Amazon SageMaker: Developer Guide
Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon’s trademarks and trade dress may not be used in connection with any product or service that is not Amazon’s, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.
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
  Model Training ............................................................................. 4
    How It Works: Next Topic ............................................................ 5
  Model Deployment ....................................................................... 5
    Considerations for Deploying Models on Amazon SageMaker Hosting Services .......................... 7
      How It Works: Next Topic ............................................................ 7
Notebook Instances and Notebooks ................................................. 8
  Creating a Notebook Instance ...................................................... 8
  Accessing Notebook Instances .................................................... 8
    How It Works: Next Topic ............................................................ 9
Validating Models ......................................................................... 9
    How It Works: Next Topic ............................................................ 10
Programming Model .................................................................. 11
    How It Works: Next Topic ............................................................ 12
Getting Started ........................................................................... 13
  Step 1: Setting Up ................................................................. 13
    Step 1.1: Create an Account .................................................. 13
    Step 1.2: Create an S3 Bucket ............................................... 14
  Step 2: Create an Amazon SageMaker Notebook Instance .......... 15
    Next Step ................................................................. 19
  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
Using Built-in Algorithms .............................................................. 34
  Common Information ............................................................... 36
    Common Parameters ............................................................. 36
    Common Data Formats .......................................................... 38
    Suggested Instance Types ...................................................... 45
    Logs ................................................................................ 45
  Linear Learner ................................................................. 46
    Input/Output Interface .......................................................... 47
    EC2 Instance Recommendation .......................................... 47
    How It Works ................................................................. 47
    Hyperparameters ............................................................... 48
    Inference Formats ............................................................... 52
  Factorization Machines .................................................. 53
    Input/Output Interface .......................................................... 53
    EC2 Instance Recommendation .......................................... 54
    How It Works ................................................................. 54
    Hyperparameters ............................................................... 55
    Inference Formats ............................................................... 58
  XGBoost Algorithm .................................................. 59
    Input/Output Interface .......................................................... 60
    EC2 Instance Recommendation .......................................... 60
    How It Works ................................................................. 60
Table of Contents

TensorFlow Model Training Code ................................................................. 112
Examples: Using Amazon SageMaker with TensorFlow ........................................... 115

Example 1: Using the tf.estimator ................................................................. 116

Using Apache MXNet ......................................................................................... 122
Apache MXNet Model Training Code ................................................................. 122
Examples: Using Amazon SageMaker with Apache MXNet ...................................... 128
Example 1: Using the Module API ................................................................. 128

Using Apache Spark ......................................................................................... 135

Downloading the Amazon SageMaker Spark Library ................................................ 135
Integrating Your Apache Spark Application with Amazon SageMaker ...................... 136
Example 1: Amazon SageMaker with Apache Spark ............................................... 137

Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark ................................................................. 141

Using the SageMakerEstimator in a Spark Pipeline ............................................. 142

Additional Examples: Amazon SageMaker with Apache Spark ................................ 143

Amazon SageMaker Libraries ............................................................................. 144

Authentication and Access Control ...................................................................... 145

Authentication ................................................................................................. 145
Access Control ................................................................................................. 146

Overview of Managing Access ........................................................................... 146

Amazon SageMaker Resources and Operations ..................................................... 147
Understanding Resource Ownership .................................................................... 147
Managing Access to Resources ......................................................................... 147
Specifying Policy Elements: Resources, Actions, Effects, and Principals ............... 149
Specifying Conditions in a Policy ..................................................................... 149

Using Identity-Based Policies (IAM Policies) ....................................................... 150

Permissions Required to Use the Amazon SageMaker Console ............................... 151

AWS Managed (Predefined) Policies for Amazon SageMaker ............................... 151

Amazon SageMaker API Permissions Reference ................................................ 152

Amazon SageMaker Roles .................................................................................. 155

CreateNotebookInstance API: Execution Role Permissions ................................. 156
CreateTrainingJob API: Execution Role Permissions ............................................. 158
CreateModel API: Execution Role Permissions ..................................................... 160

Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role ................................................................. 161

Monitoring ........................................................................................................ 163

Monitoring with CloudWatch ............................................................................. 163
Logging with CloudWatch .................................................................................. 166
Logging Amazon SageMaker API Calls with AWS CloudTrail ............................... 166
Amazon SageMaker Information in CloudTrail .................................................... 167
Understanding Amazon SageMaker Log File Entries .......................................... 167

Guidelines and Limits ....................................................................................... 169

Notebook Instance Security ............................................................................... 169

Notebook Instances Are Enabled with Internet Access .......................................... 169
Notebook Instances Provide the Best Experience for a Single User .......................... 169

Save Updated Sample Notebooks in a New Location .......................................... 170

Supported Versions ......................................................................................... 170

API Reference .................................................................................................. 171

Actions ............................................................................................................. 171

Amazon SageMaker Service ............................................................................... 172
Amazon SageMaker Runtime .............................................................................. 245

Data Types ........................................................................................................ 248

Amazon SageMaker Service ............................................................................... 248
Amazon SageMaker Runtime .............................................................................. 276

Common Errors ................................................................................................ 276

Common Parameters ......................................................................................... 277

Document History ............................................................................................. 280

AWS Glossary ................................................................................................. 281
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. 13)** – 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. 116) and Apache MXNet Example 1: Using the Module API (p. 128).
   - **Use Amazon SageMaker directly from Apache Spark** – For information, see Using Apache Spark with Amazon SageMaker (p. 135).
   - **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. 104) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.

5. **See the API Reference (p. 171)** – 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)
- Training a Model with Amazon SageMaker (p. 4)
- Deploying a Model on Amazon SageMaker Hosting Services (p. 5)
- Using an Amazon SageMaker Notebook Instance to Explore and Preprocess Data (p. 8)
- Validating Machine Learning Models (p. 9)
- The Amazon SageMaker Programming Model (p. 11)

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 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(s) 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**—You might perform additional data transformations to improve performance. 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 them 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 [Notebook Instances and Notebooks](p. 8). For more information about data preparation 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. 34).

   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**—Traditionally, you do some re-engineering of a model to integrate it with your application, before deploying the model into production. 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. 5).

   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 over time.
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.
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. 34). To try an exercise that uses an algorithm provided by Amazon SageMaker, see *Getting Started* (p. 13).

- **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. 135).

- **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. 112) and *Using Apache MXNet with Amazon SageMaker* (p. 122).

- **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. 104).

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. 190) API.

---

**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. 181) 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 desired number of ML compute instances 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. 178) 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. 175) 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. 246) 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 Serialization (p. 39).

• 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. 265).

• 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. 240) and UpdateEndpointWeightsAndCapacities (p. 242).

• 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

Using an Amazon SageMaker Notebook Instance to Explore and Preprocess Data (p. 8)
Using an Amazon SageMaker Notebook Instance to 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. 23).

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.

Creating a Notebook Instance

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.

To create a notebook instance, use either the Amazon SageMaker console or the CreateNotebookInstance (p. 184) API. In your request, provide a subnet and one or more security groups.

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. 169).

- **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, enables traffic between your VPC and the notebook instance.

- **Installs Anaconda packages and libraries for common deep learning platforms**—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**—Attaches a 5 GB ML storage 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.

- **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 a browser and sign in to the Amazon SageMaker console. 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 (p. 188) 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 Jupyter notebook dashboard to create and manage notebooks and write code. For more information about Jupyter notebooks, see http://jupyter.org/documentation.html. Note the following in the dashboard:

- Under the Files tab, there is a sample_notebook folder. It contains a set of example notebooks that Amazon SageMaker provides. For more information about these examples, see GitHub repository.
- Customers can select a kernel for their notebook using the New drop-down. Amazon SageMaker provides several kernels for Jupyter including support for Python 2 and 3, MXNet, TensorFlow, and PySpark.
- Use the API—to get the URL for the notebook instance, call the CreatePresignedNotebookInstanceUrl (p. 188) API.

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 a 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 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.

There are several articles and books about model evaluation. 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, and assess how effectively the model generalizes what it learned in the initial training, as opposed to using model "memory." This approach to validation gives you an idea of how often the model is able to infer the correct answer.

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

  Typically, the holdout dataset is of 20-30% of the training data.

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

How It Works: Next Topic

The Amazon SageMaker Programming Model (p. 11)
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. 171).

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. 122) and Using TensorFlow with Amazon SageMaker (p. 112).

  - **The AWS SDK**—The SDKs provide methods that correspond to the Amazon SageMaker API (see Actions (p. 171)). Use the SDKs to programmatically start a model training job and host the model in Amazon SageMaker. SDK clients authenticate your requests by using your access keys, so you don’t need to write authentication code. They are available in multiple languages and platforms. For more information, see SDKs.

In *Getting Started (p. 13)*, 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. 13).* For more information about these libraries, see *Amazon SageMaker Libraries (p. 144).*

- **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. 135).
How It Works: Next Topic

Getting Started (p. 13)
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 Notebook Instances and Notebooks (p. 8).

Topics
• Step 1: Setting Up (p. 13)
• Step 2: Create an Amazon SageMaker Notebook Instance (p. 15)
• 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. 13)
• Step 1.2: Create an S3 Bucket (p. 14)

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. 13)
• Step 1.1.2: Create an IAM Administrator User and Sign In (p. 14)

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.
Note
This might be unavailable in your browser if you previously signed into the AWS Management Console. In that case, choose Sign in to a different account, and then choose Create a new AWS account.

2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a 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. 145).

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. 14)

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.
Note
Amazon SageMaker needs your permissions to access this bucket. You grant these permissions 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. 15)

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 Notebook Instances and Notebooks (p. 8).

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, and then choose Create notebook instance.
3. On the **Create notebook instance** page, provide the following information:

   - For **Notebook instance name**, type `ExampleNotebookInstance`.
   - For **Instance type**, choose `ml.t2.medium`.
   - For **IAM role**, create an IAM role.
     1. Choose **Create a new role**.
2. Choose additional S3 buckets that you want to use with Amazon SageMaker.

In the preceding section, you created a bucket with `sagemaker` in its name. This IAM role automatically has the S3 permissions to use that bucket through the `AmazonSageMakerFullAccess` policy that Amazon SageMaker attaches to the role. The bucket you created is sufficient for the model training exercise in Getting Started. However, as you explore Amazon SageMaker, you might access other S3 buckets from your notebook instance. Amazon SageMaker will need permissions to access those buckets.

You can also choose to access additional S3 buckets from your Amazon SageMaker notebook instance:

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

- 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.

- Restrict access by choosing **None**. To explicitly control access, 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. 161).
3. Choose Create role.

Amazon SageMaker creates an IAM role. The role name has this form: AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmmSS. For example, AmazonSageMaker-ExecutionRole-201711125T090800.

You can view this role in the IAM console. Observe the following policies attached to the role:

- The trust policy allows Amazon SageMaker to assume the role.
- The role has the AmazonSageMakerFullAccess AWS managed policy attached.

If you specified additional S3 bucket(s) when creating the IAM role, view the customer managed policy attached to the role. The name of the customer managed policy has this form: AmazonSageMaker-ExecutionPolicy-YYYYMMDDTHHmmSS.

For more information about creating your own IAM role, see Amazon SageMaker Roles (p. 155).

- Choosing a Virtual Private Cloud (VPC) is optional for this exercise.

  **Note**
  If you want to access resources in your VPC from the notebook instance, choose a VPC and a SubnetId. For Security Group, choose the default security group of the selected VPC. The inbound and outbound rules of the default security group are sufficient for the exercises in this guide.
  To connect to a resource in your VPC, the resource must resolve to a private IP address in your VPC. For example, to ensure that an Amazon Redshift DNS name resolves to a private IP address, ensure one of the following:
  - 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 information on how to set those parameters, see Managing Clusters in an Amazon Virtual Private Cloud (VPC)

- Specifying a KMS encryption key is optional for this exercise. Specify one if you want Amazon SageMaker to use the key to encrypt data in the ML storage volume attached to the notebook instance.

4. Choose Create notebook instance.

This launches a ML compute instance, in this case, a notebook instance, and attaches an ML storage volume to it. For more information, see the CreateNotebookInstance (p. 184) API.

5. It takes a few minutes to create the notebook instance. The notebook instance provides you with a preconfigured Jupyter notebook server and a set of Anaconda libraries. When the status of the notebook instance is InService, choose Open next to the notebook instance's name. The Juypter dashboard appears:
The following are available in the dashboard:

- In Files, there is a sample_notebook folder, which contains a number of sample notebooks. For more information about them, see GitHub repository.
- Amazon SageMaker provides several kernels for Jupyter, including ones that provide support for Python 2 and 3, MXNet, TensorFlow, and PySpark. To choose a kernel for your notebook, use the New menu.

For more information, see The Jupyter notebook.

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. 79).

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. 11).

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
  ```

**Topics**

- Step 3.1: Create a Jupyter Notebook and Initialize Variables (p. 20)
- Step 3.2: Download, Explore, and Transform the Training Data (p. 21)
- Step 3.3: Train a Model (p. 23)
- Step 3.4: Deploy the Model to Amazon SageMaker Hosting Services (p. 26)
- Step 3.5: Validate the Model (p. 28)

**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:
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. 13), 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. 22)
- Step 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 23)

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:
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. 22)

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.

%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 31th image from the training dataset.
Next Step

Step 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 23)

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` 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. 24)
- Step 3.3.2: Create a Training Job (p. 24)
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. 34).

Next Step

Step 3.3.2: Create a Training Job (p. 24)

Step 3.3.2: Create a Training Job

To train a model, Amazon SageMaker provides the CreateTrainingJob (p. 190) 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. 38). 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. 82).
- The input and output configuration—Provide the S3 bucket where training data is stored and where Amazon SageMaker saves the results of model training (the model artifacts).

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

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

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

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

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

```python
from sagemaker import KMeans

data_location = 's3://{}/kmeans_highlevel_example/data'.format(bucket)
output_location = 's3://{}/kmeans_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. 82).

• **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. 190) 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"
    }
}
```
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` (p. 181) 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` (p. 178) 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 (p. 175) API using this endpoint configuration.

3. Create an endpoint— Send a CreateEndpoint (p. 175) 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.

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

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

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

- **Use the SDK for Python.**

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

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

     ```
     %%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'])
     ```

27
2. Create an Amazon SageMaker endpoint configuration by specifying the ML compute instances that you want to deploy your model to.

