
Amazon SageMaker

Developer Guide



Amazon SageMaker: Developer Guide

Copyright © 2020 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
Amazon SageMaker Features	1
Amazon SageMaker Pricing	2
Are You a First-time User of Amazon SageMaker?	2
How It Works	3
Machine Learning with Amazon SageMaker	3
How It Works: Next Topic	5
Explore, Analyze, and Process Data	5
How It Works: Next Topic	5
Model Training	5
How It Works: Next Topic	8
Model Deployment	8
Hosting Services	8
Batch Transform	11
Validating Models	12
Model Monitoring	14
Set Up Amazon SageMaker	15
Create an AWS Account	15
Create an IAM Administrator User and Group	15
Onboard to Studio	16
Onboard Using Quick Start	17
Onboard Using SSO	18
Onboard Using IAM	19
Delete a Domain	20
Get Started with Amazon SageMaker	22
Get Started with Studio	22
Studio Tour	22
UI Overview	35
Features	38
Get Started with the Console	39
Step 1: Create an Amazon S3 Bucket	40
Step 2: Create an Amazon SageMaker Notebook Instance	41
Step 3: Create a Jupyter Notebook	42
Step 4: Download, Explore, and Transform Data	42
Step 5: Train a Model	45
Step 6: Deploy the Model	49
Step 7: Validate the Model	54
Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications	58
Step 9: Clean Up	58
Use Autopilot to Automate Model Development	60
Get Started	60
Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio	60
SageMaker Autopilot Notebook Output	61
Candidate Generation Notebook	61
Data Exploration Notebook	62
SageMaker Autopilot Problem Types	62
Linear regression	62
Binary classification	62
Multi-class classification	62
Automatic problem type detection	63
Prepare and Label Data	64
Use Ground Truth for Labeling	64
Are You a First-time User of Ground Truth?	65
Getting started	65

Built in Task Types	69
Auto-Segmentation Tool	86
Data Labeling	89
Use Input and Output Data	99
Creating Instruction Pages	112
Creating Custom Labeling Workflows	114
Monitor Labeling Job Status	135
Use Augmented AI for Human Review Loops	137
What is Amazon Augmented AI?	137
Get Started with Amazon Augmented AI	138
Use Task Types	140
Create a Flow Definition	143
Create and Start a Human Loop	159
Create a Worker UI	161
Monitor and Manage Your Human Loop	170
Permissions and Security	171
CloudWatch Events	175
API References	176
Create and Manage Workforces	177
Using the Amazon Mechanical Turk Workforce	177
Managing Vendor Workforces	178
Use a Private Workforce	179
HTML Elements Reference	189
Amazon SageMaker crowd-elements	189
Augmented AI crowd-elements	232
Process Data and Evaluate Models	240
Sample Notebooks	240
CloudWatch Logs and Metrics	241
Data Processing and Model Evaluation (Scikit-Learn)	241
Use Your Own Processing Code	241
Run Scripts with a Processing Container	242
Build Your Own Processing Container	242
Build Models	247
Use Amazon SageMaker Notebooks	247
Compare Notebook Features	248
Get Started	248
Create a Notebook	249
Run and Manage Notebooks	250
Share Notebooks	250
Use Notebook Instances	252
Create a Notebook Instance	252
Access Notebook Instances	255
Customize a Notebook Instance Using a Lifecycle Configuration Script	256
Control an Amazon EMR Spark Instance Using a Notebook	258
Use Example Notebooks	260
Notebook Instance Software Updates	261
Set the Notebook Kernel	262
Install External Libraries and Kernels in Notebook Instances	262
Associate Git Repositories with Amazon SageMaker Notebook Instances	264
Get Notebook Instance Metadata	271
Monitor Jupyter Logs in Amazon CloudWatch Logs	271
Choose an Algorithm	272
Use Built-in Algorithms	272
Use Machine Learning Frameworks	466
Use Your Own Algorithms or Models	478
Buy and Sell Algorithms and Models	519
Topics	519

Amazon SageMaker Algorithms	519
Amazon SageMaker Model Packages	519
Sell Amazon SageMaker Algorithms and Model Packages	519
Find and Subscribe to Algorithms and Model Packages on AWS Marketplace	522
Train Models	524
Manage Machine Learning Experiments	524
Organize	525
Track	525
Compare and Evaluate	526
Autopilot	526
Track and Evaluate an Experiment	526
Search	531
Debugger	536
Debugger Sample Notebooks	537
Supported Algorithms and Frameworks	537
Debugger CloudWatch Metrics	539
How It Works	539
Save Tensor Data	541
Prebuilt Docker Images for Rules	543
Built-in Rules	545
Use Built-in Rules	568
Programming Model	569
Use Custom Rules	570
Visualize Model Analysis	572
Reference	573
Perform Automatic Model Tuning	575
How Hyperparameter Tuning Works	576
Define Metrics	577
Define Hyperparameter Ranges	578
Example: Hyperparameter Tuning Job	580
Stop Training Jobs Early	588
Run a Warm Start Hyperparameter Tuning Job	590
Automatic Model Tuning Resource Limits	593
Best Practices for Hyperparameter Tuning	594
Tune Multiple Algorithms	595
Get Started	595
Managing Hyperparameter Tuning Jobs	596
Create a new single or multi-algorithm HPO tuning job	596
Use Reinforcement Learning	598
Why is Reinforcement Learning Important?	598
Markov Decision Process (MDP)	599
Key Features of Amazon SageMaker RL	599
Sample RL Workflow Using Amazon SageMaker RL	601
RL Environments in Amazon SageMaker	602
Distributed Training with Amazon SageMaker RL	603
Hyperparameter Tuning with Amazon SageMaker RL	604
Train a deep graph network	604
What Is a Deep Graph Network?	604
Get Started	605
Run a Graph Network Training Example	605
Monitor and Analyze Using Metrics	606
Sample Notebooks	607
Defining Training Metrics	607
Monitoring Training Job Metrics (Console)	609
Monitoring Training Job Metrics (Amazon SageMaker Console)	609
Example: Viewing a Training and Validation Curve	612
Incremental Training	613

Perform Incremental Training (Console)	614
Perform Incremental Training (API)	616
Managed Spot Training	618
Using Managed Spot Training	618
Managed Spot Training Lifecycle	619
Use Checkpoints	619
Use Augmented Manifest Files	619
Augmented Manifest File format	620
Augmented Manifest File format	620
Use an Augmented Manifest File (Console)	621
Use an Augmented Manifest File (API)	622
Deploy Models	624
Prerequisites	624
What do you want to do?	624
Manage Model Deployments	624
Deploy Your Own Inference Code	625
Guide to Amazon SageMaker	625
Multi-Model Endpoints	625
Sample Notebooks	626
.....	626
Security	627
Instance Recommendations for Multi-Model Endpoint Deployments	627
CloudWatch Metrics for Multi-Model Endpoint Deployments	628
Create a Multi-Model Endpoint	628
Bring Your Own Container	631
Invoke a Multi-Model Endpoint	634
Add or Remove Models	634
Model Monitor	635
How It Works	635
Capture Data	636
Create a Baseline	638
Schedule Monitoring Jobs	639
Interpret Results	642
Advanced Topics	648
Inference Pipelines	661
Sample Notebooks	661
Process Features with Spark ML and Scikit-learn	662
Create a Pipeline Model	662
Real-time Inference	665
Batch Transform	667
Logs and Metrics	668
Troubleshooting	673
Use Batch Transform	675
Use Batch Transform to Get Inferences from Large Datasets	675
Speed up a Batch Transform Job	677
Use Batch Transform to Test Production Variants	677
Batch Transform Errors	677
Sample Notebooks	677
Associate Prediction Results with Input	678
Compile and Deploy Models with Neo	682
Sample Notebooks	683
Compile Models	684
Deploy Models	688
Request Inferences	696
Troubleshoot Errors	696
Elastic Inference	702
How EI Works	703

Choose an EI Accelerator Type	703
Use EI in a Amazon SageMaker Notebook Instance	704
Use EI on a Hosted Endpoint	704
Frameworks that Support EI	704
Use EI with Amazon SageMaker Built-in Algorithms	705
EI Sample Notebooks	705
Set Up to Use EI	705
Attaching EI to a Notebook Instance	708
Endpoints with Elastic Inference	710
Automatically Scale Models	713
Automatic Scaling Components	713
Before You Begin	716
Related Topics	716
Configure Automatic Scaling for a Production Variant	716
Edit a Scaling Policy	722
Delete a Scaling Policy	723
Update Endpoints that Use Automatic Scaling	724
Load Testing	725
Additional Considerations	726
Troubleshoot Deployments	727
CPU Detection Errors with a JVM	727
Deployment Best Practices	728
Deploy Multiple Instances	728
Host Instance Storage Volumes	729
Monitoring	731
Monitoring with CloudWatch	731
Logging with CloudWatch	738
Log Amazon SageMaker API Calls with AWS CloudTrail	739
Amazon SageMaker Information in CloudTrail	740
Operations Performed by Automatic Model Tuning	740
Understanding Amazon SageMaker Log File Entries	741
React to Amazon SageMaker Job Status Changes with CloudWatch Events	742
Security	743
Data Protection	743
Protect Data at Rest Using Encryption	744
Protecting Data in Transit with Encryption	745
Key Management	747
Internetwork Traffic Privacy	747
Identity and Access Management	747
Audience	748
Authenticating with Identities	748
Managing Access Using Policies	750
How Amazon SageMaker Works with IAM	751
Identity-Based Policy Examples	754
Amazon SageMaker Roles	776
AWS Managed (Predefined) Policies for Amazon SageMaker	790
Amazon SageMaker API Permissions Reference	790
Troubleshooting	795
Logging and Monitoring	797
Compliance Validation	797
Resilience	798
Infrastructure Security	798
Connect a Notebook Instance to Resources in a VPC	798
Training and Inference Containers Run in Internet-Free Mode	799
Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security	800
Vulnerabilities	800
Connect to Amazon SageMaker Through a VPC Interface Endpoint	800

Give Amazon SageMaker Processing Jobs Access to Resources in Your Amazon VPC	805
Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC	808
Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC	811
Give Batch Transform Jobs Access to Resources in Your Amazon VPC	814
Limits and Supported Regions	818
Supported Instance Types	818
US East (Ohio) us-east-2	819
US East (N. Virginia) us-east-1	821
US West (N. California) us-west-1	824
US West (Oregon) us-west-2	826
Asia Pacific (Hong Kong) ap-east-1	828
Asia Pacific (Mumbai) ap-south-1	830
Asia Pacific (Seoul) ap-northeast-2	833
Asia Pacific (Singapore) ap-southeast-1	835
Asia Pacific (Sydney) ap-southeast-2	837
Asia Pacific (Tokyo) ap-northeast-1	839
Canada (Central) ca-central-1	841
EU (Frankfurt) eu-central-1	844
EU (Ireland) eu-west-1	846
EU (London) eu-west-2	848
EU (Paris) eu-west-3	850
EU (Stockholm) eu-north-1	852
Middle East (Bahrain) me-south-1	854
South America (Sao Paulo) sa-east-1	856
AWS GovCloud (US-Gov-West) us-gov-west-1	859
API and SDK Reference	862
Overview	862
Programming Model for Amazon SageMaker	862
Document History	864
AWS Glossary	866

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.

