.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>early_stopping_rounds</td>
<td>The model will train until the validation score stops improving. Validation error needs to decrease at least every early_stopping_rounds to continue training. Amazon SageMaker hosting will use the best model for inference.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</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>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
</tbody>
</table>
Tuning a 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. 34).

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:rmse</td>
<td>Root mean square error.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mae</td>
<td>Mean absolute error.</td>
<td>You must choose one of them as an objective to optimize when tuning the algorithm with hyperparameter values. Minimize</td>
</tr>
<tr>
<td>validation:logloss</td>
<td>Negative log-likelihood.</td>
<td>Minimize</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:dropouterror</td>
<td>Multiclass classification error rate, calculated as #/(wrong cases)/#(all cases).</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:mlogloss</td>
<td>Negative log-likelihood for multiclass classification.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:auc</td>
<td>Area under the curve.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:ndcg</td>
<td>Normalized Discounted Cumulative Gain.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:map</td>
<td>Mean average precision.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable Hyperparameters

Tune the XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on XGBoost objective metrics are: alpha, min_child_weight, subsample, eta, and nrounds.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_round</td>
<td>IntegerParameterRanges</td>
<td>[1, 4000]</td>
</tr>
<tr>
<td>eta</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.1, MaxValue: 0.5</td>
</tr>
</tbody>
</table>
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:

- Imagenet - Image database
- Image classification in MXNet

Input/Output Interface

The Amazon SageMaker Image Classification algorithm supports both RecordIO (application/x-recordio) and image (application/x-image) content types for training. The algorithm supports only application/x-image for inference.
Training 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. 265) 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.

Training 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. 265) 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:

<table>
<thead>
<tr>
<th>Image Index</th>
<th>Class Index</th>
<th>Relative Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>your_image_directory/train_img_dog1.jpg</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>your_image_directory/train_img_cat1.jpg</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>your_image_directory/train_img_dog2.jpg</td>
</tr>
</tbody>
</table>

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.

Inference with Image Format

The generated models can be hosted for inference and support encoded .jpg and .png image formats as application/x-image content-type. The output is the probability values for all classes encoded in JSON format.

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

EC2 Instance Recommendation

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.

Topics
- How Image Classification Works (p. 91)
- Hyperparameters (p. 91)
- Tuning an Image Classification Model (p. 95)
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.

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>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</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>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>use_pretrained_model</td>
<td>Flag to indicate whether 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>Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>checkpoint_frequency</td>
<td>Period to store model parameters (in number of epochs).</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer no greater than epochs.</td>
</tr>
<tr>
<td></td>
<td>Default value: epochs (save checkpoint at the last epoch)</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>resize</td>
<td>Resize the image before using it for training. The images are resized so that the shortest side is of this parameter. If the parameter is not set, then the training data is used as such without resizing. Note: This option is available only for inputs specified as application/x-image content-type in training and validation channels.</td>
</tr>
<tr>
<td>epochs</td>
<td>Number of training epochs.</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Initial learning rate.</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>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 &quot;10, 20&quot;, 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 &quot;,&quot;.</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| optimizer         | The optimizer types. For more details of the parameters for the optimizers, please refer to MXNet's API.  
Valid values: One of *sgd*, *adam*, *rmsprop*, or *nag*.  
Default value: *sgd* |
| momentum          | The momentum for *sgd* and *nag*, ignored for other optimizers.  
Valid values: Float. Range in [0, 1].  
Default value: 0 |
| weight_decay      | The coefficient weight decay for *sgd* and *nag*, ignored for other optimizers.  
Valid values: Float. Range in [0, 1].  
Default value: 0.0001 |
| beta_1            | The beta1 for *adam*, in other words, exponential decay rate for the first moment estimates.  
Valid values: Float. Range in [0, 1].  
Default value: 0.9 |
| beta_2            | The beta2 for *adam*, in other words, exponential decay rate for the second moment estimates.  
Valid values: Float. Range in [0, 1].  
Default value: 0.999 |
| eps               | The epsilon for *adam* and *rmsprop*. It is usually set to a small value to avoid division by 0.  
Valid values: Float. Range in [0, 1].  
Default value: 1e-8 |
| gamma             | The gamma for *rmsprop*. A decay factor of moving average of the squared gradient.  
Valid values: Float. Range in [0, 1].  
Default value: 0.9 |
| mini_batch_size   | 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.  
Valid values: positive integer  
Default value: 32 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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. Valid values: string Default value: '3, 224, 224'</td>
</tr>
</tbody>
</table>
| augmentation_type | Data augmentation type. The input images can be augmented in multiple ways as specified below.  
  • crop: Randomly crop the image and flip the image horizontally  
  • 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  
  • 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 rotation is 10 degrees, the maximum shear ratio is 0.1, and the maximum aspect changing ratio is 0.25. 
  Valid values: One of crop, crop_color, crop_color_transform. Default value: - |
| top_k | Report 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. Valid values: Positive integer larger than 1. Default value: - |
### Parameter Name: kv_store

**Description:** 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.

- **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 local 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.

Valid values: Either dist_sync or dist_async.

Default value: none

---

### Tuning 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. 34).

### 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 one of these metrics as the objective. 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>train:accuracy</td>
<td>The ratio of the number of correct predictions to the total number of predictions made.</td>
<td>Maximize</td>
</tr>
<tr>
<td>validation:accuracy</td>
<td>The ratio of the number of correct predictions to the total number of predictions made.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

### Tunable 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 
Hyperparameters (p. 91).

<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: 8, MaxValue: 512</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 1e-6, MaxValue: 0.5</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'rmsprop', 'nag']</td>
</tr>
<tr>
<td>momentum</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<tr>
<td>weight_decay</td>
<td>ContinuousParameterRanges</td>
<td>MinValue: 0.0, MaxValue: 0.999</td>
</tr>
<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>
</tbody>
</table>

Sequence2Sequence

Amazon SageMaker seq2seq 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 is based on 
the Sockeye package, which uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network 
(CNN) models with attention as encoder-decoder architectures.

**Note**

Although Amazon SageMaker seq2seq is based on the Sockeye package, it uses a different 
input data format and renames some hyperparameters to work more effectively in Amazon 
SageMaker.

Input/Output Interface

**Training**

Although the Amazon SageMaker seq2seq algorithm relies on the Sockeye package, there are certain 
notable differences.
• It expects data in recordio-protobuf format similar to other Amazon SageMaker algorithms, whereas Sockeye expects it in a tokenized text format.
• It renames certain hyperparameters to work more effectively in Amazon SageMaker.
• It supports a subset of training and inference options that Sockeye currently offers.

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, which expects data in three channels. This algorithm expects an additional channel, vocab.

• 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 the 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**

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: {attention_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 application/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.

Please refer to the notebook for additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference.

**EC2 Instance Recommendation**

Currently Amazon SageMaker seq2seq is only set up to train on a single machine, but it does offer support for multiple GPUs.

**Topics**

• How Sequence2Sequence Works (p. 98)
• Sequence2Sequence Hyperparameters (p. 98)
• Tuning a Sequence to Sequence Model (p. 104)
How Sequence2Sequence 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.

Sequence2Sequence Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>encoder_type</td>
<td>Encoder type. The <em>rnn</em> architecture is based on attention mechanism by Bahdanau et al. and <em>cnn</em> architecture is based on Gehring et al.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>decoder_type</td>
<td>Decoder type.</td>
</tr>
<tr>
<td>num_layers_encoder</td>
<td>Number of layers for Encoder \textit{rnn} or \textit{cnn}.</td>
</tr>
<tr>
<td>num_layers_decoder</td>
<td>Number of layers for Decoder \textit{rnn} or \textit{cnn}.</td>
</tr>
<tr>
<td>rnn_num_hidden</td>
<td>The number of \textit{rnn} hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional LSTM by default.</td>
</tr>
<tr>
<td>rnn_cell_type</td>
<td>Specific type of \textit{rnn} architecture.</td>
</tr>
<tr>
<td>rnn_decoder_state_init</td>
<td>How to initialize \textit{rnn} decoder states from encoders.</td>
</tr>
<tr>
<td>rnn_residual_connections</td>
<td>Add residual connection to stacked \textit{rnn}. Number of layers should be more than 1.</td>
</tr>
<tr>
<td>rnn_first_residual_layer</td>
<td>First \textit{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>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>cnn_kernel_width_encoder</code></td>
<td>Kernel width for the <code>cnn</code> encoder.</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_kernel_width_decoder</code></td>
<td>Kernel width for the <code>cnn</code> decoder.</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_num_hidden</code></td>
<td>Number of <code>cnn</code> hidden units for encoder and decoder.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><code>cnn_activation_type</code></td>
<td>The <code>cnn</code> activation type to be used.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <code>glu</code>, <code>relu</code>, <code>softrelu</code>, <code>sigmoid</code>, or <code>tanh</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>glu</code></td>
</tr>
<tr>
<td><code>cnn_hidden_dropout</code></td>
<td>Dropout probability for dropout between convolutional layers.</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>num_embed_source</code></td>
<td>Embedding size for source tokens.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><code>num_embed_target</code></td>
<td>Embedding size for target tokens.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 512</td>
</tr>
<tr>
<td><code>embed_dropout_source</code></td>
<td>Dropout probability for source side embeddings.</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>Valid values: Float. Range in [0,1].</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>
</tbody>
</table>
| `rnn_attention_type`          | Attention model for encoders. *mlp* refers to concat and *bilinear* refers to general from the Luong et al. paper.  
                             | Valid values: String. One of *dot*, *fixed*, *mlp*, or *bilinear*.  
                             | Default value: *mlp*                                                                                                                                 |
| `rnn_attention_num_hidden`    | Number of hidden units for attention layers. defaults to *rnn_num_hidden*.  
                             | Valid values: positive integer  
                             | Default value: *rnn_num_hidden*                                                                                                                                 |
| `rnn_attention_in_upper_layers` | Pass the attention to upper layers of *rnn*, like Google NMT paper. Only applicable if more than one layer is used.  
                             | Valid values: boolean (*true* or *false*)  
                             | Default value: *true*                                                                                                                                 |
| `rnn_decoder_hidden_dropout`  | Dropout probability for hidden state that combines the context with the *rnn* hidden state in the decoder.  
                             | Valid values: Float. Range in [0,1].  
                             | Default value: 0                                                                                                                                 |
| `batch_size`                  | Mini batch size for gradient descent.  
                             | Valid values: positive integer  
                             | Default value: 64                                                                                                                                 |
| `bucketing_enabled`           | Set to *false* to disable bucketing, unroll to maximum length.  
                             | Valid values: *true* or *false*  
                             | Default value: *true*                                                                                                                                 |
| `bucket_width`                | 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*.  
                             | Valid values: positive integer  
                             | Default value: 10                                                                                                                                 |
| `loss_type`                   | Loss function for training.  
                             | Valid values: String (*cross-entropy*)  
                             | Default value: *cross-entropy*                                                                                                                                 |

101
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>training_metric</td>
<td>Metrics to track on training on validation data.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <em>perplexity</em> or <em>accuracy</em>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>perplexity</em></td>
</tr>
<tr>
<td>optimized_metric</td>
<td>Metrics to optimize with early stopping.</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>bleu_sample_size</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></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>max_num_batches</td>
<td>Maximum number of updates/batches to process. -1 for infinite.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -1</td>
</tr>
<tr>
<td>checkpoint_frequency_num_batches</td>
<td>Checkpoint and evaluate every x batches.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</td>
</tr>
<tr>
<td>checkpoint_threshold</td>
<td>Maximum number of checkpoints model is allowed to not improve in optimized_metric on validation dataset before training is stopped.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</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>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>max_num_epochs</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>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>
</tbody>
</table>
| optimizer_type      | Optimizer to choose from.  
Valid values: String. One of `adam`, `sgd`, or `rmsprop`.  
Default value: `adam` |
| weight_init_type    | Type of weight initialization.  
Valid values: String. Either `uniform` or `xavier`.  
Default value: `xavier` |
| weight_init_scale   | Weight initialization scale (for `uniform` and `xavier` initialization).  
Valid values: float  
Default value: 2.34 |
| xavier_factor_type  | Xavier factor type.  
Valid values: String. One of `in`, `out`, or `avg`.  
Default value: `in` |
| learning_rate       | Initial learning rate.  
Valid values: float  
Default value: 0.0003 |
| weight_decay        | Weight decay constant.  
Valid values: float  
Default value: 0 |
| momentum            | Momentum constant used for `sgd`. Don't pass this parameter if you are using `adam` or `rmsprop`.  
Valid values: float  
Default value: none |
| clip_gradient       | Clip absolute gradient values greater than this. Set to negative to disable.  
Valid values: float  
Default value: 1 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr_scheduler_type</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>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>plateau_reduce_lr_factor</td>
<td>Factor to multiply learning rate with (for plateau_reduce).</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 plateau_reduce scheduler, multiply learning rate with reduce factor if optimized_metric didn't improve for this many checkpoints.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>fixed_rate_lr_half_life</td>
<td>Half life for learning rate in terms of number of checkpoints for fixed_rate_* schedulers.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</td>
</tr>
<tr>
<td>beam_size</td>
<td>Length of the beam for beam search. Used during training for computing bleu and used during inference.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
</tbody>
</table>

### Tuning 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. 34)*.

### 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:perplexity</td>
<td>Perplexity computed on the validation dataset.</td>
<td>Minimize</td>
</tr>
<tr>
<td>validation:bleu</td>
<td>Bleu score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the bleu_sample_size parameter to specify the subsample.</td>
<td>Maximize</td>
</tr>
</tbody>
</table>

Tunable Hyperparameters

You can tune the following hyperparameters for the sequence to sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective 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</td>
</tr>
</tbody>
</table>
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. 107).

Input/Output Interface

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>networks (CNNs)</td>
<td></td>
<td>[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>
<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>
and CSV formats are supported for training. k-means 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 are supported. k-means returns a closest_cluster label and the distance_to_cluster for each observation.

Please see the example notebooks for more details on k-means data formats.

**EC2 Instance Recommendation**

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.

**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 MNIST 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. 265)`. 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 \( k \times x \) 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. 108)
- Step 2: Iterate over the Training Dataset and Calculate Cluster Centers (p. 109)
- Step 3: Reduce the Clusters from \( K \) to \( k \) (p. 109)

**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. 265). For an example, see Step 3.3.2: Create a Training Job (p. 23).
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 $c_1 = 100$ previously assigned points. You added 25 points from the mini-batch in this step.
   - Cluster $c_2 = 150$ previously assigned points. You added 40 points from the mini-batch in this step.
   - Cluster $c_3 = 450$ 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:

   $$
p_1 = \text{proportion of points assigned to } c_1 = \frac{25}{100+25}
$$

   $$
p_2 = \text{proportion of points assigned to } c_2 = \frac{40}{150+40}
$$

   $$
p_3 = \text{proportion of points assigned to } c_3 = \frac{5}{450+5}
$$

   c. Compute the center of the new points added to each cluster:

   $$
d_1 = \text{center of the new points added to cluster } 1
$$

   $$
d_2 = \text{center of the new points added to cluster } 2
$$

   $$
d_3 = \text{center of the new points added to cluster } 3
$$

   d. Compute the weighted average to find the updated cluster centers as follows:

   $$\text{Center of cluster 1} = ((1 - p_1) \times \text{center of cluster 1}) + (p_1 \times d_1)
$$

   $$\text{Center of cluster 2} = ((1 - p_2) \times \text{center of cluster 2}) + (p_2 \times d_2)
$$

   $$\text{Center of cluster 3} = ((1 - p_3) \times \text{center of cluster 3}) + (p_3 \times d_3)
$$

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. 265) 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. 107).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>Number of required clusters (also known as $k$). Required.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td><code>feature_dim</code></td>
<td>Dimension of the input vectors. Required.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td><code>mini_batch_size</code></td>
<td>Number of examples in a mini-batch. Required.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5000</td>
</tr>
<tr>
<td><code>init_method</code></td>
<td>The method by which we choose the initial centers.</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><code>extra_center_factor</code></td>
<td>The algorithm creates $num_clusters \times extra_center_factor$ as it runs and reduces the number of centers to $k$ when finalizing.</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><code>local_lloyd_max_iter</code></td>
<td>Maximum iterations for Lloyds EM procedure in the local kmeans used in the finalize stage.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 300</td>
</tr>
<tr>
<td><code>local_lloyd_tol</code></td>
<td>Tolerance for change in ssd for early stopping in local kmeans.</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><code>local_lloyd_init_method</code></td>
<td>Initialization method for local version.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Either random or kmeans++.</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
| **local_lloyd_num_trials** | Local version is run multiple times and the one with the best loss is chosen. This determines how many times. Valid values: Either a positive integer or *auto*. Default value: *auto*  
| **half_life_time_size** | The points can have a decayed weight. When a point is observed its weight, with regard to the computation of the cluster mean is 1. This weight decays exponentially as we observe more points. The exponent coefficient is chosen so that after observing half_life_time_size points after the mentioned point, its weight will become 1/2. If set to 0, there is no decay. Valid values: non-negative integer Default value: 0  
| **epochs** | Number of passes done over the training data. Valid values: positive integer Default value: 1  
| **eval_metrics** | Valid values: Either msd or ssd. Default value: msd  

**Tuning 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. 34).

**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.
**Tunable Hyperparameters**

Tune the Amazon SageMaker k-means model with the following hyperparameters. The hyperparameters that have the greatest impact on k-means objective metrics are: mini\_batch\_size, extra\_center\_factor, and init\_method. Tuning the hyperparameter epochs generally results in minor improvements.

<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: 3000, MaxValue: 15000</td>
</tr>
<tr>
<td>extra_center_factor</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 4, MaxValue: 10</td>
</tr>
<tr>
<td>init_method</td>
<td>CategoricalParameterRanges</td>
<td>['kmeans++', 'random']</td>
</tr>
<tr>
<td>epochs</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 10</td>
</tr>
</tbody>
</table>

**k-means Response Formats**

**JSON**

```json
{
    "predictions": [
        {
            "closest_cluster": 1.0,
            "distance_to_cluster": 3.0,
        },
        {
            "closest_cluster": 2.0,
            "distance_to_cluster": 5.0,
        },
        ....
    ]
}
```

**RECORDIO**

```json
[  Record = {
      features = {},
      label = {
```
Principal Component Analysis (PCA)

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.

**Input/Output Interface**

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 file formats are supported. PCA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

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."

Please refer to the example notebooks for additional details on training and inference formats.

**EC2 Instance Recommendation**

PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

**Topics**

- [How PCA Works](#)
- [PCA Hyperparameters](#)
- [PCA Response Formats](#)
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 $\text{False}$, 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 $X_i$ 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 $X_i$ of dimension $b \times d$, the worker draws a random matrix $H_i$ 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 $\pm 1$. 