```python
from time import gmtime, strftime

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.

```python
%%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. 246) 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(train_set[0][30:31])
   print(result)
   ```

   Example response:

   ```json
   [label {
     key: "closest_cluster"
     value {
       float32_tensor {
         values: 3.0
       }
     }
   }
   label {
     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
   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
def np2csv(arr):
    csv = io.BytesIO()
    numpy.savetxt(csv, arr, delimiter=',', fmt='%g')
    return csv.getvalue().decode().rstrip()

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:
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. 82).

- 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. 46).

- 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. 65).

- 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. 122) and Using TensorFlow with Amazon SageMaker (p. 112).

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.
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. 46) and the *XGBoost Algorithm* (p. 59). 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. 46) or the *XGBoost Algorithm* (p. 59) 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. 53) 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. 79).

"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. 85) 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. 65)—Use this algorithm to classify images. It uses example data with answers (referred to as supervised algorithm).

• Amazon SageMaker Sequence2Sequence (p. 71)—This supervised algorithm is commonly used for neural machine translation.

• Latent Dirichlet Allocation (LDA) (p. 88)—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. 92)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

Topics
• Algorithms Provided by Amazon SageMaker: Common Information (p. 36)
• Linear Learner (p. 46)
• Factorization Machines (p. 53)
• XGBoost Algorithm (p. 59)
• Image Classification Algorithm (p. 65)
• Amazon SageMaker Sequence2Sequence (p. 71)
• K-Means Algorithm (p. 79)
• Principal Component Analysis (PCA) (p. 85)
• Latent Dirichlet Allocation (LDA) (p. 88)
• Neural Topic Model (NTM) (p. 92)
• DeepAR Forecasting (p. 95)
• BlazingText (p. 101)

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. 36)
• Algorithms Provided by Amazon SageMaker: Common Data Formats (p. 38)
• Algorithms Provided by Amazon SageMaker: Suggested Instance Types (p. 45)
• Algorithms Provided by Amazon SageMaker: Logs (p. 45)

Algorithms Provided by Amazon SageMaker: Common Parameters

The following table lists parameters for each of the algorithms provided by Amazon SageMaker. The Registry paths use <latest>, for example: /kmeans:1.

<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>&lt;ecr_path&gt;/kmeans:latest</td>
<td>File</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU</td>
</tr>
<tr>
<td>PCA</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/pca:latest</td>
<td>File</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>LDA</td>
<td>train and (optionally) test</td>
<td>&lt;ecr_path&gt;/lda:latest</td>
<td>File</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>&lt;ecr_path&gt;/factorization-machines:latest</td>
<td>File</td>
<td>recordIO-protobuf</td>
<td>CPU (GPU for dense data)</td>
</tr>
</tbody>
</table>
## Common Parameters

<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>Linear Learner</td>
<td>train and (optionally) validation, test, or both</td>
<td><code>&lt;ecr_path&gt;/linear-learner:latest</code></td>
<td>File</td>
<td>recordIO-protobuf or CSV</td>
<td>CPU or GPU</td>
</tr>
<tr>
<td>Neural Topic Model</td>
<td>train and (optionally) validation, test, or both</td>
<td><code>&lt;ecr_path&gt;/ntm:latest</code></td>
<td>File</td>
<td>recordIO-protobuf or CSV</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>Seq2Seq Modeling</td>
<td>train, validation, and vocab</td>
<td><code>&lt;ecr_path&gt;/seq2seq:latest</code></td>
<td>File</td>
<td>recordIO-protobuf</td>
<td>GPU (single instance only)</td>
</tr>
<tr>
<td>XGBoost</td>
<td>train and (optionally) validation</td>
<td><code>&lt;ecr_path&gt;/xgboost:latest</code></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><code>&lt;ecr_path&gt;/image-classification:latest</code></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><code>&lt;ecr_path&gt;/forecasting-deepar:latest</code></td>
<td>File</td>
<td>JSON Lines or Parquet</td>
<td>GPU or CPU</td>
</tr>
<tr>
<td>BlazingText</td>
<td>train</td>
<td><code>&lt;ecr_path&gt;/blazingtext:latest</code></td>
<td>File</td>
<td>Text file (one sentence per line with 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, depending on algorithm and region use one of the following values for `<ecr_path>`.

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>AWS Region</th>
<th>Training Image and Inference Image Registry Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-means, PCA, Factorization Machines, Linear Learner, and Neural Topic Model</td>
<td>us-west-2</td>
<td>174872318107.dkr.ecr.us-west-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-1</td>
<td>382416733822.dkr.ecr.us-east-1.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>us-east-2</td>
<td>404615174143.dkr.ecr.us-east-2.amazonaws.com</td>
</tr>
<tr>
<td></td>
<td>eu-west-1</td>
<td>438346466558.dkr.ecr.eu-west-1.amazonaws.com</td>
</tr>
</tbody>
</table>
## 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. 38)
- **Common Data Formats—Inference** (p. 41)

### 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 Serialization**

Amazon SageMaker algorithms work best with the optimized protobuf recordIO format. In the following code, each observation in the dataset is converted into a binary representation as a set of 4-byte floats and is then loaded to the protobuf “values” field.

**Note**  
Step 3.2.3: Transform the Training Dataset and Upload It to S3 (p. 23) shows how to convert a numPy array into the protobuf recordIO format.

The schema for the protocol buffers is:

```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 not empty then 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];

    // Optional shape which will allow the vector to represent a matrix.
    // e.g. if shape = [ 10, 20 ] then floor(keys[i] / 10) will give the row
    // and keys[i] % 20 will give the column.
    // This also supports n-dimensional tensors.
    // NB. this must be specified if the tensor is sparse.
    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 not empty then 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];
}
```

39
Amazon SageMaker Developer Guide
Common Data Formats

// Optional shape which will allow the vector to represent a matrix.
// e.g. if shape = [10, 20] then floor(keys[i] / 10) will give the row
// and keys[i] % 20 will give the column.
// This also supports n-dimensional tensors.
// NB. this must be specified if the tensor is sparse.
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 not empty then 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];

  // Optional shape which will allow the vector to represent a matrix.
  // e.g. if shape = [10, 20] then floor(keys[i] / 10) will give the row
  // and keys[i] % 20 will give the column.
  // This also supports n-dimensional tensors.
  // NB. this must be specified if the tensor is sparse.
  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;

  // Stores the content type of the data if known.
  // This will allow 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;

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

  // Unique identifier for this record in the dataset.
  //
  // Whilst not necessary, this allows better
  // debugging where there are data issues.
After creating the protocol buffer is created, store it in an S3 location that is accessible by Amazon SageMaker, and 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 EASE 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-protobuf.
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, 14.0, 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:

```json
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:

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

Multi-record inference:

```json
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:

```json
{
    "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

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

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

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. 87)
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

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

Failed validating u'oneOf' in
schema[u'properties'][u'feature_dim']:

{{u'pattern': u'^([1-9]\[0-9]*$)', u'type': u'string'},
{u'minimum': 1, u'type': u'integer'}}
```

3. Inaccurate protobuf file format.

From the CloudWatch log:

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

**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. 190). 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 (target variable for prediction).

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 training 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.</td>
</tr>
<tr>
<td></td>
<td>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 mini batch.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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_selection_criteria</td>
<td>Pick the model with best criteria from the validation dataset for predictor_type binary_classifier:</td>
</tr>
<tr>
<td></td>
<td>• accuracy: model with highest accuracy.</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
| • \( f_1 \): model with highest \( f_1 \) score.  
- \( \text{precision}_\text{at}_\text{target}_\text{recall} \): model with highest precision at a given recall target.  
- \( \text{recall}_\text{at}_\text{target}_\text{precision} \): model with highest recall at a given precision target.  
- \( \text{cross}_\text{entropy}_\text{loss} \): model with lowest cross entropy loss. | Valid values: \( \text{accuracy}, f_1, \text{precision}_\text{at}_\text{target}_\text{recall}, \text{recall}_\text{at}_\text{target}_\text{precision}, \text{cross}_\text{entropy}_\text{loss} \)  
Default value: \( \text{accuracy} \)
| target\_recall | Target recall. Applicable only if \( \text{binary}_\text{classifier}_\text{model}_\text{selection}_\text{criteria} \) is \( \text{precision}_\text{at}_\text{target}_\text{recall} \).  
Valid values: Number between 0 and 1.0  
Default value: 0.8
| target\_precision | Target precision. Only applicable if \( \text{binary}_\text{classifier}_\text{model}_\text{selection}_\text{criteria} \) is \( \text{recall}_\text{at}_\text{target}_\text{precision} \).  
Valid values: Number between 0 and 1.0  
Default value: 0.8
| num\_models | Description: Number of models to train in parallel. For the default \( \text{auto} \), 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 \( \text{auto} \)  
Default values: \( \text{auto} \)
| num\_calibration\_samples | Number of observations to use from the validation dataset for model calibration (finding the best threshold).  
Valid values: Positive integer or \( \text{auto} \)  
Default value: \( \text{auto} \)
| init\_method | Function to use to set the initial model weights.  
- \( \text{uniform} \): uniformly between \(-\text{scale}, +\text{scale}\)  
- \( \text{normal} \): normal, with mean 0 and sigma | Valid values: \( \text{uniform} \) or \( \text{normal} \)  
Default value: \( \text{uniform} \)
### Parameter Name | Description
---|---
`init_scale` | Scale. Applies only when `init_method` is set to *uniform*.  
Valid values: numbers -1.0 and 1.0  
Default value: 0.07

`init_sigma` | Standard deviation. Applies only when `init_method` is set to *normal*. Optional.  
Valid values: number between 0 and 1  
Default value: 0.01

`init_bias` | Initial weight for bias term.  
Valid values: number  
Default value: 0

`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*, or *absolute_loss*  
Default value: *auto*

`wd` | L2 regularization parameter. In other words, the weight decay parameter. Use 0 for no L2 regularization.  
Valid values: number between 0 and 1.0, *auto*  
Default value: *auto*

`l1` | L1 regularization parameter. Use 0 for no L1 regularization.  
Valid values: number between 0 and 1.0, *auto*  
Default value: *auto*

`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: number between 0 and 1.0, *auto*  
Default value: *auto*
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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  
                      Default value: 0.999 |
| bias_lr_mult   | 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 |
| bias_wd_mult   | 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 |
| use_lr_scheduler | If true, uses a scheduler for the learning rate.  
                      Valid values: `(true or false)`.  
                      Default value: `true` |
| lr_scheduler_step | The number of steps between decreases of the learning rate. Only applies to learning rate scheduler.  
                      Valid values: positive integer  
                      Default value: 100 |
| lr_scheduler_factor | 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 |
| lr_scheduler_minimum_lr | 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 |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normalize_data</td>
<td>Normalizes the features before training to have \text{std_dev} of 1. Optional.</td>
</tr>
<tr>
<td></td>
<td>Valid values: \text{true}, \text{false}, or \text{auto}</td>
</tr>
<tr>
<td></td>
<td>Default value: \text{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 \text{true} during classification, this parameter is ignored.</td>
</tr>
<tr>
<td></td>
<td>Valid values: \text{true}, \text{false}, or \text{auto}</td>
</tr>
<tr>
<td></td>
<td>Default value: \text{auto}</td>
</tr>
<tr>
<td>unbias_data</td>
<td>Unbiases the features before training so the mean is 0. By default data is unbiased if use_bias is set to \text{true}.</td>
</tr>
<tr>
<td></td>
<td>Valid values: \text{true}, \text{false}, or \text{auto}</td>
</tr>
<tr>
<td></td>
<td>Default value: \text{auto}</td>
</tr>
<tr>
<td>unbias_label</td>
<td>Unbiases labels before training so the mean is 0. Only done for regression if use_bias is set to \text{true}.</td>
</tr>
<tr>
<td></td>
<td>Valid values: \text{true}, \text{false}, or \text{auto}</td>
</tr>
<tr>
<td></td>
<td>Default value: \text{auto}</td>
</tr>
<tr>
<td>num_point_for_scaler</td>
<td>Number of data points to use for calculating normalization or unbiasing of terms.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10,000</td>
</tr>
</tbody>
</table>

**Linear Learner Response Formats**

**JSON**

**Binary classification**

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

**Regression**

```javascript
let response = {
  "predictions": [
    {
```
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 parsimoniously capture interactions between features within high dimensional sparse datasets. 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 machines algorithm can be run in either in binary classification mode or regression mode. In both modes, a dataset can be provided to the test channel along with the train channel dataset. In regression mode, the testing dataset is scored using RMSE (Root Mean Square Error). 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.

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

We recommend training and inference with CPU instances for both sparse and dense datasets. In some circumstances, training with 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. 54)
- Factorization Machines Hyperparameters (p. 55)
- Factorization Machine Response Formats (p. 58)

### How Factorization Machines Work

The factorization machine model with pairwise feature interactions can be written as follows:

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

For regression tasks, the model is trained by minimizing the squared error between model prediction and the target value (in other words, 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 log loss:

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

where:

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

The model has three components.

- Bias terms
- Linear terms
- Factorization (interaction) terms

Bias and linear terms are the same as in a linear model. Pairwise feature interactions are modeled 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 high. In other words, their embedding vectors would be close to each other in cosine similarity.

## Factorization Machines Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>feature_dim</td>
<td>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: -</td>
</tr>
<tr>
<td>num_factors</td>
<td>Dimensionality of factorization. Required. Valid values: Positive integer. Suggested value range: [2,1000]. Default value: -</td>
</tr>
<tr>
<td>predictor_type</td>
<td>Type of predictor. Required. Valid values: String: binary_classifier or regressor. Default value: -</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Size of mini-batch used for training. Valid values: positive integer. Default value: 1000</td>
</tr>
<tr>
<td>epochs</td>
<td>Number of training epochs to run. Valid values: positive integer. Default value: 1</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>Optimizer parameter. Clip the gradient by projecting onto the box [-clip_gradient, +clip_gradient]. Valid values: float. Default value: -</td>
</tr>
<tr>
<td>eps</td>
<td>Optimizer parameter. Small value to avoid division by 0. Valid values: float. Default value: -</td>
</tr>
<tr>
<td>rescale_grad</td>
<td>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: -</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>bias_lr</td>
<td>Learning rate for the bias term.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>linear_lr</td>
<td>Learning rate for linear terms.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>factors_lr</td>
<td>Learning rate for factorization terms.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0001</td>
</tr>
<tr>
<td>bias_wd</td>
<td>Weight decay for the bias term.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</td>
</tr>
<tr>
<td>linear_wd</td>
<td>Weight decay for linear terms.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.001</td>
</tr>
<tr>
<td>factors_wd</td>
<td>Weight decay for factorization terms.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.00001</td>
</tr>
<tr>
<td>bias_init_method</td>
<td>Initialization method for the bias term.</td>
</tr>
<tr>
<td></td>
<td>• normal: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by bias_init_sigma.</td>
</tr>
<tr>
<td></td>
<td>• uniform: Initializes weights with random values uniformly sampled from a range specified by [-bias_init_scale, +bias_init_scale].</td>
</tr>
<tr>
<td></td>
<td>• constant: Initializes the weights to a scalar value specified by bias_init_value.</td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform, normal, or constant</td>
</tr>
<tr>
<td></td>
<td>Default value: normal</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bias_init_scale</td>
<td>Range for initialization of the bias term. Takes effect if bias_init_method is set to uniform.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: -</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 normal.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</td>
</tr>
<tr>
<td>bias_init_value</td>
<td>Initial value of the bias term. Takes effect if bias_init_method is set to constant.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Suggested value range: [1e-8, 512]/</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>linear_init_method</td>
<td>Initialization method for linear terms.</td>
</tr>
<tr>
<td></td>
<td>• normal Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by linear_init_sigma.</td>
</tr>
<tr>
<td></td>
<td>• uniform Initializes weights with random values uniformly sampled from a range specified by [-linear_init_scale, +linear_init_scale].</td>
</tr>
<tr>
<td></td>
<td>• constant Initializes the weights to a scalar value specified by linear_init_value.</td>
</tr>
<tr>
<td></td>
<td>Valid values: uniform, normal, or constant.</td>
</tr>
<tr>
<td></td>
<td>Default value: normal</td>
</tr>
<tr>
<td>linear_init_scale</td>
<td>Range for initialization of linear terms. Takes effect if linear_init_method is set to uniform.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: -</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 normal.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.01</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 \([-\text{factors_init_scale}, +\text{factors_init_scale}]\).
- **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: -

---

**Factorization Machine Response Formats**

**JSON**

**Binary classification**

```javascript
let response = {
    "predictions": [
        {
            ...
        }
    ]
}```
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. In fact, XGBoost can be tweaked to resemble an algorithm
closer to random forests than gradient boosted trees. 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 *libsvm* or *csv*.
- For Inference ContentType, valid inputs are *text/x-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.