Topics

- [Amazon SageMaker Features \(p. 1\)](#)
- [Amazon SageMaker Pricing \(p. 2\)](#)
- [Are You a First-time User of Amazon SageMaker? \(p. 2\)](#)

Amazon SageMaker Features

Amazon SageMaker includes the following features:

[Amazon SageMaker Studio](#)

An integrated machine learning environment where you can build, train, deploy, and analyze your models all in the same application.

[Amazon SageMaker Ground Truth](#)

High-quality training datasets by using workers along with machine learning to create labeled datasets.

[Amazon Augmented AI](#)

Human-in-the-loop reviews

[Amazon SageMaker Studio Notebooks](#)

The next generation of Amazon SageMaker notebooks that include SSO integration, fast start-up times, and single-click sharing.

[Preprocessing](#)

Analyze and pre-process data, tackle feature engineering, and evaluate models.

[Amazon SageMaker Experiments](#)

Experiment management and tracking. You can use the tracked data to reconstruct an experiment, incrementally build on experiments conducted by peers, and trace model lineage for compliance and audit verifications.

[Amazon SageMaker Debugger](#)

Inspect training parameters and data throughout the training process. Automatically detect and alert users to commonly occurring errors such as parameter values getting too large or small.

Amazon SageMaker Autopilot

Users without machine learning knowledge can quickly build classification and regression models.

Reinforcement Learning

Maximize the long-term reward that an agent receives as a result of its actions.

Batch Transform

Preprocess datasets, run inference when you don't need a persistent endpoint, and associate input records with inferences to assist the interpretation of results.

Amazon SageMaker Model Monitor

Monitor and analyze models in production (endpoints) to detect data drift and deviations in model quality.

Amazon SageMaker Neo

Train machine learning models once, then run anywhere in the cloud and at the edge.

Amazon SageMaker Elastic Inference

Speed up the throughput and decrease the latency of getting real-time inferences.

Amazon SageMaker Pricing

As with other AWS products, there are no contracts or minimum commitments for using Amazon SageMaker. For more information about the cost of using Amazon SageMaker, see [Amazon SageMaker Pricing](#).

Are You a First-time User of Amazon SageMaker?

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

1. **Read How Amazon SageMaker Works (p. 3)** – This section provides an overview of Amazon SageMaker, explains key concepts, and describes the core components involved in building AI solutions with Amazon SageMaker. We recommend that you read this topic in the order presented.
2. **Read Get Started with Amazon SageMaker (p. 22)** – 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 [Get Started with Amazon SageMaker \(p. 22\)](#).
4. **Explore other topics** – Depending on your needs, do the following:
 - **Submit Python code to train with deep learning frameworks** – In Amazon SageMaker, you can use your own training scripts to train models. For information, see [Use Machine Learning Frameworks with Amazon SageMaker \(p. 466\)](#).
 - **Use Amazon SageMaker directly from Apache Spark** – For information, see [Use Apache Spark with Amazon SageMaker \(p. 466\)](#).
 - **Use Amazon AI to train and/or deploy your own custom algorithms** – Package your custom algorithms with Docker so you can train and/or deploy them in Amazon SageMaker. See [Use Your Own Algorithms or Models with Amazon SageMaker \(p. 478\)](#) to learn how Amazon SageMaker interacts with Docker containers, and for the Amazon SageMaker requirements for Docker images.
5. **See the Reference** – This section describes the Amazon SageMaker API operations.

How Amazon SageMaker Works

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

Topics

- [Machine Learning with Amazon SageMaker \(p. 3\)](#)
- [Explore, Analyze, and Process Data \(p. 5\)](#)
- [Train a Model with Amazon SageMaker \(p. 5\)](#)
- [Deploy a Model in Amazon SageMaker \(p. 8\)](#)
- [Monitoring a Model in Production \(p. 14\)](#)

How It Works: Next Topic

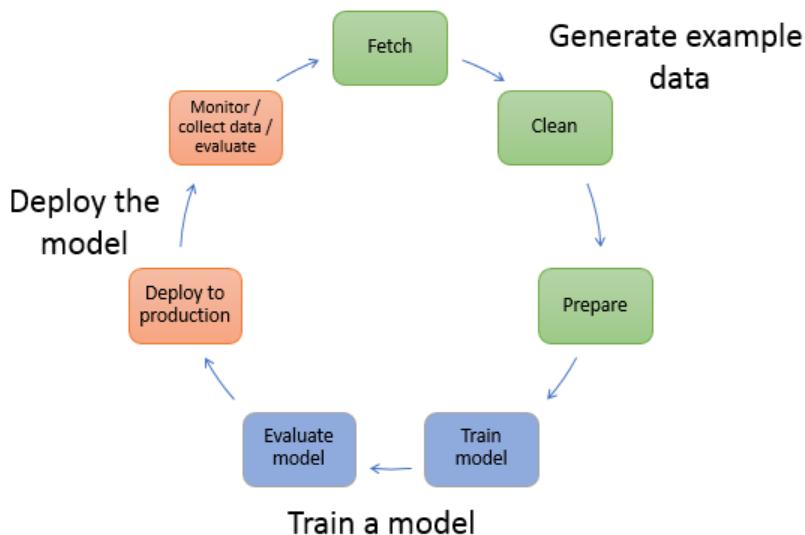
[Machine Learning with Amazon SageMaker \(p. 3\)](#)

Machine Learning with Amazon SageMaker

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

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

The following diagram illustrates the typical workflow for creating a machine learning model:



As the diagram illustrates, you typically perform the following activities:

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

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

- a. **Fetch the data**— You might have in-house example data repositories, or you might use datasets that are publicly available. Typically, you pull the dataset or datasets into a single repository.
- b. **Clean the data**—To improve model training, inspect the data and clean it as needed. For example, if your data has a country name attribute with values United States and US, you might want to edit the data to be consistent.
- c. **Prepare or transform the data**—To improve performance, you might perform additional data transformations. For example, you might choose to combine attributes. If your model predicts the conditions that require de-icing an aircraft, instead of using temperature and humidity attributes separately, you might combine those attributes into a new attribute to get a better model.

In Amazon SageMaker, you preprocess example data in a Jupyter notebook on your notebook instance. You use your notebook to fetch your dataset, explore it, and prepare it for model training. For more information, see [Explore, Analyze, and Process Data \(p. 5\)](#). For more information about preparing data in AWS Marketplace, see [data preparation](#).

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

- **Training the model**— To train a model, you need an algorithm. The algorithm you choose depends on a number of factors. For a quick, out-of-the-box solution, you might be able to use one of the algorithms that Amazon SageMaker provides. For a list of algorithms provided by Amazon SageMaker and related considerations, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).

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 general-purpose instance to a distributed cluster of GPU instances. For more information, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

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

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

3. **Deploy the model**— You traditionally re-engineer a model before you integrate it with your application and deploy it. With Amazon SageMaker hosting services, you can deploy your model independently, decoupling it from your application code. For more information, see [Deploy a Model on Amazon SageMaker Hosting Services \(p. 8\)](#).

Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, 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. You do this by retraining the model with the new dataset. As more and more example data becomes available, you continue retraining your model to increase accuracy.

How It Works: Next Topic

[Explore, Analyze, and Process Data \(p. 5\)](#)

Explore, Analyze, and Process Data

Before using a dataset to train a model, data scientists typically explore, analyze, 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 4.3: Transform the Training Dataset and Upload It to Amazon S3 \(p. 44\)](#).

To preprocess data use one of the following methods:

- Use a Jupyter notebook on an Amazon SageMaker notebook instance. You can also use the notebook instance to do the following:
 - Write code to create model training jobs
 - Deploy models to Amazon SageMaker hosting
 - Test or validate your models

For more information, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#)

- You can use a model to transform data by using Amazon SageMaker batch transform. For more information, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Amazon SageMaker Processing enables running jobs to preprocess and postprocess data, perform feature engineering, and evaluate models on Amazon SageMaker easily and at scale. When combined with the other critical machine learning tasks provided by Amazon SageMaker, such as training and hosting, Processing provides you with the benefits of a fully managed machine learning environment, including all the security and compliance support built into Amazon SageMaker. With Processing, you have the flexibility to use the built-in data processing containers or to bring your own containers and submit custom jobs to run on managed infrastructure. After you submit a job, Amazon SageMaker launches the compute instances, processes and analyzes the input data, and releases the resources upon completion. For more information, see [Process Data and Evaluate Models \(p. 240\)](#).