114
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 $H^*$, the sum of columns of $H_1, \ldots, H_T$ ( $t$ being the total number of mini-batches), and $s$, 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 $C = B - (1/n) h h^* s$, 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. 114).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>num_components</td>
<td>The number of principal components to compute. Required.</td>
</tr>
<tr>
<td>feature_dim</td>
<td>Input dimension. Required.</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of rows in a mini-batch. Required.</td>
</tr>
<tr>
<td>algorithm_mode</td>
<td>Mode for computing the principal components.</td>
</tr>
<tr>
<td>subtract_mean</td>
<td>Indicates whether the data should be unbiased both during training and at inference.</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.</td>
</tr>
</tbody>
</table>
PCA Response Formats

JSON

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]
    },
    ...
  ]
}
```

RECORDIO

Accept—application/x-recordio-protobuf

```python
[ 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]
      }
    }
  }
]
```

Latent Dirichlet Allocation (LDA)

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.

**Input/Output Interface**

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. LDA inference returns application/json or application/x-recordio-protobuf predictions, which include the topic_mixture vector for each observation.

Please see the example notebooks for more detail on training and inference formats.

**EC2 Instance Recommendation**

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

**Topics**
- How LDA Works (p. 117)
- LDA Hyperparameters (p. 119)
- Tuning an LDA Model (p. 120)

**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')
> # modify alpha and beta
> mx.nd.save('new_model.tar.gz', [new_alpha, new_beta])
> # upload to S3 and create new SageMaker model using the console
```

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. 117).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_topics</code></td>
<td>The number of topics for LDA to find within the data. Required. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td><code>feature_dim</code></td>
<td>The size of the vocabulary of the input document corpus. Required. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td><code>mini_batch_size</code></td>
<td>The total number of documents in the input document corpus. Required. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td><code>alpha0</code></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. Valid values: Positive float</td>
</tr>
</tbody>
</table>
### Parameter Name

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>max_restarts</strong></td>
<td>The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm. Can be used to find better quality local minima at the expense of additional computation, but typically should not be adjusted. Valid values: Positive integer</td>
</tr>
<tr>
<td><strong>max_iterations</strong></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. Valid values: Positive integer</td>
</tr>
<tr>
<td><strong>tol</strong></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. Valid values: Positive float</td>
</tr>
</tbody>
</table>

### Tuning 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. 34).

### 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><code>test:pwll</code></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 Hyperparameters

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, \( \alpha_0 \) and \( \text{num\_topics} \), can affect the LDA objective metric (\( \text{test:pwll} \)). 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>( \text{num_topics} )</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 150</td>
</tr>
</tbody>
</table>

Neural Topic Model (NTM)

Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. The latent representation of each document is provided in terms of a probability distribution over a fixed set of aspects, often referred to as topics. Each topic, in turn, can be represented in terms of a probability distribution over words in the vocabulary. The semantics of topics are usually inferred by examining the top ranking words in each topic. 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.

Input/Output Interface

Amazon SageMaker Neural Topic Model supports three data channels: train, validation, and test. The validation and test data channels are optional. If you specify the validation channel, the test channel, or both, set the value of the S3DataDistributionType parameter for each of these channels 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. NTM supports 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 \( \text{number of records} \times \text{vocabulary size} \). NTM 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. NTM inference returns application/json or application/x-recordio-protobuf predictions, which include the topic_weights vector for each observation.

Please see the example notebooks for more detail on training and inference formats.

**EC2 Instance Recommendation**

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.

**Topics**
- NTM Hyperparameters (p. 122)
- Tuning an NTM Model (p. 123)
- NTM Response Formats (p. 125)

**NTM Hyperparameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>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></td>
<td>Default value: -</td>
</tr>
<tr>
<td>num_topics</td>
<td>Number of required topics</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 2, max: 1000)</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>encoder_layers</td>
<td>Represents 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.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Comma-separated list of positive integers or auto</td>
</tr>
<tr>
<td></td>
<td>Default value: auto</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of examples in each mini batch.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1, max: 10000)</td>
</tr>
<tr>
<td></td>
<td>Default value: 256</td>
</tr>
<tr>
<td>epochs</td>
<td>Maximum number of passes over the training data.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1, max: 100)</td>
</tr>
<tr>
<td></td>
<td>Default value: 50</td>
</tr>
<tr>
<td>encoder_layers_activation</td>
<td>Activation function to use in the encoder layers.</td>
</tr>
<tr>
<td></td>
<td>Valid values: One of sigmoid, tanh, or relu</td>
</tr>
<tr>
<td><strong>Parameter Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
</tr>
</tbody>
</table>
| **optimizer**     | Optimizer to use for training.  
|                   | Default value: *adadelta*  
| **tolerance**     | Maximum relative change in the loss function within the last num_patience_epochs number of epochs below which early stopping is triggered.  
|                   | Default value: 0.001  
| **num_patience_epochs** | Number of successive epochs over which early stopping criterion is evaluated.  
|                   | Default value: 3  
| **batch_norm**    | Whether to use batch normalization during training.  
|                   | Default value: *false*  
| **rescale_gradient** | Rescale factor for gradient.  
|                   | Default value: 1.0  
| **clip_gradient** | Maximum magnitude for each gradient component.  
|                   | Default value: Infinity  
| **weight_decay**  | Weight decay coefficient. Adds L2 regularization.  
|                   | Default value: 0.0  
| **learning_rate** | Learning rate for the optimizer.  
|                   | Default value: 0.001 |

**Tuning 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. 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 can fit 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 is generalizable, that it doesn't overfit too much to the training data. 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 reduces the generality of the model.

For more information about model tuning, see Automatic Model Tuning (p. 34).

### 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 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. 122).

<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: 16, MaxValue:2048</td>
</tr>
<tr>
<td>encoder_layers_activation</td>
<td>CategoricalParameterRanges</td>
<td>['sigmoid', 'tanh', 'relu']</td>
</tr>
<tr>
<td>optimizer</td>
<td>CategoricalParameterRanges</td>
<td>['sgd', 'adam', 'adadelta']</td>
</tr>
<tr>
<td>learning_rate</td>
<td>ContinuousParameterRange</td>
<td>MinValue: 1e-4, MaxValue: 0.1</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

JSON

```
{
    "predictions": [
        {
            "topic_weights": [0.02, 0.1, 0,...],
            "topic_weights": [0.25, 0.067, 0,...]
        }
    ]
}
```

RECORDIO

```
[  
  Record = {
    features = {},
    label = {
      'topic_weights': {
        keys: [],
        values: [0.25, 0.067, 0, ...]  # float32
      }
    }
  },
  Record = {
    features = {},
    label = {
      'topic_weights': {
        keys: [],
        values: [0.25, 0.067, 0, ...]  # float32
      }
    }
  }
]
```

DeepAR Forecasting

Amazon SageMaker DeepAR is a supervised learning algorithm for forecasting scalar time series using recurrent neural networks (RNN). Classical forecasting methods, such as Autoregressive Integrated Moving Average (ARIMA) or Exponential Smoothing (ETS), fit one model to each individual time series, and then use that model to extrapolate the time series into the future. In many applications, however, you might have many similar time series across a set of cross-sectional units (for example, demand for different products, load of servers, requests for web pages, and so on). In this case, it can be beneficial to train a single model jointly over all of these time series. DeepAR takes this approach, training a model for predicting a time series over a large set of (related) time series.

For the training phase, the dataset consists of one or preferably more than one time series, and an optional categorical grouping variable of which the time series is a member. The model learns entirely from these values. The DeepAR algorithm currently accepts no other external features. The model is then trained by randomly selecting time points from the provided time series and using them as training examples.

For inference, the trained model takes as input an individual time series, which might or might not have been used during training, and generates a forecast for the time series. This forecast takes into account what typically happened for similar time series in the training set.
DeepAR supports two data channels. The train channel is used for training a model and is required. The test channel is optional. If the test channel is present, the algorithm uses it to calculate accuracy metrics for the model after training. You can provide datasets as JSON or Parquet files.

By default, the model determines the input format from the file extension (either .json or .parquet). If you provide input files with different extensions, you can specify the file type by setting the `ContentType` parameter of the Channel (p. 354) data type.

If you use a JSON file, it must be in the JSON Lines format, where each record contains the following fields:

- "start" whose value is a string of the format YYYY-MM-DD HH:MM:SS.
- "target", whose value is an array of floats (or integers) that represent the time series variable's values.
- "cat" (optional), whose value is an integer that encodes the categorical grouping that record's time series is a member of. The categorical feature allows the model to learn typical behavior for that group. This can increase accuracy.

The following is an example of JSON data:

```
{{"start":"2009-11-01 00:00:00", "target": [4.3, 10.3, ...], "cat": 0}
{{"start":"2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": 2}
{{"start":"1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": 0}
```

For Parquet, you use the same three fields as columns. In addition, "start" can be the datetime type. gzip and snappy compression types are also supported.

For training data:

- All time series must have the same time unit: minutes, hours, days, weeks, or months.
- To train an accurate model, the training set should contain a sufficient number of time series (typically at least a few hundred) and should cover a representative time range. For example, one or more years when yearly seasonal patterns occur.
- The training file should be shuffled. In other words, the time series should occur in a random order in the file.
- If you use the categorical feature ("cat"), all time series must have this feature. It's required that you provide the largest value of ("cat"), and all values between 0 and this largest value must be present in the training data.

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:

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

where \(y_{it}\) is the true value of time series \(i\) at time \(t\) and \(\hat{y}_{it}\) 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 (see DeepAR Hyperparameters (p. 129)).
In addition, the accuracy of the forecast distribution is evaluated using weighted quantile loss. For a quantile in the range \([0, 1]\), the weighted quantile loss is defined as follows:

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

Here, \(q_{i,t}^{(\tau)}\) is the \(\tau\)-quantile of the distribution that the model predicts. Set the \text{test_quantiles} hyperparameter to specify which quantiles for which the algorithm calculates quantile loss. 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. 129).

If you have a set of time series, a simple way to prepare training and test datasets is as follows:

- Use the full dataset in the test channel.
- In the training channel, remove the last \text{prediction_length} points from each time series.

This ensures that the model does not see the removed points during training, and then those points are used for calculating the accuracy of the model.

For inference, DeepAR accepts JSON format with an "instances" field which includes one or more time series in JSON Lines format, and a name of "configuration", which includes parameters for generating the forecast. For details, see DeepAR Request and Response Formats (p. 133).

### DeepAR Instance Recommendations

You can train DeepAR on both GPU and CPU instances, in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, c4.xlarge or c4.2xlarge), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves performance only when the model has more than 100 cells in each hidden layer and/or a mini-batch size greater than 1000.

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

### DeepAR Common Questions

**Q: Can the model handle unobserved, missing values, or nan values?**

No, unobserved, missing values, and nan values are not currently supported.

**Q: Do all time-series require the same length or the same starting point?**

No, the time-series can have arbitrary starting points and arbitrary length. (Note, however, that time series shorter than \text{prediction_length} are ignored during training).

**Q: Is there a one-to-one relation between the training set and the test set?**

No, time-series in the training set are used to train the model. After that, the trained model can be used to generate forecasts for the future of the time series used in the training set, or for other time series that were not previously included.

**Q: Do I need to train one model per time series?**

We recommend that more than one time series be used when training a model.

**Q: Can I pass time-dependent features or scalar features?**
Currently, only signal categorical features are supported. In particular we do not support time-dependent co-variates. However, time-dependent features such as day of month are generated internally. Because of this, it is important to set the `start` field to the right value rather than simply a dummy date.

**Q: How do I use the categorical feature?**

The categorical feature `cat` can be used to encode a grouping. If the time-series belong to $N$ different groups, you can encode each such group by a number ($0$ to $N - 1$). The model can then use the categorical feature to generate better forecasts. To use this feature, the parameter `cardinality` has to be set to the number of groups (e.g. $N$) and the `embedding_dimension` parameter also has to be set. If either of these two hyperparameters is not set, then the cat field in train/test time series is ignored. The embedding dimension is typically smaller than cardinality, for instance $\log(N)$. It is important to remember that, in the training set, all categories from $0$ to $N - 1$ must be present in the training data or an otherwise an exception will be thrown. This is occurs because during inference, we can only forecast for categories which we have previously seen in training.

**Q: Can I pass multiple files?**

Yes. The training folder and the test folder can each contain multiple files. The file names can be arbitrary, but the file ending should be `.json`, `.gz.json` or `.parquet`. For example: `s3://mybucket/myfolder/train/data-1.parquet` or `s3://mybucket/myfolder/train/data-2.parquet`.

**Q: Can I use this to train on a single time-series?**

Models need sufficient data in order to learn typical behavior. A single or small number of time-series are typically not sufficient for training the neural network (unless the time-series are very long). While a DeepAR model trained on a single time-series will usually still generate sensible forecasts, standard forecasting methods such as ARIMA or ETS may be more accurate and stable. Where the DeepAR approach starts to outperform the standard methods is when your dataset contains hundreds of time-series and thus can be significantly more accurate with more data.

**Q: Do I need to split data into train/test set for evaluation?**

It can be useful. The time series in the `train` channel is used for training the model. The time-series in the `test` channel are used for evaluation after the model is trained. For the evaluation, the last `prediction_length` points of each time-series in the test set are withheld and a prediction is generated. The forecast is then compared with the actual last `prediction_length` points. Starting from a dataset of time-series, the simplest train / test split can be created by using the entire dataset in the test channel (in other words, all-time series of full length). For the train channel you can then remove the last `prediction_length` points from each time-series so that the model does not see these points during training. You can create more complex evaluations by repeating time-series multiple times in the test set, but cutting them at different end points, resulting in accuracy metrics that are averaged over multiple forecasts from different time points.

**Q: Can the forecast horizon be changed later?**

No. The forecast horizon (`prediction_length`) is fixed when a model is trained and it cannot be changed later.

**Q: Can the time series in the dataset have different frequencies?**

No, all time series in the dataset have to have the same frequency (for example, hourly).

**Q: Do I have to split my individual time series?**

No. You should not split individual time series into pieces. Each time series should be provided as a whole unit in the dataset (see training format). It is also important to make sure the start point is accurate for each time series.
Q: What is `context_length` and how should I set it?

The `context_length` corresponds to the number of data points the algorithm gets to see before making a prediction. Typical values are of the same order of magnitude as the forecast length. Note that the algorithm also uses a set of so-called "lags" that take into account observations that are farther back in time. For instance, with a daily time series where you want to predict for one week, you might set the `context_length` to 14 days. The lags are then automatically set depending on the frequency you set. For daily data, in addition to current data, they will take into account observations 1 month previously as well as 1 year previously. As a result, for the prediction, the algorithm will read in the last 14 days, 14 days one month ago, and 14 days one year ago. Because of this, it is important to provide the entire time series when training and when doing inference.

**Topics**
- DeepAR Hyperparameters (p. 129)
- Tuning a DeepAR Model (p. 132)
- DeepAR Request and Response Formats (p. 133)

### DeepAR Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| `time_freq`          | The granularity of the time-series in the dataset. Use `time_freq` to select appropriate date features and lags. The model supports only the listed frequencies. It doesn't support multiples, such as two days. Required. Choose one of the following values:  
  - `M`: monthly  
  - `W`: weekly  
  - `D`: daily  
  - `H`: hourly  
  - `min`: every minute  
  
  Valid values: One of `M`, `W`, `D`, `H`, or `min`.  
  Default value: - |
| `prediction_length`  | 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. Required.  
  
  Valid values: positive integer  
  Default value: - |
| `context_length`     | 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. The lag values that the model picks depend on the frequency of the  
  Default value: - |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| time series | 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  
Default value: - |
| 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.  
Valid values: One of *gaussian*, *beta*, *negative-binomial*, *student-T*, or *deterministic-L1*.  
Default value: *student-T* |
| epochs | 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. Required.  
Valid values: positive integer  
Default value: - |
| cardinality | If you include categorical features, *cardinality* specifies the number of categories (groups).  
The *cat* field in time series must range between 0 and *cardinality*-1 (inclusive). Required if you use categorical features.  
Valid values: positive integer  
Default value: - |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>embedding_dimension</td>
<td>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. Required if you use categorical features. Valid values: positive integer Default value: -</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>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 40</td>
</tr>
<tr>
<td>num_layers</td>
<td>The number of hidden layers in the RNN. Typical values range from 1 to 4.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The size of mini-batches used during training. Typical values range from 32 to 512.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 32</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>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1e-3</td>
</tr>
<tr>
<td>dropout_rate</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>early_stopping_patience</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Valid values: integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
`test_quantiles` | Quantiles for which to calculate quantile loss.  
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]

---

### Tuning 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. 34).

### 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 *DeepAR Common Questions* (p. 127).

<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 between forecast and actual target computed on the test set.</td>
<td>Minimize</td>
</tr>
<tr>
<td>**test:**mean_wQuantileLoss</td>
<td>Average overall quantile losses computed on the test set. Setting the <code>test_quantiles</code> hyperparameter controls which quantiles are used.</td>
<td>Minimize</td>
</tr>
<tr>
<td>**train:**final_loss</td>
<td>Training negative log-likelihood loss averaged over the last training epoch for the model.</td>
<td>Minimize</td>
</tr>
</tbody>
</table>

### Tunable Hyperparameters

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact on DeepAR objective metrics are: `learning_rate`, `num_cells`, and `context_length`, and `epochs`.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Recommended Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mini_batch_size</code></td>
<td><code>IntegerParameterRanges</code></td>
<td>MinValue: 32, MaxValue: 1028</td>
</tr>
<tr>
<td><code>epochs</code></td>
<td><code>IntegerParameterRanges</code></td>
<td>MinValue: 1, MaxValue: 1000</td>
</tr>
<tr>
<td><code>context_length</code></td>
<td><code>IntegerParameterRanges</code></td>
<td>MinValue: 1, MaxValue: 200</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>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 Request and Response Formats**

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

```
{
    "instances": [
        { "start": "2009-11-01 00:00:00", "target": [4.0, 10.0, 50.0, 100.0, 113.0], "cat": 0 },
        { "start": "2012-01-30", "target": [1.0], "cat": 2 },
        { "start": "1999-01-30", "target": [2.0, 1.0], "cat": 1 }
    ],
    "configuration": {
        "num_samples": 50,
        "output_types": ["mean", "quantiles", "samples"],
        "quantiles": ["0.5", "0.9"]
    }
}
```

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 (embedding_dimension and cardinality > 0), you must provide `cat` in the request. If the model was trained without the `cat` field, it can be omitted.

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.

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

```
{
    "predictions": [
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]
        }
    ]
}
```
The Amazon SageMaker BlazingText algorithm is an implementation of the Word2vec algorithm, which learns high-quality distributed vector representations of words in a large collection of documents. Many natural language processing (NLP) applications use word embeddings that are trained on large collections of documents. A word embedding represents each word in a collection of documents as a vector of numbers. Words that are similar have vectors that are similar. That is, their vectors have relatively short distances between them. These pretrained vector representations provide information about word distributions that typically improves the generalization of other models that are subsequently trained on a 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 BlazingText, you can learn word embeddings on your own datasets. Similar to Word2vec, it provides the Skipgram and continuous bag-of-words (CBOW) training architectures.

Amazon SageMaker provides the following features:

- Acceleration of training on multiple GPUs using highly optimized CUDA kernels. For more information, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.
- A new batch_skipgram mode that allows faster training and distributed computation across multiple CPU nodes. Batch Skipgram trains mini-batches using the Negative Sample Sharing strategy to convert level-1 Basic Linear Algebra Subprograms (BLAS) operations to level-3 BLAS operations. This conversion uses the multiply-add instructions of modern architectures. For more information, see Parallelizing Word2Vec in Shared and Distributed Memory.

**Input/Output Interface**

BlazingText expects you to provide a single preprocessed text file on the train channel. The text file must contain space-separated tokens, and each line of the file should contain a single sentence.

BlazingText outputs a text file named `vectors.txt`, which contains the trained word-to-vectors mapping. If the value of the `evaluation` hyperparameter is `true`, BlazingText also creates a JSON file named `eval.json`. This file contains the similarity evaluation results (Spearman's rank correlation coefficients) for the WordSim353 dataset. BlazingText also reports the number of words from the WordSim353 dataset that are not present in the training document collection.

BlazingText doesn't support model hosting or inference.
For more detail on training formats, see the example notebook.

**EC2 Instance Recommendation**

For CBOW and skipgram modes, BlazingText supports single CPU and single GPU instances. 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 more information about the mathematics behind BlazingText, see BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs.