**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) or a memory-optimized instance (for example, R4) 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. 60)
- XGBoost Hyperparameters (p. 61)

**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>
</table>
| num_round      | The number of rounds to run the training. Required.  
|                | Valid values: integer  
|                | Default value: - |
| num_class      | Required if objective is set to multi:softmax or multi:softprob.  
|                | Valid values: integer  
|                | Default value: - |
| booster        | Which booster to use. The gbtree and dart values use a tree-based model, while gblinear uses a linear function.  
|                | Valid values: String. One of gbtree, gblinear, or dart.  
|                | Default value: gbtree |
| silent         | 0 means print running messages, 1 means silent mode.  
|                | Valid values: 0 or 1  
|                | Default value: 0 |
| nthread        | Number of parallel threads used to run xgboost.  
|                | Valid values: integer  
|                | Default value: Maximum number of threads. |
| eta            | 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].  
|                | Default value: 0.3 |
| 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,∞).  
|                | Default value: 0 |
## Parameter Name | Description
--- | ---
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 `grow_policy=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 `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 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

lambda | L2 regularization term on weights. Increasing this value makes models more conservative.

Valid values: float

Default value: 1
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>L1 regularization term on weights. Increasing this value makes models more conservative. Valid values: float. Default value: 1</td>
</tr>
<tr>
<td>tree_method</td>
<td>The tree construction algorithm used in XGBoost. Valid values: One of <em>auto</em>, <em>exact</em>, <em>approx</em>, or <em>hist</em>. Default value: <em>auto</em></td>
</tr>
<tr>
<td>sketch_eps</td>
<td>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</td>
</tr>
<tr>
<td>scale_pos_weight</td>
<td>Controls the balance of positive and negative weights. It’s useful for unbalanced classes. A typical value to consider: $\frac{\text{sum(negative cases)}}{\text{sum(positive cases)}}$. Valid values: float. Default value: 1</td>
</tr>
<tr>
<td>updater</td>
<td>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</td>
</tr>
<tr>
<td>refresh_leaf</td>
<td>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</td>
</tr>
<tr>
<td>process_type</td>
<td>The type of boosting process to run. Valid values: String. Either <em>default</em> or <em>update</em>. Default value: <em>default</em></td>
</tr>
</tbody>
</table>

63
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| **grow_policy** | Controls the way that new nodes are added to the tree. Currently supported only if `tree_method` is set to `hist`.  
  Valid values: String. Either `depthwise` or `lossguide`.  
  Default value: `depthwise` |
| **max_leaves** | Maximum number of nodes to be added. Relevant only if `grow_policy` is set to `lossguide`.  
  Valid values: integer  
  Default value: 0 |
| **max_bin** | Maximum number of discrete bins to bucket continuous features. Used only if `tree_method` is set to `hist`.  
  Valid values: integer  
  Default value: 256 |
| **sample_type** | Type of sampling algorithm.  
  Valid values: Either `uniform` or `weighted`.  
  Default value: `uniform` |
| **normalize_type** | Type of normalization algorithm.  
  Valid values: Either `tree` or `forest`.  
  Default value: `tree` |
| **rate_drop** | Dropout rate (a fraction of previous trees to drop during the dropout).  
  Valid values: Float. Range: [0.0, 1.0].  
  Default value: 0.0 |
| **one_drop** | When this flag is enabled, at least one tree is always dropped during the dropout.  
  Valid values: 0 or 1  
  Default value: 0 |
| **skip_drop** | Probability of skipping the dropout procedure during a boosting iteration.  
  Valid values: Float. Range: [0.0, 1.0].  
  Default value: 0.0 |
| **lambda_bias** | L2 regularization term on bias.  
  Valid values: Float. Range: [0.0, 1.0].  
  Default value: 0 |
### 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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tweedie_variance_power</strong></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><strong>objective</strong></td>
<td>Specifies the learning task and the corresponding learning objective.</td>
</tr>
<tr>
<td></td>
<td>Examples: <em>reg:linear</em>, <em>reg:logistic</em>, <em>multi:softmax</em>. 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: <em>reg:linear</em></td>
</tr>
<tr>
<td><strong>base_score</strong></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><strong>eval_metric</strong></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><strong>seed</strong></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>
</tbody>
</table>
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. 190) 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. 190) 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 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>2</td>
<td>your_image_directory/train_img_dog1.jpg</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>your_image_directory/train_img_cat1.jpg</td>
</tr>
<tr>
<td>22</td>
<td>2</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, and ml.p2.16xlarge. However, both CPU and GPU instances can be used for the inference. We recommend using GPU instances with more memory for training with large batch sizes. For hosting, C4 CPU instances and P2 GPU instances can be used. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training.
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. 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. 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. 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. Valid values: 0 or 1</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>checkpoint_frequency</td>
<td>Period to store model parameters (in number of epochs).</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>num_layers</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 CFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200].</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110].</td>
</tr>
<tr>
<td></td>
<td>Default value: 152</td>
</tr>
<tr>
<td>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></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: -</td>
</tr>
<tr>
<td>epochs</td>
<td>Number of training epochs.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 30</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Initial learning rate.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>lr_scheduler_factor</td>
<td>The ratio to reduce learning rate used in conjunction with the lr_scheduler_step parameter, defined as lr_new = lr_old * lr_scheduler_factor.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in [0, 1].</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.1</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lr_scheduler_step</td>
<td>The epochs at which to reduce the learning rate. As explained in the lr_scheduler_factor parameter, the learning rate is reduced by lr_scheduler_factor at these epochs. For example, if the value is set to &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>optimizer</td>
<td>The optimizer types. For more details of the parameters for the optimizers, please refer to MXNet's API.</td>
</tr>
<tr>
<td>momentum</td>
<td>The momentum for sgd and nag, ignored for other optimizers.</td>
</tr>
<tr>
<td>weight_decay</td>
<td>The coefficient weight decay for sgd and nag, ignored for other optimizers.</td>
</tr>
<tr>
<td>beta_1</td>
<td>The beta1 for adam, in other words, exponential decay rate for the first moment estimates.</td>
</tr>
<tr>
<td>beta_2</td>
<td>The beta2 for adam, in other words, exponential decay rate for the second moment estimates.</td>
</tr>
<tr>
<td>eps</td>
<td>The epsilon for adam and rmsprop. It is usually set to a small value to avoid division by 0.</td>
</tr>
<tr>
<td>gamma</td>
<td>The gamma for rmsprop. A decay factor of moving average of the squared gradient.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>The batch size for training. In a single-machine multi-GPU setting, each GPU handles mini_batch_size/num_gpu training samples. For the multi-machine training in dist_sync mode, the actual batch size is mini_batch_size*number of machines. See MXNet docs for more details. Valid values: positive integer Default value: 32</td>
</tr>
<tr>
<td>image_shape</td>
<td>The input image dimensions, which is the same size as the input layer of the network. The format is defined as 'num_channels, height, width'. The image dimension can take on any value as the network can handle varied dimensions of the input. However, there may be memory constraints if a larger image dimension is used. Typical image dimensions for image classification are '3, 224, 224'. This is similar to the ImageNet dataset. Valid values: string Default value: '3, 224, 224'</td>
</tr>
<tr>
<td>augmentation_type</td>
<td>Data augmentation type. The input images can be augmented in multiple ways as specified below.</td>
</tr>
<tr>
<td></td>
<td>• crop: Randomly crop the image and flip the image horizontally</td>
</tr>
<tr>
<td></td>
<td>• crop_color: In addition to ‘crop’, three random values in the range [-36, 36], [-50, 50], and [-50, 50] are added to the corresponding Hue-Saturation-Lightness channels respectively</td>
</tr>
<tr>
<td></td>
<td>• crop_color_transform: In addition to crop_color, random transformations, including rotation, shear, and aspect ratio variations are applied to the image. The maximum angle of rotation is 10 degrees, the maximum shear ratio is 0.1, and the maximum aspect changing ratio is 0.25.</td>
</tr>
<tr>
<td></td>
<td>Valid values: One of crop, crop_color, crop_color_transform. Default value: -</td>
</tr>
<tr>
<td>top_k</td>
<td>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: -</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
kv_store | 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

---

**Amazon SageMaker 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 modes support 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. 72)
- Sequence2Sequence Hyperparameters (p. 73)

**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 are 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 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 paper by Luong et al. that explains and simplifies calculations for various attention mechanisms. This paper by Google describes their 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 rnn architecture is based on attention mechanism by Bahdanau et al. and cnn architecture is based on Gehring et al.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td>decoder_type</td>
<td>Decoder type.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either rnn or cnn.</td>
</tr>
<tr>
<td></td>
<td>Default value: rnn</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>num_layers_encoder</td>
<td>Number of layers for Encoder <em>rnn</em> or <em>cnn</em>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>num_layers_decoder</td>
<td>Number of layers for Decoder <em>rnn</em> or <em>cnn</em>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>rnn_num_hidden</td>
<td>Number of <em>rnn</em> hidden units for encoder and decoder.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1024</td>
</tr>
<tr>
<td>rnn_cell_type</td>
<td>Specific type of <em>rnn</em> architecture.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <em>lstm</em> or <em>gru</em>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>lstm</em></td>
</tr>
<tr>
<td>rnn_decoder_state_init</td>
<td>How to initialize <em>rnn</em> decoder states from encoders.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <em>last</em>, <em>avg</em>, or <em>zero</em>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>last</em></td>
</tr>
<tr>
<td>rnn_residual_connections</td>
<td>Add residual connection to stacked <em>rnn</em>. Number of layers should be more</td>
</tr>
<tr>
<td></td>
<td>than 1.</td>
</tr>
<tr>
<td></td>
<td>Valid values: boolean (<em>true</em> or <em>false</em>)</td>
</tr>
<tr>
<td></td>
<td>Default value: <em>false</em></td>
</tr>
<tr>
<td>rnn_first_residual_layer</td>
<td>First <em>rnn</em> layer to have a residual connection, only applicable if number</td>
</tr>
<tr>
<td></td>
<td>of layers in encoder or decoder is more than 1.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 2</td>
</tr>
<tr>
<td>cnn_kernel_width_encoder</td>
<td>Kernel width for the <em>cnn</em> encoder.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</td>
</tr>
<tr>
<td>cnn_kernel_width_decoder</td>
<td>Kernel width for the <em>cnn</em> decoder.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 5</td>
</tr>
</tbody>
</table>
## Hyperparameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cnn_num_hidden</code></td>
<td>Number of ( cnn ) 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 ( cnn ) activation type to be used.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of glu, relu, softrelu, sigmoid, or tanh.</td>
</tr>
<tr>
<td></td>
<td>Default value: glu</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><code>rnn_attention_type</code></td>
<td>Attention model for encoders. ( mlp ) refers to concat and bilinear</td>
</tr>
<tr>
<td></td>
<td>refers to general from the Luong et al. paper.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of dot, fixed, mlp, or bilinear.</td>
</tr>
<tr>
<td></td>
<td>Default value: mlp</td>
</tr>
<tr>
<td><code>rnn_attention_num_hidden</code></td>
<td>Number of hidden units for attention layers.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: ( \text{rnn_num_hidden} )</td>
</tr>
</tbody>
</table>

---

75
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rnn_attention_in_upper_layers</code></td>
<td>Pass the attention to upper layers of <code>rnn</code>, like Google NMT paper. Only applicable if more than one layer is used.</td>
</tr>
<tr>
<td></td>
<td>Valid values: boolean (<code>true</code> or <code>false</code>)</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>true</code></td>
</tr>
<tr>
<td><code>rnn_decoder_hidden_dropout</code></td>
<td>Dropout probability for hidden state that combines the context with the <code>rnn</code> hidden state in the decoder.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Float. Range in <code>[0,1]</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>0</code></td>
</tr>
<tr>
<td><code>batch_size</code></td>
<td>Mini batch size for gradient descent.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>64</code></td>
</tr>
<tr>
<td><code>bucketing_enabled</code></td>
<td>Set to <code>false</code> to disable bucketing, unroll to maximum length.</td>
</tr>
<tr>
<td></td>
<td>Valid values: <code>true</code> or <code>false</code></td>
</tr>
<tr>
<td></td>
<td>Default value: <code>true</code></td>
</tr>
<tr>
<td><code>bucket_width</code></td>
<td>Returns (source,target) buckets up to <code>(max_seq_len_source, max_seq_len_target)</code>.</td>
</tr>
<tr>
<td></td>
<td>The longer side of the data uses steps of <code>bucket_width</code> while the shorter side uses steps scaled down by the average target/source length ratio.</td>
</tr>
<tr>
<td></td>
<td>If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of <code>max_len</code>.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>10</code></td>
</tr>
<tr>
<td><code>loss_type</code></td>
<td>Loss function for training.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String (<code>cross-entropy</code>)</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>cross-entropy</code></td>
</tr>
<tr>
<td><code>training_metric</code></td>
<td>Metrics to track on training on validation data.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. Either <code>perplexity</code> or <code>accuracy</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>perplexity</code></td>
</tr>
<tr>
<td><code>optimized_metric</code></td>
<td>Metrics to optimize with early stopping.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of <code>perplexity</code>, <code>accuracy</code>, or <code>bleu</code>.</td>
</tr>
<tr>
<td></td>
<td>Default value: <code>perplexity</code></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</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). Valid values: integer. Default value: 0</td>
</tr>
<tr>
<td>max_num_batches</td>
<td>Maximum number of updates/batches to process. -1 for infinite. Valid values: integer. Default value: -1</td>
</tr>
<tr>
<td>checkpoint_frequency_num_batches</td>
<td>Checkpoint and evaluate every x batches. Valid values: positive integer. 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. Valid values: positive integer. Default value: 3</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. Valid values: positive integer. Default value: none</td>
</tr>
<tr>
<td>optimizer_type</td>
<td>Optimizer to choose from. Valid values: String. One of adam, sgd, or rmsprop. Default value: adam</td>
</tr>
<tr>
<td>weight_init_type</td>
<td>Type of weight initialization. Valid values: String. Either uniform or xavier. Default value: xavier</td>
</tr>
<tr>
<td>weight_init_scale</td>
<td>Weight initialization scale (for uniform and xavier initialization). Valid values: float. Default value: 2.34</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>xavier_factor_type</td>
<td>Xavier factor type.</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of in, out, or avg.</td>
</tr>
<tr>
<td></td>
<td>Default value: in</td>
</tr>
<tr>
<td>learning_rate</td>
<td>Initial learning rate.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0.0003</td>
</tr>
<tr>
<td>weight_decay</td>
<td>Weight decay constant.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 0</td>
</tr>
<tr>
<td>momentum</td>
<td>Momentum constant used for sgd. Don't pass this parameter if</td>
</tr>
<tr>
<td></td>
<td>you are using adam or rmsprop.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: none</td>
</tr>
<tr>
<td>clip_gradient</td>
<td>Clip absolute gradient values greater than this. Set to</td>
</tr>
<tr>
<td></td>
<td>negative to disable.</td>
</tr>
<tr>
<td></td>
<td>Valid values: float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1</td>
</tr>
<tr>
<td>lr_scheduler_type</td>
<td>Learning rate scheduler type. plateau_reduce means reduce</td>
</tr>
<tr>
<td></td>
<td>the learning rate whenever optimized_metric on</td>
</tr>
<tr>
<td></td>
<td>validation_accuracy plateaus. inv_t is inverse time</td>
</tr>
<tr>
<td></td>
<td>decay. learning_rate/(1+decay_rate*t)</td>
</tr>
<tr>
<td></td>
<td>Valid values: String. One of plateau_reduce,</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>reduce factor if optimized_metric didn't improve for this</td>
</tr>
<tr>
<td></td>
<td>many checkpoints.</td>
</tr>
<tr>
<td></td>
<td>Valid values: positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 3</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. 80).