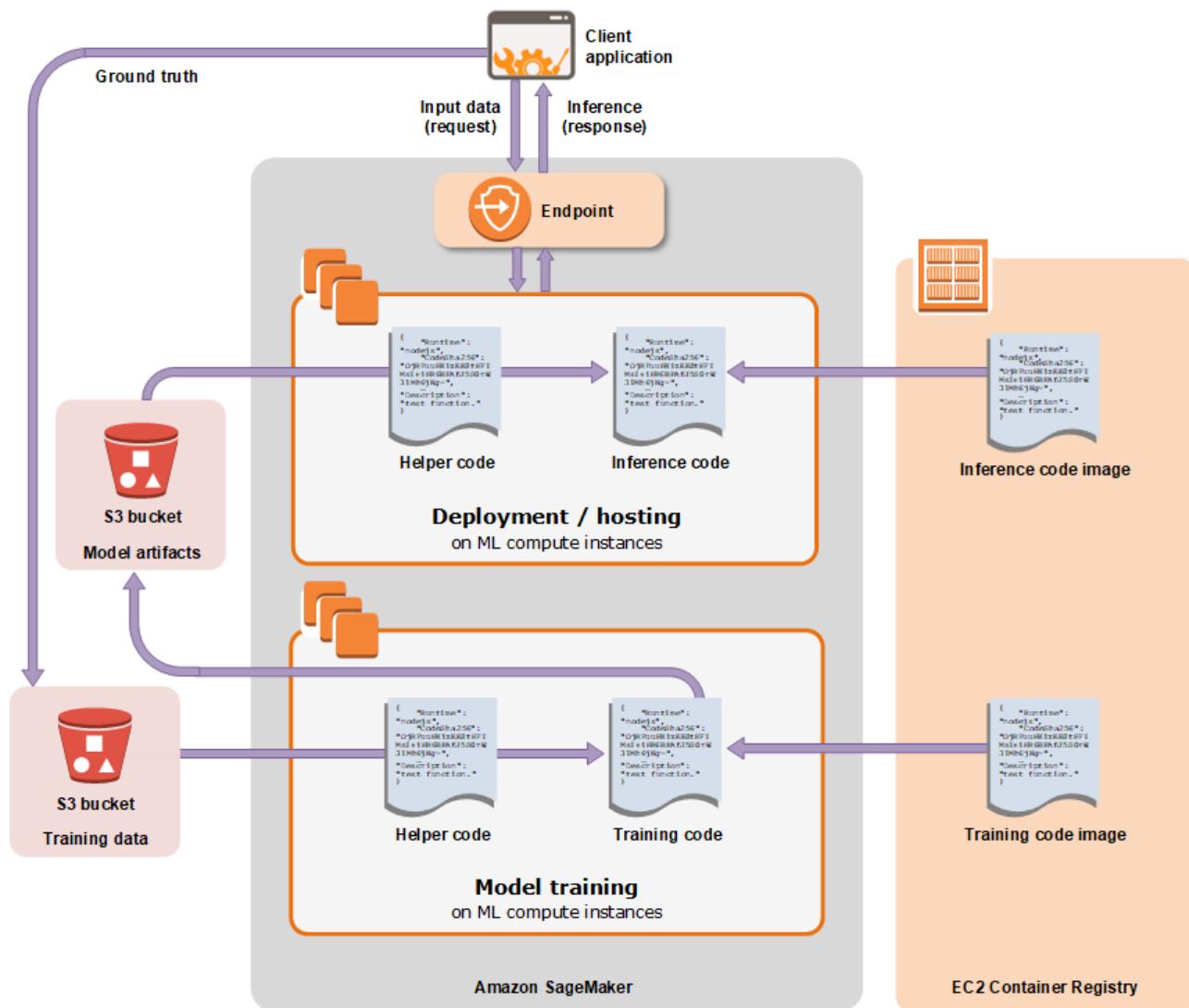
- For information about how to run your own data processing scripts, [Data Processing and Model Evaluation with Scikit-Learn \(p. 241\)](#).
- For information about how to build your own processing container to run scripts, see [Build Your Own Processing Container \(p. 242\)](#).

How It Works: Next Topic

[Train a Model with Amazon SageMaker \(p. 5\)](#)

Train a Model with Amazon SageMaker

The following diagram shows how you train and deploy a model with Amazon SageMaker:



The area labeled Amazon SageMaker highlights the two components of Amazon SageMaker: model training and model deployment.

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

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The compute resources that you want Amazon SageMaker to use for model training. Compute resources are ML compute instances that are managed by Amazon SageMaker.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see [Common Parameters for Built-In Algorithms \(p. 274\)](#).

You have the following options for a training algorithm:

- **Use an algorithm provided by Amazon SageMaker**—Amazon SageMaker provides training algorithms. If one of these meets your needs, it's a great out-of-the-box solution for quick model

training. For a list of algorithms provided by Amazon SageMaker, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#). To try an exercise that uses an algorithm provided by Amazon SageMaker, see [Get Started with Amazon SageMaker \(p. 22\)](#).

- **Use Amazon SageMaker Debugger**—to inspect training parameters and data throughout the training process when working with the TensorFlow, PyTorch, and Apache MXNet learning frameworks or the XGBoost algorithm. Debugger automatically detects and alerts users to commonly occurring errors such as parameter values getting too large or small. For more information about using Debugger, see [Amazon SageMaker Debugger \(p. 536\)](#). Debugger sample notebooks are available at [Amazon SageMaker Debugger Samples](#).
- **Use Apache Spark with Amazon SageMaker**—Amazon SageMaker provides a library that you can use in Apache Spark to train models with Amazon SageMaker. Using the library provided by Amazon SageMaker is similar to using Apache Spark MLLib. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 466\)](#).
- **Submit custom code to train with deep learning frameworks**—You can submit custom Python code that uses TensorFlow or Apache MXNet for model training. For more information, see [Use TensorFlow with Amazon SageMaker \(p. 475\)](#) and [Use Apache MXNet with Amazon SageMaker \(p. 476\)](#).
- **Use your own custom algorithms**—Put your code together as a Docker image and specify the registry path of the image in an Amazon SageMaker CreateTrainingJob API call. For more information, see [Use Your Own Algorithms or Models with Amazon SageMaker \(p. 478\)](#).
- **Use an algorithm that you subscribe to from AWS Marketplace**—For information, see [Find and Subscribe to Algorithms and Model Packages on AWS Marketplace \(p. 522\)](#).

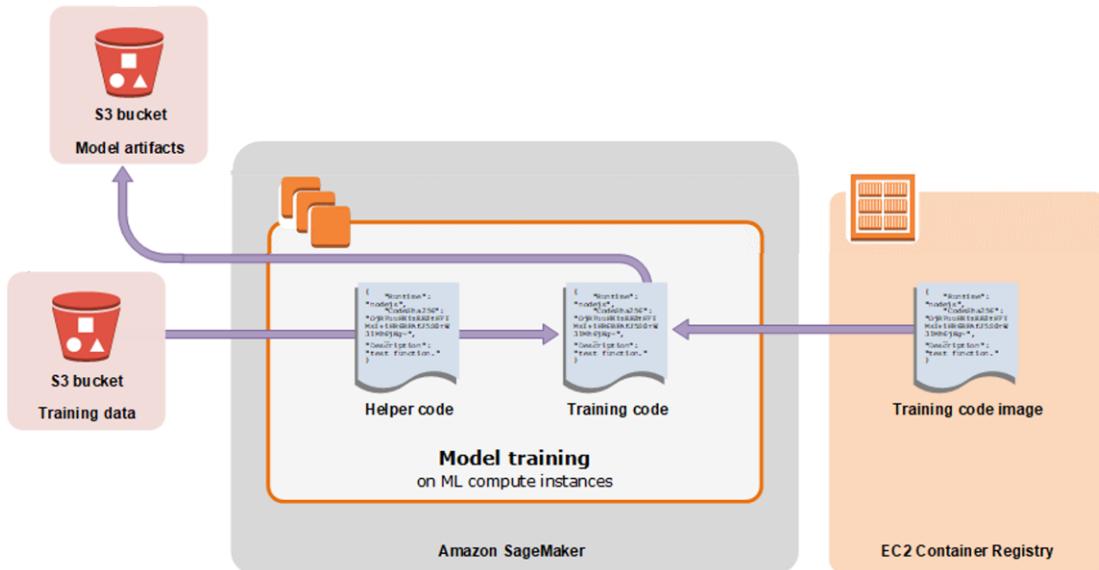
After you create the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

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 API](#).

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

Important

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



How It Works: Next Topic

[Deploy a Model in Amazon SageMaker \(p. 8\)](#)