**Topics**
- BlazingText Hyperparameters (p. 135)
- Tuning a BlazingText Model (p. 136)

**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 following table lists the hyperparameters for the BlazingText 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. Required.</td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>batch_skipgram</code>, <code>skipgram</code>, or <code>cbow</code></td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>min_count</td>
<td>The minimum number of times a word must appear in the collection to be included in training and output.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The initial learning rate.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.05</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 in training.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>vector_dim</td>
<td>The dimension of the word vectors that the algorithm learns.</td>
</tr>
<tr>
<td></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></td>
<td>Default value: 100</td>
</tr>
<tr>
<td>epochs</td>
<td>The number of complete passes through the training data.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
<tr>
<td>negative_samples</td>
<td>The number of noise words in the output layer for which the algorithm updates weights.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</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>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 11</td>
</tr>
<tr>
<td>sampling_threshold</td>
<td>To counter the imbalance between rare and very frequent words, words with frequency higher than the value of this parameter are discarded with some probability.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive fraction. Suggested values are less than .001.</td>
</tr>
<tr>
<td></td>
<td>Default value: .0001</td>
</tr>
<tr>
<td>evaluation</td>
<td>Whether the trained model is evaluated using the WordSim353 test.</td>
</tr>
<tr>
<td></td>
<td>Valid values: true or false.</td>
</tr>
<tr>
<td></td>
<td>Default value: true</td>
</tr>
</tbody>
</table>

**Tuning a BlazingText Model**

BlazingText is the Amazon SageMaker implementation of the Word2Vec algorithm that is used to generate word embeddings from a large number of documents. Word embeddings represent each unique word in the entire collection of text documents as a vector of numbers. Words that are similar will have similar vectors. This algorithm is used in a variety of Natural Language Understanding tasks.

*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. 34).*

**Metrics Computed by the BlazingText Algorithm**

The BlazingText algorithm reports on a single metric during training: the `train:mean_rho`. This metric is computed on WS-353 word similarity datasets. When tuning the hyperparameter values, use this metric as the objective.
## Random Cut Forest

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.
Input/Output Interface

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. RCF can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

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 Algorithms Provided by Amazon SageMaker: Common Data Formats (p. 55) 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.

Please see the example notebooks for demonstrations of training and inference with different content types.

Instance Recommendations

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.

Topics
- How RCF Works (p. 139)
- RCF Hyperparameters (p. 141)
- Tuning a RCF Model (p. 142)
- RCF Response Formats (p. 143)
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].

Randomly Sampling Data

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 $K$ from $N$ total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw $K$ 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 $S = \{S_1, \ldots, S_K\}$ 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 $S = \{S_1, \ldots, S_K\}$
- 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].

Training and Inference

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:

![Root](image)

Figure 1. 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.
Figure 2. 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.

Choosing 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 describe 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 \( \frac{1}{num\_samples\_per\_tree} \) 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. 139).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>num_samples_per_tree</code></td>
<td>Number of random samples given to each tree from the training data set.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer (min: 1, max: 2048)</td>
</tr>
<tr>
<td></td>
<td>Default value: 256</td>
</tr>
</tbody>
</table>
### Tuning a 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. 34).

### 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 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_trees</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 50, MaxValue: 1000</td>
</tr>
<tr>
<td>num_samples_per_tree</td>
<td>IntegerParameterRanges</td>
<td>MinValue: 1, MaxValue: 2048</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. Below, is a list the available output formats.

**JSON**

ACCEPT: application/json.

```
{
    "scores": [
        {
            "score": 0.02
        },
        {
            "score": 0.25
        }
    ]
}
```

**RECORDIO**

ACCEPT: application/x-recordio-protobuf.

```
[
    Record = {
```
features = {},

label = {

'score': {

keys: [],

values: [0.25] # float32

}

}

}

},

Record = {

features = {},

label = {

'score': {

keys: [],

values: [0.23] # float32

}

}

}

}
Using 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 Using Apache MXNet with Amazon SageMaker (p. 178) and Using TensorFlow with Amazon SageMaker (p. 168).

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. 147)
- Using Your Own Inference Code (p. 151)
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. 147)
- How Amazon SageMaker Provides Training Information (p. 148)
- Signalling Algorithm Success and Failure (p. 150)
- How Amazon SageMaker Processes Training Output (p. 150)
- Next Step (p. 151)

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:

  \[
  \text{docker run image train}
  \]

  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:

  \[
  \text{ENTRYPOINT ["executable", "param1", "param2"]}
  \]

  For example:

  \[
  \text{ENTRYPOINT ["python", "k-means-algorithm.py"]}
  \]

  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. 265) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.

  - The StopTrainingJob (p. 337) API issues the equivalent of the docker stop, with a 2 minute timeout, command to gracefully stop the specified container:

  \[
  \text{docker stop -t120}
  \]
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](https://github.com/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](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html) 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](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html).

**Topics**

- Hyperparameters (p. 148)
- Input Data Configuration (p. 148)
- Training Data (p. 149)
- Distributed Training Configuration (p. 149)

## Hyperparameters

Amazon SageMaker makes the hyperparameters in a [CreateTrainingJob](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html) request available in the Docker container in the `/opt/ml/input/config/hyperparameters.json` file.

## Input Data Configuration

You specify data channel information in the `InputDataConfig` parameter in a [CreateTrainingJob](https://docs.aws.amazon.com/sagemaker/latest/dg/API_CreateTrainingJob.html) 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
{
}
```
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"]
}
```

### Signalling 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. 151)

Using Your Own Inference Code

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code. Use this information to write inference code and create a Docker image.

Topics

- How Amazon SageMaker Runs Your Inference Image (p. 151)
- How Amazon SageMaker Loads Your Model Artifacts (p. 152)
- How Containers Serve Requests (p. 152)
- How Your Container Should Respond to Inference Requests (p. 152)
- How Your Container Should Respond to Health Check (Ping) Requests (p. 153)

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. 243) 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` APIs), Amazon SageMaker runs a command to stop the Docker container on the instances being terminated. The command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

If you update the endpoint (by calling the `UpdateEndpoint` 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 `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

- Amazon SageMaker uses the container definition that you provided in your `CreateModel` request to set environment variables and the DNS hostname for the container as follows:
  - It sets environment variables using the `ContainerDefinition.Environment` string-to-string map.
  - It sets the DNS hostname using the `ContainerDefinition.ContainerHostname`.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateEndpointConfig` 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` 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, anything else will result in failure to download the file.

Amazon SageMaker stores the model artifact as a single compressed tar file in Amazon S3. Amazon SageMaker uncompressed this tar file into the `/opt/ml/model` directory before your container starts. If you used Amazon SageMaker to train the model, the files will appear just as you left them.

### How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/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 `InvokeEndpoint` 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 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.

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.

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
Automatically Scaling 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 3.4: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).

Topics

- Automatic Scaling Components (p. 154)
- Before You Begin (p. 157)
- Related Topics (p. 157)
- Configure Automatic Scaling for a Variant (p. 157)
- Editing a Scaling Policy (p. 163)
- Deleting a Scaling Policy (p. 164)
- Load Testing for Variant Automatic Scaling (p. 165)
- Additional Considerations for Configuring Automatic Scaling (p. 166)

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. 213).

If you are using a custom permission policy, you must include the following permissions:

```
{
  "Effect": "Allow",
  "Action": [
    "sagemaker:DescribeEndpoint",
    "sagemaker:DescribeEndpointConfig",
    "sagemaker:UpdateEndpointWeightsAndCapacities"
  ],
  "Resource": "**"
}
{
  "Action": [
    "application-autoscaling:*"
  ],
  "Effect": "Allow",
  "Resource": "**"
}
{
  "Action": "iam:CreateServiceLinkedRole",
  "Effect": "Allow",
  "Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
  "Condition": {
    "StringLike": { "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com" }
  }
}
{
  "Effect": "Allow",
  "Action": [
    "cloudwatch:PutMetricAlarm",
    "cloudwatch:DescribeAlarms",
    "cloudwatch:DeleteAlarms"
  ],
  "Resource": "**"
}
```

Service-Linked Role

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

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 calculates 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

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

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.
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 3.4: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).

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 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 Variant (Console) (p. 157)
- Configure Automatic Scaling for a Variant (AWS CLI or the Application Auto Scaling API) (p. 158)

Configure Automatic Scaling for a Variant (Console)

To configure automatic scaling 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 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. 165).

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 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.

**Registering a 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 Variant (AWS CLI)**

To register your endpoint, use the `register-scalable-target` AWS CLI command with the following parameters:
Configure Automatic Scaling for a Variant
(AWS CLI or the Application Auto Scaling API)

- `--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:

```
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 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`.
- `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
```
Defining 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**
- Using a Predefined Metric (p. 160)
- Using a Custom Metric (p. 160)
- Adding a Cooldown Period (p. 161)
- Disabling Scale-in Activity (p. 161)

**Using 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.

```
{
  "TargetValue": 70.0,
  "PredefinedMetricSpecification":
  {
    "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
  }
}
```

**Using 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.

```
{
  "TargetValue": 50,
  "CustomizedMetricSpecification":
  {
    "MetricName": "CPUUtilization",
    "Namespace": "/aws/sagemaker/Endpoints",
    "Dimensions": [
      {
        "Name": "EndpointName", "Value": "my-endpoint"
      },
      {
        "Name": "VariantName","Value": "my-variant"
      }
    ],
    "Statistic": "Average",
    "Unit": "Percent"
  }
}
```

Adding 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.

```
{
  "TargetValue": 70.0,
  "PredefinedMetricSpecification":
  {
    "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
  },
  "ScaleInCooldown": 600,
  "ScaleOutCooldown": 300
}
```

Disabling 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
    }
}
```

**Applying 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.

**Applying a Scaling Policy (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`.

```bash
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
```
Applying a Scaling Policy (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": {
    "TargetValue": 70.0,
    "PredefinedMetricSpecification": {
      "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
    }
  }
}
```

Editing a Scaling Policy

You can edit a variant scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

**Editing 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 variant (Console) (p. 157).
Editing 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 Applying a Scaling Policy to a Production Variant (p. 162).

Deleting a Scaling Policy

You can delete a scaling policy with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Deleting 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.

Deleting 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.

Deleting 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
    --resource-id endpoint/MyEndpoint/variant/MyVariant
    --service-namespace sagemaker
    --scalable-dimension sagemaker:variant:DesiredInstanceCount
```

Deleting 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 Variant Automatic Scaling
Perform load tests to choose an automatic scaling configuration that works the way you want.

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 Variant (p. 166)
Determine the Performance Characteristics of a 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 3.4: Deploy the Model to Amazon SageMaker Hosting Services (p. 26).
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-minute 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.

Additional Considerations for Configuring Automatic Scaling

When configuring automatic scaling, consider the following general guidelines.
Test 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. 339) 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
Using TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train a model using custom TensorFlow code. If you choose to deploy your code using Amazon SageMaker hosting services, you can also provide custom TensorFlow inference code. This section provides guidelines for writing custom TensorFlow code for both model training and inference, and an example that includes sample TensorFlow code and instructions for model training and deployment.

For information about TensorFlow supported versions, see Supported Versions (p. 237). The container source code can be found at the GitHub repository at https://github.com/aws/sagemaker-tensorflow-containers.

Topics
- Writing Custom TensorFlow Model Training and Inference Code (p. 168)
- Examples: Using Amazon SageMaker with TensorFlow (p. 171)

Writing Custom TensorFlow Model Training and Inference Code

To train a model on Amazon SageMaker using custom TensorFlow code and deploy it on Amazon SageMaker, you need to implement training and inference code interfaces in your code.

Your TensorFlow training script must be a Python 2.7 source file. The current default TensorFlow version is 1.6. This training/inference script must contain the following functions:

- **model_fn**: Defines the model that will be trained.
- **train_input_fn**: Preprocess and load training data.
- **eval_input_fn**: Preprocess and load evaluation data.
- **serving_input_fn**: Defines the features to be passed to the model during prediction.

For more information, see https://github.com/aws/sagemaker-python-sdk#tensorflow-sagemaker-estimators.

Implement the following training code interface:

```python
def model_fn(features, labels, mode, hyperparameters):
    ""
    Implement code to do the following:
    1. Configure the model with TensorFlow operations
    2. Define the loss function for training/evaluation
    3. Define the training operation/optimizer
    4. Generate predictions
    5. Return predictions/loss/train_op/eval_metric_ops in EstimatorSpec object
    ""
    For more information on how to create a model_fn, see https://www.tensorflow.org/extend/estimators#constructing_the_model_fn.
```
def train_input_fn(training_dir, hyperparameters):
    
    Implement code to do the following:
    1. Read the **training** dataset files located in training_dir
    2. Preprocess the dataset
    3. Return 1) a mapping of feature columns to Tensors with the corresponding feature data, and 2) a Tensor containing labels

    For more information on how to create a input_fn, see https://www.tensorflow.org/get_started/input_fn.

    Args:
        training_dir: Directory where the dataset is located inside the container.
        hyperparameters: The hyperparameters passed to your Amazon SageMaker TrainingJob that runs your TensorFlow training script. You can use this to pass hyperparameters to your training script.

    Returns: (data, labels) tuple

def eval_input_fn(training_dir, hyperparameters):
    
    Implement code to do the following:
    1. Read the **evaluation** dataset files located in training_dir
    2. Preprocess the dataset
    3. Return 1) a mapping of feature columns to Tensors with the corresponding feature data, and 2) a Tensor containing labels

    For more information on how to create a input_fn, see https://www.tensorflow.org/get_started/input_fn.

    Args:
        training_dir: The directory where the dataset is located inside the container.
        hyperparameters: The hyperparameters passed to your Amazon SageMaker TrainingJob that runs your TensorFlow training script. You can use this to pass hyperparameters to your training script.

    Returns: (data, labels) tuple
def serving_input_fn(hyperparameters):
    """During training, a train_input_fn() ingests data and prepares it for use by the model. At the end of training, similarly, a serving_input_fn() is called to create the model that will be exported for Tensorflow Serving.

Use this function to do the following:

- Add placeholders to the graph that the serving system will feed with inference requests.
- Add any additional operations needed to convert data from the input format into the feature Tensors expected by the model.

The function returns a tf.estimator.export.ServingInputReceiver object, which packages the placeholders and the resulting feature Tensors together.

Typically, inference requests arrive in the form of serialized tf.Examples, so the serving_input_receiver_fn() creates a single string placeholder to receive them. The serving_input_receiver_fn() is then also responsible for parsing the tf.Examples by adding a tf.parse_example operation to the graph.

For more information on how to create a serving_input_fn, see https://github.com/tensorflow/tensorflow/blob/18003982ff9c809ab8e9b76dd4c9b9ebc795f4b8/tensorflow/docs_src/programmers_guide/saved_model.md#preparing-serving-inputs.

Args:
hyperparameters: The hyperparameters passed to your TensorFlow Amazon SageMaker estimator that deployed your TensorFlow inference script. You can use this to pass hyperparameters to your inference script.
"""

Optionally implement the following inference code interface:

def input_fn(data, content_type):
    """[Optional] Prepares data for transformation. Amazon SageMaker invokes your input_fn definition in response to an InvokeEndpoint operation on an Amazon SageMaker endpoint containing this script. Amazon SageMaker passes in the data in the InvokeEndpoint request, and the InvokeEndpoint ContentType argument. If you omit this function, Amazon SageMaker provides a default input_fn for you. The default input_fn supports protobuf, CSV, or JSON-encoded array data. It returns the input in the format expected by TensorFlow serving. For more information about the default input_fn, see the Amazon SageMaker Python SDK GitHub documentation.

Args:
data: An Amazon SageMaker InvokeEndpoint request data content_type: An Amazon SageMaker InvokeEndpoint ContentType value for data.

Returns:
object: A deserialized object that will be used by TensorFlow serving as input. Must be in the format defined by the placeholders in your serving_input_fn.
"""
def output_fn(data, accepts):
    
    [Optional]
    Serializes the result of prediction in response to an InvokeEndpoint request. This
    function should return a serialized object. This serialized object is returned in the
    response to an
    InvokeEndpoint request. If you omit this function, Amazon SageMaker provides a default
    output_fn for you.
    The default function works with protobuf, CSV, and JSON accept types and serializes data
to either,
    depending on the specified accepts.

    Args:
    data: A result from TensorFlow Serving
    accepts: The Amazon SageMaker InvokeEndpoint Accept value. The content type the
    response
    object should be serialized to.
    
    Returns:
    object: The serialized object that will be send to back to the client.

Examples: Using Amazon SageMaker with TensorFlow

Topics

- TensorFlow Example 1: Using the tf.estimator (p. 172)

Amazon SageMaker provides the following example notebooks in your Amazon SageMaker notebook
instance:

- **tf.estimator** (iris_dnn_classifier)—This introductory example shows how to use the Amazon
  SageMaker sagemaker.tensorflow.TensorFlow estimator class. It uses this class to create a
  model for classifying a flower, based on four numerical features. The custom training code provided
  for this example uses the DNNClassifier to create a neural network with three hidden layers.

- **tf.layers** (abalone_using_layers)—This example shows how to use the TensorFlow layers
  module. It uses the module to create a model that predicts the age of a sea snail, based on seven
  numerical features. The custom code creates a neural network by individually specifying its layers.

- **tf.contrib.keras** (abalone_using_keras)—This example shows how to use the TensorFlow Keras
  library. It uses the library to create a model that predicts the age of a sea snail.

- **distributed TensorFlow** (distributed_mnist)—In this example, you explore distributed TensorFlow
  training in Amazon SageMaker. You use a convolutional neural network (CNN) to classify handwritten
  numbers and multiple GPU hosts for distributed training.

- **ResNet CIFAR-10 with Tensorboard** (resnet_cifar10_with_tensorboard)—In this example, you
  use Tensorboard with SageMaker. You use a ResNet model to train the CIFAR-10 dataset and evaluate
  it using TensorBoard.

In the examples, only the TensorFlow custom model training code differs. You interact with Amazon
SageMaker the same way in each.
The examples use the high-level Python library provided by Amazon SageMaker. This library is available in the Amazon SageMaker notebook instance that you created in Getting Started. If you use your own terminal, download and install the library using one of the following options:

- Use pip to install it:

  ```
  $ pip install sagemaker
  ```

The following topic explains one TensorFlow example in detail. All of the TensorFlow example notebooks that Amazon SageMaker provides follow the same pattern. They differ only in the custom TensorFlow code that they use for model training.

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 notebooks, open and run the example notebook that Amazon SageMaker provides in the notebook instance.

**Topics**

- TensorFlow Example 1: Using the tf.estimator (p. 172)

**TensorFlow Example 1: Using the tf.estimator**

This introductory TensorFlow example demonstrates how to use the Amazon SageMaker `sagemaker.tensorflow.TensorFlow` estimator class. This class provides the `fit` method for model training in Amazon SageMaker and the `deploy` method to deploy the resulting model in Amazon SageMaker. It is included in the Amazon SageMaker high-level Python library.

TensorFlow provides a high-level machine learning API (`tf.estimator`) to easily configure, train, and evaluate a variety of machine learning models. In this exercise, you construct a neural network classifier using this API. You then train it on Amazon SageMaker using the **Iris flower data set**.

In the example code, you do the following:

1. Train the model. During training, the following occurs:
   a. Amazon SageMaker loads the Docker image that contains the TensorFlow framework (this starts the Docker container).
   b. Amazon SageMaker reads training data from an Amazon S3 bucket into the container's file system.
   c. Your custom training code constructs a neural network classifier, `tf.estimator.DNNClassifier`.
   d. The Amazon SageMaker code in the container runs training. Your training code reads the training data.
2. Deploy the model using Amazon SageMaker hosting service. Amazon SageMaker returns an endpoint that you send requests to get inferences.
3. Test the model. You send requests to the model to classify flower samples you send in the request.

**About the Training Dataset**

The Iris flower training dataset used in this exercise contains 150 rows of data, with 50 samples from each of the three related Iris species (Iris setosa, Iris virginica, and Iris versicolor). Each row in the dataset contains the following data for each of the flower samples:
For this exercise, the dataset is randomized and split into two .csv files:

- A training dataset of 120 samples (iris_training.csv)
- A test dataset of 30 samples (iris_test.csv)

Next Step

Step 1: Create a Notebook and Initialize Variables (p. 173)

Step 1: Create a Notebook and Initialize Variables

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

1. Follow the instructions in Step 1: Setting Up (p. 12) to create the S3 bucket and IAM role.

   For simplicity, we suggest that you use one S3 bucket to store both the training code and the results of model training.

2. Create the notebook.

   a. If you haven’t already done so, sign in to the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

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

   c. To create a notebook, choose Files, New, and conda_tensorflow_p36. This pre-installed environment includes the default Anaconda installation, Python 3, and TensorFlow.

   d. Name the notebook.

3. To initialize variables, copy, paste, and run the following code in your notebook. Provide the name of the S3 bucket that contains your custom code. The get_execution_role function retrieves the IAM role you created at the time of creating your notebook instance. You can use the bucket that you created in Step 1: Setting Up (p. 12), or create a new bucket.