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 and CSV are supported for training.

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 \((K = k \times x)\). 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. 190). 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 \(k \times x\), 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. 81)
- Step 2: Iterate over the Training Dataset and Calculate Cluster Centers (p. 81)
- Step 3: Reduce the Clusters from \(K\) to \(k\) (p. 82)
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. 190). For an example, see Step 3.3.2: Create a Training Job (p. 24).

You now have an initial set of cluster centers.

Step 2: Iterate over the Training Dataset and Calculate Cluster Centers

The cluster centers that you created in the preceding step are mostly random, with some consideration for the training dataset. In this step, you use the training dataset to move these centers toward the true cluster centers. The algorithm iterates over the training dataset, and recalculates the $K$ cluster centers.

1. Read a mini-batch of observations (a small, randomly chosen subset of all records) from the training dataset and do the following.
Note
When creating a model training job, you specify the batch size in the `mini_batch_size` string in the `HyperParameters` string map.

a. Assign all of the observations in the mini-batch to one of the clusters with the closest cluster center.

b. Calculate the number of observations assigned to each cluster. Then, calculate the proportion of new points assigned per cluster.

For example, consider the following clusters:

Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.

Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.

Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

Calculate the proportion of new points assigned to each of clusters as follows:

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Previously Assigned Points</th>
<th>New Points Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>100</td>
<td>25</td>
</tr>
<tr>
<td>c2</td>
<td>150</td>
<td>40</td>
</tr>
<tr>
<td>c3</td>
<td>450</td>
<td>5</td>
</tr>
</tbody>
</table>

For example, consider the following clusters:

Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.

Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.

Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

Calculate the proportion of new points assigned to each of clusters as follows:

\[
p_1 = \text{proportion of points assigned to c1} = \frac{25}{100+25} \\
p_2 = \text{proportion of points assigned to c2} = \frac{40}{150+40} \\
p_3 = \text{proportion of points assigned to c3} = \frac{5}{450+5}
\]

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:

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Center Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1</td>
<td>((1 - p_1) \times \text{center of cluster 1} + (p_1 \times d_1))</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>((1 - p_2) \times \text{center of cluster 2} + (p_2 \times d_2))</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>((1 - p_3) \times \text{center of cluster 3} + (p_3 \times d_3))</td>
</tr>
</tbody>
</table>

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` 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.
<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. Valid values: positive integer. Default value: -</td>
</tr>
<tr>
<td>feature_dim</td>
<td>Dimension of the input vectors. Required. Valid values: positive integer. Default value: -</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of examples in a mini-batch. Required. Valid values: positive integer. Default value: 5000</td>
</tr>
<tr>
<td>init_method</td>
<td>The method by which we choose the initial centers. Valid values: Either random or kmeans++. Default value: random</td>
</tr>
<tr>
<td>extra_center_factor</td>
<td>The algorithm creates num_clusters * extra_center_factor as it runs and reduces the number of centers to k when finalizing. Valid values: Either a positive integer or auto. Default value: auto</td>
</tr>
<tr>
<td>local_lloyd_max_iter</td>
<td>Maximum iterations for Lloyds EM procedure in the local kmeans used in the finalize stage. Valid values: positive integer. Default value: 300</td>
</tr>
<tr>
<td>local_lloyd_tol</td>
<td>Tolerance for change in ssd for early stopping in local kmeans. Valid values: Float. Range in [0, 1]. Default value: 0.0001</td>
</tr>
<tr>
<td>local_lloyd_init_method</td>
<td>Initialization method for local version. Valid values: Either random or kmeans++. Default value: kmeans++</td>
</tr>
<tr>
<td>local_lloyd_num_trials</td>
<td>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</td>
</tr>
<tr>
<td>half_life_time_size</td>
<td>The points can have a decayed weight. When a point is observed its weight, with regard to the computation of the cluster mean is</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| 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 |

**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**

```python
[  
    Record = {
        features = {},
        label = {
            'closest_cluster': {
                keys: [],
                values: [1.0, 2.0]  # float32
            },
            'distance_to_cluster': {
                keys: [],
                values: [3.0, 5.0]  # float32
            }
        }
    }
]```
Principal Component Analysis (PCA)

PCA is an unsupervised 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.

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 (p. 85)
- PCA Hyperparameters (p. 87)
- PCA Response Formats (p. 87)

**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 \( t = \text{num\_components} + \text{extra\_components} \). Given a mini-batch \( X_i \) of dimension \( b \times d \), the worker draws a random matrix \( H_i \) of dimension \( t \times b \). Depending on whether the environment uses a GPU or CPU and the dimension size, the matrix is either a random sign matrix where each entry is \( \pm 1 \) or a \( \text{FJLT} \) (fast Johnson Lindenstrauss transform; for information, see \( \text{FJLT\ Transforms} \) and the follow-up papers). The worker then computes \( H_i X_i \) and maintains \( B = \sum H_i X_i \). The worker also maintains \( h^T \), the sum of columns of \( H_1, \ldots, H_T \) (\( t \) being the total number of mini-batches), and \( s \), 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 \( B^1, h^1, s^1, n^1 \). The server computes \( B, h, s, n \), then computes \( C = B - (1/n)h^T s \), and finds its singular value decomposition. The top-right singular vectors and singular values of \( C \) are used as the approximate solution to the problem.
PCA Hyperparameters

In the CreateTrainingJob request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see How PCA Works (p. 85).

<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. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td>feature_dim</td>
<td>Input dimension. Required. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td>mini_batch_size</td>
<td>Number of rows in a mini-batch. Required. Valid values: Positive integer Default value: -</td>
</tr>
<tr>
<td>algorithm_mode</td>
<td>Mode for computing the principal components. Valid values: regular or randomized Default value: regular</td>
</tr>
<tr>
<td>subtract_mean</td>
<td>Indicates whether the data should be unbiased both during training and at inference. Valid values: One of true or false Default value: true</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. Valid values: Non-negative integer or -1 Default value: -1</td>
</tr>
</tbody>
</table>

PCA Response Formats

JSON

Accept—application/json

```json
{
}
```
Latent Dirichlet Allocation (LDA)

Amazon SageMaker LDA 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 different documents with similar topic mixtures will not be the same. But overall, you’d 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>
</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 be of dimension `number of records * vocabulary size`.

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. 89)
- LDA Hyperparameters (p. 91)

### 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 preprocessing 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
```
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. 89).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| num_topics         | The number of topics for LDA to find within the data. Required. Valid values: Positive integer  
                      Default value: -                                                                                                          |
| feature_dim        | The size of the vocabulary of the input document corpus. Required. Valid values: Positive integer  
                      Default value: -                                                                                                          |
| mini_batch_size    | The total number of documents in the input document corpus. Required. Valid values: Positive integer  
                      Default value: -                                                                                                          |
| alpha0             | 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  
                      Default value: 0.1                                                                                                           |
<p>| max_restarts       | 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. |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 10</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.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive integer</td>
</tr>
<tr>
<td></td>
<td>Default value: 1000</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.</td>
</tr>
<tr>
<td></td>
<td>Valid values: Positive float</td>
</tr>
<tr>
<td></td>
<td>Default value: 1e-8</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 the latent representations of documents that the topic model learns as 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.

**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 the dimension number of records * vocabulary size.
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. 93)
- NTM Response Formats (p. 94)

**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</td>
</tr>
<tr>
<td></td>
<td>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></td>
<td>Default value: sigmoid</td>
</tr>
</tbody>
</table>
### Parameter Name | Description
--- | ---
**optimizer** | Optimizer to use for training.
- Valid values: One of `sgd`, `adam`, `adagrad`, `adadelta`, or `rmsprop`
- 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.
- Valid values: float, min: 1e-6, max: 0.1
- Default value: 0.001

**num_patience_epochs** | Number of successive epochs over which early stopping criterion is evaluated.
- Valid values: Positive integer (min: 1, max: 10)
- Default value: 3

**batch_norm** | Whether to use batch normalization during training.
- Valid values: `true` or `false`
- Default value: `false`

**rescale_gradient** | Rescale factor for gradient.
- Valid values: float, min: 1e-3, max: 1.0
- Default value: 1.0

**clip_gradient** | Maximum magnitude for each gradient component.
- Valid values: float, min: 1e-3
- Default value: Infinity

**weight_decay** | Weight decay coefficient. Adds L2 regularization.
- Valid values: float, min: 0.0, max: 1.0
- Default value: 0.0

**learning_rate** | Learning rate for the optimizer.
- Valid values: float, min: 1e-6, max: 1.0
- Default value: 0.001

### NTM Response Formats

**JSON**

```
{
}
```
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 more time series, and, optionally a categorical grouping variable that the time series is a member of. The model learns entirely from these values. The DeepAR algorithm 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.

Input/Output Interface

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. 251) 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.

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:

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

where $y_{i,t}$ is the true value of time series $i$ at time $t$ and $\hat{y}_{i,t}$ 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 forecast horizon by setting the prediction_length hyperparameter (see DeepAR Hyperparameters (p. 97)).

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:

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

Here, $q_{i,t}^{(\tau)}$ is the $\tau$-quantile of the distribution that the model predicts. Set the test_quantiles hyperparameter to specify which quantiles for which the algorithm calculates quantile loss. For information, see DeepAR Hyperparameters (p. 97).

If you have a set of time series, as simple way to prepare, test, and train datasets is as follows:
• Use the full dataset in the test channel.
• In the train channel, remove the last `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. 100).

**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.

**Topics**
- DeepAR Hyperparameters (p. 97)
- DeepAR Request and Response Formats (p. 100)

**DeepAR Hyperparameters**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>time_freq</code></td>
<td>The granularity of the time series in the dataset. Use <code>time_freq</code> 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:</td>
</tr>
</tbody>
</table>
|                     | • `M`: monthly  
|                     | • `W`: weekly  
|                     | • `D`: daily  
|                     | • `H`: hourly  
|                     | • `min`: every minute  

Valid values: One of `M`, `W`, `D`, `H`, or `min`.  
Default value: -

<table>
<thead>
<tr>
<th><code>prediction_length</code></th>
<th>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.</th>
</tr>
</thead>
</table>
|                     | Valid values: positive integer  
<p>|                     | Default value: - |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context_length</td>
<td>The number of time-points that the model gets to see before making the prediction. The value for this parameter should be about the same as the prediction_length. The model also receives lagged inputs from the target, so context_length can be much smaller than typical seasonalities. For example, a daily time series can have yearly seasonality. The model automatically includes a lag of one year, so the context length can be shorter than a year. The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year. Required. Valid values: positive integer Default value: -</td>
</tr>
</tbody>
</table>
| likelihood | The model generates a probabilistic forecast, and can provide quantiles of the distribution and return samples. Depending on your data, select an appropriate likelihood (noise model) that is used for uncertainty estimates. The following likelihoods can be selected:  
  * gaussian: Use for real-valued data.  
  * beta: Use for real-valued targets between 0 and 1 inclusive.  
  * negative-binomial: Use for count data (non-negative integers).  
  * student-T: An alternative for real-valued data that works well for bursty data.  
  * deterministic-L1: A loss function that does not estimate uncertainty and only learns a point forecast. Valid values: One of gaussian, beta, negative-binomial, student-T, or deterministic-L1. Default value: gaussian |
<p>| 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). Required if you use categorical features. Valid values: positive integer Default value: - |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Valid values:</th>
<th>Default value:</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 <code>embedding_dimension</code> for each group, capturing the common properties of all time series in the group. A larger <code>embedding_dimension</code> allows the model to capture more complex patterns. However, because increasing the <code>embedding_dimension</code> 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.</td>
<td>positive integer</td>
<td>-</td>
</tr>
<tr>
<td>num_cells</td>
<td>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</td>
<td>positive integer</td>
<td>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>
<td>positive integer</td>
<td>3</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>
<td>positive integer</td>
<td>32</td>
</tr>
<tr>
<td>learning_rate</td>
<td>The learning rate used in training. Typical values range from $10^{-4}$ to $10^{-1}$.</td>
<td>float</td>
<td>$10^{-3}$</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>
<td>float</td>
<td>0.05</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>
<td>integer</td>
<td>-</td>
</tr>
</tbody>
</table>
Parameter Name | Description
---|---
test_quantiles | Quantiles for which to calculate quantile loss.

Valid values: array of floats
Default value: [0.5, 0.9]

DeepAR Request and Response Formats

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

```json
{
    "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:

```json
{
    "predictions": [
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]}
        ,
        {
            "quantiles": {
                "0.9": [...],
                "0.5": [...]
            },
            "samples": [...],
            "mean": [...]}
        ,
        {
            "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 (p. 271)` object that you pass to `CreateTrainingJob (p. 190)` 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. 102)

## 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>
</table>
| **mode**       | The Word2vec architecture used for training. Required.  
|                | Valid values: `batch_skipgram`, `skipgram`, or `cbow`  
|                | Default value: - |
| **min_count**  | The minimum number of times a word must appear in the collection to be included in training and output.  
|                | Valid values: Non-negative integer  
|                | Default value: 5 |
| **learning_rate** | The initial learning rate.  
|                  | Valid values: Positive float  
|                  | Default value: 0.05 |
| **window_size** | The size of the context window. The context window is the number of words surrounding the target word used in training.  
|                  | Valid values: Positive integer  
|                  | Default value: 5 |
| **vector_dim**  | The dimension of the word vectors that the algorithm learns.  
|                | Valid values: Positive integer  
|                | Default value: 100 |
| **epochs**      | The number of complete passes through the training data.  
|                | Valid values: Positive integer  
<p>|                | Default value: 5 |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
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. 122) and Using TensorFlow with Amazon SageMaker (p. 112).