Deploy a Model in Amazon SageMaker

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

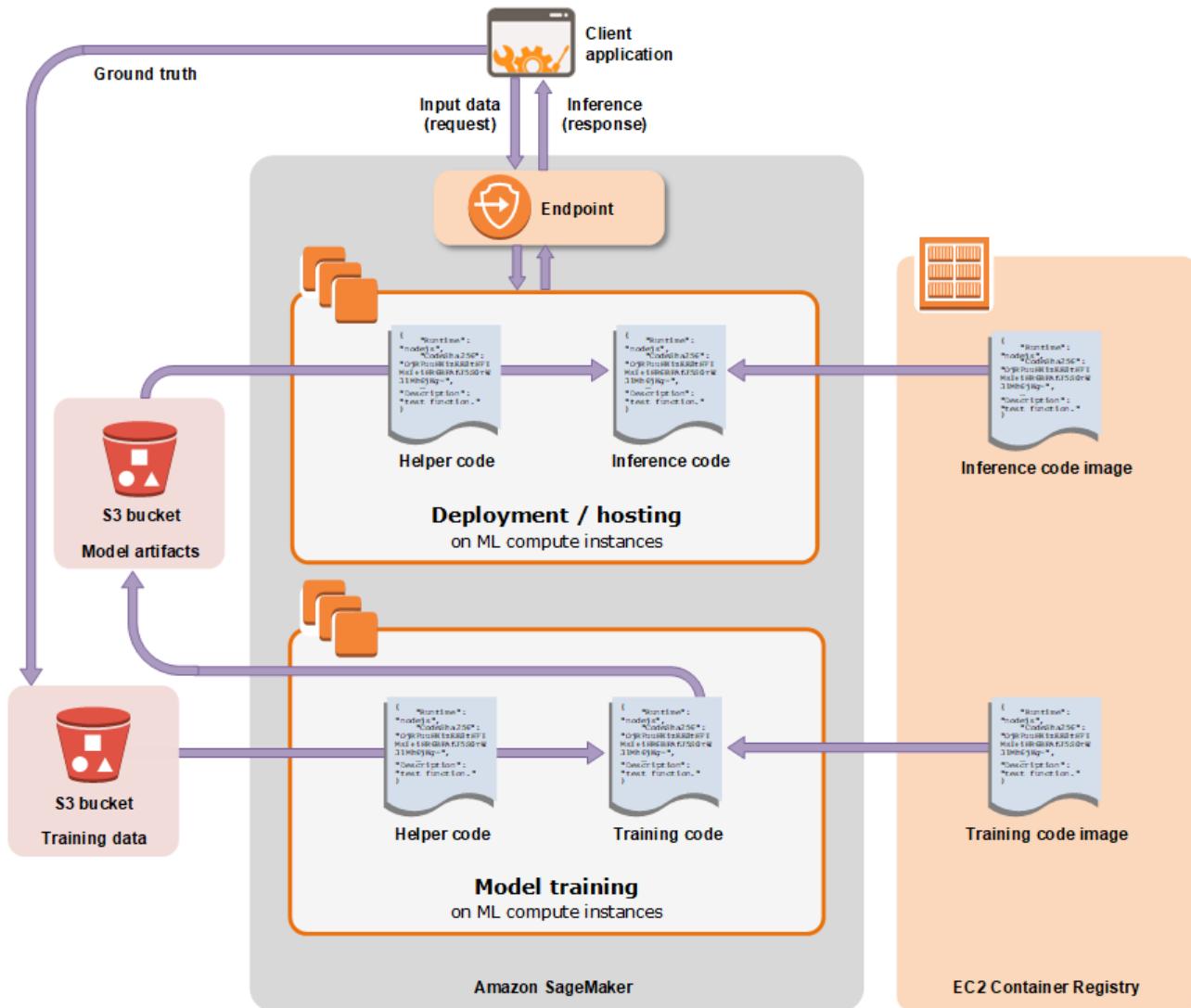
- To set up a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

- [Deploy a Model on Amazon SageMaker Hosting Services \(p. 8\)](#)
- [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#)
- [Validate a Machine Learning Model \(p. 12\)](#)

Deploy a Model on Amazon SageMaker Hosting Services

Amazon SageMaker also provides model hosting services for model deployment, as shown in the following diagram. Amazon SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences.



Deploying a model using Amazon SageMaker hosting services is a three-step process:

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

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

Note

Endpoints are scoped to an individual AWS account, and are not public. The URL does not contain the account ID, but Amazon SageMaker determines the account ID from the authentication token that is supplied by the caller.

For an example of how to use Amazon API Gateway and AWS Lambda to set up and deploy a web service that you can call from a client application that is not within the scope of your account, see [Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda](#) in the *AWS Machine Learning Blog*.

Note

When you create an endpoint, Amazon SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. For a list of instance types that Amazon SageMaker hosting service supports, see [AWS Service Limits](#). For a list of the sizes of the storage volumes that Amazon SageMaker attaches to each instance, see [Host Instance Storage Volumes \(p. 729\)](#).

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

Best Practices for Deploying Models on Amazon SageMaker Hosting Services

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

- Typically, a client application sends requests to the Amazon SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.
- You can deploy a model trained with Amazon SageMaker to your own deployment target. To do that, you need to know the algorithm-specific format of the model artifacts that were generated by model training. For more information about output formats, see the section corresponding to the algorithm you are using in [Training Data Formats \(p. 281\)](#).
- 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](#).
- You can configure a `ProductionVariant` to use Application Auto Scaling. For information about configuring automatic scaling, see [Automatically Scale Amazon SageMaker Models \(p. 713\)](#).
- 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](#) and [UpdateEndpointWeightsAndCapacities](#).

- Changing or deleting model artifacts or changing inference code after deploying a model produces unpredictable results. If you need to change or delete model artifacts or change inference code, modify the endpoint by providing a new endpoint configuration. Once you provide the new endpoint configuration, you can change or delete the model artifacts corresponding to the old endpoint configuration.
- If you want to get inferences on entire datasets, consider using batch transform as an alternative to hosting services. For information, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#)

How It Works: Next Topic

[Validate a Machine Learning Model \(p. 12\)](#)

Get Inferences for an Entire Dataset with Batch Transform

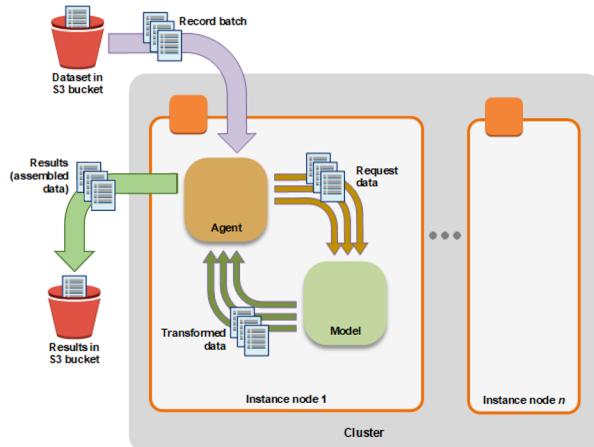
To get inferences for an entire dataset, use batch transform. With batch transform, you create a batch transform job using a trained model and the dataset, which must be stored in Amazon S3. Amazon SageMaker saves the inferences in an S3 bucket that you specify when you create the batch transform job. Batch transform manages all of the compute resources required to get inferences. This includes launching instances and deleting them after the batch transform job has completed. Batch transform manages interactions between the data and the model with an object within the instance node called an agent.

Use batch transform when you:

- Want to get inferences for an entire dataset and index them to serve inferences in real time
- Don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences
- Don't need the subsecond latency that Amazon SageMaker hosted endpoints provide

You can also use batch transform to preprocess your data before using it to train a new model or generate inferences.

The following diagram shows the workflow of a batch transform job:



To perform a batch transform, create a batch transform job using either the Amazon SageMaker console or the API. Provide the following:

- The path to the S3 bucket where you've stored the data that you want to transform.
- The compute resources that you want Amazon SageMaker to use for the transform job. *Compute resources* are machine learning (ML) compute instances that are managed by Amazon SageMaker.
- The path to the S3 bucket where you want to store the output of the job.
- The name of the Amazon SageMaker model that you want to use to create inferences. You must use a model that you have already created either with the [CreateModel](#) operation or the console.

The following is an example of what a dataset file might look like.

```
An example of input file content:
AttributeM      Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-
AttributeM      Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-
AttributeM      Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-
AttributeM      ...
AttributeM      RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-
```

A record is a single input data unit. For information about how to delimit records for batch transform jobs, see [SplitType](#).

For an example of how to use batch transform, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

How It Works: Next Topic

[Validate a Machine Learning Model \(p. 12\)](#)

Validate a Machine Learning Model

After training a model, evaluate it to determine whether its performance and accuracy allow you to achieve your business goals. You might generate multiple models using different methods and evaluate each. For example, you could apply different business rules for each model, and then apply various

measures to determine each model's suitability. You might consider whether your model needs to be more sensitive than specific (or vice versa).

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

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

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

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

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

Options for offline model evaluation include:

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

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

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

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

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

How It Works: Next Topic

[Programming Model for Amazon SageMaker \(p. 862\)](#)

Monitoring a Model in Production

After you deploy a model into your production environment, Amazon SageMaker Model Monitor can be used to continuously monitor the quality of Amazon SageMaker machine learning models. It enables developers to set alerts for when there are deviations in the model quality, such as data drift and anomalies. Early and pro-active detection of these deviations enables you to take corrective actions. For more information, see [Amazon SageMaker Model Monitor \(p. 635\)](#).

Set Up Amazon SageMaker

In this section, you sign up for an AWS account, create an IAM admin user, and onboard to Amazon SageMaker Studio.

If you're new to Amazon SageMaker, we recommend that you read [How Amazon SageMaker Works \(p. 3\)](#).

Topics

- [Create an AWS Account \(p. 15\)](#)
- [Create an IAM Administrator User and Group \(p. 15\)](#)
- [Onboard to Amazon SageMaker Studio \(p. 16\)](#)

Create an AWS Account

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

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

To create an AWS account

1. Open <https://portal.aws.amazon.com/billing/signup>.
2. Follow the online instructions.

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

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

Create an IAM Administrator User and Group

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

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

To create an administrator user

- 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 [Authenticating with Identities \(p. 748\)](#).

Onboard to Amazon SageMaker Studio

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Note

Amazon SageMaker Studio is available only in the following AWS Regions:

- US East (Ohio), us-east-2

To change the Region in the Amazon SageMaker console, use the Region selector at the top right of the page, next to your AWS account number.

To use Amazon SageMaker Studio and Amazon SageMaker Studio Notebooks, you must complete the Studio onboarding process using the Amazon SageMaker console.

When onboarding, you can choose to use either AWS Single Sign-On (AWS SSO) or AWS Identity and Access Management (IAM) for authentication methods. When you use IAM authentication, you can choose either the **Quick start** or the **Standard setup** procedure.

The simplest way to create an Amazon SageMaker Studio account is to follow the **Quick start** procedure. For more control, including the option of using AWS SSO authentication, use the **Standard setup** procedures.

AWS SSO Authentication

To use AWS SSO authentication with Amazon SageMaker Studio, you must onboard to an AWS SSO organization. The SSO organization account must be in an AWS Region supported by Studio. In addition, an AWS SSO account can't be in multiple Regions.