```python
from sagemaker import get_execution_role

#Bucket location to save your custom code in tar.gz format.
custom_code_upload_location = 's3://yourbucket/customcode/tensorflow_iris'
```
Example 1: Using the tf.estimator

```python
#Bucket location where results of model training are saved.
model_artifacts_location = 's3://yourbucket/artifacts'

#IAM execution role that gives SageMaker access to resources in your AWS account.
role = get_execution_role()
```

Next Step

Step 2: Train a Model on Amazon SageMaker Using TensorFlow Custom Code (p. 174)

**Step 2: Train a Model on Amazon SageMaker Using TensorFlow Custom Code**

The high-level Python library provides the `TensorFlow` class, which has two methods: `fit` (for training a model) and `deploy` (for deploying a model).

**To train a model**

1. Create an instance of the `sagemaker.tensorflow.TensorFlow` class by copying, pasting, and running the following code:

```python
from sagemaker.tensorflow import TensorFlow
iris_estimator = TensorFlow(entry_point='/home/ec2-user/sample-notebooks/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/iris_dnn_classifier.py',
role=role,
output_path=model_artifacts_location,
code_location=custom_code_upload_location,
train_instance_count=1,
train_instance_type='ml.c4.xlarge',
training_steps=1000,
evaluation_steps=100)
```

Some of these constructor parameters are sent in the `fit` method call for model training in the next step.

**Details:**

- `entry_point`—The example uses only one source file (`iris_dnn_classifier.py`) and it is already provided for you on your notebook instance. If your custom training code is stored in a single file, specify only the `entry_point` parameter. If it's stored in multiple files, also add the `source_dir` parameter.

  **Note**
  Specify only the source file that contains your custom code. The `sagemaker.tensorflow.TensorFlow` object determines which Docker image to use for model training when you call the `fit` method in the next step.

- `output_path`—Identifies the S3 location where you want to save the result of model training (model artifacts).
- `code_location`—S3 location where you want the `fit` method (in the next step) to upload the tar archive of your custom TensorFlow code.
- `role`—Identifies the IAM role that Amazon SageMaker assumes when performing tasks on your behalf, such as downloading training data from an S3 bucket for model training and uploading training results to an S3 bucket.
- `hyperparameters`—Any hyperparameters that you specify to influence the final quality of the model. Your custom training code uses these parameters.
• `train_instance_type` and `train_instance_count`—Identify the type and number of ML Compute instances to launch for model training.

You can also train your model on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see https://github.com/aws/sagemaker-python-sdk#local-mode in the Amazon SageMaker Python SDK.

2. Start model training by copying, pasting, and running the following code:

```python
%%time
import boto3
region = boto3.Session().region_name
train_data_location = 's3://sagemaker-sample-data-{}.tensorflow/iris'.format(region)
iris_estimator.fit(train_data_location)
```

The `fit` method parameter identifies the S3 location where the training data is stored. The `fit` method sends a CreateTrainingJob (p. 265) request to Amazon SageMaker.

You can get the training job information by calling the DescribeTrainingJob (p. 298) or viewing it in the console. The following is an example response:

```json
{
  "InputDataConfig": [
    {
      "ChannelName": "training",
      "DataSource": {
        "S3DataSource": {
          "S3DataType": "S3Prefix",
          "S3DataDistributionType": "FullyReplicated",
          "S3Uri": "s3://sagemaker-sample-data-us-west-2/tensorflow/iris"
        }
      }
    },
    "OutputDataConfig": {
      "S3OutputPath": "s3://sagemakermv/artifacts"
    },
    "StoppingCondition": {
      "MaxRuntimeInSeconds": 86400
    },
    "TrainingJobName": "sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686",
    "AlgorithmSpecification": {
      "TrainingInputMode": "File",
      "TrainingImage": "142577830533.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-py2-cpu:1.0.5"
    },
    "HyperParameters": {
      "sagemaker_program": ""iris_dnn_classifier.py"",
      "checkpoint_path": "s3://sagemakermv/artifacts/checkpoints",
      "sagemaker_job_name": "sagemaker-tensorflow-py2-cpu-2017-11-18-03-11-11-686",
      "sagemaker_region": "us-west-2",
      "training_steps": "100",
      "sagemaker_container_log_level": "20"
    },
    "ResourceConfig": {
      "VolumeSizeInGB": 30,
      "InstanceCount": 1,
```
Details:

- **TrainingImage**—Amazon SageMaker runs this image to create a container for model training. You don't explicitly identify this image in your request. The `fit` method dynamically chooses the correct image by inspecting the Python version in the interpreter and the GPU capability of the ML compute instance type that you specified when creating the TensorFlow object.

- **Hyperparameters**—The request includes the hyperparameters that you specified when you created the `sagemaker.tensorflow.TensorFlow` object. It also includes the following additional hyperparameters, which have the prefix `sagemaker`. Amazon SageMaker uses these hyperparameters to set up the training environment.
  - **sagemaker_submit_directory**—Identifies the S3 location of the custom training code. The high-level Python library does several things for you. In this case, the `fit` method creates a gzipped tar archive from the custom code file(s), and uploads the archive to an S3 bucket. You specify this archive in this hyperparameter.
  - **sagemaker_program**—Identifies the primary module that your training functions will be loaded from. This is the `entry_point` parameter that you specified when you created the `sagemaker.tensorflow.TensorFlow` object.
  - **sagemaker_container_log_level**—Sets the Python logging level.
  - **sagemaker_job_name**—Amazon SageMaker uses the job name to publish CloudWatch metrics in your account.
  - **sagemaker_checkpoint_path**—In distributed training, TensorFlow uses this S3 location as a shared file system for the ML compute instances running the training.

- **InputDataConfig**—Specifies one channel. A channel is a named input source that the training code consumes.

- **OutputDataConfig**—Identifies the S3 location where you want to save training results (model artifacts).

By default, the training job runs synchronously (you see the output in the notebook). If you want it to run asynchronously, set the `wait` value to `false` in the call to the `fit` method or when you create the `sagemaker.tensorflow.TensorFlow` instance.

**Next Step**

**Step 3: Deploy the Trained Model**

**Step 3: Deploy the Trained Model**

To deploy the model, use Amazon SageMaker hosting services. During deployment, Amazon SageMaker launches the ML compute instances and deploys the model (the model artifacts and inference code) on them. In response, you get an endpoint. To get inferences from the model, your application sends requests to the endpoint.

For fast deployment, use the `deploy` method in the `sagemaker.tensorflow.TensorFlow` class. The class is included in the high-level Python library provided by Amazon SageMaker. The `deploy` method does the following:

1. Creates an Amazon SageMaker model by calling the `CreateModel` API. The model contains important information, such as the location of the model artifacts and the inference code image.
2. Creates an endpoint configuration by calling the `CreateEndpointConfig` API. This configuration specifies the name of the model (which was created in the preceding step), and the resource configuration (the type and number of ML compute instances to launch for hosting).

3. Creates the endpoint by calling the `CreateEndpoint` API and specifying the endpoint configuration. Amazon SageMaker launches ML compute instances as specified in the endpoint configuration, and deploys the model on them.

To deploy the model, copy, paste, and run the following code:

```python
%%time
iris_predictor = iris_estimator.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge')
```

When the status of the endpoint is INSERVICE, your model has been deployed. The API returns a `TensorFlowPredictor` object. To get inferences, you will use the `predict` method of this object.

**Note**
You can deploy your model to an endpoint hosted on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see https://github.com/aws/sagemaker-python-sdk#local-mode in the Amazon SageMaker Python SDK.

**Next Step**

**Step 4: Invoke the Endpoint to Get Inferences**

To get inferences, send requests using the `predict` method of the `TensorFlowPredictor` object. (This object was returned in Step 3). The `predict` method calls the Amazon SageMaker `InvokeEndpoint` API.

Run the `predict` method:

```python
iris_predictor.predict([6.4, 3.2, 4.5, 1.5]) # expected label to be 1
```

For more information about the input features, see About the Training Dataset (p. 172).

The model predicts that the flower species is Iris versicolor (label 1) with a probability of 97%:

```json
{u'result': {u'classifications': [{u'classes': [{u'label': u'0', u'score': 0.009605729021131992}, {u'label': u'1', u'score': 0.9699361324310303}, {u'label': u'2', u'score': 0.02045818418264389}]]}}
```

Your Amazon SageMaker notebook instance includes additional examples.
Using Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train a model using your own custom Apache MXNet training code. If you choose to use Amazon SageMaker hosting services, you can also provide your own custom Apache MXNet inference code. This section provides guidelines for writing custom Apache MXNet code for both model training and inference and an example that includes sample Apache MXNet code and instructions for model training and deployment.

For information about Apache MXNet supported versions, see Supported Versions (p. 237). The container source code can be found at the GitHub repository at https://github.com/aws/sagemaker-mxnet-containers.

Topics
- Writing Custom Apache MXNet Model Training and Inference Code (p. 178)
- Examples: Using Amazon SageMaker with Apache MXNet (p. 184)

Writing Custom Apache MXNet Model Training and Inference Code

To train a model on Amazon SageMaker using custom Apache MXNet code and deploy it on Amazon SageMaker, your code must implement the following training code interface and inference code interface.

- Implement the following training code interface:

```python
# Training functions
# ----------------------------------------------------------------------------#
def train(
    hyperparameters,
    input_data_config,
    channel_input_dirs,
    output_data_dir,
    model_dir,
    num_gpus,
    num_cpus,
    hosts,
    current_host,
    **kwargs):
    """
    [Required]

    Runs Apache MXNet training. Amazon SageMaker calls this function with information about the training environment. When called, if this function returns an object, that object is passed to a save function. The save function
```
can be used to serialize the model to the Amazon SageMaker training job model directory.

The **kwargs parameter can be used to absorb any Amazon SageMaker parameters that your training job doesn't need to use. For example, if your training job doesn't need to know anything about the training environment, your function signature can be as simple as train(**kwargs).

Amazon SageMaker invokes your train function with the following python kwargs:

**Args:**
- hyperparameters: The Amazon SageMaker Hyperparameters dictionary. A dict of string to string.
- input_data_config: The Amazon SageMaker input channel configuration for this job.
- channel_input_dirs: A dict of string-to-string maps from the Amazon SageMaker algorithm input channel name to the directory containing files for that input channel. Note, if the Amazon SageMaker training job is run in PIPE mode, this dictionary will be empty.
- output_data_dir: The Amazon SageMaker output data directory. After the function returns, data written to this directory is made available in the Amazon SageMaker training job output location.
- model_dir: The Amazon SageMaker model directory. After the function returns, data written to this directory is made available to the Amazon SageMaker training job model location.
- num_gpus: The number of GPU devices available on the host this script is being executed on.
- num_cpus: The number of CPU devices available on the host this script is being executed on.
- hosts: A list of hostnames in the Amazon SageMaker training job cluster.
- current_host: This host's name. It will exist in the hosts list.
- kwargs: Other keyword args.

**Returns:**
- (object): Optional. An Apache MXNet model to be passed to the model save function. If you do not return anything (or return None), the save function is not called.

```
""
pass
```

```python
def save(model, model_dir):
    ""
    [Optional]

    Saves an Apache MXNet model after training. This function is called with the return value of train, if there is one. You are free to implement this to perform your own saving operation.

    Amazon SageMaker provides a default save function for Apache MXNet models. The default save function serializes 'Apache MXNet Module <https://mxnet.incubator.apache.org/api/python/module.html>' models. To rely on the default save function, omit a definition of 'save' from your script. The default save function is discussed in more detail in the Amazon SageMaker Python SDK GitHub documentation.

    If you are using the Gluon API, you should provide your own save function, or save your model in the train function and let the train function complete without returning anything.

    **Arguments:**
    - model (object): The return value from train.
    - model_dir: The Amazon SageMaker model directory.
```
```
• Implement the following inference code interface:

```python
# Hosting functions
# ------------------

def model_fn(model_dir):
    
    """[Optional]
    Loads a model from disk, reading from model_dir. Called once by each
    inference service worker when it is started.

    If you want to take advantage of Amazon SageMaker's default request handling
    functions, the returned object should be a `Gluon Block` or MXNet `Module`,
    described below. If you are providing your own transform_fn,
    then your model_fn can return anything that is compatible with your
    transform_fn.

    Amazon SageMaker provides a default model_fn that works with the serialization
    format used by the Amazon SageMaker default save function, discussed above. If
    you saved your model using the default save function, you do not need to
    provide a model_fn in your hosting script.

    Args:
    - model_dir: The Amazon SageMaker model directory.

    Returns:
    - (object): Optional. The deserialized model.

    """  

    pass

def transform_fn(model, input_data, content_type, accept):
    """[Optional]
    Transforms input data into a prediction result. Amazon SageMaker invokes your
    transform_fn in response to an InvokeEndpoint operation on an Amazon SageMaker
    endpoint containing this script. Amazon SageMaker passes in the model, previously
    loaded with model_fn, along with the input data, request content type, and accept
    content type from the InvokeEndpoint request.

    The input data is expected to have the given content_type.

    The output returned should have the given accept content type.

    This function should return a tuple of (transformation result, content
    type). In most cases, the returned content type should be the same as the
    accept content type, but it might differ. For example, when your code needs to
    return an error response.

    If you provide a transform_fn in your hosting script, it will be used
    to handle the entire request. You don't need to provide any other
    request handling functions (input_fn, predict_fn, or output_fn).
    If you do provide them, they will be ignored.

    Amazon SageMaker provides default transform_fn implementations that work with
    Gluon and Module models. These support JSON input and output, and for Module
```

models, also CSV. To use the default transform_fn, provide a
hosting script without a transform_fn or any other request handling
functions. For more information about the default transform_fn,
see the SageMaker Python SDK GitHub documentation.

Args:
- input_data: The input data from the payload of the
  InvokeEndpoint request.
- content_type: The content type of the request.
- accept: The content type from the request's Accept header.

Returns:
- (object, string): A tuple containing the transformed result
  and its content type

""
pass

# Request handlers for Gluon models
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

def input_fn(input_data, content_type):
    ""
    [Optional]
    Prepares data for transformation. Amazon SageMaker invokes your input_fn in
    response to an InvokeEndpoint operation on an Amazon SageMaker endpoint that contains
    this script. Amazon SageMaker passes in input data and content type from the
    InvokeEndpoint request.

    The function should return an NDArray that can be passed to the
    predict_fn.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default input_fn converts JSON-encoded array data into an NDArray.
    For more information about the default input_fn, see the SageMaker
    Python SDK GitHub documentation.

    Args:
    - input_data: The input data from the payload of the
      InvokeEndpoint request.
    - content_type: The content type of the request.

    Returns:
    - (NDArray): An NDArray

    ""
pass

def predict_fn(block, array):
    ""
    [Optional]
    Performs prediction on an NDArray object. In response to an InvokeEndpoint request,
    Amazon SageMaker invokes your
    predict_fn with the model returned by your model_fn and the result
    of the input_fn.

    The function should return an NDArray containing the prediction results.

    If you omit this function, Amazon SageMaker provides a default implementation.
    The default predict_fn call passes the array to the forward
    method of the Gluon block, for example, block(array).
Args:
- block (Block): The loaded Gluon model; the result of calling model_fn on this script.
- array (NDArray): The result of a call to input_fn in response to an Amazon SageMaker InvokeEndpoint request.

Returns:
- (NDArray): An NDArray containing the prediction result.

```python
def output_fn(ndarray, accept):
    """
    [Optional]
    Serializes prediction results. Amazon SageMaker invokes your output_fn with the NDArray returned by your predict_fn and the content type from the InvokeEndpoint request's accept header.

    This function should return a tuple of (transformation result, content type). In most cases, the returned content type should be the same as the accept content type, but it might differ; for example, when your code needs to return an error response.

    If you omit this function, Amazon SageMaker provides a default implementation. The default output_fn converts the prediction result into a JSON-encoded array data. For more information about the default input_fn, see the Amazon SageMaker Python SDK GitHub documentation.

    Args:
    - ndarray: NDArray. The result of calling predict_fn.
    - content_type: string. The content type from the InvokeEndpoint request's Accept header.

    Returns:
    - (object, string): A tuple containing the transformed result and its content type
    """
    pass
```

# Request handlers for Module models
# ----------------------------------------

def input_fn(model, input_data, content_type):
    """
    [Optional]
    Prepares data for transformation. Amazon SageMaker invokes your input_fn in response to an InvokeEndpoint operation on an Amazon SageMaker endpoint that contains this script. Amazon SageMaker passes the MXNet Module returned by your model_fn, along with the input data and content type from the InvokeEndpoint request.

    The function should return an NDArray. Amazon SageMaker wraps the returned NDArray in a DataIter with a batch size that matches your model, and then passes it to your predict_fn.

    If you omit this function, Amazon SageMaker provides a default implementation. The default input_fn converts a JSON or CSV-encoded array data into an NDArray. For more information about the default input_fn, see the Amazon SageMaker Python SDK GitHub documentation.

    Args:
```
- model: A Module; the result of calling model_fn on this script.
- input_data: The input data from the payload of the
  InvokeEndpoint request.
- content_type: The content type of the request.

Returns:
  - (NDArray): an NDArray

```
pass
```

def predict_fn(module, data):
  ""
  [Optional]
  Performs prediction on an MXNet DataIter object. In response to an
  InvokeEndpoint request, Amazon SageMaker invokes your
  predict_fn with the model returned by your model_fn and DataIter
  that contains the result of the input_fn.

  The function should return a list of NDArray or a list of list of NDArray
  containing the prediction results. For more information, see the MXNet Module API
  <https://mxnet.incubator.apache.org/api/python/

  If you omit this function, Amazon SageMaker provides a default implementation.
  The default predict_fn calls module.predict on the input
data and returns the result.

  Args:
    - module (Module): The loaded MXNet Module; the result of calling
      model_fn on this script.
    - data (DataIter): A DataIter containing the results of a
      call to input_fn.

  Returns:
    - (object): A list of NDArray or list of list of NDArray
      containing the prediction results.

  ""
  pass
```

def output_fn(data, accept):
  ""
  [Optional]
  Serializes prediction results. Amazon SageMaker invokes your output_fn with the
  results of predict_fn and the content type from the InvokeEndpoint
  request's accept header.

  This function should return a tuple of (transformation result, content
type). In most cases, the returned content type should be the same as the
accept content type, but it might differ. For example, when your code needs to
return an error response.

  If you omit this function, Amazon SageMaker provides a default implementation.
The default output_fn converts the prediction result into JSON or CSV-
encoded array data, depending on the value of the accept header. For more
information about the default input_fn, see the Amazon SageMaker Python SDK
GitHub documentation.

  Args:
    - data: A list of NDArray or list of list of NDArray. The result of
calling predict_fn.
    - content_type: A string, The content type from the InvokeEndpoint
      request's Accept header.
Examples: Using Amazon SageMaker with Apache MXNet

Amazon SageMaker provides the following example notebooks in your Amazon SageMaker notebook instance:

- **The Apache MXNet Module API**— In this example, you use the Amazon SageMaker `sagemaker.mxnet.MXNet` estimator class to train a model. The custom Apache MXNet training code trains a multilayer perceptron neural network that predicts the number in images of single-digit handwritten numbers. It uses images of handwritten numbers as training data.

- **The Apache MXNet Gluon API**— This example uses the Gluon API to do the same thing that the Apache MXNet Module API does.

These examples use the high-level Python library provided by Amazon SageMaker. This library is available on the Amazon SageMaker notebook instance you created as part of Getting Started. For more information, see Step 2: Create an Amazon SageMaker Notebook Instance (p. 14). However, if you choose to use your own terminal, you need to download and install the library using one of the following options:

- Install it using pip:

  ```bash
  $ pip install sagemaker
  ```

This documentation explains one Apache MXNet example in detail. All of the Apache MXNet example notebooks that Amazon SageMaker provides follow the same pattern. They differ only in the custom Apache MXNet code they use in model training.