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

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. 105)
- How Amazon SageMaker Provides Training Information (p. 106)
- Signalling Algorithm Success and Failure (p. 108)
- How Amazon SageMaker Processes Training Output (p. 108)
- Next Step (p. 109)

How Amazon SageMaker Runs Your Training Image

To configure a Docker container to run as an executable, use an ENTRYPOINT instruction in a Dockerfile. Note the following:

- For model training, Amazon SageMaker runs the container as follows:

  ```
  docker run image train
  ```

  Amazon SageMaker overrides any default CMD statement in a container by specifying the train argument after the image name. The train argument also overrides arguments that you provide using CMD in the Dockerfile.

- Use the exec form of the ENTRYPOINT instruction:

  ```
  ENTRYPOINT ["executable", "param1", "param2"]
  ```

  For example:

  ```
  ENTRYPOINT ["python", "k-means-algorithm.py"]
  ```

  The exec form of the ENTRYPOINT instruction starts the executable directly, not as a child of /bin/sh. This enables it to receive signals like SIGTERM and SIGHUP from Amazon SageMaker APIs. Note the following:

  - The CreateTrainingJob (p. 190) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.

  - The StopTrainingJob (p. 238) API issues the equivalent of the docker stop, with a 2 minute timeout, command to gracefully stop the specified container:
docker stop -t120

The command attempts to stop the running container by sending a **SIGTERM** signal. After the 2 minute timeout, **SIGKILL** is sent and the containers are forcibly stopped. If the container handles the **SIGTERM** gracefully and exits within 120 seconds from receiving it, no **SIGKILL** is sent.

**Note**

If you want access to the intermediate model artifacts after Amazon SageMaker stops the training, add code to handle saving artifacts in your **SIGTERM** handler.

- If you plan to use GPU devices for model training, make sure that your containers are **nvidia-docker** compatible. Only the CUDA toolkit should be included on containers; don't bundle NVIDIA drivers with the image. For more information about **nvidia-docker**, see [NVIDIA/nvidia-docker](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. 106)
- **Input Data Configuration** (p. 106)
- **Training Data** (p. 107)
- **Distributed Training Configuration** (p. 108)

## Hyperparameters

Amazon SageMaker makes the hyperparameters in a `CreateTrainingJob` 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` 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:
How Amazon SageMaker Provides Training Information

```json
{
  "train" : {
    "ContentType": "trainingContentType",
    "TrainingInputMode": "File",
    "S3DistributionType": "FullyReplicated",
    "RecordWrapperType": "None"},
  "evaluation" : {
    "ContentType": "evalContentType",
    "TrainingInputMode": "File",
    "S3DistributionType": "FullyReplicated",
    "RecordWrapperType": "None"},
  "validation" : {
    "TrainingInputMode": "File",
    "S3DistributionType": "FullyReplicated",
    "RecordWrapperType": "None"}
}
```

**Note**

Amazon SageMaker provides only relevant information about each data channel (for example, the channel name and the content type) to the container, as shown.

**Training Data**

The `TrainingInputMode` parameter in a `CreateTrainingJob` request specifies how to make data available for model training: in `FILE` mode or `PIPE` mode. Depending on the specified input mode, Amazon SageMaker does the following:

- **FILE mode**—Amazon SageMaker makes the data for the channel available in the `/opt/ml/input/data/channel_name` directory in the Docker container. For example, if you have three channels named training, validation, and testing, Amazon SageMaker makes three directories in the Docker container:
  - `/opt/ml/input/data/training`
  - `/opt/ml/input/data/validation`
  - `/opt/ml/input/data/testing`

- **PIPE mode**—Amazon SageMaker makes data for the channel available from the named pipe: `/opt/ml/input/data/channel_name_epoch_number`. For example, if you have three channels named training, validation, and testing, you will need to read from the following pipes:
  - `/opt/ml/input/data/training_0`, `/opt/ml/input/data/training_1`, ...
  - `/opt/ml/input/data/validation_0`, `/opt/ml/input/data/validation_1`, ...
  - `/opt/ml/input/data/testing_0`, `/opt/ml/input/data/testing_1`, ...

Read the pipes sequentially. For example, if you have a channel called `training`, read the pipes in this sequence:

1. Open `/opt/ml/input/data/training_0` in read mode and read it to EOF (or if you are done with the first epoch, close the file early).
2. After closing the first pipe file, look for `/opt/ml/input/data/training_1` and read it to go through the second epoch, and so on.

If the file for a given epoch doesn't exist yet, your code may need to retry until the pipe is created. There is no sequencing restriction across channel types. That is, you can read multiple epochs for the `training` channel, for example, and only start reading the `validation` channel when you are ready. Or, you can read them simultaneously if your algorithm requires that.
Distributed Training Configuration

If you're performing distributed training with multiple containers, Amazon SageMaker makes information about all containers available in the /opt/ml/input/config/resourceconfig.json file.

To enable inter-container communication, this JSON file contains DNS 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 DNS 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 DNS 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.

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/data—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. 109)

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. 109)
• How Amazon SageMaker Loads Your Model Artifacts (p. 110)
• How Containers Serve Requests (p. 110)
• How Your Container Should Respond to Inference Requests (p. 110)
• How Your Container Should Respond to Health Check (Ping) Requests (p. 111)

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. 175) API to create an endpoint, Amazon SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. Amazon SageMaker runs the Docker container on those instances.
If you reduce the number of instances backing the endpoint (by calling the UpdateEndpointWeightsAndCapacities (p. 242) 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 (p. 240) 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 (p. 181) request to set environment variables and the DNS hostname for the container as follows:
  - It sets environment variables using the ContainerDefintion.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 (p. 181) 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 uncompresses 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 (p. 246) 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 Developer Guide

How Your Container Should Respond to Health Check (Ping) Requests

- 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.

**Note**
Amazon SageMaker does not currently run this health check continuously. This is intended to prevent an unstable container going into service against an endpoint at startup.

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/scikitbring_your_own/scikitbring_your_own.ipynb
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. 170).

Topics
• Writing Custom TensorFlow Model Training and Inference Code (p. 112)
• Examples: Using Amazon SageMaker with TensorFlow (p. 115)

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 supported TensorFlow version is 1.4.0. 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.
    """
```
Args:

- features: A dict containing the features passed to the model with `train_input_fn` in training mode, with `eval_input_fn` in evaluation mode, and with `serving_input_fn` in predict mode.
- labels: Tensor containing the labels passed to the model with `train_input_fn` in training mode and `eval_input_fn` in evaluation mode. It is empty for predict mode.
- mode: One of the following `tf.estimator.ModeKeys` string values indicating the context in which the model_fn was invoked:
  - TRAIN: The model_fn was invoked in training mode.
  - EVAL: The model_fn was invoked in evaluation mode.
  - PREDICT: The model_fn was invoked in predict mode.
- 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: An `EstimatorSpec`, which contains evaluation and loss function.

```python
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 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/18003982ff9c809ab8e9b78d8dc999ebc795f4b8/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. 116)

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. 116)

**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:
Example 1: Using the tf.estimator

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. 117)

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. 13) 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. 13), or create a new bucket.

```
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. 118)

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-py/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.
Example 1: Using the tf.estimator

- `train_instance_type` and `train_instance_count`—Identify the type and number of ML Compute instances to launch for model training.

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. 190) request to Amazon SageMaker.

You can get the training job information by calling the `DescribeTrainingJob` (p. 212) or viewing it in the console. The following is an example response:

```json
{}
```

```
119
```
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** (p. 120)

**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` (p. 181) 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` (p. 178) 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` (p. 175) 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 had been deployed. The API returns a TensorFlowPredictor object. To get inferences, you will use the `predict` method of this object.

**Next Step**

**Step 4: Invoke the Endpoint to Get Inferences (p. 121)**

**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 (p. 246).

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. 116)*.

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

```json
{u'result': {u'classifications': [{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. 170).

Topics
- Writing Custom Apache MXNet Model Training and Inference Code (p. 122)
- Examples: Using Amazon SageMaker with Apache MXNet (p. 128)

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
# ----------------------------------------------------------------------------#
# Required
# ----------------------------------------------------------------------------#

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.

```
```
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' 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.
    """    pass
```

- Implement the following inference code interface:

```python
# your inference code here
```
# Hosting functions

```python
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 <https://mxnet.incubator.apache.org/api/python/gluon/gluon.html#mxnet.gluon.Block>` or MXNet `Module <https://mxnet.incubator.apache.org/api/python/module.html>`, 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.


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

   ""

# 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 in 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.

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

pass
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 uses 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. 15). 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 details. 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. 128)

**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:

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. 129)
- Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code (p. 130)
- Step 3: Deploy the Trained Model (p. 132)
- Step 4: Invoke the Endpoint to Get Inferences (p. 133)

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. 13) 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. 13), 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.
role = get_execution_role()
```
Next Step

Step 2: Train a Model on Amazon SageMaker Using Apache MXNet Custom Code (p. 130)

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.

   - 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)
   ```
Example 1: Using the Module API

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. 190) request to Amazon SageMaker.

You can get the training job information by calling the `DescribeTrainingJob` (p. 212) or viewing it in the console. The following is an example response:

```
{
   "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://sagemakermv/artifacts"
   },
   "TrainingJobName": "sagemaker-mxnet-py2-cpu-2017-11-18-02-02-18-586",
   "StoppingCondition": {
      "MaxRuntimeInSeconds": 86400
   },
   "ResourceConfig": {
      "InstanceCount": 1,
      "InstanceType": "ml.m4.xlarge",
      "VolumeSizeInGB": 30
   },
   "RoleArn": "arn:aws:iam::account-id:role/SageMakerRole"
}
```

Details:
Example 1: Using the Module API

• **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. 132)

**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. 181)` 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. 178)` 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. 175)` 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
```
Example 1: Using the Module API

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

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

**Next Step**

**Step 4 : Invoke the Endpoint to Get Inferences**  (p. 133)

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

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. 246).

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 raw 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: {}'.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.

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).

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).
Example 1: Using the Module API

- 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 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. 170).

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.Estimators 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
    ```
  - If you are using a Jupyter notebook on your Amazon SageMaker notebook instance to develop applications, the PySpark library is also available on your notebook instance.

- You can get the Scala library from Maven. Add the Spark library to your project by adding the following dependency to your pom.xml file:

```xml
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>sagemaker-spark_2.11</artifactId>
  <version>spark_2.2.0-1.0</version>
</dependency>
```
Integrating Your Apache Spark Application with Amazon SageMaker

The following is a high-level summary of the steps for integrating your Apache Spark application with Amazon SageMaker.

1. Continue data preprocessing using the Apache Spark library that you are familiar with. Your dataset remains a DataFrame in your Spark cluster.

   **Note**
   Load your data into a DataFrame and preprocess it so that you have a `features` column with `org.apache.spark.ml.linalg.Vector` of `Doubles`, and an optional `label` column with values of `Double` type.

2. Use the estimator in the Amazon SageMaker Spark library to train your model. For example, if you choose the k-means algorithm provided by Amazon SageMaker for model training, you call the `KMeansSageMakerEstimator.fit` method.

   Provide your DataFrame as input. The estimator returns a `SageMakerModel` object.

   **Note**
   `SageMakerModel` extends the `org.apache.spark.ml.Model`.

   The `fit` method does the following:
   a. Converts the input DataFrame to the protobuf format by selecting the `features` and `label` columns from the input DataFrame and uploading the protobuf data to an Amazon S3 bucket. The protobuf format is efficient for model training in Amazon SageMaker.
   b. Starts model training in Amazon SageMaker by sending an Amazon SageMaker `CreateTrainingJob` request. After model training has completed, Amazon SageMaker saves the model artifacts to an S3 bucket.
   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 to Amazon SageMaker.
      ii. Sends a `CreateEndpointConfig` request to Amazon SageMaker.
      iii. Sends a `CreateEndpoint` request 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`.

   Provide an input DataFrame with features as input. The `transform` method transforms it to a DataFrame containing inferences. Internally, the `transform` method sends a request to the `InvokeEndpoint` Amazon SageMaker 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. 141)
- Using the SageMakerEstimator in a Spark Pipeline (p. 142)

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. 135). 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. 79).

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 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"

val estimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(784)

// train
val model = estimator.fit(trainingData)

val transformedData = model.transform(testData)
transformedData.show
```

The code does the following:

- Loads the MNIST dataset from an S3 bucket provided by Amazon SageMaker (awsai-sparksdk-dataset) into a Spark DataFrame (mnistTrainingDataFrame):

```scala
trainingData.show()
```

The `show` method displays the first 20 rows in the data frame:

```
+-----+--------------------+
|label|            features|
+-----+--------------------+
|  5.0|[784,[152,153,154...|
|  0.0|[784,[127,128,129...|
|  4.0|[784,[160,161,162...|
|  1.0|[784,[158,159,160...|
|  9.0|[784,[208,209,210...|
|  2.0|[784,[155,156,157...|
|  1.0|[784,[124,125,126...|
|  3.0|[784,[151,152,153...|
|  1.0|[784,[152,153,154...|
|  4.0|[784,[134,135,161...|
|  3.0|[784,[123,124,125...|
|  5.0|[784,[216,217,218...|
|  3.0|[784,[143,144,145...|
|  6.0|[784,[72,73,74,99...|
|  1.0|[784,[151,152,153...|
```
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.
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

```java
// 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. 136). 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.