AWS SSO authentication provides the following benefits over IAM authentication:

- Members given access to Studio have a unique sign-on URL that directly opens Studio, and they sign-on with their SSO credentials. When you use IAM authentication, you must sign on through the Amazon SageMaker console.
- Organizations manage their members in AWS SSO instead of Studio. You can assign multiple members access to Studio at the same time. When you use IAM authentication, you must add and manage members manually one at time using the Studio Control Panel.

Note

If you onboard using IAM authentication and want to switch to AWS SSO authentication later, you must delete the domain created for you by Amazon SageMaker Studio. Then, you need to manually re-import all notebooks and other user data that you created. For more information, see [Delete an Amazon SageMaker Studio Domain \(p. 20\)](#).

Topics

- [Onboard to Amazon SageMaker Studio Using Quick Start \(p. 17\)](#)
- [Onboard to Amazon SageMaker Studio Using AWS SSO \(p. 18\)](#)

- [Onboard to Amazon SageMaker Studio Using IAM \(p. 19\)](#)
- [Delete an Amazon SageMaker Studio Domain \(p. 20\)](#)

Onboard to Amazon SageMaker Studio Using Quick Start

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Note

Amazon SageMaker Studio is available only in specific AWS Regions. To view the list of supported Regions, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

This procedure describes how to onboard to Amazon SageMaker using the **Quick start** procedure, which uses AWS Identity and Access Management (IAM) authentication. The following procedures describe how to onboard additional users and how to access Studio after you have been onboarded. For information on how to onboard using the **Standard setup** procedure, see [Onboard Using SSO \(p. 18\)](#) or [Onboard Using IAM \(p. 19\)](#).

To onboard to Studio using Quick start

1. Open the [Amazon SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio** page, under **Get started**, choose **Quick start**.
4. For **User name**, keep the default name or create a new name. The name can be up to 63 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).
5. For **Execution role**, choose an option from the role selector.

If you choose **Enter a custom IAM role ARN**, the role must have the `AmazonSageMakerFullAccess` policy attached.

If you choose **Create a new role**, the **Create an IAM role** dialog opens:

- For **S3 buckets you specify**, specify additional S3 buckets that users of your notebooks can access. If you don't want to add access to more buckets, choose **None**.
 - Choose **Create role**. Amazon SageMaker creates a new IAM role with the `AmazonSageMakerFullAccess` policy attached.
6. Choose **Submit**.

On the **Amazon SageMaker Studio Control Panel** page, under **Studio Summary**, wait for **Status** to change to **Ready**.

When **Status** is **Ready**, the user name that you specified is enabled and chosen. The **Add user** and **Delete user** buttons, and the **Open Studio** link are also enabled.

7. Choose **Open Studio**. The **Amazon SageMaker Studio** loading page displays.

When Studio opens you can start using it.

Now that you've onboarded to Amazon SageMaker Studio, use the following steps to access Studio later.

To access Studio after you onboard

1. Open the [Amazon SageMaker console](#).

2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio Control Panel** page, choose your user name and then choose **Open Studio**.

To add more users

1. On the **Amazon SageMaker Studio Control Panel**, choose **Add user**.
2. Repeat steps 4 and 5 from the first procedure, "To onboard to Studio using Quick start."
3. Choose **Submit**.

For information about using Amazon SageMaker Studio, see [Get Started with Studio \(p. 22\)](#).

Onboard to Amazon SageMaker Studio Using AWS SSO

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Note

Amazon SageMaker Studio is available only in specific AWS Regions. To view the list of supported Regions, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

This procedure describes how to onboard to Amazon SageMaker Studio using AWS SSO authentication. For information on how to onboard using AWS Identity and Access Management (IAM) authentication, see [Onboard Using Quick Start \(p. 17\)](#) or [Onboard Using IAM \(p. 19\)](#).

To onboard to Studio using AWS SSO

1. Open the [Amazon SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio** page, under **Get started**, choose **Standard setup**.
4. For **Authentication method**, choose **AWS Single Sign-On (SSO)**. A message tells you whether you have an AWS SSO account in an AWS Region supported by Amazon SageMaker Studio.
5. If you don't have an AWS SSO account in a supported Region, you must create an AWS SSO account in a supported Region before proceeding. To continue to onboard without creating a new AWS SSO account, choose the **AWS Identity and Access Management (IAM)** authentication method or the **Quick start** procedure, which also uses IAM.

For information about setting up AWS SSO for use with Studio, see [Set Up AWS SSO for Use with Amazon SageMaker Studio \(p. 19\)](#).

6. To continue with SSO, under **Permission**, for **Execution role for all users**, choose an option from the role selector.

If you choose **Create a new role**, the **Create an IAM role** dialog opens:

- For **S3 buckets you specify**, specify additional S3 buckets that users of your notebooks can access. If you don't want to add access to more buckets, choose **None**.
 - Choose **Create role**. Amazon SageMaker creates a new IAM role with the [AmazonSageMakerFullAccess](#) policy attached.
7. Choose **Submit**.

On the **Amazon SageMaker Studio Control Panel** page, under **Studio Summary**, the **Status** shows as **Pending** while Studio creates an Amazon SageMaker Studio application in your AWS SSO domain. When **Status** changes to **Ready**, the **Assign users** button is enabled.

8. Choose **Assign users**. The **Assign users** page opens and displays a list of your organization's members.
9. To assign users access to Amazon SageMaker Studio, choose the check box next to their user name and choose **Assign users**.
10. Send each assigned user the **Studio address** link shown under **Studio Summary**. Your AWS SSO users go to this address to access Studio.

To access Studio after onboarding

In your browser, go to the **Studio address**. On the first visit, wait for your user profile to be created. Then, and on subsequent visits, you only need to wait for Amazon SageMaker Studio to load.

For information about using Amazon SageMaker Studio, see [Get Started with Studio \(p. 22\)](#).

Set Up AWS SSO for Use with Amazon SageMaker Studio

To use AWS SSO authentication, you must belong to an AWS SSO organization. If you don't belong to an AWS SSO organization, you can create one with the following procedure.

To create an AWS SSO organization

1. On the Studio **Get started** page, choose the [set up an account](#) link to open the AWS Single Sign-On (AWS SSO) console.
2. Choose **Enable AWS SSO**, and then choose **Create AWS organization**.

When your organization has been created, the AWS SSO dashboard opens. AWS also sends you email to verify the email address associated with the organization.

3. To add a user to your AWS SSO organization, in the navigation pane, choose **Users**.
4. On the **Users** page, choose **Add user**. Under **User details**, specify all required fields. For **Password**, choose **Send an email to the user**.
5. Choose **Next: Groups** and then **Add user**. AWS sends an email to the user inviting them to create a password and activate their AWS SSO account.
6. To add more users, repeat steps 4 through 6.

Return to Amazon SageMaker Studio to continue to onboard using AWS SSO authentication.

Onboard to Amazon SageMaker Studio Using IAM

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Note

Amazon SageMaker Studio is available only in specific AWS Regions. To view the list of supported Regions, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

This procedure describes how to onboard to Amazon SageMaker using the standard setup process for AWS Identity and Access Management (IAM) authentication. For a procedure to onboard faster using IAM, see [Onboard Using Quick Start \(p. 17\)](#). For information about onboarding using AWS SSO authentication, see [Onboard Using SSO \(p. 18\)](#).

To onboard to Studio using IAM

1. Open the [Amazon SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio** page, under **Get started**, choose **Standard setup**.
4. For **Authentication method**, choose **AWS Identity and Access Management (IAM)**.
5. Under **Permission**, for **Execution role for all users**, choose an option from the role selector.

If you choose **Create a new role**, the **Create an IAM role** dialog opens:

- For **S3 buckets you specify**, specify additional S3 buckets that users of your notebooks can access. If you don't want to add access to more buckets, choose **None**.
 - Choose **Create role**. Amazon SageMaker creates a new IAM role with the [AmazonSageMakerFullAccess](#) policy attached.
6. Choose **Submit**.
- On the **Amazon SageMaker Studio Control Panel** page, under **Studio Summary**, wait for **Status** to change to **Ready** and the **Add user** button to be enabled.
7. Choose **Add user**.
 8. On the **Add user** page, keep the default name or create a new name. A name can be up to 63 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).
- For **Execution role**, choose an option from the role selector.
9. Choose **Submit**. The Studio Control Panel home page opens with the new user listed and chosen. The **Delete user** button and the **Open Studio** link are both enabled.
 10. To add more users, repeat steps 7 through 9.

When multiple users are listed, none of the users in the user list is chosen. Choose a user and the **Delete user** button and the **Open Studio** link for that user are enabled.

11. Choose **Open Studio**. The **Amazon SageMaker Studio** loading page displays.

When Amazon SageMaker Studio opens, you can start using Studio.

Now that you've onboarded to Amazon SageMaker Studio, use the following steps to subsequently access Studio.

Access Studio after you onboard

1. Open the [Amazon SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio Control Panel** page, choose your user name and then choose **Open Studio**.

For information about using Amazon SageMaker Studio, see [Get Started with Studio \(p. 22\)](#).

Delete an Amazon SageMaker Studio Domain

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

When you onboard to Amazon SageMaker Studio using IAM authentication, Studio creates a domain for your account. A domain consists of a list of authorized users, configuration settings, and an Amazon

Elastic File System (Amazon EFS) volume, which contains data for the users, including notebooks, resources, and artifacts.

To return Studio to the state it was in before you onboarded, you must delete this domain. This is required if you want to switch authentication modes from IAM to AWS SSO.

When a domain has been deleted, all members of the domain lose access to the Amazon EFS volume and the data it contains. The data is not deleted, but you must manually import the notebooks and other resources to reuse them in Studio.

To delete a domain, the domain cannot contain any user profiles or apps. To delete a user profile, the profile cannot contain any apps.

Note

Amazon SageMaker Studio is available only in the US East (Ohio) Region.
You must have admin permission to delete a domain.