**Topics**

- [Apache MXNet Example 1: Using the Module API](#) (p. 184)

### Apache MXNet Example 1: Using the Module API

This introductory Apache MXNet example demonstrates using Amazon SageMaker `sagemaker.mxnet.MXNet` estimator class, provided as part of Amazon SageMaker high-level Python library. It provides the `fit` method for model training in Amazon SageMaker and `deploy` method to deploy resulting model in Amazon SageMaker.

In this exercise, you construct a neural network classifier using the Apache MXNet Module API. You then train the model using the **The MNIST Database** dataset, which Amazon SageMaker provides in an S3 bucket.

In this example, you do the following:

```python
Returns:
- (object, string): A tuple containing the transformed result and its content type.

# pass
```
1. Train the model. During training, the following occurs:
   a. Amazon SageMaker loads the Docker image containing the Apache MXNet framework.
   b. Amazon SageMaker reads training data from the S3 bucket into the container's file system.
   c. Your custom training code constructs a neural network classifier (using the `mxnet.module.Module` class).
   d. The Amazon SageMaker code in the container runs training. Your training code reads the training data for model training.

2. Deploy the model using Amazon SageMaker hosting services. Amazon SageMaker returns an endpoint that you send requests to to get inferences.

3. Test the model. The example provides an HTML canvas in the notebook where you can write a single-digit number using your mouse. The image of the number is then sent to the model for inference.

Topics
- Step 1: Create a Notebook and Initialize Variables (p. 185)
- Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code (p. 186)
- Step 3: Deploy the Trained Model (p. 188)
- Step 4: Invoke the Endpoint to Get Inferences (p. 189)

Step 1: Create a Notebook and Initialize Variables

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

1. Follow the instructions in Step 1: Setting Up (p. 12) to create the S3 bucket and IAM role.

   For simplicity, we suggest that you use one S3 bucket to store both the training code and the results of model training.

2. Create the notebook.

   a. If you haven’t already done so, sign in to the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.

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

   c. To create a notebook, choose Files, New, and `conda_mxnet_p36`. This pre-installed environment includes the default Anaconda installation, Python 3, and MXNet.

   d. Name the notebook.

3. To initialize variables, copy, paste, and run the following code in your notebook. Provide the name of the S3 bucket that contains your custom code. The `get_execution_role` function retrieves the IAM role you created at the time of creating your notebook instance. You can use the bucket that you created in Step 1: Setting Up (p. 12), or create a new bucket.

   ```python
   from sagemaker import get_execution_role
   #Bucket location to save your custom code in tar.gz format.
   custom_code_upload_location = 's3://your-bucket-name/customcode/mxnet'

   #Bucket location where results of model training are saved.
   model_artifacts_location = 's3://your-bucket-name/artifacts'

   #IAM execution role that gives Amazon SageMaker access to resources in your AWS account.
   #We can use the Amazon SageMaker Python SDK to get the role from our notebook environment.
   ```
Next Step

Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code (p. 186)

Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code

The high-level Python library provides the MXNet class with two methods: `fit` (for training a model) and `deploy` (for deploying a model).

To train the model:

1. Create an instance of the `sagemaker.mxnet.MXNet` class by copying, pasting, and running the following code:

```python
from sagemaker.mxnet import MXNet

mnist_estimator = MXNet(entry_point='/home/ec2-user/sample-notebooks/sagemaker-python-sdk/mxnet_mnist/mnist.py',
                        role=role,
                        output_path=model_artifacts_location,
                        code_location=custom_code_upload_location,
                        train_instance_count=1,
                        train_instance_type='ml.m4.xlarge',
                        hyperparameters={'learning_rate': 0.1})
```

Some of these constructor parameters are sent in the `fit` method call for model training in the next step.

Details:

- `entry_point` — The example uses only one source file (`mnist.py`) and it is already provided for you on your notebook instance. If your custom code is in one file, you specify only the `entry_point` parameter. If your training code consists of multiple files, you also add the `source_dir` parameter.

  **Note**
  You specify the source of only your custom code. The `sagemaker.mxnet.MXNet` object determines which Docker image to use for model training.

- `role` — The IAM role that Amazon SageMaker assumes when performing tasks on your behalf, such as downloading training data from an S3 bucket for model training and uploading training results to an S3 bucket.

- `code_location` — S3 location where you want the `fit` method (in the next step) to upload the tar archive of your custom Apache MXNet code.

- `output_path` — Identifies S3 location where you want to the result of model training (model artifacts) saved.

- `train_instance_count` and `train_instance_type` — You specify the number and type of instances to use for model training.

You can also train your model on your local computer by specifying `local` as the value for `train_instance_type` and `1` as the value for `train_instance_count`. For more information about local mode, see [https://github.com/aws/sagemaker-python-sdk#local-mode](https://github.com/aws/sagemaker-python-sdk#local-mode) in the Amazon SageMaker Python SDK.
• Hyperparameters — Any hyperparameters that you specify to influence the final quality of the model. Your custom training code uses these parameters.

2. Start model training by copying, pasting, and running the following code:

```python
%%time
import boto3
region = boto3.Session().region_name
train_data_location = 's3://sagemaker-sample-data-{}//mxnet/mnist/train'.format(region)
test_data_location = 's3://sagemaker-sample-data-{}//mxnet/mnist/test'.format(region)

mnist_estimator.fit({'train': train_data_location, 'test': test_data_location})
```

In the `fit` call, you specify the S3 URI strings that identify where the training and test dataset are stored. The `fit` method sends a CreateTrainingJob (p. 265) request to Amazon SageMaker.

You can get the training job information by calling the DescribeTrainingJob (p. 298) or viewing it in the console. The following is an example response:

```json
{
    "AlgorithmSpecification": {
        "TrainingImage": "520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-mxnet-py2-cpu:1.0",
        "TrainingInputMode": "File"
    },
    "HyperParameters": {
        "learning_rate": "0.11",
        "sagemaker_program": "mnist.py",
        "sagemaker_container_log_level": "20",
        "sagemaker_job_name": "sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586",
        "sagemaker_region": "us-west-2"
    },
    "InputDataConfig": [
        {
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://sagemaker-sample-data-us-west-2/mxnet/mnist/train"
                }
            },
            "ChannelName": "train"
        },
        {
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://sagemaker-sample-data-us-west-2/mxnet/mnist/test"
                }
            },
            "ChannelName": "test"
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemaker-mv/artifacts"
    },
    "TrainingJobName": "sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586",
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    }
}
```
Example 1: Using the Module API

```
{
  "ResourceConfig": {
    "InstanceCount": 1,
    "InstanceType": "ml.m4.xlarge",
    "VolumeSizeInGB": 30
  },
  "RoleArn": "arn:aws:iam::account-id:role/SageMakerRole"
}
```

Details:

- **TrainingImage**—Amazon SageMaker runs this image to create a container for model training. You don't explicitly identify this image in your request. Instead, the method dynamically chooses the appropriate image. It inspects the Python version in the interpreter and the GPU capability of the ML compute instance type that you specified when creating the Apache MXNet object.

- **Hyperparameters**—The request includes the hyperparameters that you specified when creating the `sagemaker.mxnet.MXNet` object. In addition, the request includes the following additional hyperparameters (all beginning with the prefix `sagemaker`). The hyperparameters starting with prefix "sagemaker" are used by Amazon SageMaker to set up the training environment.
  - `sagemaker_submit_directory`—Identifies the custom training code in S3 location to use for model training.
  - `sagemaker_program`—Identifies the primary module from which your training functions will be loaded (it is the `entry_point` parameter you specified when creating the `sagemaker.mxnet.MXNet` object.
  - `sagemaker_container_log_level`—Sets the Python logging level.

- **InputDataConfig**—Specifies two channels (train and test). Each channel is a named input source the training code consumes.

- **OutputDataConfig**—Identifies the S3 location where you want Amazon SageMaker to save training results (model artifacts).

By default, the training job runs synchronously (you see the output in the notebook).

Next Step

Step 3 : Deploy the Trained Model  (p. 188)

**Step 3 : Deploy the Trained Model**

You can use Amazon SageMaker hosting services to deploy the model. During deployment, Amazon SageMaker launches ML compute instances and deploys the model (model artifacts and inference code) on them. In response, you get an endpoint. To get inferences from the model, your application code can send requests to the endpoint.

The `sagemaker.mxnet.MXNet` class in the high-level Python library provides the `deploy` method for quickly deploying your model. The `deploy` method does the following in order:

1. Creates an Amazon SageMaker model by calling the [CreateModel](p. 253) API. The model that you create in Amazon SageMaker holds information such as location of the model artifacts and the inference code image.
2. Creates an endpoint configuration by calling the [CreateEndpointConfig](p. 246) API. This configuration holds necessary information including the name of the model (which was created in
the preceding step) and the resource configuration (the type and number of ML compute instances to launch for hosting).

3. Creates the endpoint by calling the CreateEndpoint (p. 243) API and specifying the endpoint configuration created in the preceding step. Amazon SageMaker launches ML compute instances as specified in the endpoint configuration, and deploys the model on them.

Copy, paste, and run the following code:

```python
%%time
predictor = mnist_estimator.deploy(initial_instance_count=1,
instance_type='ml.m4.xlarge')
```

When the status of the endpoint is INSERVICE the API returns a `MXNetPredictor` object. Use the `predict` method of this object to obtain inferences.

**Note**
You can deploy a model to an endpoint hosted on your local computer by specifying `local` as the value for `train_instance_type` and 1 as the value for `train_instance_count`. For more information about local mode, see https://github.com/aws/sagemaker-python-sdk#local-mode in the Amazon SageMaker Python SDK.

**Next Step**

**Step 4 : Invoke the Endpoint to Get Inferences** (p. 189)

Your model is now deployed on Amazon SageMaker. To get inferences, send requests using the `predict` method provided by the `MXNetPredictor` object (returned in the preceding section). The method calls the Amazon SageMaker `InvokeEndpoint` (p. 347).

This example provides an HTML canvas that you can use to draw a number using your mouse. The test code sends this image to the model for inference.

1. Display the canvas by copying, pasting, and running the following code:

   ```python
   from IPython.display import HTML
   HTML(open("/home/ec2-user/sample-notebooks/sagemaker-python-sdk/mxnet_mnist/
input.html").read())
   ```

2. Use your mouse to draw a single-digit number on the canvas.
3. Run the `predict` method to get inference from the model.

   ```python
   response = predictor.predict(data)
   print('Raw prediction result:')
   print(response)
   labeled_predictions = list(zip(range(10), response[0]))
   print('Labeled predictions: ')
   print(labeled_predictions)
   labeled_predictions.sort(key=lambda label_and_prob: 1.0 - label_and_prob[1])
   print('Most likely answer: {}\n'.format(labeled_predictions[0]))
   ```

   The following is an example of output. It shows the number that was inferred (7) and a number that represents the probability that the inference is correct.
Example 1: Using the Module API

Raw prediction result:

```
[[1.7489463002839933e-11, 0.006231508683413267, 8.953022916102782e-05,
  0.0872468426823616, 0.0001965702831512317, 1.7784617739380337e-05,
  3.312719196180147e-11, 0.7383657097816467, 0.009811942465603352, 0.15804021060466766]]
```

Labeled predictions:

```
[(0, 1.7489463002839933e-11), (1, 0.006231508683413267), (2, 8.953022916102782e-05),
 (3, 0.0872468426823616), (4, 0.0001965702831512317), (5, 1.7784617739380337e-05),
 (6, 3.312719196180147e-11), (7, 0.7383657097816467), (8, 0.009811942465603352), (9,
  0.15804021060466766)]
```

Most likely answer: (7, 0.7383657097816467)

In the result:

- The **Raw prediction result** is a list of 10 probability values that the model returned as inference, corresponding to the digits 0 through 9. From these values, the input digit is 7 based on the highest probability value (0.7383657097816467).
- The values are listed in order, one value for each digit (0 through 9). The model added labels to it and returned **Labeled predictions**.
- Based on the highest probability, our code returned the **Most likely answer** (digit 7).

You can now delete the resources that you created in this exercise. For more information, see Step 4: Clean up (p. 32).

Your Amazon SageMaker notebook instance includes additional examples.
Using Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. Amazon SageMaker provides an open-source container that makes writing a Chainer script and running it in Amazon SageMaker easier. For information about how to build and use the Amazon SageMaker Chainer container, see the GitHub repository at https://github.com/aws/sagemaker-chainer-container.

For information about Chainer versions supported by the Amazon SageMaker Chainer container, see Supported Versions (p. 237).

For information about writing Chainer training scripts and using Chainer estimators with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk#chainer-sagemaker-estimators.
Using PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. Amazon SageMaker provides an open-source container that makes writing a PyTorch script and running it in Amazon SageMaker easier. For information about how to build and use the Amazon SageMaker PyTorch container, see the GitHub repository at https://github.com/aws/sagemaker-pytorch-container.

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see Supported Versions (p. 237).

For information about writing PyTorch training scripts and using PyTorch estimators with Amazon SageMaker, see https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst.
Using 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 Supported Versions (p. 237).

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` 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.

Downloading 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:
    ```bash
    $ 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.

- 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>
  ...
</dependency>
```
Integrating 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 (p. 265) request. After model training has completed, Amazon SageMaker saves the model artifacts to an S3 bucket.

      Amazon SageMaker assumes the IAM role that you specified for model training to perform tasks on your behalf, for example, to read training data from an S3 bucket and to write model artifacts to an S3 bucket.
   c. Creates and returns a SageMakerModel object. The constructor does the following tasks, which are related to deploying your model to Amazon SageMaker.
      i. Sends a CreateModel request (p. 253) to Amazon SageMaker.
      ii. Sends a CreateEndpointConfig request (p. 246) to Amazon SageMaker.
      iii. Sends a CreateEndpoint request (p. 243) to Amazon SageMaker, which then launches the specified resources, and hosts the model on them.

3. You can get inferences from your model hosted in Amazon SageMaker with the SageMakerModel.transform method.

   Provide an input DataFrame with features as input. The transform method transforms it to a DataFrame containing inferences. Internally, the transform methods sends a request to the InvokeEndpoint API to get inferences. The transform method appends the inferences to the input DataFrame.
Example 1: Using Amazon SageMaker for Training and Inference with Apache Spark

Topics

- Using Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark (p. 199)
- Using the SageMakerEstimator in a Spark Pipeline (p. 200)

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 Using Apache Spark with Amazon SageMaker (p. 193). 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. 106).

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")
```
Example 1: Amazon SageMaker with Apache Spark

```scala
// load mnist data as a dataframe from libsvm
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):

```
// 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|      |label|            features|
+-----+--------------------+      +-----+--------------------+
|  5.0|(784,[152,153,154...|      |  6.0|(784,[151,152,153...|
|  0.0|(784,[127,128,129...|      |  1.0|(784,[124,125,126...|
|  4.0|(784,[160,161,162...|      |  6.0|(784,[72,73,74,99...|
|  1.0|(784,[151,152,153...|      |  1.0|(784,[151,152,153...|
|  4.0|(784,[134,135,161...|      |  1.0|(784,[151,152,153...|
|  3.0|(784,[123,124,125...|      |  1.0|(784,[151,152,153...|
|  5.0|(784,[216,217,218...|      |  1.0|(784,[151,152,153...|
|  3.0|(784,[143,144,145...|      |  6.0|(784,[72,73,74,99...|
|  6.0|(784,[72,73,74,99...|
+-----+--------------------+      +-----+--------------------+
```
Example 1: Amazon SageMaker with Apache Spark

|  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...]) |

*only showing top 20 rows*

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.

  **Note**
  The KMeansSageMakerEstimator extends the Amazon SageMaker SageMakerEstimator, which extends the Apache Spark Estimator.

```scala
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, such as requests to create a training job and API calls for deploying the model using Amazon SageMaker hosting services, to Amazon SageMaker.
Example 1: Amazon SageMaker with Apache Spark

- After the KMeansSageMakerEstimator object has been created, you set the following parameters, are 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, 0-9.
  - Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel image, making 784 features).

- Call 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, Integrating Your Apache Spark Application with Amazon SageMaker (p. 194). 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>1671.5716659921875</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>
</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.

SageMaker Spark Github Readme provides information on how to run these examples. For more information, see https://github.com/aws/sagemaker-spark/blob/master/README.md.

### Using Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark

In Example 1: Using Amazon SageMaker for Training and Inference with Apache Spark (p. 195), 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 code sample 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.

```scala
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`.

### Using the SageMakerEstimator in a Spark Pipeline

You can use `org.apache.spark.ml.Estimator` estimators and `org.apache.spark.ml.Model` models, and `SageMakerEstimator` estimators 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
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)
```
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>1386.24682611875</td>
<td>9.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[158,159,160...</td>
<td>[-42.328192193771...</td>
<td>1277.0736083984375</td>
<td>5.0</td>
</tr>
<tr>
<td>9.0</td>
<td>(784,[208,209,210...</td>
<td>[374.04390208233...</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>1496.15795894375</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>1165.123652265625</td>
<td>3.0</td>
</tr>
<tr>
<td>3.0</td>
<td>(784,[123,124,125...</td>
<td>[610.596447285397...</td>
<td>1325.93369140625</td>
<td>7.0</td>
</tr>
<tr>
<td>5.0</td>
<td>(784,[216,217,218...</td>
<td>[142.959601818422...</td>
<td>1353.493049921875</td>
<td>5.0</td>
</tr>
<tr>
<td>3.0</td>
<td>(784,[143,144,145...</td>
<td>[1036.71862533658...</td>
<td>1460.431585546875</td>
<td>7.0</td>
</tr>
<tr>
<td>6.0</td>
<td>(784,[72,73,74,99...</td>
<td>[996.740157453575...</td>
<td>1159.86315926875</td>
<td>2.0</td>
</tr>
<tr>
<td>1.0</td>
<td>(784,[151,152,153...</td>
<td>[-107.26076167417...</td>
<td>960.93623046875</td>
<td>5.0</td>
</tr>
<tr>
<td>7.0</td>
<td>(784,[211,212,213...</td>
<td>[619.773820430940...</td>
<td>1245.13623046875</td>
<td>6.0</td>
</tr>
<tr>
<td>2.0</td>
<td>(784,[151,152,153...</td>
<td>[-850.15210181716...</td>
<td>1304.437744140625</td>
<td>8.0</td>
</tr>
<tr>
<td>8.0</td>
<td>(784,[159,160,161...</td>
<td>[-370.041887230547...</td>
<td>1192.4781494140625</td>
<td>0.0</td>
</tr>
<tr>
<td>6.0</td>
<td>(784,[100,101,102...</td>
<td>[546.674328209335...</td>
<td>1277.0908203125</td>
<td>2.0</td>
</tr>
<tr>
<td>9.0</td>
<td>(784,[209,210,211...</td>
<td>[-29.259112927426...</td>
<td>1245.8182373046875</td>
<td>6.0</td>
</tr>
</tbody>
</table>
```

Additional Examples: Using Amazon SageMaker with Apache Spark

Additional examples of using Amazon SageMaker with Apache Spark are available at https://github.com/aws/sagemaker-spark/tree/master/examples.
Amazon SageMaker Libraries

The Amazon SageMaker libraries and related information is available at the following locations:

- Amazon SageMaker high-level Python library - https://github.com/aws/sagemaker-python-sdk
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. 203)
- Access Control (p. 204)

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.
• **AWS service access** – You can use an IAM role in your account to grant an AWS service permissions to access your account’s resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.

• **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances 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. 204)
- Using Identity-based Policies (IAM Policies) for Amazon SageMaker (p. 208)
- Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 210)

### 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. 205)
- Understanding Resource Ownership (p. 205)
- Managing Access to Resources (p. 205)
- Specifying Policy Elements: Resources, Actions, Effects, and Principals (p. 207)
- Specifying Conditions in a Policy (p. 207)
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>Notebook Instance</td>
<td>arn:aws:sagemaker:region:account-id:notebook-instance/instance</td>
</tr>
<tr>
<td>Model</td>
<td>arn:aws:sagemaker:region:account-id:model/modelName</td>
</tr>
<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>
</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. 238).

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?
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. 206)
- Resource-Based Policies (p. 207)

### 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": [
        {
            "Action": ["sagemaker:CreateModel"],
            "Effect": "Allow",
            "Resource": "arn:aws:sagemaker:region:account-id:model/modelName"
        },
        {
            "Action": "iam:PassRole",
            "Effect": "Allow",
```
Specifying Policy Elements: Resources, Actions, Effects, and Principals

For more information about using identity-based policies with Amazon SageMaker, see Using Identity-based Policies (IAM Policies) for Amazon SageMaker (p. 208). 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. 205) and API Reference (p. 238).