```java
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`:

```
+-----+--------------------+-------------------+---------------+
|label|            features|distance_to_cluster|closest_cluster|
+-----+--------------------+-------------------+---------------+
|  5.0|(784,[152,153,154...|  1767.897705078125|            4.0|
|  0.0|(784,[127,128,129...|  1392.157470703125|            5.0|
|  4.0|(784,[160,161,162...|  1671.5711669921875|            9.0|
|  1.0|(784,[158,159,160...|  1182.6082763671875|            6.0|
|  9.0|(784,[208,209,210...|  1390.4002685546875|            0.0|
|  2.0|(784,[155,156,157...|  1713.988037109375|            1.0|
|  1.0|(784,[124,125,126...|  1246.3016357421875|            2.0|
|  3.0|(784,[151,152,153...|  1753.229248046875|            4.0|
|  1.0|(784,[152,153,154...|  978.8394165039062|            2.0|
|  4.0|(784,[134,135,161...|  1623.176513671875|            3.0|
|  3.0|(784,[123,124,125...|  1533.863525390625|            4.0|
|  5.0|(784,[216,217,218...|  1469.357177734375|            6.0|
```
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. 137), 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(integTestingRole),
  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
  ```java
  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
  ```java
  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)
```
A set of additional Apache Spark examples are available at https://github.com/aws/sagemaker-spark/tree/master/examples.

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:

| label |  | features |  | projectedFeatures |  | distance_to_cluster |  | closest_cluster |  |
|-------| |-----------| |-----------------| |-------------------| |-------------------| |-----------------| |
| 5.0   |  | [784,152,153,154... | [880.731433034386... | 1500.470703125... | 0.0 | 1142.18359375... | 4.0 |                 | |
| 0.0   |  | [784,127,128,129... | [1768.5172022166... | 1142.18359375... | 4.0 |                 | 9.0 |                 | |
| 4.0   |  | [784,160,161,162... | [704.94923632934... | 1386.246826171875 | 9.0 |                 | 3.0 |                 | |
| 1.0   |  | [784,158,159,160... | [-42.328192293771... | 1277.0736083984375 | 5.0 |                 | 3.0 |                 | |
| 9.0   |  | [784,208,209,210... | [374.043902028333... | 1211.00927734375 | 3.0 |                 | 8.0 |                 | |
| 2.0   |  | [784,155,156,157... | [941.267714528850... | 1496.157958984375 | 8.0 |                 | 5.0 |                 | |
| 1.0   |  | [784,124,125,126... | [30.284859610594... | 1327.676357421875 | 5.0 |                 | 5.0 |                 | |
| 3.0   |  | [784,151,152,153... | [1270.14374062052... | 1570.7674560546875 | 0.0 |                 | 5.0 |                 | |
| 1.0   |  | [784,152,153,154... | [-112.10792566485... | 1037.568539375... | 5.0 |                 | 3.0 |                 | |
| 4.0   |  | [784,134,135,161... | [452.068280676606... | 1165.1236572265625 | 3.0 |                 | 7.0 |                 | |
| 3.0   |  | [784,123,124,125... | [610.596447285397... | 1325.953369140625 | 7.0 |                 | 5.0 |                 | |
| 5.0   |  | [784,216,217,218... | [142.959601818422... | 1353.4930419921875 | 5.0 |                 | 7.0 |                 | |
| 3.0   |  | [784,143,144,145... | [1036.71862533658... | 1460.4351855646875 | 7.0 |                 | 2.0 |                 | |
| 6.0   |  | [784,72,73,74,99... | [996.7401574535754... | 1159.8631591796875 | 2.0 |                 | 8.0 |                 | |
| 1.0   |  | [784,151,152,153... | [-107.26076167417... | 960.963623046875 | 5.0 |                 | 6.0 |                 | |
| 7.0   |  | [784,211,212,213... | [619.7731820430940... | 1245.13623046875 | 6.0 |                 | 6.0 |                 | |
| 2.0   |  | [784,151,152,153... | [850.152101817161... | 1304.437744140625 | 8.0 |                 | 6.0 |                 | |
| 8.0   |  | [784,159,160,161... | [30.041887230547... | 1182.478494140625 | 0.0 |                 | 2.0 |                 | |
| 6.0   |  | [784,100,101,102... | [546.674328209335... | 1277.090823125 | 2.0 |                 | 6.0 |                 | |
| 9.0   |  | [784,209,210,211... | [-29.259112972426... | 1245.8182373046875 | 6.0 |                 | 6.0 |                 | |

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. 145)
• Access Control (p. 146)

Authentication

You can access AWS as any of the following types of identities:

• **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

• **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create the in Amazon SageMaker). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don’t use AWS tools, you must sign the request yourself. Amazon SageMaker supports Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

• **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

• **Federated user access** – Instead of creating an IAM user, you can use existing user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
Access Control

You can have valid credentials to authenticate your requests, but unless you have permissions you cannot create or access Amazon SageMaker resources. For example, you must have permissions to create an Amazon SageMaker notebook instance.

The following sections describe how to manage permissions for Amazon SageMaker. We recommend that you read the overview first.

- Overview of Managing Access Permissions to Your Amazon SageMaker Resources (p. 146)
- Using Identity-based Policies (IAM Policies) for Amazon SageMaker (p. 150)
- Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference (p. 152)

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. 147)
- Understanding Resource Ownership (p. 147)
- Managing Access to Resources (p. 147)
- Specifying Policy Elements: Resources, Actions, Effects, and Principals (p. 149)
- Specifying Conditions in a Policy (p. 149)
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/notebookInstanceName</td>
</tr>
<tr>
<td>Training Job</td>
<td>arn:aws:sagemaker:region:account-id:training-job/trainingJobName</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. 171).

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. 148)
- Resource-Based Policies (p. 149)

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 an 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",
```

```json
```
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. 150). 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. 147) and API Reference (p. 171).

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. 147).
- **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. 152).

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
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. 146).

Topics
• Permissions Required to Use the Amazon SageMaker Console (p. 151)
• AWS Managed (Predefined) Policies for Amazon SageMaker (p. 151)

The following is an example of a basic permissions policy:

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "AllowCreate-Describe-Delete-Models",
    "Effect": "Allow",
    "Action": [
      "sagemaker:CreateModel",
      "sagemaker:DescribeModel",
      "sagemaker:DeleteModel"
    ],
    "Resource": "*
  }
},
  "Statement": [{
    "Sid": "AdditionalIamPermission",
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": 
      "arn:aws:iam::account-id:role/role-name"
  }
]
}
```

The policy has two statements:

- The first statement grants permission for three Amazon SageMaker actions (sagemaker:CreateModel, sagemaker:DescribeModel, and sagemaker:DeleteModel) within an Amazon SageMaker notebook instance. Using the wildcard character (*) as the resource grants universal permissions for these actions across all AWS Regions and models owned by this account.
- The second statement grants permission for the iam:PassRole action, which is needed for the Amazon SageMaker action sagemaker:CreateModel, which is allowed by the first statement.

The policy doesn't specify the Principal element because in an identity-based policy you don't specify the principal who gets the permission. When you attach the policy to a user, the user is the implicit...
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. 152).

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. 146) 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. 173)**

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

**API Operation: CreateEndpoint (p. 175)**

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

**API Operation: CreateEndpointConfig (p. 178)**

- Required Permissions (API Action): sagemaker:CreateEndpointConfig

152
API Operation: CreateModel (p. 181)

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

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

API Operation: CreateNotebookInstance (p. 184)


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

API Operation: CreateTrainingJob (p. 190)

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

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

API Operation: DeleteEndpoint (p. 195)

Required Permissions (API Action): sagemaker:DeleteEndpoint

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

API Operation: DeleteEndpointConfig (p. 196)

Required Permissions (API Action): sagemaker:DeleteEndpointConfig

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

API Operation: DeleteModel (p. 197)

Required Permissions (API Action): sagemaker:DeleteModel

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

API Operation: DeleteNotebookInstance (p. 198)


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

API Operation: DeleteTags (p. 200)

Required Permissions (API Action): sagemaker:DeleteTags

Resources: *

API Operation: DescribeEndpoint (p. 202)

Required Permissions (API Action): sagemaker:DescribeEndpoint

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

API Operation: DescribeEndpointConfig (p. 205)

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

**API Operation:** DescribeModel (p. 207)

Required Permissions (API Action): sagemaker:DescribeModel

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

**API Operation:** DescribeNotebookInstance (p. 209)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

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

**API Operation:** DescribeTrainingJob (p. 212)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

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

**API Operation:** CreatePresignedNotebookInstanceUrl (p. 188)

Required Permissions (API Action): sagemaker:CreatePresignedNotebookInstanceUrl

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

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

Required Permissions (API Action): sagemaker:InvokeEndpoint

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

**API Operation:** ListEndpointConfigs (p. 217)

Required Permissions (API Action): sagemaker:ListEndpointConfigs

Resources: *

**API Operation:** ListEndpoints (p. 220)

Required Permissions (API Action): sagemaker:ListEndpoints

Resources: *

**API Operation:** ListModels (p. 223)

Required Permissions (API Action): sagemaker:ListModels

Resources: *

**API Operation:** ListNotebookInstances (p. 226)

Required Permissions (API Action): sagemaker:ListNotebookInstances

Resources: *

**API Operation:** ListTags (p. 229)

Required Permissions (API Action): sagemaker:ListTags

Resources: *

**API Operation:** ListTrainingJobs (p. 231)

Required Permissions (API Action): sagemaker:ListTrainingJobs

Resources: *

**API Operation:** StartNotebookInstance (p. 234)

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. 184), CreateTrainingJob (p. 190), and CreateModel (p. 181).

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. 161). 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. 156)
- CreateTrainingJob API: Execution Role Permissions (p. 158)
- CreateModel API: Execution Role Permissions (p. 160)
- Using the AWS Managed Permission Policy (AmazonSageMakerFullAccess) for an Execution Role (p. 161)

CreateNotebookInstance API: Execution Role Permissions

The permissions that you grant to the execution role for calling the CreateNotebookInstance API depend on what you plan to do with the notebook instance. If you plan to use it to invoke Amazon SageMaker APIs and pass the same role when calling the CreateTrainingJob and CreateModel APIs, attach the following permissions policy to the role:

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "cloudwatch:PutMetricData",
                "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:

```json
{
    "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"
            ]
        }
    ]
}
```
CreateTrainingJob 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](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch-resources-and-operations.html) in the Amazon CloudWatch User Guide.

## 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"
         ]
      }
   ]
}
```
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 `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.
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 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"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*"
    }
  ]
}
```
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` 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.

## Using the AWS Managed Permission Policy
(AmazonSageMakerFullAccess) for an Execution Role

You can create an execution role one of two ways:

- In the Amazon SageMaker console when you create a notebook instance, training job, or model.
- In the AWS Identity and Access Management (IAM) console. You then specify the role as you follow the
  notebook instance, training job, and model creation workflows in the Amazon SageMaker console.

Regardless of how you create an execution role, you can attach the AWS-managed permission policy
(AmazonSageMakerFullAccess) to the role.

When attaching the `AmazonSageMakerFullAccess` policy to a role, you must do one of the following to allow Amazon SageMaker to access your S3 bucket:

- Include the string "SageMaker" or "sagemaker" in the name of the bucket where you store training
data, or the model artifacts resulting from model training, or both.
- Include the string "SageMaker" or "sagemaker" in the object name of the training data object(s).
- Tag the S3 object with "sagemaker=true". The key and value are case sensitive. For more information, see
  `Object Tagging` in the Amazon Simple Storage Service Developer Guide.
- Add a bucket policy that allows access for the execution role. For more information, see Using Bucket
  Policies and User Policies in the Amazon Simple Storage Service Developer Guide.
You can attach additional policies that specify the resources for which you want to grant permissions for the `s3:GetObject`, `s3:PutObject`, and `s3:ListBucket` actions. In the IAM console, you can attach a customer managed policy or an inline policy to your execution role(s). Alternatively, when you create a role in the Amazon SageMaker console, you can attach a customer managed policy that specifies the S3 buckets. This resulting execution role has the prefix "AmazonSageMaker-ExecutionRole-".
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. 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. 246).

Metrics are available at a 1-minute frequency.

<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 InvokeEndpoint requests where the model returned a 4xx HTTP response code.</td>
</tr>
<tr>
<td></td>
<td>For each 4xx response, 1 is sent; otherwise, 0 is sent.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Valid statistics: Average, Sum</td>
</tr>
</tbody>
</table>
### Metric Description

**Invocation5XXErrors**

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.

- Units: Count
- Valid statistics: Average, Sum

**Invocations**

The number of InvokeEndpoint requests sent to a model.

To get the total number of requests to the endpoint variant, use the Sum statistic.

- Units: Count
- Valid statistics: Sum, Sample Count

**InvocationsPerInstance**

The number of invocations sent to a model, normalized by InstanceCount in each ProductionVariant. \( \frac{1}{\text{numberOfInstances}} \) is sent as the value on each request, where numberOfInstances is the number of active instances for the ProductionVariant behind the endpoint at the time of the request.

- Units: Count
- Valid statistics: Sum

---

### 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 <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. For endpoints, the value for this dimension has the format <code>[endpoint-name]/{production-variant-name}/{instance-id}</code>. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the <code>/aws/sagemaker/Endpoints</code> 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 and endpoints, anything an algorithm container or a model container 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><code>/aws/sagemaker/TrainingJobs</code></td>
<td><code>[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]</code></td>
</tr>
<tr>
<td><code>/aws/sagemaker/Endpoints/[EndpointName]</code></td>
<td><code>[production-variant-name]/[instance-id]</code></td>
</tr>
</tbody>
</table>

**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",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",
        "arn": "arn:aws:iam::123456789012:user/intern",
        "accountId": "123456789012",
        "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
        "userName": "intern"
    },
    "eventTime": "2018-01-02T13:39:06Z",
    "eventSource": "sagemaker.amazonaws.com",
    "eventName": "CreateEndpoint",
    "awsRegion": "us-west-2",
    "eventSourceIdentity": {
        "type": "IAMUser",
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",
        "arn": "arn:aws:iam::123456789012:user/intern",
        "accountId": "123456789012",
        "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
        "userName": "intern"
    },
    "requestParameters": {
        "x-amz-security-token": "AIXDAYQEXAMPLEUMLYNGL"
    },
    "resources": [
        {
            "type": "Endpoint"
        }
    ],
    "eventBusName": "default",
    "eventType": "AwsResourceConfigure",
    "awsRegion": "us-west-2",
    "sourceAccount": "123456789012"
}
```
The following example is a log entry for the `CreateModel` action, which creates one or more containers to host a previously trained model.

```json
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",
        "arn": "arn:aws:iam::123456789012:user/intern",
        "accountId": "123456789012",
        "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",
        "userName": "intern"
    },
    "eventTime": "2018-01-02T15:23:46Z",
    "eventSource": "sagemaker.amazonaws.com",
    "eventName": "CreateModel",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "USER_AGENT",
    "requestParameters": {
        "modelName": "ExampleModel",
        "primaryContainer": {
            "image": "174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest"
        }
    },
    "executionRoleArn": "arn:aws:iam::123456789012:role/EXAMPLEARN",
    "responseElements": {
    },
    "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, and lists supported versions of learning frameworks.

Topics
- Notebook Instance Security (p. 169)
- Save Updated Sample Notebooks in a New Location (p. 170)
- Supported Versions (p. 170)

Notebook Instance Security

Note the following security considerations for notebook instances.

Topics
- Notebook Instances Are Enabled with Internet Access (p. 169)
- Notebook Instances Provide the Best Experience for a Single User (p. 169)

Notebook Instances Are Enabled with Internet Access

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.

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",
    }
  ]
}
```
Save 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 files in the /sample-notebooks folder. To save your changes, move the relevant files and folders out of the /sample-notebook folder, to any other folder within the SageMaker folder. The SageMaker folder is the default location when you open a notebook instance. Amazon SageMaker saves any files within the SageMaker folder, except for anything in the /sample-notebooks subfolder.

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

Supported Versions

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

- TensorFlow 1.4
- Apache MXNet 0.12
- Apache Spark 2.1.1 and 2.2.0
- When using your own algorithms, you can use any version of the frameworks in your Docker image.
API Reference

This section contains the API Reference documentation.