To delete a domain

1. Retrieve the list of applications in the domain.

```
aws --region us-east-2 sagemaker list-apps \  
--domain-id-equals DomainId
```

2. Delete each application in the list.

```
aws --region us-east-2 sagemaker delete-app \  
--domain-id DomainId \  
--app-name AppName \  
--app-type AppType \  
--user-profile-name UserProfileName
```

3. Retrieve the list of user profiles in the domain.

```
aws --region us-east-2 sagemaker list-user-profiles \  
--domain-id-equals DomainId
```

4. Delete each user profile in the list.

```
aws --region us-east-2 sagemaker delete-user-profile \  
--domain-id DomainId \  
--user-profile-name UserProfileName
```

5. Delete the domain.

```
aws --region us-east-2 sagemaker delete-domain \  
--domain-id DomainId
```

Get Started with Amazon SageMaker

Topics

- [Get Started with Amazon SageMaker Studio \(p. 22\)](#)
- [Get Started with the Amazon SageMaker Console \(p. 39\)](#)

Get Started with Amazon SageMaker Studio

Amazon SageMaker Studio is a web-based, integrated development environment (IDE) for machine learning that lets you build, train, debug, deploy, and monitor your machine learning models. Studio provides all the tools you need to take your models from experimentation to production while boosting your productivity. In a single unified visual interface, customers can

- Write and execute code in Jupyter notebooks
- Build and train machine learning models
- Deploy the models and monitor the performance of their predictions
- Track and debug the machine learning experiments

The following sections provide an overview of the user interface and a description of Studio's main features.

For information on the onboarding steps to sign-on to Amazon SageMaker Studio, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

For a tutorial that demonstrates the basic features of Amazon SageMaker Studio, see [Track and Evaluate a Model Training Experiment \(p. 526\)](#).

Topics

- [Amazon SageMaker Studio Tour \(p. 22\)](#)
- [Amazon SageMaker Studio UI Overview \(p. 35\)](#)
- [Amazon SageMaker Studio Features \(p. 38\)](#)

Amazon SageMaker Studio Tour

This example shows how to use the main features of Amazon SageMaker Studio. The example is based on the https://github.com/awslabs/amazon-sagemaker-examples/blob/master/aws_sagemaker_studio/getting_started/xgboost_customer_churn_studio.ipynb sample Jupyter notebook. You must run this sample notebook in Studio.

For a series of videos that shows how to use the main features of Amazon SageMaker Studio, see [NEW! Amazon SageMaker Studio](#) on YouTube.

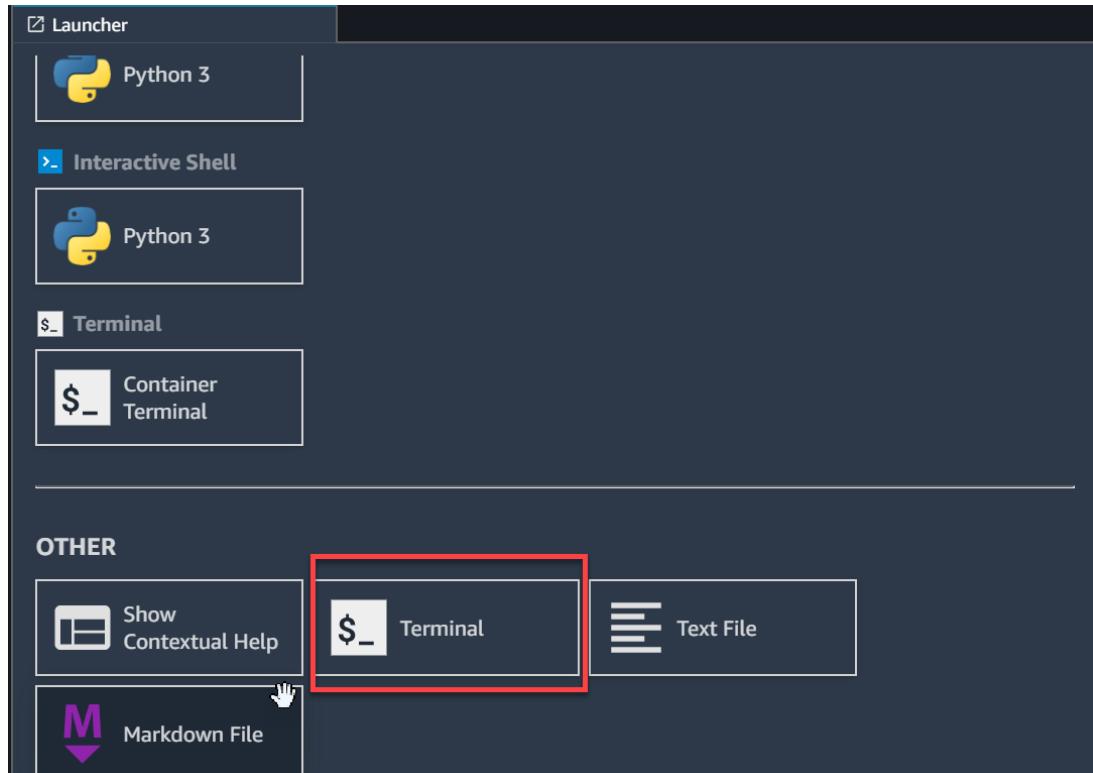
Prerequisites

To run the code for this tour, you need:

- An AWS SSO or IAM account to sign on to Studio. For information, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).
- A copy of the [awslabs/amazon-sagemaker-examples](https://github.com/awslabs/amazon-sagemaker-examples) repository in your Studio environment.

To start the tour

1. Sign in to Amazon SageMaker Studio. For instructions, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).
2. On the **Launcher** page, choose **Terminal** to launch a command terminal.



3. At the command prompt, run the following command to clone the amazon-sagemaker-examples repository.

```
git clone https://github.com/awslabs/amazon-sagemaker-examples.git
```

Topics

- [Use Amazon SageMaker Studio Notebooks \(p. 23\)](#)
- [Keep Track of Machine Learning Experiments \(p. 24\)](#)
- [Create Charts to Visualize Data \(p. 27\)](#)
- [Debug Training Jobs \(p. 30\)](#)
- [Monitor Deployed Models \(p. 31\)](#)
- [Create an Amazon SageMaker Autopilot Experiment \(p. 33\)](#)

Use Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio notebooks are collaborative Jupyter notebooks that are built into Amazon SageMaker Studio. You can launch Studio notebooks without setting up compute instances and file storage, so you can get started fast. You pay only for the resources used when you run the notebooks.

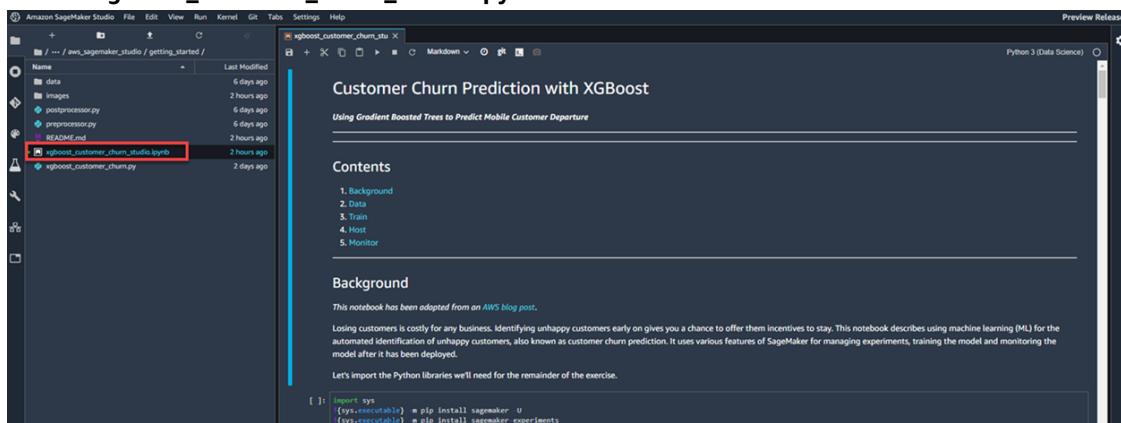
You can share notebooks with others in your organization, so that they can easily reproduce your results and collaborate while building models and exploring your data. Dependencies for your notebook are

included in the notebooks' environment settings by default, so sharing is seamless. When you share a notebook, Studio publishes a reproducible snapshot of your notebook environments. It provides your collaborators access to the notebook through a secure URL.

For more information about Amazon SageMaker Studio notebooks, see [Use Amazon SageMaker Notebooks \(p. 247\)](#).

To open the notebook

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).
2. Choose the file browser icon ().
3. Navigate to `amazon-sagemaker-examples/aws_sagemaker_studio_getting_started`.
4. Choose `xgboost_customer_churn_studio.ipynb`.



Keep Track of Machine Learning Experiments

Organize, track, compare, and evaluate your machine learning experiments with Amazon SageMaker Experiments. When you create a training job in Amazon SageMaker, you can include code that creates a *trial* for the job and associates the trial with an experiment. You can then use Amazon SageMaker Studio to organize and track your experiments. For more information about how to use Amazon SageMaker Experiments, see [Track and Evaluate a Model Training Experiment \(p. 526\)](#).