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. 205).
- **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. 210).

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. 209). 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. 204).

## Topics

- Permissions Required to Use the Amazon SageMaker Console (p. 209)
- AWS Managed (Predefined) Policies for Amazon SageMaker (p. 209)

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": "*"
      },
      {
         "Sid": "AdditionalIamPermission",
         "Effect": "Allow",
         "Action": [ "iam:PassRole" ],
         "Resource": "arn:aws:iam::account-id:role/role-name"
      }
   ],
   "Statement": [
      {
         "Sid": "AllowAdditionalIamPermission",
         "Effect": "Deny",
         "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...
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. 210).

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:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        // Populate customer VPCs, Subnets, and Security Groups for CreateNotebookInstance form
        // These permissions needed to create the notebook instance in the console
        {
            "Sid": "CreateNotebookInstanceForm",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeVpcs",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        // Create execution roles for CreateNotebookInstance, CreateTrainingJob, and CreateModel
        // Needed if creating an IAM role (for example, as part of creating a notebook instance)
        {
            "Sid": "CreateExecutionRoles",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy"
            ],
            "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.

### Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference

When you are setting up Access Control (p. 204) 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. 241)

- Required Permissions (API Action): sagemaker:AddTags
- Resources: *

**API Operation:** CreateEndpoint (p. 243)

- Required Permissions (API Action): sagemaker:CreateEndpoint
- Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

**API Operation:** CreateEndpointConfig (p. 246)

- Required Permissions (API Action): sagemaker:CreateEndpointConfig
Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: CreateModel (p. 253)

Required Permissions (API Action): sagemaker:CreateModel, iam:PassRole

Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: CreateNotebookInstance (p. 256)


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: CreateTrainingJob (p. 265)

Required Permissions (API Action): sagemaker:CreateTrainingJob, iam:PassRole

Resources: arn:aws:sagemaker:region:account-id:training-job/trainingJobName

API Operation: DeleteEndpoint (p. 270)

Required Permissions (API Action): sagemaker:DeleteEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: DeleteEndpointConfig (p. 271)

Required Permissions (API Action): sagemaker:DeleteEndpointConfig

Resources: arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName

API Operation: DeleteModel (p. 272)

Required Permissions (API Action): sagemaker:DeleteModel

Resources: arn:aws:sagemaker:region:account-id:model/modelName

API Operation: DeleteNotebookInstance (p. 273)


Resources: arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName

API Operation: DeleteTags (p. 276)

Required Permissions (API Action): sagemaker:DeleteTags

Resources: *

API Operation: DescribeEndpoint (p. 278)

Required Permissions (API Action): sagemaker:DescribeEndpoint

Resources: arn:aws:sagemaker:region:account-id:endpoint/endpointName

API Operation: DescribeEndpointConfig (p. 281)

Required Permissions (API Action): sagemaker:DescribeEndpointConfig
Resources: `arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName`

**API Operation:** DescribeModel (p. 288)

Required Permissions (API Action): sagemaker:DescribeModel

Resources: `arn:aws:sagemaker:region:account-id:model/modelName`

**API Operation:** DescribeNotebookInstance (p. 291)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

**API Operation:** DescribeTrainingJob (p. 298)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

Resources: `arn:aws:sagemaker:region:account-id:training-job/trainingJobName`

**API Operation:** CreatePresignedNotebookInstanceUrl (p. 263)

Required Permissions (API Action): sagemaker:CreatePresignedNotebookInstanceUrl

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

**API Operation:** InvokeEndpoint (p. 347)

Required Permissions (API Action): sagemaker:InvokeEndpoint

Resources: `arn:aws:sagemaker:region:account-id:endpoint/endpointName`

**API Operation:** ListEndpointConfigs (p. 303)

Required Permissions (API Action): sagemaker:ListEndpointConfigs

Resources: *

**API Operation:** ListEndpoints (p. 306)

Required Permissions (API Action): sagemaker:ListEndpoints

Resources: *

**API Operation:** ListModels (p. 313)

Required Permissions (API Action): sagemaker:ListModels

Resources: *

**API Operation:** ListNotebookInstances (p. 319)

Required Permissions (API Action): sagemaker:ListNotebookInstances

Resources: *

**API Operation:** ListTags (p. 323)

Required Permissions (API Action): sagemaker:ListTags

Resources: *

**API Operation:** ListTrainingJobs (p. 325)

Required Permissions (API Action): sagemaker:ListTrainingJobs

Resources: *

**API Operation:** StartNotebookInstance (p. 331)

Required Permissions (API Action): sagemaker:StartNotebookInstance
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. 256), CreateHyperParameterTuningJob (p. 249), CreateTrainingJob (p. 265), and CreateModel (p. 253).

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. 221). 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. 214)
- CreateHyperParameterTuningJob API: Execution Role Permissions (p. 216)
- CreateTrainingJob API: Execution Role Permissions (p. 218)
- CreateModel API: Execution Role Permissions (p. 220)
- Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role (p. 221)

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:

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "sagemaker:*",
            "ecr:GetAuthorizationToken",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability",
            "cloudwatch:PutMetricData",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:DescribeLogStreams",
            "logs:PutLogEvents",
            "logs:GetLogEvents",
            "s3:CreateBucket",
            "s3:ListBucket",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:DeleteObject"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": "sagemaker.amazonaws.com"
            }
        }
    }
    ]
}
```
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": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "sagemaker.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::inputbucket/object1",
        "arn:aws:s3:::outputbucket/path",
        "arn:aws:s3:::inputbucket/object2",
        "arn:aws:s3:::inputbucket/object3"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": [
        "arn:aws:ecr:::repository/my-repo1",
        "arn:aws:ecr:::repository/my-repo2"
      ]
    }
  ]
}
```
CreateHyperParameterTuningJob API: Execution Role Permissions

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`
    - `SuplementalContainers.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:

```
{
  "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": "arn:aws:ecr:::repository/my-repo3"
    }
  ]
}
```
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"
      ]
    },
    {
      "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.
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:

```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 the specifying "Resource": "*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```json
{
    "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": "s3://my-bucket"  
}
```
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"
    ],
    "Resource": "*"
}
```

**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",
                "logs:CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of the specifying "Resource": "*", you can 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"
            ]
        }
    ]
}
```
Using the AWS Managed Permission Policy
(AmazonSageMakerFullAccess) for an Execution Role

You can create an execution role one of two ways:

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. 253) 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:
• 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-".
Monitoring 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.

Monitoring 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. 347)`.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModelLatency</td>
<td>The latency of the model's response, as viewed from Amazon SageMaker.</td>
</tr>
<tr>
<td></td>
<td>Units: Microseconds</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum, Min, Max, Sample Count</td>
</tr>
<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: Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Invocation5XXErrors</td>
<td>The number of InvokeEndpoint 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>Invocations</td>
<td>The number of InvokeEndpoint requests sent to a model. To get the total number of requests to the endpoint variant, use the Sum statistic.</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.</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</td>
<td>Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.</td>
</tr>
</tbody>
</table>

### Training Job and Endpoint Instance Metrics

The /aws/sagemaker/TrainingJobs 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%. For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance. For training jobs, the value is the CPU utilization of the Algorithm container on the instance. Units: Percent</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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 endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance. For training jobs, the value is the memory utilization of the Algorithm container on the instance. 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%. For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance. For training jobs, the value is the GPU utilization of the Algorithm container on the instance. 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%. For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance. For training jobs, the value is the GPU memory utilization of the Algorithm container on the instance. Units: Percent</td>
</tr>
</tbody>
</table>

**Dimensions for Training Job and Endpoint Instance Metrics**

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>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 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.</td>
</tr>
</tbody>
</table>

**Training Job Instance Metrics**
The `/aws/sagemaker/TrainingJobs` namespace includes the following metrics for the training jobs instance.

Metrics are available at a 1-minute frequency.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiskUtilization</td>
<td>The percentage of disk space that the algorithm container on an instance uses. This value can range between 0% and 100%. Units: Percent</td>
</tr>
</tbody>
</table>

Dimensions for Training Job Metrics

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>The value for this dimension has the format <code>[training-job-name]/algo-[instance-number-in-cluster]</code>. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the <code>/aws/sagemaker/TrainingJobs</code> namespace.</td>
</tr>
</tbody>
</table>

Logging Amazon SageMaker 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

<table>
<thead>
<tr>
<th>Log Group Name</th>
<th>Log Stream Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>/aws/sagemaker/TrainingJobs</td>
<td><code>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</code></td>
</tr>
<tr>
<td>/aws/sagemaker/Endpoints/</td>
<td><code>[production-variant-name]/[instance-id]</code></td>
</tr>
<tr>
<td>[EndpointName]</td>
<td></td>
</tr>
<tr>
<td>/aws/sagemaker/NotebookInstances</td>
<td><code>[notebook-instance-name]/[LifecycleConfigHook]</code></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 2.1: (Optional) Customize a Notebook Instance (p. 18).
Logging 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. By creating a trail, a configuration that enables delivery of events as log files to an Amazon Simple Storage Service (Amazon S3) bucket, you can enable continuous delivery of CloudTrail events to an S3 bucket, Amazon CloudWatch Logs, and Amazon CloudWatch Events. Use the information collected by CloudTrail to 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, including how to configure and enable it, see the AWS CloudTrail User Guide.

Amazon SageMaker Information in CloudTrail

All Amazon SageMaker actions except the Amazon SageMaker Runtime action InvokeEndpoint are logged by CloudTrail. For example, calls to the CreateTrainingJob, CreateModel, and CreateEndpoint actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. This information helps you determine the following:

- Whether the request was made with root or 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.

When you create a trail, you can store your log files in your S3 bucket for as long as you want. You can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

You can aggregate Amazon SageMaker log files from multiple AWS Regions and multiple AWS accounts into a single S3 bucket. For more information, see Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts.

To be notified of log file delivery, configure CloudTrail to publish Amazon Simple Notification Service (Amazon SNS) notifications. For more information, see Configuring Amazon SNS Notifications for CloudTrail.

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",
```
The following example is a log entry for the CreateModel action, which creates one or more containers to host a previously trained model.

```
{
  "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": {
  },
  "requestID": "417b8dab-EXAMPLE",
  "eventID": "0f2b3e81-EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "444455556666"
}
```
Guidelines and Limits

This section provides guidelines for securing notebook instances and preserving changes you make to sample notebooks, allowing training and hosting instances to access data in your VPC, and lists supported versions of learning frameworks.

For Amazon SageMaker service limits, see Amazon SageMaker Limits.

Topics
- Notebook Instance Security (p. 229)
- Saving Updated Sample Notebooks in a New Location (p. 230)
- Installing External Libraries and Kernels in Notebook Instances (p. 230)
- Protect Training Jobs by Using an Amazon Virtual Private Cloud (p. 232)
- Protect Models by Using an Amazon Virtual Private Cloud (p. 234)
- Supported Versions (p. 237)

Notebook Instance Security

Note the following security considerations for notebook instances.

Topics
- Notebook Instances Are Enabled with Internet Access by Default (p. 229)
- Notebook Instances Provide the Best Experience for a Single User (p. 229)

Notebook Instances Are Enabled with Internet Access by Default

Amazon SageMaker notebook instances are Internet-enabled. This allows data scientists to download popular packages and notebooks, customize their 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 a NAT gateway and your security groups 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 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:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker:CreatePresignedNotebookInstanceUrl",
    }
  ]
}
```

Saving Updated Sample Notebooks in a New Location

Amazon SageMaker provides several sample notebooks in your notebook instance. Each of these samples provides step-by-step instructions for training a model, and deploying and validating it. As you explore these samples, you might modify their code. We recommend that you move the modified files and folders out of the `/sample-notebooks` folder.

When you stop a notebook instance, Amazon SageMaker frees up the resource (a machine learning instance). When you restart your notebook instance, Amazon SageMaker provisions a new machine learning instance with the latest sample notebooks and an updated version of the Amazon Machine Image (AMI).

As a result, when you stop an instance, you lose changes that you made in any of the files in the `/sample-notebooks` folder. To save your changes, move the relevant files and folders in the `/sample-notebook` folder to any other folder within the SageMaker folder. When you open a notebook instance, Amazon SageMaker expects to find it in the SageMaker folder by default. Amazon SageMaker saves files, except for those in the `/sample-notebooks` subfolder, in the SageMaker folder.

To move files and folders, use the Move command in the Jupyter dashboard.

Installing 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`.

For example, 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.

**For example, 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
   ```
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.

Configuring 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. 265) 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"
    ]
}
```

Configuring 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 available 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 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.

```
{
  "Version": "2008-10-17",
  "Statement": [ 
    {
      "Effect": "Allow",
      "Principal": "*",
      "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:

```
{
  "Statement": [ 
    {
      "Sid": "AmazonLinuxAMIRepositoryAccess",
      "Principal": "*",
      "Action": [ "s3:GetObject" ],
      "Effect": "Deny",
    }
  ]
}
```
"Statement": [ 
  { "Sid": "AmazonLinux2AMIRepositoryAccess", 
    "Principal": "*", 
    "Action": [ "s3:GetObject" ], 
    "Effect": "Deny", 
    "Resource": [ "arn:aws:s3:::amazonlinux.*.amazonaws.com/*" ] 
  } 
] 

Configure the VPC Security Group

In distributed training, you must allow communication between the different containers in the same training 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.

Connecting to Resources Outside Your VPC

If you configure your VPC so that it doesn’t have internet access, training jobs that use that VPC do not have access to resources outside your VPC. If your training jobs need access to resources outside your VPC, provide access with one of the following options:

- If your training 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 training 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.

Protect Models 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.
Configuring a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the VpcConfig request parameter of the CreateModel (p. 253) API or when you create a model in the Amazon SageMaker console. Amazon 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:

```
VpcConfig: {
    "Subnets": [
        "subnet-0123456789abcdef0",
        "subnet-0123456789abcdef1",
        "subnet-0123456789abcdef2"
    ],
    "SecurityGroupIds": [
        "sg-0123456789abcdef0"
    ]
}
```

Configuring 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 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 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 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.

```
{
    "Version": "2008-10-17",
    "Statement": [
        { "Effect": "Allow",
          "Principal": "*",
          "Action": [ "s3:GetObject",
                        "s3:PutObject",
                        "s3:ListBucket",
                        "s3:GetBucketLocation",
                        "s3:GetObjectVersion"
                      ]
        }
    ]
}
```
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/*"
            ]
        }
    ]
}
```

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 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.

Supported Versions

Amazon SageMaker supports the following versions of learning frameworks and computing systems.

Deep Learning Framework Containers

Amazon SageMaker Deep Learning containers support:

- Apache MXNet 0.12, 1.0, and 1.1
- Chainer 4.0.0
- PyTorch 0.4.0
- TensorFlow 1.4, 1.5, 1.6, 1.7, and 1.8

Containers use the most recent version of a framework by default. To use an earlier version, set the value of the framework_version parameter in the TensorFlow and MXNet estimators. For more information about using TensorFlow and MXNet estimators, see the Amazon SageMaker Python SDK.

When using your own algorithms, you can use any version of the frameworks in your Docker image.

Notebook Instances

Amazon SageMaker notebook instances are installed with kernels that support:

- TensorFlow 1.8
- Apache MXNet 1.2
- Apache Spark 2.1.1 and 2.2.0
- Chainer 4.0
- PyTorch 0.4
API Reference

This section contains the API Reference documentation.

Topics

- Actions (p. 238)
- Data Types (p. 349)

Actions

The following actions are supported by Amazon SageMaker Service:

- AddTags (p. 241)
- CreateEndpoint (p. 243)
- CreateEndpointConfig (p. 246)
- CreateHyperParameterTuningJob (p. 249)
- CreateModel (p. 253)
- CreateNotebookInstance (p. 256)
- CreateNotebookInstanceLifecycleConfig (p. 260)
- CreatePresignedNotebookInstanceUrl (p. 263)
- CreateTrainingJob (p. 265)
- DeleteEndpoint (p. 270)
- DeleteEndpointConfig (p. 271)
- DeleteModel (p. 272)
- DeleteNotebookInstance (p. 273)
- DeleteNotebookInstanceLifecycleConfig (p. 275)
- DeleteTags (p. 276)
- DescribeEndpoint (p. 278)
- DescribeEndpointConfig (p. 281)
- DescribeHyperParameterTuningJob (p. 283)
- DescribeModel (p. 288)
- DescribeNotebookInstance (p. 291)
- DescribeNotebookInstanceLifecycleConfig (p. 295)
- DescribeTrainingJob (p. 298)
- ListEndpointConfigs (p. 303)
- ListEndpoints (p. 306)
- ListHyperParameterTuningJobs (p. 309)
- ListModels (p. 313)
- ListNotebookInstanceLifecycleConfigs (p. 316)
- ListNotebookInstances (p. 319)
- ListTags (p. 323)
- ListTrainingJobs (p. 325)
- ListTrainingJobsForHyperParameterTuningJob (p. 328)
• StartNotebookInstance (p. 331)
• StopHyperParameterTuningJob (p. 333)
• StopNotebookInstance (p. 335)
• StopTrainingJob (p. 337)
• UpdateEndpoint (p. 339)
• UpdateEndpointWeightsAndCapacities (p. 341)
• UpdateNotebookInstance (p. 343)
• UpdateNotebookInstanceLifecycleConfig (p. 345)

The following actions are supported by Amazon SageMaker Runtime:

• InvokeEndpoint (p. 347)

Amazon SageMaker Service

The following actions are supported by Amazon SageMaker Service:

• AddTags (p. 241)
• CreateEndpoint (p. 243)
• CreateEndpointConfig (p. 246)
• CreateHyperParameterTuningJob (p. 249)
• CreateModel (p. 253)
• CreateNotebookInstance (p. 256)
• CreateNotebookInstanceLifecycleConfig (p. 260)
• CreatePresignedNotebookInstanceUrl (p. 263)
• CreateTrainingJob (p. 265)
• DeleteEndpoint (p. 270)
• DeleteEndpointConfig (p. 271)
• DeleteModel (p. 272)
• DeleteNotebookInstance (p. 273)
• DeleteNotebookInstanceLifecycleConfig (p. 275)
• DeleteTags (p. 276)
• DescribeEndpoint (p. 278)
• DescribeEndpointConfig (p. 281)
• DescribeHyperParameterTuningJob (p. 283)
• DescribeModel (p. 288)
• DescribeNotebookInstance (p. 291)
• DescribeNotebookInstanceLifecycleConfig (p. 295)
• DescribeTrainingJob (p. 298)
• ListEndpointConfigs (p. 303)
• ListEndpoints (p. 306)
• ListHyperParameterTuningJobs (p. 309)
• ListModels (p. 313)
• ListNotebookInstanceLifecycleConfigs (p. 316)
• ListNotebookInstances (p. 319)
• ListTags (p. 323)
• ListTrainingJobs (p. 325)
• ListTrainingJobsForHyperParameterTuningJob (p. 328)
• StartNotebookInstance (p. 331)
• StopHyperParameterTuningJob (p. 333)
• StopNotebookInstance (p. 335)
• StopTrainingJob (p. 337)
• UpdateEndpoint (p. 339)
• UpdateEndpointWeightsAndCapacities (p. 341)
• UpdateNotebookInstance (p. 343)
• UpdateNotebookInstanceLifecycleConfig (p. 345)
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, 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 Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

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. 402).