Topics

- Actions (p. 171)
- Data Types (p. 248)

Actions

The following actions are supported by Amazon SageMaker Service:

- AddTags (p. 173)
- CreateEndpoint (p. 175)
- CreateEndpointConfig (p. 178)
- CreateModel (p. 181)
- CreateNotebookInstance (p. 184)
- CreatePresignedNotebookInstanceUrl (p. 188)
- CreateTrainingJob (p. 190)
- DeleteEndpoint (p. 195)
- DeleteEndpointConfig (p. 196)
- DeleteModel (p. 197)
- DeleteNotebookInstance (p. 198)
- DeleteTags (p. 200)
- DescribeEndpoint (p. 202)
- DescribeEndpointConfig (p. 205)
- DescribeModel (p. 207)
- DescribeNotebookInstance (p. 209)
- DescribeTrainingJob (p. 212)
- ListEndpointConfigs (p. 217)
- ListEndpoints (p. 220)
- ListModels (p. 223)
- ListNotebookInstances (p. 226)
- ListTags (p. 229)
- ListTrainingJobs (p. 231)
- StartNotebookInstance (p. 234)
- StopNotebookInstance (p. 236)
- StopTrainingJob (p. 238)
- UpdateEndpoint (p. 240)
- UpdateEndpointWeightsAndCapacities (p. 242)
- UpdateNotebookInstance (p. 244)

The following actions are supported by Amazon SageMaker Runtime:
The following actions are supported by Amazon SageMaker Service:

- AddTags (p. 173)
- CreateEndpoint (p. 175)
- CreateEndpointConfig (p. 178)
- CreateModel (p. 181)
- CreateNotebookInstance (p. 184)
- CreatePresignedNotebookInstanceUrl (p. 188)
- CreateTrainingJob (p. 190)
- DeleteEndpoint (p. 195)
- DeleteEndpointConfig (p. 196)
- DeleteModel (p. 197)
- DeleteNotebookInstance (p. 198)
- DeleteTags (p. 200)
- DescribeEndpoint (p. 202)
- DescribeEndpointConfig (p. 205)
- DescribeModel (p. 207)
- DescribeNotebookInstance (p. 209)
- DescribeTrainingJob (p. 212)
- ListEndpointConfigs (p. 217)
- ListEndpoints (p. 220)
- ListModels (p. 223)
- ListNotebookInstances (p. 226)
- ListTags (p. 229)
- ListTrainingJobs (p. 231)
- StartNotebookInstance (p. 234)
- StopNotebookInstance (p. 236)
- StopTrainingJob (p. 238)
- UpdateEndpoint (p. 240)
- UpdateEndpointWeightsAndCapacities (p. 242)
- UpdateNotebookInstance (p. 244)
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. 277).

The request accepts the following data in JSON format.

**ResourceArn (p. 173)**

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. 173)**

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. 274) 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. 173)

A list of tags associated with the Amazon SageMaker resource.

Type: Array of Tag (p. 274) 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. 276).

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. 277).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 175)**

The name of an endpoint configuration. For more information, see CreateEndpointConfig.

Type: String

Length Constraints: Maximum length of 63.

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

Required: Yes

**EndpointName (p. 175)**

The name of the endpoint. The name must be unique within an AWS Region in your AWS account.

Type: String

Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9][-]*[a-zA-Z0-9]*)*

Required: Yes

Tags (p. 175)

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. 274) objects

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

Required: No

Response Syntax

```json
{
  "EndpointArn": "string"
}
```

Response Elements

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

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

EndpointArn (p. 176)

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. 276).

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. 277).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 178)**

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. 178)**

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. 178)**

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

Type: Array of `ProductionVariant` (p. 265) objects

Array Members: Minimum number of 1 item.

Required: Yes

**Tags (p. 178)**

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. 274) objects

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

Required: No

**Response Syntax**

```json
{
  "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. 179)**

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. 276).

**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"
        }
    ]
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**ExecutionRoleArn (p. 181)**

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.

Type: String

Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/?[a-zA-Z-A-Z0-9+=,.@-_/]+$  
Required: Yes  

ModelName (p. 181)

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. 181)

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. 253) object  
Required: Yes

Tags (p. 181)

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

Response Syntax

```
{
   "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. 182)

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. 276).
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 an ML compute instance running on a Jupyter notebook.

In a CreateNotebookInstance request, you 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, creates a network interface in your own VPC, which is inferred from the subnet ID that you provide in the input. When creating this network interface, Amazon SageMaker attaches the security group that you specified in the request to the network interface that it creates in your VPC.
3. Launches an EC2 instance of the type specified in the request in the Amazon SageMaker VPC. If you specified SubnetId of your VPC, Amazon SageMaker specifies both network interfaces when launching this instance. This enables inbound traffic from your own VPC to the notebook instance, assuming that the security groups allow it.

After creating the notebook instance, Amazon SageMaker returns its Amazon Resource Name (ARN).

After Amazon SageMaker creates the notebook instance, you can connect to the Jupyter server and work in Jupyter notebooks. For example, you can write code to explore a dataset that you can use for model training, train a model, host models by creating Amazon SageMaker endpoints, and validate hosted models.

For more information, see How It Works.

Request Syntax

```json
{
    "InstanceType": "string",
    "KmsKeyId": "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. 277).

The request accepts the following data in JSON format.
**InstanceType (p. 184)**

The type of ML compute instance to launch for the notebook instance.

Type: String

Valid Values: ml.t2.medium | ml.m4.xlarge | ml.p2.xlarge

Required: Yes

**KmsKeyId (p. 184)**

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

**NotebookInstanceName (p. 184)**

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. 184)**

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.

Type: String


Pattern: ^arn:aws[a-zA-Z\-\*]:iam::\d{12}:role/\?a-zA-Z\-0-9+=,\.,@\-\/]+$

Required: Yes

**SecurityGroupIds (p. 184)**

The VPC security group IDs, in the form sg-xxxxxxx. 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. 184)**

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. 184)**
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. 274) objects
Array Members: Minimum number of 0 items. Maximum number of 50 items.
Required: No

**Response Syntax**

```
{
   "NotebookInstanceArn": "string"
}
```

**Response Elements**

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

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

**NotebookInstanceArn (p. 186)**
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. 276).

**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. 277).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 188)**

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. 188)**

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. 188)**

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. 276).

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"

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

AlgorithmSpecification (p. 190)

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 Bring Your Own Algorithms.

Type: AlgorithmSpecification (p. 250) object

Required: Yes

HyperParameters (p. 190)

Algorithm-specific parameters. You set hyperparameters before you start the learning process. Hyperparameters influence the quality of the model. 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. 190)

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. 251) objects

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

Required: Yes

**OutputDataConfig (p. 190)**

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

Type: OutputDataConfig (p. 264) object

Required: Yes

**ResourceConfig (p. 190)**

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. 269) object

Required: Yes

**RoleArn (p. 190)**

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.

Type: String


Pattern: ^arn:aws[a-z\-]*:iam::\d{12}:role/\?[a-zA-Z\-_0-9+=,.@\-_/\]+$  

Required: Yes

**StoppingCondition (p. 190)**

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` object

Required: Yes

**Tags**

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

Type: Array of `Tag` objects

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

Required: No

**TrainingJobName**

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\-\(\)*\]\ ]*`

Required: Yes

**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**

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

Type: String

Length Constraints: Maximum length of 256.

Pattern: `arn:aws:sagemaker:\[\p{Alnum}\-\)*[0-9]\{12\}:training-job/.*`
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. 277).

The request accepts the following data in JSON format.

**EndpointName (p. 195)**

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-]*$ [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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteEndpointConfig
Service: Amazon SageMaker Service

Deletes an endpoint configuration. The DeleteEndpointConfig API deletes only the specified configuration. It does not delete endpoints created using the configuration.

Request Syntax

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

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 196)**

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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteModel
Service: Amazon SageMaker Service

Deletes a model. The DeleteModel API deletes only the model entry that was created in Amazon SageMaker when you called the CreateModel API. It does not delete model artifacts, inference code, or the IAM role that you specified when creating the model.

Request Syntax

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

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**ModelName (p. 197)**

The name of the model to delete.

Type: String

Length Constraints: Maximum length of 63.

Pattern: `^[a-zA-Z0-9\(-*\[a-zA-Z0-9\-\]0-9)]*`

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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
DeleteNotebookInstance
Service: Amazon SageMaker Service

Deletes an Amazon SageMaker notebook instance. Before you can delete a notebook instance, you must call the StopNotebookInstance API.

**Important**
When you delete a notebook instance, you lose all of your data. Amazon SageMaker removes the ML compute instance, and deletes the ML storage volume and the network interface associated with the notebook instance.

**Request Syntax**

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

**Request Parameters**

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 198)**

The name of the Amazon SageMaker notebook instance to delete.

Type: String

Length Constraints: Maximum length of 63.

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

Required: Yes

**Response Elements**

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

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 276).

**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. 277).

The request accepts the following data in JSON format.

**ResourceArn (p. 200)**

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. 200)**

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. 276).

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**

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

**Request Parameters**

For information about the parameters that are common to all actions, see [Common Parameters](p. 277).

The request accepts the following data in JSON format.

**EndpointName (p. 202)**

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**

```
{
   "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. 202)**

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

**EndpointArn (p. 202)**

The Amazon Resource Name (ARN) of the endpoint.

Type: String


**EndpointConfigName (p. 202)**

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. 202)**

Name of the endpoint.

Type: String

Length Constraints: Maximum length of 63.

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

**EndpointStatus (p. 202)**

The status of the endpoint.

Type: String

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

**FailureReason (p. 202)**

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

Type: String

Length Constraints: Maximum length of 1024.

**LastModifiedTime (p. 202)**

A timestamp that shows when the endpoint was last modified.

Type: Timestamp

**ProductionVariants (p. 202)**

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

Type: Array of ProductionVariantSummary (p. 267) objects

Array Members: Minimum number of 1 item.

---

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 276).
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

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

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 205)**

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

```json
{
   "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. 205)**

A timestamp that shows when the endpoint configuration was created.

Type: Timestamp
EndpointConfigArn (p. 205)

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

Type: String


EndpointConfigName (p. 205)

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. 205)

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. 205)

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

Type: Array of ProductionVariant (p. 265) objects

Array Members: Minimum number of 1 item.

Errors

For information about the errors that are common to all actions, see Common Errors (p. 276).

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. 277).

The request accepts the following data in JSON format.

**ModelName (p. 207)**

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. 207)**

A timestamp that shows when the model was created.

Type: Timestamp
ExecutionRoleArn (p. 207)

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. 207)

The Amazon Resource Name (ARN) of the model.

Type: String


ModelName (p. 207)

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. 207)

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. 253) object

Errors

For information about the errors that are common to all actions, see Common Errors (p. 276).

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. 277).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 209)**

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])*  

Required: Yes

Response Syntax

```json
{
    "CreationTime": number,
    "FailureReason": "string",
    "InstanceType": "string",
    "KmsKeyId": "string",
    "LastModifiedTime": number,
    "NetworkInterfaceId": "string",
    "NotebookInstanceArn": "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. 209)**

A timestamp. Use this parameter to return the time when the notebook instance was created.

Type: Timestamp
FailureReason (p. 209)
If status is failed, the reason it failed.
Type: String
Length Constraints: Maximum length of 1024.

InstanceType (p. 209)
The type of ML compute instance running on the notebook instance.
Type: String
Valid Values: ml.t2.medium | ml.m4.xlarge | ml.p2.xlarge

KmsKeyId (p. 209)
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. 209)
A timestamp. Use this parameter to retrieve the time when the notebook instance was last modified.
Type: Timestamp

NetworkInterfaceId (p. 209)
Network interface IDs that Amazon SageMaker created at the time of creating the instance.
Type: String

NotebookInstanceArn (p. 209)
The Amazon Resource Name (ARN) of the notebook instance.
Type: String
Length Constraints: Maximum length of 256.

NotebookInstanceName (p. 209)
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. 209)
The status of the notebook instance.
Type: String
Valid Values: Pending | InService | Stopping | Stopped | Failed | Deleting

RoleArn (p. 209)
Amazon Resource Name (ARN) of the IAM role associated with the instance.
Type: String

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

**SecurityGroups (p. 209)**  
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. 209)**  
The ID of the VPC subnet.

Type: String

Length Constraints: Maximum length of 32.

**Url (p. 209)**  
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. 276)].

**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. 277).

The request accepts the following data in JSON format.

**TrainingJobName (p. 212)**

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": {
        "S3ModelArtifacts": "string"
    }
}
```
Response Elements

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

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

**AlgorithmSpecification (p. 212)**

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

Type: AlgorithmSpecification (p. 250) object

**CreationTime (p. 212)**

A timestamp that indicates when the training job was created.

Type: Timestamp

**FailureReason (p. 212)**

If the training job failed, the reason it failed.

Type: String

Length Constraints: Maximum length of 1024.

**HyperParameters (p. 212)**

Algorithm-specific parameters.

Type: String to string map

Key Length Constraints: Maximum length of 256.

Value Length Constraints: Maximum length of 256.

**InputDataConfig (p. 212)**

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

Type: Array of Channel (p. 251) objects

Array Members: Minimum number of 1 item. Maximum number of 8 items.
**LastModifiedTime (p. 212)**

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

Type: Timestamp

**ModelArtifacts (p. 212)**

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

Type: ModelArtifacts (p. 260) object

**OutputDataConfig (p. 212)**

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. 264) object

**ResourceConfig (p. 212)**

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

Type: ResourceConfig (p. 269) object

**RoleArn (p. 212)**

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. 212)**

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. 212)**

The condition under which to stop the training job.

Type: StoppingCondition (p. 273) object

**TrainingEndTime (p. 212)**

A timestamp that indicates when model training ended.

Type: Timestamp

**TrainingJobArn (p. 212)**

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

Type: String

Length Constraints: Maximum length of 256.

Pattern: arn:aws:sagemaker:[\p{Alnum}\-]*:[0-9]{12}:training-job/.*
**TrainingJobName (p. 212)**

Name of the model training job.

Type: String


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

**TrainingJobStatus (p. 212)**

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. 212)**

A timestamp that indicates when training started.

Type: Timestamp

**Errors**

For information about the errors that are common to all actions, see Common Errors (p. 276).

**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
ListEndpointConfigs
Service: Amazon SageMaker Service

Lists endpoint configurations.

Request Syntax

```
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 217)

A filter that returns only endpoint configurations created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 217)

A filter that returns only endpoint configurations created before the specified time (timestamp).

Type: Timestamp

Required: No

MaxResults (p. 217)

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. 217)

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. 217)

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. 217)

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime

Required: No

SortOrder (p. 217)

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. 218)

An array of endpoint configurations.

Type: Array of EndpointConfigSummary (p. 257) objects

NextToken (p. 218)

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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListEndpoints
Service: Amazon SageMaker Service

Lists endpoints.