To create an experiment with training trials

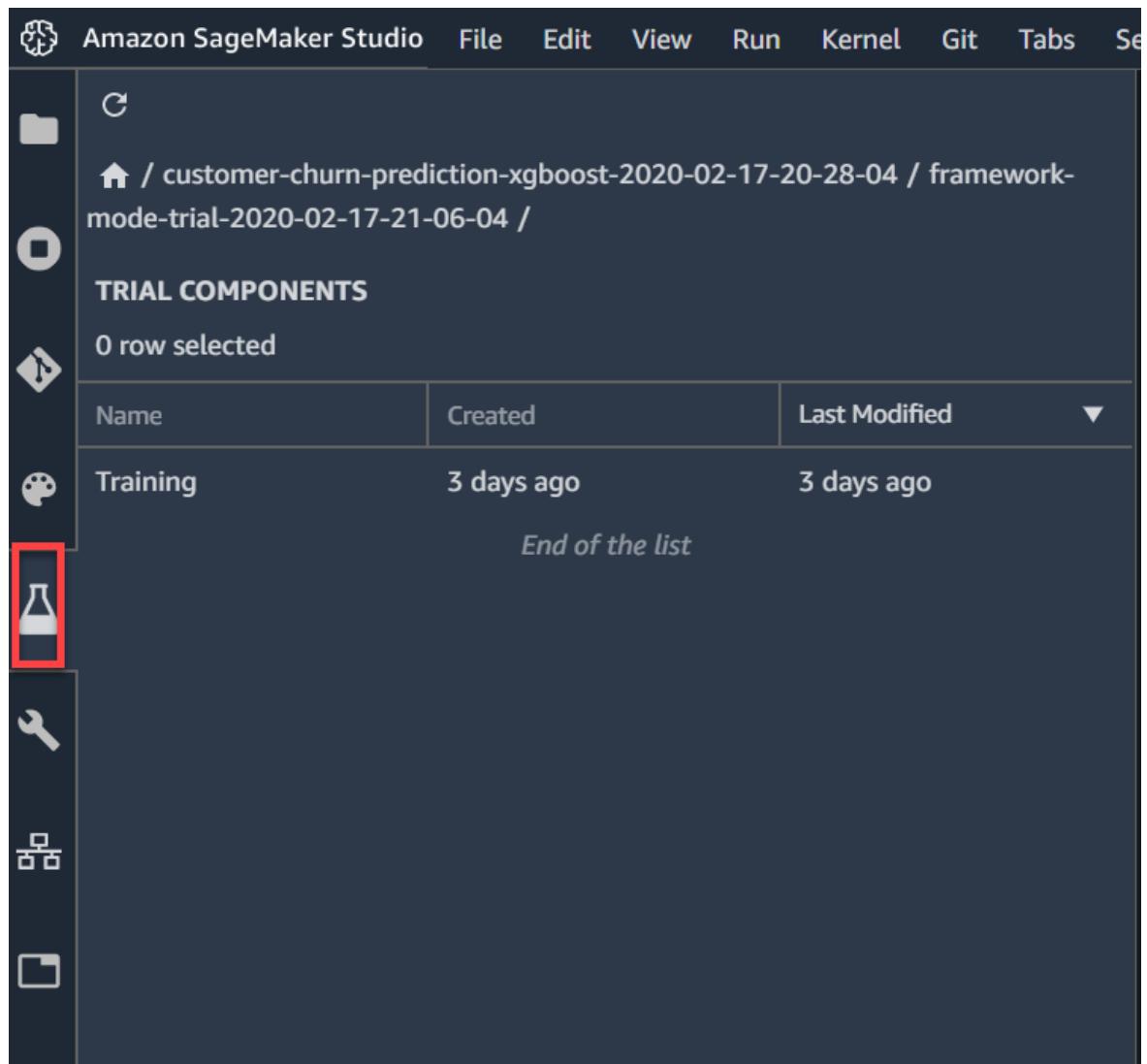
1. Run the following cell to create an experiment by calling the `create` method of the `Experiment` class.

```
sess = sagemaker.session.Session()

create_date = strftime("%Y-%m-%d-%H-%M-%S", gmtime())
customer_churn_experiment = Experiment.create(experiment_name="customer-churn-
prediction-xgboost-{}".format(create_date),
                                              description="Using xgboost to predict
                                              customer churn",
                                              sagemaker_boto_client=boto3.client('sagemaker'))
```

2. Create trials and associate them with the experiment that you created in the previous step, and then run the trials by calling the `fit` method of the `Estimator` class. Here, we'll look at the cell in the notebook named `Trying other hyperparameter values` so that we generate several trials.

After the training jobs finish, you can see the new experiment and trials in the Studio experiments list by choosing the SageMaker Experiment List icon ().



To see the trials associated with your experiment, open the context (right click) menu on the experiment name and choose **Open in trial component list**.

This screenshot shows a modal window titled "Trial Component List" within the Amazon SageMaker Studio interface. The modal contains a table of "TRIAL COMPONENTS" with 10 rows. The columns are: Status, Experiment, Type, Trial, Trial component, Created on, and Last modified. All entries are "Completed" and belong to the experiment "customer-churn-prediction-xgboost-2020-02-17-20-28-04". The table ends with "End of the list". On the left side of the modal, there's a sidebar with icons for folder, file, experiments, and other studio functions. A context menu is open over the experiment name "customer-churn-prediction-xgboost-2020-02-17-20-28-04", listing options: "Open in trial component list", "Copy cell contents", and "Shift+Right Click for Browser Menu".

On this page, you can see the details about the trials, compare trials to find the best performing model, and create charts to visualize training results.

Create Charts to Visualize Data

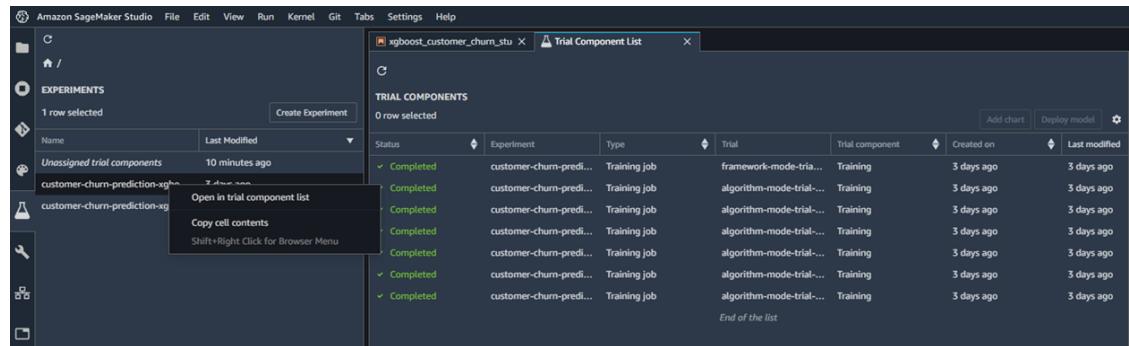
To visualize data after training jobs run, you can create charts in Amazon SageMaker Studio. In this example notebook, the training jobs run for a very short time, so they don't create much data. Because of this, we'll create a scatter plot of the `validation:error_last` metric (final validation error) for each of the `min_child_weight` hyperparameter values that we specified in the training jobs.

To create the scatter plot

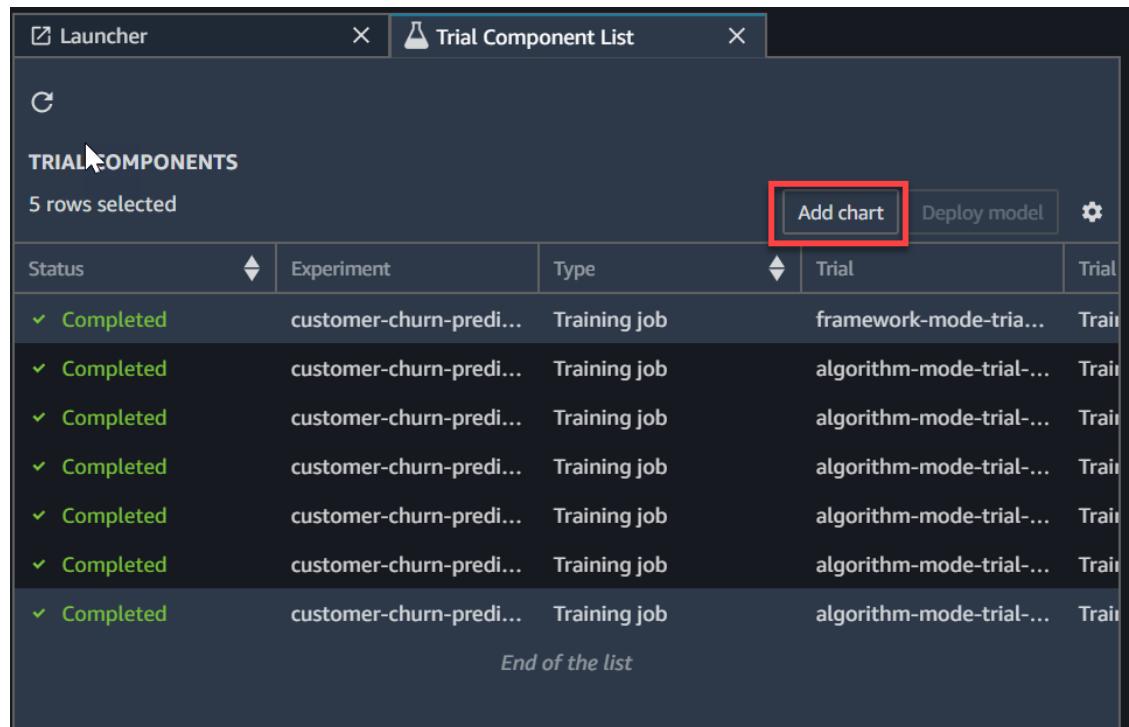
1.

In Studio, choose the SageMaker Experiment List icon ().

2. Open the context (right-click) menu for the experiment, and then choose **Open in trial component list**.



3. In the **Trial Components** list, multi-select the five trials that we varied the `min_child_weight` for, then choose **Add chart**.



4. Choose **Settings**, then configure the chart properties as follows:

- For **Data type**, choose **Summary statistics**.
- For **Chart type**, choose **Scatter plot**.

- For **X-axis**, choose **min_child_weight**.
- For **Y-axis**, choose **validation:error_last**.