The request accepts the following data in JSON format.

ResourceArn (p. 241)

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. 241)

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. 395) 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. 241)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of Tag (p. 395) 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. 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.

**Request Syntax**

```json
{
  "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. 402).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 243)**

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][a-zA-Z0-9]`*

Required: Yes

**EndpointName (p. 243)**

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][-\[a-zA-Z0-9\]]*\^[a-zA-Z0-9][-\[a-zA-Z0-9\]]*

Required: Yes

Tags (p. 243)

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. 395) objects

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

Required: No

Response Syntax

```
{
  "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. 244)

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. 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
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": [
    {
      "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. 402).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 246)**

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](-*[a-zA-Z0-9])*  
Required: Yes
KmsKeyId (p. 246)

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. 246)

An array of ProductionVariant objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 385) objects

Array Members: Minimum number of 1 item.

Required: Yes

Tags (p. 246)

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. 395) 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. 247)

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. 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
CreateHyperParameterTuningJob
Service: Amazon SageMaker Service

Starts a hyperparameter tuning job.

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"
    },
    "HyperParameterTuningJobName": "string",
    "Tags": [
        {
            "Key": "string",
            "Value": "string"
        }
    ],
    "TrainingJobDefinition": {
        "AlgorithmSpecification": {
            "MetricDefinitions": [
                {
                    "Name": "string",
                    "Regex": "string"
                }
            ],
            "TrainingImage": "string",
            "TrainingInputMode": "string"
        },
        "InputDataConfig": [
            {
                "ChannelName": "string",
                "CompressionType": "string",
                "ContentType": "string"
            }
        ]
    }
}
```
Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

HyperParameterTuningJobConfig (p. 249)

The HyperParameterTuningJobConfig (p. 370) object describes the tuning job, including the search strategy, metric used to evaluate training jobs, ranges of parameters to search, and resource limits for the tuning job.

Type: HyperParameterTuningJobConfig (p. 370) object

Required: Yes

HyperParameterTuningJobName (p. 249)

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. Names are not case sensitive, and must be between 1-32 characters.

Type: String


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

Required: Yes
Tags (p. 249)

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 Using Cost Allocation Tags in the AWS Billing and Cost Management User Guide.

Type: Array of Tag (p. 395) objects

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

Required: No

TrainingJobDefinition (p. 249)

The HyperParameterTrainingJobDefinition (p. 366) 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 (p. 366) object

Required: Yes

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. 251)

The Amazon Resource Name (ARN) of the 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/.*

Errors

For information about the errors that are common to all actions, see Common Errors (p. 400).

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 one or more containers. For each 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 into production.

Use this API to create a model only if you want to use Amazon SageMaker hosting services. 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.

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. 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
{
  "ExecutionRoleArn": "string",
  "ModelName": "string",
  "PrimaryContainer": {
    "ContainerHostname": "string",
    "Environment": {
      "string": "string"
    },
    "Image": "string",
    "ModelDataUrl": "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. 402).

The request accepts the following data in JSON format.

**ExecutionRoleArn (p. 253)**

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. 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-z-]*:iam::\d{12}:role/?[a-zA-Z0-9+=,.@-_]+$  

Required: Yes

**ModelName (p. 253)**
The name of the new model.

Type: String

Length Constraints: Maximum length of 63.

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

Required: Yes

**PrimaryContainer (p. 253)**
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 into production.

Type: ContainerDefinition (p. 356) object

Required: Yes

**Tags (p. 253)**
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. 395) objects

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

Required: No

**VpcConfig (p. 253)**
A VpcConfig (p. 400) 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. For more information, see Protect Models by Using an Amazon Virtual Private Cloud (p. 234).

Type: VpcConfig (p. 400) 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. 254)**

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. 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
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

```
{
    "DirectInternetAccess": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LifecycleConfigName": "string",
    "NotebookInstanceName": "string",
    "RoleArn": "string",
    "SecurityGroupIds": [ "string" ],
    "SubnetId": "string",
    "Tags": [ 
        { 
            "Key": "string",
            "Value": "string"
        } 
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).
The request accepts the following data in JSON format.

**DirectInternetAccess (p. 256)**

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 Enabled with Internet Access by Default (p. 229). 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. 256)**

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.m4.xlarge` | `ml.m4.2xlarge` | `ml.m4.4xlarge` | `ml.m4.10xlarge` | `ml.m4.16xlarge` | `ml.p2.xlarge` | `ml.p2.8xlarge` | `ml.p2.16xlarge` | `ml.p3.2xlarge` | `ml.p3.8xlarge` | `ml.p3.16xlarge`

Required: Yes

**KmsKeyId (p. 256)**

If you provide a 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.

Type: String

Length Constraints: Maximum length of 2048.

Required: No

**LifecycleConfigName (p. 256)**

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 (p. 18).

Type: String

Length Constraints: Maximum length of 63.

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

Required: No

**NotebookInstanceName (p. 256)**

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. 256)**

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

Required: Yes

**SecurityGroupIds (p. 256)**

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. 256)**

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. 256)**

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. 395) objects

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

Required: No

**Response Syntax**

```json
{
    "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. 258)**

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. 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
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 (p. 18).

**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. 402).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 260)**

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. 260)**

A shell script that runs only once, when you create a notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 379) objects

**NotebookInstanceLifecycleHook (p. 379)**

A lifecycle hook to run during the lifecycle of a notebook instance.
Array Members: Maximum number of 1 item.

Required: No

OnStart (p. 260)

A shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 379) 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. 261)

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. 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
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.

Request Syntax

```json
{
    "NotebookInstanceName": "string",
    "SessionExpirationDurationInSeconds": number
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 263)**

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. 263)**

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. 263)**

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. 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
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": {
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "HyperParameters": {
      "string": "string"
   },
   "InputDataConfig": [
      {
         "ChannelName": "string",
         "CompressionType": "string",
         "ContentType": "string",
         "DataSource": {
            "S3DataSource": {
               "S3DataDistributionType": "string",
               "S3DataType": "string",
               "S3Uri": "string"
            } },
         "RecordWrapperType": "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. 402).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 265)

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 (p. 146).

Type: AlgorithmSpecification (p. 352) object

Required: Yes

HyperParameters (p. 265)

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. 265)

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. 354) objects

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

Required: Yes

OutputDataConfig (p. 265)

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

Type: OutputDataConfig (p. 383) object

Required: Yes

ResourceConfig (p. 265)

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. 389) object

Required: Yes

RoleArn (p. 265)

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. 265)

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. 394) object

Required: Yes

Tags (p. 265)

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. 395) objects

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

Required: No

TrainingJobName (p. 265)

The name of the training job. The name must be unique within an AWS Region in an AWS account. It appears in the Amazon SageMaker console.

Type: String


Pattern: ^[a-zA-Z0-9\-[a-zA-Z0-9]*]$ (p. 265)

Required: Yes

VpcConfig (p. 265)

A VpcConfig (p. 400) 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 (p. 232)

Type: VpcConfig (p. 400) 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. 268)

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. 400).

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
DeleteEndpoint
Service: Amazon SageMaker Service

Deletes an endpoint. Amazon SageMaker frees up all of the resources that were deployed when the endpoint was created.

Request Syntax

```
{
   "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

EndpointName (p. 270)

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. 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
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

```json
{
    "EndpointConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 271)**

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. 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
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

```
{
  "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**ModelName (p. 272)**

The name of the model 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. 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
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

```json
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 273)**

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])* (\[p. 273\])

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. 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
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. 402).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 275)**

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. 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
**DeleteTags**  
Service: Amazon SageMaker Service

Deletes the specified tags from an Amazon SageMaker resource.

To list a resource's tags, use the `ListTags` 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. 402).

The request accepts the following data in JSON format.

**ResourceArn (p. 276)**

  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. 276)**

  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: `^((?!aws:)[\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. 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. 402).

The request accepts the following data in JSON format.

**EndpointName (p. 278)**

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,
            "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. 278)**

A timestamp that shows when the endpoint was created.
Type: Timestamp

EndpointArn (p. 278)

The Amazon Resource Name (ARN) of the endpoint.

Type: String


EndpointConfigName (p. 278)

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. 278)

Name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

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

EndpointStatus (p. 278)

The status of the endpoint.

Type: String

Valid Values: OutOfService | Creating | Updating | RollingBack | InService | Deleting | Failed

FailureReason (p. 278)

If the status of the endpoint is Failed, the reason why it failed.

Type: String

Length Constraints: Maximum length of 1024.

LastModifiedTime (p. 278)

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

ProductionVariants (p. 278)

An array of ProductionVariant objects, one for each model hosted behind this endpoint.

Type: Array of ProductionVariantSummary (p. 387) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 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
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. 402).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 281)**

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": [
    {
      "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. 281)**

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp
**EndpointConfigArn (p. 281)**

The Amazon Resource Name (ARN) of the endpoint configuration.

Type: String


**EndpointConfigName (p. 281)**

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. 281)**

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. 281)**

An array of ProductionVariant objects, one for each model that you want to host at this endpoint.

Type: Array of ProductionVariant (p. 385) objects

Array Members: Minimum number of 1 item.

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 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
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. 402).

The request accepts the following data in JSON format.

HyperParameterTuningJobName (p. 283)

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"
        },
        "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"  
},  
"HyperParameterTuningJobName": "string",  
"HyperParameterTuningJobStatus": "string",  
"LastModifiedTime": number,  
"ObjectiveStatusCounters": {  
  "Failed": number,  
  "Pending": number,  
  "Succeeded": number  
},  
"TrainingJobDefinition": {  
  "AlgorithmSpecification": {  
    "MetricDefinitions": [  
      {  
        "Name": "string",  
        "Regex": "string"  
      }  
    ],  
    "TrainingImage": "string",  
    "TrainingInputMode": "string"  
  },  
  "InputDataConfig": [  
    {  
      "ChannelName": "string",  
      "CompressionType": "string",  
      "ContentType": "string",  
      "DataSource": {  
        "S3DataSource": {  
          "S3DataDistributionType": "string",  
          "S3DataType": "string",  
          "S3Uri": "string"  
        }  
      },  
      "RecordWrapperType": "string"  
    }  
  ],  
  "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  
}  
}

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. 283)**

A TrainingJobSummary (p. 398) object that describes the training job that completed with the best current HyperParameterTuningJobObjective (p. 371).

Type: HyperParameterTrainingJobSummary (p. 368) object

**CreationTime (p. 283)**

The date and time that the tuning job started.

Type: Timestamp

**FailureReason (p. 283)**

If the tuning job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**HyperParameterTuningEndTime (p. 283)**

The date and time that the tuning job ended.

Type: Timestamp

**HyperParameterTuningJobArn (p. 283)**

The Amazon Resource Name (ARN) of the 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/.*`

**HyperParameterTuningJobConfig (p. 283)**

The `HyperParameterTuningJobConfig` object that specifies the configuration of the tuning job.

Type: `HyperParameterTuningJobConfig` object

**HyperParameterTuningJobName (p. 283)**

The name of the tuning job.

Type: `String`


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

**HyperParameterTuningJobStatus (p. 283)**

The status of the tuning job: InProgress, Completed, Failed, Stopping, or Stopped.

Type: `String`

Valid Values: Completed | InProgress | Failed | Stopped | Stopping

**LastModifiedTime (p. 283)**

The date and time that the status of the tuning job was modified.

Type: `Timestamp`

**ObjectiveStatusCounters (p. 283)**

The `ObjectiveStatusCounters` object that specifies the number of training jobs, categorized by the status of their final objective metric, that this tuning job launched.

Type: `ObjectiveStatusCounters` object

**TrainingJobDefinition (p. 283)**

The `HyperParameterTrainingJobDefinition` object that specifies the definition of the training jobs that this tuning job launches.

Type: `HyperParameterTrainingJobDefinition` object

**TrainingJobStatusCounters (p. 283)**

The `TrainingJobStatusCounters` object that specifies the number of training jobs, categorized by status, that this tuning job launched.

Type: `TrainingJobStatusCounters` object

**Errors**

For information about the errors that are common to all actions, see `Common Errors (p. 400)`.

**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

```json
{
    "ModelName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**ModelName (p. 288)**

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

```json
{
    "CreationTime": number,
    "ExecutionRoleArn": "string",
    "ModelArn": "string",
    "ModelName": "string",
    "PrimaryContainer": {
        "ContainerHostname": "string",
        "Environment": { "string": "string" }
    },
    "Image": "string",
    "ModelDataUrl": "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. 288)**

A timestamp that shows when the model was created.
Type: Timestamp

**ExecutionRoleArn (p. 288)**

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. 288)**

The Amazon Resource Name (ARN) of the model.

Type: String


**ModelName (p. 288)**

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. 288)**

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. 356) object

**VpcConfig (p. 288)**

A VpcConfig (p. 400) object that specifies the VPC that this model has access to. For more information, see Protect Models by Using an Amazon Virtual Private Cloud (p. 234)

Type: VpcConfig (p. 400) object

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 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
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. 402).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 291)**

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])*\n
Required: Yes

Response Syntax

```json
{
    "CreationTime": number,
    "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"
}
```

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. 291)**

A timestamp. Use this parameter to return the time when the notebook instance was created.
DirectInternetAccess (p. 291)

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 Enabled with Internet Access by Default (p. 229).

Type: String

Valid Values: Enabled | Disabled

FailureReason (p. 291)

If status is failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

InstanceType (p. 291)

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.m4.xlarge
- ml.m4.2xlarge
- ml.m4.4xlarge
- ml.m4.10xlarge
- ml.m4.16xlarge
- ml.p2.xlarge
- ml.p2.8xlarge
- ml.p2.16xlarge
- ml.p3.2xlarge
- ml.p3.8xlarge
- ml.p3.16xlarge

KmsKeyId (p. 291)

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. 291)

A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.

Type: Timestamp

NetworkInterfaceId (p. 291)

Network interface IDs that Amazon SageMaker created at the time of creating the instance.

Type: String

NotebookInstanceArn (p. 291)

The Amazon Resource Name (ARN) of the notebook instance.

Type: String

Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 291)

Returns the name of a notebook instance lifecycle configuration.

For information about notebook instance lifecycle configurations, see Step 2.1: (Optional) Customize a Notebook Instance (p. 18).
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
**NotebookInstanceName (p. 291)**
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. 291)**
The status of the notebook instance.

Type: String
Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting

**RoleArn (p. 291)**
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-ZA-Z0-9+=,.@\-_\/]+$

**SecurityGroups (p. 291)**
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. 291)**
The ID of the VPC subnet.

Type: String
Length Constraints: Maximum length of 32.

**Url (p. 291)**
The URL that you use to connect to the Jupyter notebook that is running in your notebook instance.

Type: String

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 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
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 (p. 18).

Request Syntax

```
{
    "NotebookInstanceLifecycleConfigName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 295)**

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

```
{
    "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. 295)
A timestamp that tells when the lifecycle configuration was created.
Type: Timestamp

LastModifiedTime (p. 295)
A timestamp that tells when the lifecycle configuration was last modified.
Type: Timestamp

NotebookInstanceLifecycleConfigArn (p. 295)
The Amazon Resource Name (ARN) of the lifecycle configuration.
Type: String
Length Constraints: Maximum length of 256.

NotebookInstanceLifecycleConfigName (p. 295)
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. 295)
The shell script that runs only once, when you create a notebook instance.
Type: Array of NotebookInstanceLifecycleHook (p. 379) objects
Array Members: Maximum number of 1 item.

OnStart (p. 295)
The shell script that runs every time you start a notebook instance, including when you create the notebook instance.
Type: Array of NotebookInstanceLifecycleHook (p. 379) objects
Array Members: Maximum number of 1 item.

Errors
For information about the errors that are common to all actions, see Common Errors (p. 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
DescribeTrainingJob
Service: Amazon SageMaker Service

Returns information about a training job.

Request Syntax

```json
{
   "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

TrainingJobName (p. 298)

The name of the training job.

Type: String


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

Required: Yes

Response Syntax

```json
{
   "AlgorithmSpecification": {
      "TrainingImage": "string",
      "TrainingInputMode": "string"
   },
   "CreationTime": number,
   "FailureReason": "string",
   "HyperParameters": {
      "string": "string"
   },
   "InputDataConfig": [
      {
         "ChannelName": "string",
         "CompressionType": "string",
         "ContentType": "string",
         "DataSource": {
            "S3DataSource": {
               "S3DataDistributionType": "string",
               "S3DataType": "string",
               "S3Uri": "string"
            }
         },
         "RecordWrapperType": "string"
      }
   ],
   "LastModifiedTime": number,
   "ModelArtifacts": {
   }
}
```
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. 298)**

Information about the algorithm used for training, and algorithm metadata.

Type: AlgorithmSpecification (p. 352) object

**CreationTime (p. 298)**

A timestamp that indicates when the training job was created.

Type: Timestamp

**FailureReason (p. 298)**

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**HyperParameters (p. 298)**

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.
InputDataConfig (p. 298)

An array of Channel objects that describes each data input channel.

Type: Array of Channel (p. 354) objects

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

LastModifiedTime (p. 298)

A timestamp that indicates when the status of the training job was last modified.

Type: Timestamp

ModelArtifacts (p. 298)

Information about the Amazon S3 location that is configured for storing model artifacts.

Type: ModelArtifacts (p. 376) object

OutputDataConfig (p. 298)

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. 383) object

ResourceConfig (p. 298)

Resources, including ML compute instances and ML storage volumes, that are configured for model training.

Type: ResourceConfig (p. 389) object

RoleArn (p. 298)

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+=,.@\-_/]+$

SecondaryStatus (p. 298)

Provides granular information about the system state. For more information, see TrainingJobStatus.

Type: String

Valid Values: Starting | Downloading | Training | Uploading | Stopping | Stopped | MaxRuntimeExceeded | Completed | Failed

StoppingCondition (p. 298)

The condition under which to stop the training job.

Type: StoppingCondition (p. 394) object

TrainingEndTime (p. 298)

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. 298)**

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/.*`

**TrainingJobName (p. 298)**

Name of the model training job.

Type: String


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

**TrainingJobStatus (p. 298)**

The status of the training job.

For the InProgress status, Amazon SageMaker can return these secondary statuses:
- Starting - Preparing for training.
- Downloading - Optional stage for algorithms that support File training input mode. It indicates data is being downloaded to ML storage volumes.
- Training - Training is in progress.
- Uploading - Training is complete and model upload is in progress.

For the Stopped training status, Amazon SageMaker can return these secondary statuses:
- MaxRuntimeExceeded - Job stopped as a result of maximum allowed runtime exceeded.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

**TrainingStartTime (p. 298)**

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. 298)**

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. 298)**

A `VpcConfig (p. 400)` 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. 232).
Type: VpcConfig (p. 400) object

Errors
For information about the errors that are common to all actions, see Common Errors (p. 400).

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
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. 402).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 303)**

A filter that returns only endpoint configurations created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 303)**

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 303)**

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. 303)**

A string in the endpoint configuration name. This filter returns only endpoint configurations whose name contains the specified string.

Type: String

Pattern: [a-zA-Z0-9-]+

Required: No
**NextToken (p. 303)**

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. 303)**

The field to sort results by. The default is `CreationTime`.

Type: String

Valid Values: Name | CreationTime

Required: No

**SortOrder (p. 303)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**Response Syntax**

```json
{
  "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. 304)**

An array of endpoint configurations.

Type: Array of `EndpointConfigSummary (p. 361)` objects

**NextToken (p. 304)**

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. 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
ListEndpoints
Service: Amazon SageMaker Service

Lists endpoints.

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. 402).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 306)

A filter that returns only endpoints that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 306)

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 306)

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 306)

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

MaxResults (p. 306)

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. 306)**

A string in endpoint names. This filter returns only endpoints whose name contains the specified string.

Type: String

Pattern: \[a-zA-Z0-9-]+

Required: No

**NextToken (p. 306)**

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. 306)**

Sorts the list of results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 306)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 306)**

A filter that returns only endpoints with the specified status.