Request Syntax

```json
{
   "CreationTimeAfter": number,
   "CreationTimeBefore": number,
   "LastModifiedTimeAfter": number,
   "LastModifiedTimeBefore": number,
   "MaxResults": number,
   "NameContains": "string",
   "NextToken": "string",
   "SortBy": "string",
   "SortOrder": "string",
   "StatusEquals": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 220)**

A filter that returns only endpoints that were created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 220)**

A filter that returns only endpoints that were created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter (p. 220)**

A filter that returns only endpoints that were modified after the specified timestamp.

Type: Timestamp

Required: No

**LastModifiedTimeBefore (p. 220)**

A filter that returns only endpoints that were modified before the specified timestamp.

Type: Timestamp

Required: No

**MaxResults (p. 220)**

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. 220)

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. 220)

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. 220)

Sorts the list of results. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime | Status

Required: No
SortOrder (p. 220)

The sort order for results. The default is Ascending.

Type: String
Valid Values: Ascending | Descending

Required: No
StatusEquals (p. 220)

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",
      "Name": "string",
      "Status": "string",
      "StatusUpdateReason": "string",
      "LastUpdatedTime": number,
      "LastStoppedTime": number,
      "LastErrorCause": "string",
      "Metrics": {
        "Metrics": {
          "MetricName": "string",
          "MetricValue": number,
          "Unit": "string"
        }
      }
    }
  ]
}
```
"EndpointName": "string",
"EndpointStatus": "string",
"LastModifiedTime": number

],
"NextToken": "string"
}

Response Elements

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

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

Endpoints (p. 221)

An array or endpoint objects.

Type: Array of EndpointSummary (p. 258) objects

NextToken (p. 221)

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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListModels
Service: Amazon SageMaker Service

Lists models created with the CreateModel API.

Request Syntax

```
{
  "CreationTimeAfter": number,
  "CreationTimeBefore": number,
  "MaxResults": number,
  "NameContains": "string",
  "NextToken": "string",
  "SortBy": "string",
  "SortOrder": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

**CreationTimeAfter (p. 223)**

A filter that returns only models created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore (p. 223)**

A filter that returns only models created before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults (p. 223)**

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. 223)**

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. 223)

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. 223)

Sorts the list of results. The default is CreationTime.

Type: String
Valid Values: Name | CreationTime
Required: No

SortOrder (p. 223)

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. 224)

An array of ModelSummary objects, each of which lists a model.

Type: Array of ModelSummary (p. 261) objects

NextToken (p. 224)

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. 276).

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

```
{
  "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. 277).

The request accepts the following data in JSON format.

CreationTimeAfter (p. 226)

A filter that returns only notebook instances that were created after the specified time (timestamp).

Type: Timestamp

Required: No

CreationTimeBefore (p. 226)

A filter that returns only notebook instances that were created before the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeAfter (p. 226)

A filter that returns only notebook instances that were modified after the specified time (timestamp).

Type: Timestamp

Required: No

LastModifiedTimeBefore (p. 226)

A filter that returns only notebook instances that were modified before the specified time (timestamp).

Type: Timestamp

Required: No
MaxResults (p. 226)
The maximum number of notebook instances to return.
Type: Integer
Valid Range: Minimum value of 1. Maximum value of 100.
Required: No

NameContains (p. 226)
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. 226)
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

SortBy (p. 226)
The field to sort results by. The default is Name.
Type: String
Valid Values: Name | CreationTime | Status
Required: No

SortOrder (p. 226)
The sort order for results.
Type: String
Valid Values: Ascending | Descending
Required: No

StatusEquals (p. 226)
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

```json
{
    "NextToken": "string",
    "NotebookInstances": [
        {
            "CreationTime": number,
            "InstanceType": "string",
            "LastModifiedTime": number,
            "NotebookInstanceArn": "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. 228)**

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. 228)**

An array of NotebookInstanceSummary objects, one for each notebook instance.

Type: Array of NotebookInstanceSummary (p. 262) objects

Errors

For information about the errors that are common to all actions, see [Common Errors (p. 276)](#).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V2
ListTags
Service: Amazon SageMaker Service

Returns the tags for the specified Amazon SageMaker resource.

Request Syntax

```json
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceArn": "string"
}
```

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

MaxResults (p. 229)

Maximum number of tags to return.

Type: Integer

Valid Range: Minimum value of 50.

Required: No

NextToken (p. 229)

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. 229)

The Amazon Resource Name (ARN) of the resource whose tags you want to retrieve.

Type: String

Length Constraints: Maximum length of 256.

Required: Yes

Response Syntax

```json
{
   "NextToken": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```
Response Elements

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

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

**NextToken (p. 229)**

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. 229)**

An array of Tag objects, each with a tag key and a value.

Type: Array of Tag (p. 274) 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. 276).

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. 277).

The request accepts the following data in JSON format.

**CreationTimeAfter** (p. 231)

A filter that only training jobs created after the specified time (timestamp).

Type: Timestamp

Required: No

**CreationTimeBefore** (p. 231)

A filter that returns only training jobs created before the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeAfter** (p. 231)

A filter that returns only training jobs modified after the specified time (timestamp).

Type: Timestamp

Required: No

**LastModifiedTimeBefore** (p. 231)

A filter that returns only training jobs modified before the specified time (timestamp).

Type: Timestamp

Required: No

**MaxResults** (p. 231)

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. 231)**

A string in the training job name. This filter returns only models whose name contains the specified string.

Type: String

Length Constraints: Maximum length of 63.

Pattern: [a-zA-Z0-9-]+

Required: No

**NextToken (p. 231)**

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. 231)**

The field to sort results by. The default is CreationTime.

Type: String

Valid Values: Name | CreationTime | Status

Required: No

**SortOrder (p. 231)**

The sort order for results. The default is Ascending.

Type: String

Valid Values: Ascending | Descending

Required: No

**StatusEquals (p. 231)**

A filter that retrieves only training jobs with a specific status.

Type: String

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

Required: No

**Response Syntax**

```json
{
    "NextToken": "string",
    "TrainingJobSummaries": [
        {
        
```
Response Elements

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

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

**NextToken (p. 232)**

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. 232)**

An array of `TrainingJobSummary` objects, each listing a training job.

Type: Array of `TrainingJobSummary (p. 275)` objects

Errors

For information about the errors that are common to all actions, see Common Errors (p. 276).

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. 277).

The request accepts the following data in JSON format.

NotebookInstanceName (p. 234)

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. 276).

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
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. 277).

The request accepts the following data in JSON format.

**NotebookInstanceName (p. 236)**

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. 276).

See Also

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

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
• AWS SDK for Ruby V2
StopTrainingJob
Service: Amazon SageMaker Service

Stops a training job. To stop a job, Amazon SageMaker sends the algorithm the SIGTERM signal, which delays job termination for 120 seconds. Algorithms might use this 120-second window to save the model artifacts, so the results of the training is not lost.

Training algorithms provided by Amazon SageMaker save the intermediate results of a model training job. This intermediate data is a valid model artifact. You can use the model artifacts that are saved when Amazon SageMaker stops a training job to create a model.

When it receives a StopTrainingJob request, Amazon SageMaker changes the status of the job to Stopping. After Amazon SageMaker stops the job, it sets the status to Stopped.

Request Syntax

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

Request Parameters

For information about the parameters that are common to all actions, see Common Parameters (p. 277).

The request accepts the following data in JSON format.

TrainingJobName (p. 238)

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. 276).

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. 277).

The request accepts the following data in JSON format.

**EndpointConfigName (p. 240)**

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. 240)**

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. 240)**

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. 276)](#).

**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. 277).

The request accepts the following data in JSON format.

DesiredWeightsAndCapacities (p. 242)

An object that provides new capacity and weight values for a variant.

Type: Array of DesiredWeightAndCapacity (p. 256) objects

Array Members: Minimum number of 1 item.

Required: Yes

EndpointName (p. 242)

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. 242)**

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. 276).

**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. 277).

The request accepts the following data in JSON format.

**InstanceType (p. 244)**

The Amazon ML compute instance type.

Type: String

Valid Values: ml.t2.medium | ml.m4.xlarge | ml.p2.xlarge

Required: No

**NotebookInstanceName (p. 244)**

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. 244)**

Amazon Resource Name (ARN) of the IAM role to associate with the instance.

Type: String


Pattern: ^arn:aws[a-zA-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. 276).

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. 246)
**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. 246)**

The desired MIME type of the inference in the response.

Length Constraints: Maximum length of 1024.

**ContentType (p. 246)**

The MIME type of the input data in the request body.

Length Constraints: Maximum length of 1024.

**EndpointName (p. 246)**

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. 246)**

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. 247)

The MIME type of the inference returned in the response body.

Length Constraints: Maximum length of 1024.

InvokedProductionVariant (p. 247)

Identifies the production variant that was invoked.

Length Constraints: Maximum length of 1024.

The response returns the following as the HTTP body.

Body (p. 247)

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. 276).

InternalFailure

An internal failure occurred.

HTTP Status Code: 500

ModelError

Model (owned by the customer in the container) returned an error 500.

HTTP Status Code: 424

ServiceUnavailable

The service is unavailable. Try your call again.

HTTP Status Code: 503

ValidationError

Inspect your request and try again.
HTTP Status Code: 400

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. 250)
- Channel (p. 251)
- ContainerDefinition (p. 253)
- DataSource (p. 255)
- DesiredWeightAndCapacity (p. 256)
- EndpointConfigSummary (p. 257)
- EndpointSummary (p. 258)
- ModelArtifacts (p. 260)
- ModelSummary (p. 261)
- NotebookInstanceSummary (p. 262)
- OutputDataConfig (p. 264)
- ProductionVariant (p. 265)
- ProductionVariantSummary (p. 267)
- ResourceConfig (p. 269)
- S3DataSource (p. 271)
- StoppingCondition (p. 273)
- Tag (p. 274)
- TrainingJobSummary (p. 275)

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. 250)
- Channel (p. 251)
- ContainerDefinition (p. 253)
- DataSource (p. 255)
- DesiredWeightAndCapacity (p. 256)
- EndpointConfigSummary (p. 257)
- EndpointSummary (p. 258)
- ModelArtifacts (p. 260)
- ModelSummary (p. 261)
- NotebookInstanceSummary (p. 262)
- OutputDataConfig (p. 264)
- ProductionVariant (p. 265)
- ProductionVariantSummary (p. 267)
- ResourceConfig (p. 269)
- S3DataSource (p. 271)
- StoppingCondition (p. 273)
- Tag (p. 274)
- TrainingJobSummary (p. 275)
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 Bring Your Own Algorithms.

Contents

TrainingImage

The registry path of the Docker image that contains the training algorithm. For information about using your own algorithms, see Docker Registry Paths for Algorithms Provided by Amazon SageMaker.

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
**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. 255) 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\-]$

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_]\d[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
**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](#) 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

EndpointConfigName
The name of the endpoint configuration.
Type: String
Length Constraints: Maximum length of 63.
Pattern: ^[a-zA-Z0-9-]*[^a-zA-Z0-9]*$  
Required: Yes

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

- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for Ruby V2
**EndpointSummary**  
Service: Amazon SageMaker Service

Provides summary information for an endpoint.

**Contents**

**CreationTime**

A timestamp that shows when the endpoint was created.

Type: Timestamp  
Required: Yes

**EndpointArn**

The Amazon Resource Name (ARN) of the endpoint.

Type: String  
Required: Yes

**EndpointName**

The name of the endpoint.

Type: String  
Length Constraints: Maximum length of 63.  
Pattern: ^[a-zA-Z0-9](-*[a-zA-Z0-9])*  
Required: Yes

**EndpointStatus**

The status of the endpoint.

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
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
**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.m4.xlarge | ml.p2.xlarge

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

**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
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 the configuration of the output S3 bucket requires server-side encryption for objects, and you don’t provide the KMS key ID, Amazon SageMaker uses the default service key. 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
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.c4.2xlarge | ml.c4.8xlarge | ml.c4.xlarge | ml.c5.2xlarge | ml.c5.9xlarge | ml.c5.xlarge | ml.m4.xlarge | ml.p2.xlarge | ml.p3.2xlarge | ml.t2.medium
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 `UpdateWeightAndCapacities` 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 `UpdateWeightAndCapacities` request.  
Type: Integer  
Valid Range: Minimum value of 1.  
Required: No

**DesiredWeight**  
The requested weight, as specified in the `UpdateWeightAndCapacities` 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.m4.4xlarge | ml.m4.10xlarge | ml.m4.xlarge
| ml.m4.2xlarge | ml.m4.8xlarge | ml.m4.xlarge
| ml.p2.16xlarge | ml.p2.8xlarge | ml.p2.16xlarge
| ml.p2.16xlarge | ml.p3.2xlarge | ml.p3.8xlarge | ml.p3.16xlarge
| ml.c5.xlarge | ml.c5.2xlarge | ml.c5.4xlarge | ml.c5.9xlarge | ml.c5.18xlarge

Required: Yes

VolumeKmsKeyId

The 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
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 1/n of the number of S3 objects. In this case, model training on each machine uses only the subset of training data.

Don't choose more ML compute instances for training than available S3 objects. If you do, some nodes won't get any data and you will pay for nodes that aren't getting any training data. This applies in both FILE and PIPE modes. Keep this in mind when developing algorithms.

In distributed training, where you use multiple ML compute EC2 instances, you might choose ShardedByS3Key. If the algorithm requires copying training data to the ML storage volume (when TrainingInputMode is set to File), this copies 1/n of the number of objects.

Type: String

Valid Values: FullyReplicated | ShardedByS3Key

Required: No

S3DataType

If you choose S3Prefix, S3Uri identifies a key name prefix. Amazon SageMaker uses all objects 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
**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:sagemaker:[\p{Alnum}\-]*:[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

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
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

**X-Amz-Date**

The date that is used to create the signature. The format must be ISO 8601 basic format (YYYYMMDD'T'HHMMSS'Z'). For example, the following date time is a valid X-Amz-Date value: 20120325T120000Z.

Condition: X-Amz-Date is optional for all requests; it can be used to override the date used for signing requests. If the Date header is specified in the ISO 8601 basic format, X-Amz-Date is not required. When X-Amz-Date is used, it always overrides the value of the Date header. For more information, see Handling Dates in Signature Version 4 in the Amazon Web Services General Reference.

Type: string

Required: Conditional
X-Amz-Security-Token

The temporary security token that was obtained through a call to AWS Security Token Service (AWS STS). For a list of services that support temporary security credentials from AWS Security Token Service, go to AWS Services That Work with IAM in the IAM User Guide.

Condition: If you're using temporary security credentials from the AWS Security Token Service, you must include the security token.

Type: string

Required: Conditional

X-Amz-Signature

Specifies the hex-encoded signature that was calculated from the string to sign and the derived signing key.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional

X-Amz-SignedHeaders

Specifies all the HTTP headers that were included as part of the canonical request. For more information about specifying signed headers, see Task 1: Create a Canonical Request For Signature Version 4 in the Amazon Web Services General Reference.

Condition: Specify this parameter when you include authentication information in a query string instead of in the HTTP authorization header.

Type: string

Required: Conditional
# Document History for Amazon SageMaker

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 <a href="#">Amazon SageMaker Developer Guide</a>.</td>
<td>November 29, 2017</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.