Type: String

Valid Values: OutOfService | Creating | Updating | RollingBack | InService | Deleting | Failed

Required: No

**Response Syntax**

```json
{
  "Endpoints": [
    {
      "CreationTime": number,
      "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.

Endpoints (p. 307)

An array or endpoint objects.

Type: Array of EndpointSummary (p. 362) objects

NextToken (p. 307)

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. 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
ListHyperParameterTuningJobs
Service: Amazon SageMaker Service

Gets a list of HyperParameterTuningJobSummary (p. 372) 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. 402).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 309)

A filter that returns only tuning jobs that were created after the specified time.

Type: Timestamp

Required: No

CreationTimeBefore (p. 309)

A filter that returns only tuning jobs that were created before the specified time.

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 309)

A filter that returns only tuning jobs that were modified after the specified time.

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 309)

A filter that returns only tuning jobs that were modified before the specified time.

Type: Timestamp

Required: No

MaxResults (p. 309)

The maximum number of tuning jobs to return.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

**NameContains (p. 309)**
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. 309)**
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. 309)**
The field to sort results by. The default is Name.
Type: String
Valid Values: Name | Status | CreationTime
Required: No

**SortOrder (p. 309)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 309)**
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"
]
• 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

```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. 402).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 313)

A filter that returns only models created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 313)

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 313)

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. 313)

A string in the training job name. This filter returns only models in the training job whose name contains the specified string.

Type: String

Pattern: [a-zA-Z0-9-]+

Required: No
NextToken (p. 313)

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. 313)

Sorts the list of results. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime
Required: No

SortOrder (p. 313)

The sort order for results. The default is Ascending.

Type: String
Valid Values: Ascending | Descending
Required: No

Response Syntax

```
{
    "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. 314)

An array of ModelSummary objects, each of which lists a model.

Type: Array of ModelSummary (p. 377) objects

NextToken (p. 314)

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. 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
ListNotebookInstanceLifecycleConfigs
Service: Amazon SageMaker Service

Lists notebook instance lifestyle configurations created with the CreateNotebookInstanceLifecycleConfig (p. 260) 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. 402).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 316)

A filter that returns only lifecycle configurations that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 316)

A filter that returns only lifecycle configurations that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 316)

A filter that returns only lifecycle configurations that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 316)

A filter that returns only lifecycle configurations that were modified before the specified time (timestamp).

Type: Timestamp

Required: No
MaxResults (p. 316)

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. 316)

A string in the lifecycle configuration name. This filter returns only lifecycle configurations whose name contains the specified string.

Type: String

Pattern: [a-zA-Z0-9-]+

Required: No

NextToken (p. 316)

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. 316)

Sorts the list of results. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | LastModifiedTime

Required: No

SortOrder (p. 316)

The sort order for results.

Type: String

Valid Values: Ascending | Descending

Required: No

Response Syntax

```
{
   "NextToken": "string",
   "NotebookInstanceLifecycleConfigs": [
   {
      "CreationTime": number,
      "LastModifiedTime": number,
      "NotebookInstanceLifecycleConfigArn": "string",
      "NotebookInstanceLifecycleConfigName": "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. 317)**

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. 317)**

An array of `NotebookInstanceLifecycleConfiguration` objects, each listing a lifecycle configuration.

Type: Array of `NotebookInstanceLifecycleConfigSummary` (p. 378) objects

Errors

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

```json
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "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. 402).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 319)**

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 319)**

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 319)**

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 319)**

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp

Required: No
MaxResults (p. 319)

The maximum number of notebook instances to return.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

NameContains (p. 319)

A string in the notebook instances' name. This filter returns only notebook instances whose name contains the specified string.

Type: String

Pattern: [a-zA-Z0-9-]+

Required: No

NextToken (p. 319)

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. 319)

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. 319)

The field to sort results by. The default is Name.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

SortOrder (p. 319)

The sort order for results.

Type: String
Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 319)**

A filter that returns only notebook instances with the specified status.

Type: String

Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting

Required: No

**Response Syntax**

```
{
   "NextToken": "string",
   "NotebookInstances": [
      {
         "CreationTime": number,
         "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. 321)**

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. 321)**

An array of NotebookInstanceSummary objects, one for each notebook instance.

Type: Array of NotebookInstanceSummary (p. 380) objects

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 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
ListTags
Service: Amazon SageMaker Service

Returns the tags for the specified Amazon SageMaker resource.

Request Syntax

```
{
  "MaxResults": number,
  "NextToken": "string",
  "ResourceArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

MaxResults (p. 323)

- Maximum number of tags to return.
- Type: Integer
- Valid Range: Minimum value of 50.
- Required: No

NextToken (p. 323)

- 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. 323)

- 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

```
{
  "NextToken": "string",
  "Tags": [
    {
      "Key": "string",
      "Value": "string"
    }
  ]
}
```

323
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. 323)

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. 323)

An array of Tag objects, each with a tag key and a value.

Type: Array of Tag (p. 395) 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. 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
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. 402).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 325)

A filter that only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 325)

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 325)

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 325)

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 325)

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. 325)**

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. 325)**

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. 325)**

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 325)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 325)**

A filter that retrieves only training jobs with a specific status.

Type: String

Valid Values: InProgress | Completed | Failed | Stopping | Stopped

Required: No

**Response Syntax**

```
{
   "NextToken": "string",
   "TrainingJobSummaries": [
   
   
```

326
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. 326)

- 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. 326)

- An array of TrainingJobSummary objects, each listing a training job.

  Type: Array of TrainingJobSummary (p. 398) objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 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
ListTrainingJobsForHyperParameterTuningJob
Service: Amazon SageMaker Service

Gets a list of TrainingJobSummary (p. 398) objects that describe the training jobs that a hyperparameter tuning job launched.

Request Syntax

```json
{
  "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. 402).

The request accepts the following data in JSON format.

**HyperParameterTuningJobName (p. 328)**

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. 328)**

The maximum number of training jobs to return.

Type: Integer

Valid Range: Minimum value of 1. Maximum value of 100.

Required: No

**NextToken (p. 328)**

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. 328)**

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. 328)**
The sort order for results. The default is Ascending.
Type: String
Valid Values: Ascending | Descending
Required: No

**StatusEquals (p. 328)**
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"
      }
    }
  ]
}
```

**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. 329)**
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. 329)*

A list of *TrainingJobSummary (p. 398)* objects that describe the training jobs that the *ListTrainingJobsForHyperParameterTuningJob* request returned.

Type: Array of *HyperParameterTrainingJobSummary (p. 368)* objects

**Errors**

For information about the errors that are common to all actions, see *Common Errors (p. 400).*

**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
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

```json
{
   "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 331)**

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. 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
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

```json
{
   "HyperParameterTuningJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**HyperParameterTuningJobName (p. 333)**

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. 400).

**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

```
{
  "NotebookInstanceName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 335)**

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. 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
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

```json
{
  "TrainingJobName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**TrainingJobName (p. 337)**

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. 400).

**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
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.

Request Syntax

```json
{
    "EndpointConfigName": "string",
    "EndpointName": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 339)**

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. 339)**

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. 339)**

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. 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
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. 402).

The request accepts the following data in JSON format.

**DesiredWeightsAndCapacities (p. 341)**

An object that provides new capacity and weight values for a variant.

Type: Array of DesiredWeightAndCapacity (p. 360) objects

Array Members: Minimum number of 1 item.

Required: Yes

**EndpointName (p. 341)**

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. 341)**

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. 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
**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
{
    "InstanceType": "string",
    "NotebookInstanceName": "string",
    "RoleArn": "string"
}
```

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 402).

The request accepts the following data in JSON format.

**InstanceType (p. 343)**

The Amazon ML compute instance type.

- Type: String
- 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.p2.xlarge | ml.p2.8xlarge | ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge
- Required: No

**NotebookInstanceName (p. 343)**

The name of the notebook instance to update.

- Type: String
- Length Constraints: Maximum length of 63.
- Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*$  
- Required: Yes

**RoleArn (p. 343)**

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-z-]*:iam::d(12):role/?[a-zA-Z0-9+,.,@\-_]+$
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. 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
UpdateNotebookInstanceLifecycleConfig

Service: Amazon SageMaker Service

Updates a notebook instance lifecycle configuration created with the CreateNotebookInstanceLifecycleConfig (p. 260) 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 (p. 402).

The request accepts the following data in JSON format.

**NotebookInstanceLifecycleConfigName (p. 345)**

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. 345)**

The shell script that runs only once, when you create a notebook instance

Type: Array of NotebookInstanceLifecycleHook (p. 379) objects

Array Members: Maximum number of 1 item.

Required: No

**OnStart (p. 345)**

The shell script that runs every time you start a notebook instance, including when you create the notebook instance.

Type: Array of NotebookInstanceLifecycleHook (p. 379) 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. 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

Amazon SageMaker Runtime

The following actions are supported by Amazon SageMaker Runtime:

- InvokeEndpoint (p. 347)
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.

Request Syntax

```
POST /endpoints/EndpointName/invocations HTTP/1.1
Content-Type: ContentType
Accept: Accept

Body
```

URI Request Parameters

The request requires the following URI parameters.

Accept (p. 347)

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

ContentType (p. 347)

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

EndpointName (p. 347)

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. 347)

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

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. 348)
The MIME type of the inference returned in the response body.
Length Constraints: Maximum length of 1024.

InvokedProductionVariant (p. 348)
Identifies the production variant that was invoked.
Length Constraints: Maximum length of 1024.

The response returns the following as the HTTP body.

Body (p. 348)
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. 400).

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.
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. 352)
- CategoricalParameterRange (p. 353)
- Channel (p. 354)
- ContainerDefinition (p. 356)
- ContinuousParameterRange (p. 358)
- DataSource (p. 359)
- DesiredWeightAndCapacity (p. 360)
- EndpointConfigSummary (p. 361)
- EndpointSummary (p. 362)
- FinalHyperParameterTuningJobObjectiveMetric (p. 364)
- HyperParameterAlgorithmSpecification (p. 365)
- HyperParameterTrainingJobDefinition (p. 366)
- HyperParameterTrainingJobSummary (p. 368)
- HyperParameterTuningJobConfig (p. 370)
- HyperParameterTuningJobObjective (p. 371)
- HyperParameterTuningJobSummary (p. 372)
- IntegerParameterRange (p. 374)
- MetricDefinition (p. 375)
- ModelArtifacts (p. 376)
- ModelSummary (p. 377)
- NotebookInstanceLifecycleConfigSummary (p. 378)
- NotebookInstanceLifecycleHook (p. 379)
- NotebookInstanceSummary (p. 380)
- ObjectiveStatusCounters (p. 382)
- OutputDataConfig (p. 383)
- ParameterRanges (p. 384)
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. 352)
- CategoricalParameterRange (p. 353)
- Channel (p. 354)
- ContainerDefinition (p. 356)
- ContinuousParameterRange (p. 358)
- DataSource (p. 359)
- DesiredWeightAndCapacity (p. 360)
- EndpointConfigSummary (p. 361)
- EndpointSummary (p. 362)
- FinalHyperParameterTuningJobObjectiveMetric (p. 364)
- HyperParameterAlgorithmSpecification (p. 365)
- HyperParameterTrainingJobDefinition (p. 366)
- HyperParameterTrainingJobSummary (p. 368)
- HyperParameterTuningJobConfig (p. 370)
- HyperParameterTuningJobObjective (p. 371)
- HyperParameterTuningJobSummary (p. 372)
- IntegerParameterRange (p. 374)
- MetricDefinition (p. 375)
- ModelArtifacts (p. 376)
- ModelSummary (p. 377)
- NotebookInstanceLifecycleConfigSummary (p. 378)
- NotebookInstanceLifecycleHook (p. 379)
- NotebookInstanceSummary (p. 380)
- ObjectiveStatusCounters (p. 382)
- OutputDataConfig (p. 383)
- ParameterRanges (p. 384)
- ProductionVariant (p. 385)
- ProductionVariantSummary (p. 387)
- ResourceConfig (p. 389)
- ResourceLimits (p. 391)
- S3DataSource (p. 392)
- StoppingCondition (p. 394)
- Tag (p. 395)
- TrainingJobStatusCounters (p. 396)
- TrainingJobSummary (p. 398)
- VpcConfig (p. 400)
• ResourceLimits (p. 391)
• S3DataSource (p. 392)
• StoppingCondition (p. 394)
• Tag (p. 395)
• TrainingJobStatusCounters (p. 396)
• TrainingJobSummary (p. 398)
• VpcConfig (p. 400)
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 (p. 146).

Contents

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 (p. 52).

Type: String

Length Constraints: Maximum length of 255.

Required: Yes

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
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
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. 359) object
Required: Yes

RecordWrapperType

Specify RecordIO as the value when input data is in raw format but the training algorithm requires the RecordIO format, in which 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
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**

The DNS host name for the container after Amazon SageMaker deploys it.

Type: String

Length Constraints: Maximum length of 63.

Pattern: \^[a-zA-Z0-9](-*[a-zA-Z0-9])*\n
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. For more information, see Using Your Own Algorithms with Amazon SageMaker

Type: String

Length Constraints: Maximum length of 255.

Pattern: [\S]+

Required: Yes

**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).

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
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
**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. 392) 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
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

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.
Type: String
Valid Values: OutOfService | Creating | Updating | 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
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. 370).

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
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

MetricDefinitions

An array of MetricDefinition (p. 375) objects that specify the metrics that the algorithm emits.

Type: Array of MetricDefinition (p. 375) 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 (p. 52).

Type: String

Length Constraints: Maximum length of 255.

Required: Yes

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
HyperParameterTrainingJobDefinition

Service: Amazon SageMaker Service

Defines the training jobs launched by a hyperparameter tuning job.

Contents

AlgorithmSpecification

The HyperParameterAlgorithmSpecification (p. 365) object that specifies the algorithm to use for the training jobs that the tuning job launches.

Type: HyperParameterAlgorithmSpecification (p. 365) object

Required: Yes

InputDataConfig

An array of Channel (p. 354) objects that specify the input for the training jobs that the tuning job launches.

Type: Array of Channel (p. 354) objects

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

Required: Yes

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. 383) 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. 389) 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-Z0-9+=,.@\-_\/]++$  

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. 394) object

Required: Yes

VpcConfig

The VpcConfig (p. 400) 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 (p. 232).

Type: VpcConfig (p. 400) 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. 364) object that specifies the value of the objective metric of the tuning job that launched this training job.

Type: FinalHyperParameterTuningJobObjectiveMetric (p. 364) 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

The date and time that the training job ended.

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]*\[a-zA-Z0-9]*)*

Required: 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

**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. 371) object that specifies the objective metric for this tuning job.

Type: HyperParameterTuningJobObjective (p. 371) object

Required: Yes

ParameterRanges

The ParameterRanges (p. 384) object that specifies the ranges of hyperparameters that this tuning job searches.

Type: ParameterRanges (p. 384) object

Required: Yes

ResourceLimits

The ResourceLimits (p. 391) object that specifies the maximum number of training jobs and parallel training jobs for this tuning job.

Type: ResourceLimits (p. 391) object

Required: Yes

Strategy

Specifies the search strategy for hyperparameters. Currently, the only valid value is Bayesian.

Type: String

Valid Values: Bayesian

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
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-z\-]*:sagemaker:[a-z0-9\-]*:\d{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. 382) object that specifies the numbers of training jobs, categorized by objective metric status, that this tuning job launched.

Type: ObjectiveStatusCounters (p. 382) object

Required: Yes

ResourceLimits

The ResourceLimits (p. 391) object that specifies the maximum number of training jobs and parallel training jobs allowed for this tuning job.

Type: ResourceLimits (p. 391) 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. 396) object that specifies the numbers of training jobs, categorized by status, that this tuning job launched.

Type: TrainingJobStatusCounters (p. 396) 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
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
MetricDefinition
Service: Amazon SageMaker Service

Specifies a metric that the training algorithm writes to stderr or stdout. Amazon SageMakerHyperparameter 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 (p. 35).

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
**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
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 (p. 18).

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**

**CreationTime**

A timestamp that shows when the notebook instance was created.

Type: Timestamp

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.m4.xlarge` | `ml.m4.2xlarge` | `ml.m4.4xlarge` | `ml.m4.10xlarge` |
| `ml.m4.16xlarge` | `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  (p. 18).

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

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
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.

Note
If you don’t provide the 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 Amazon Simple Storage Service developer guide.

Note
The KMS key policy must grant permission to the IAM role you specify in your
CreateTrainingJob request. 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
ParameterRanges
Service: Amazon SageMaker Service

Specifies ranges of integer, continuous, and categorical hyperparameters that a hyperparameter tuning job searches.

Contents

CategoricalParameterRanges

The array of CategoricalParameterRange (p. 353) objects that specify ranges of categorical hyperparameters that a hyperparameter tuning job searches.

Type: Array of CategoricalParameterRange (p. 353) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

ContinuousParameterRanges

The array of ContinuousParameterRange (p. 358) objects that specify ranges of continuous hyperparameters that a hyperparameter tuning job searches.

Type: Array of ContinuousParameterRange (p. 358) objects

Array Members: Minimum number of 0 items. Maximum number of 20 items.

Required: No

IntegerParameterRanges

The array of IntegerParameterRange (p. 374) objects that specify ranges of integer hyperparameters that a hyperparameter tuning job searches.

Type: Array of IntegerParameterRange (p. 374) 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
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

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.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

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

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
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.p2.4xlarge | ml.p2.8xlarge | ml.p3.16xlarge |
| ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge |
ml.c5.18xlarge

Required: Yes

VolumeKmsKeyId
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(s) that run the training job.

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

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 \( \frac{1}{n} \) 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 \( \frac{1}{n} \) 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 with 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.

Type: String

Valid Values: ManifestFile | S3Prefix

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-1
```

The complete set of `s3uris` in this manifest constitutes the input data for the channel for this datasource. The object that each `s3uris` points to must 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
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 5 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
Tag
Service: Amazon SageMaker Service

Describes a tag.

Contents

Key

The tag key.

Type: String


Pattern: ^((?!aws:)[\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
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 a 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-z\-]*:sagemaker:[a-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])*
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
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 Models by Using an Amazon Virtual Private Cloud (p. 234) and Protect Training Jobs by Using an Amazon Virtual Private Cloud (p. 232).

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

Amazon SageMaker Runtime

The following data types are supported by Amazon SageMaker Runtime:

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.

**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
The following table describes the documentation for this release of Amazon SageMaker.

- **Latest documentation update:** November 29, 2017

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>New guide</td>
<td>This is the first release of the <em>Amazon SageMaker Developer Guide.</em></td>
<td>November 29, 2017</td>
</tr>
<tr>
<td>DeepAR algorithm</td>
<td>Amazon SageMaker now supports the [DeepAR Forecasting](p. 125) algorithm.</td>
<td>January 8, 2018</td>
</tr>
<tr>
<td>CloudTrail support</td>
<td>Amazon SageMaker now supports loggin with [AWS CloudTrail](p. 227).</td>
<td>January 11, 2018</td>
</tr>
<tr>
<td>KMS encryption support for training and hosting</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 [CreateEndpointConfig](p. 246). 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 [OutputDataConfig](p. 383) object you use to configure your training job.</td>
<td>January 17, 2018</td>
</tr>
<tr>
<td>BlazingText algorithm</td>
<td>Amazon SageMaker now supports the [BlazingText](p. 134) algorithm.</td>
<td>January 18, 2018</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>TensorFlow 1.5 and MXNet 1.0</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>support</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Auto Scaling</td>
<td>Amazon SageMaker now supports Application Auto Scaling for production variants. (p. 154)</td>
<td>February 28, 2018</td>
</tr>
<tr>
<td>support</td>
<td></td>
<td></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 (p. 229).</td>
<td>March 15, 2018</td>
</tr>
<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 Step 2.1: (Optional) Customize a Notebook Instance (p. 18).</td>
<td>March 15, 2017</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.