CHART PROPERTIES

Data type

- Time series
- Summary statistics

Chart type

- Bar
- Histogram
- Line
- Scatter plot

X-axis

min_child_weight ▾

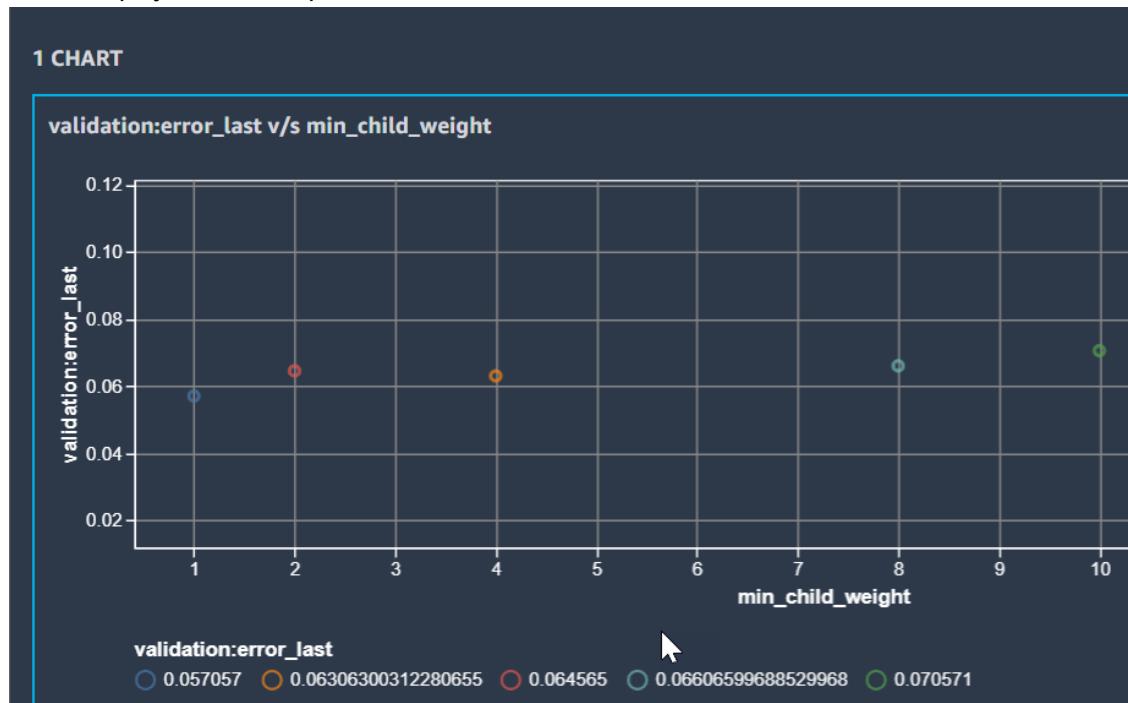
Y-axis

validation:error_last ▾

Color

Select a column ▾

Studio displays the scatter plot.



Debug Training Jobs

Amazon SageMaker Debugger helps you analyze your training jobs and find problems. It monitors, records, and analyzes tensor data from training jobs and checks the training tensors against a set of rules that you specify. You can choose from a list of built-in rules, or create your own custom rules. For more information, see [Amazon SageMaker Debugger \(p. 536\)](#).

To use Amazon SageMaker Debugger to debug a training job

1. Specify one or more rules that you want to use to analyze your training job. In this notebook, we use the following rules.

```
debug_rules = [Rule.sagemaker(rule_configs.loss_not_decreasing()),
               Rule.sagemaker(rule_configs.overtraining()),
               Rule.sagemaker(rule_configs.overfit())
              ]
```

2. Create a trial to track the new training job, and run the training job with the specified Debugger rules.

```
entry_point_script = "xgboost_customer_churn.py"

trial = Trial.create(trial_name="framework-mode-trial-{}".format(strftime("%Y-%m-%d-%H-%M-%S", gmtime())),
                     experiment_name=customer_churn_experiment.experiment_name,
                     sagemaker_boto_client=boto3.client('sagemaker'))

framework_xgb = sagemaker.xgboost.XGBoost(image_name=docker_image_name,
                                           entry_point=entry_point_script,
                                           role=role,
                                           framework_version="0.90-2",
```

```

        py_version="py3",
        hyperparameters=hyperparams,
        train_instance_count=1,
        train_instance_type='ml.m4.xlarge',
        output_path='s3://{}//{}'
    output'.format(bucket, prefix),
    base_job_name="demo-xgboost-customer-churn",
    sagemaker_session=sess,
    rules=debug_rules
)

framework_xgb.fit({'train': s3_input_train,
                    'validation': s3_input_validation},
                    experiment_config={
                        "ExperimentName": customer_churn_experiment.experiment_name,
                        "TrialName": trial.trial_name,
                        "TrialComponentDisplayName": "Training",
})

```

You can see the results for each Debugger rule that you specified by navigating in Studio to the trial that you just created in the **Trial Components** list, and then choosing **Debugger** on the **Describe Trial Component** page for the training job.

Status	Last modified	Rule name	Job ARN
No Issues Found	3 days ago	LossNotDecreasing	arn:aws:sagemaker:us-east-2:279
No Issues Found	3 days ago	Overtraining	arn:aws:sagemaker:us-east-2:279
No Issues Found	3 days ago	Overfit	arn:aws:sagemaker:us-east-2:279

Notice that the training job passed all three of the rules that were configured for the job. If Debugger had found issues, you could choose the rule in the list to see more information in the **Debugger Details** page.

Status	Last modified	Rule name	Job ARN
No Issues Found	3 days ago	LossNotDecreasing	arn:aws:sagemaker:us-east-2:279
No Issues Found	3 days ago	Overtraining	arn:aws:sagemaker:us-east-2:279
No Issues Found	3 days ago	Overfit	arn:aws:sagemaker:us-east-2:279

Monitor Deployed Models

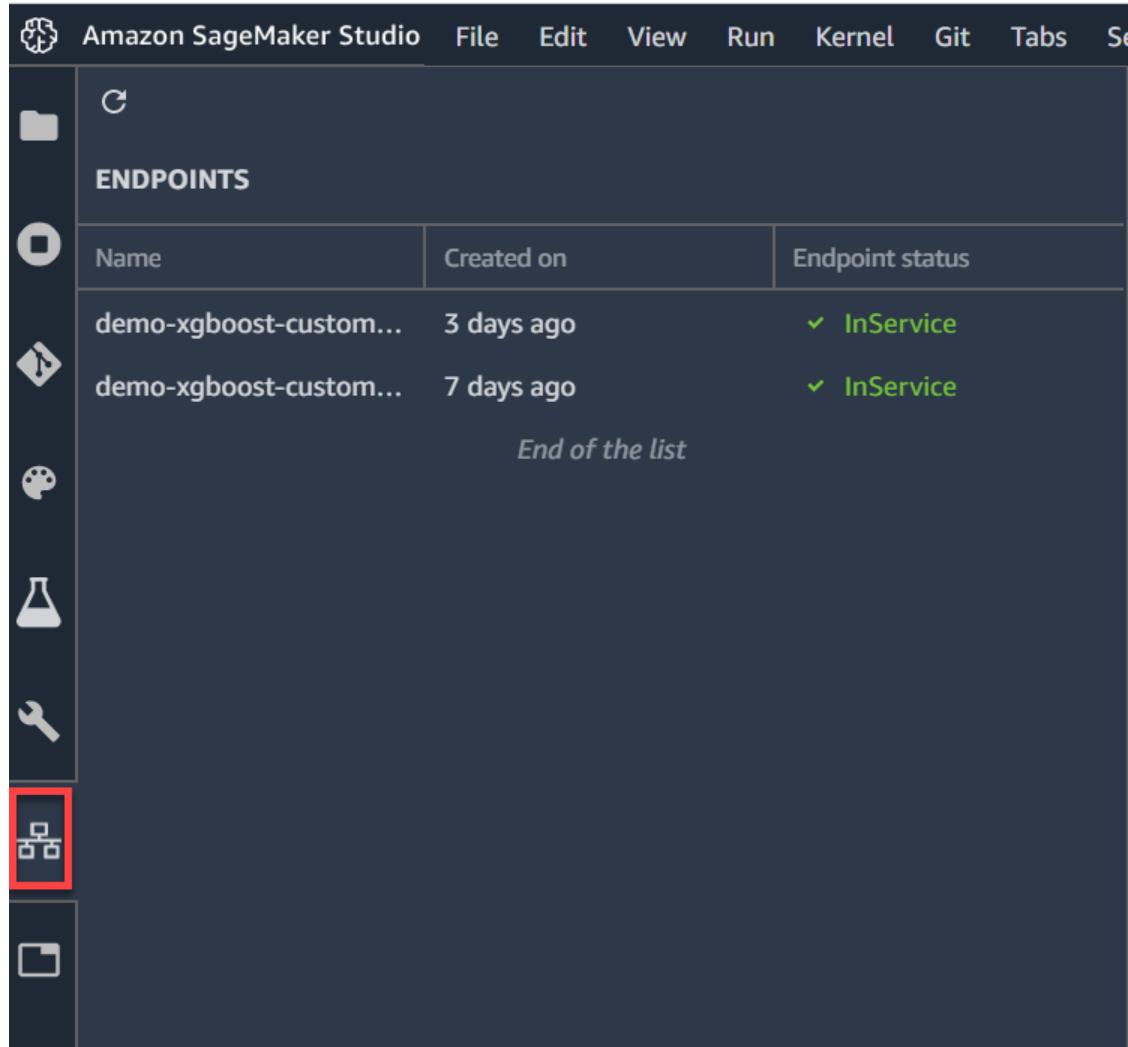
Monitor the quality of your deployed models with Amazon SageMaker Model Monitor. Model Monitor runs monitoring jobs on the endpoints where models are deployed. You can use its built-in monitoring capabilities, which don't require coding, or you can write code for custom analysis. In this exercise, we'll look only at the results of a scheduled monitor job for the endpoint that you created in the example

notebook. For information about what to do to monitor models in production, see [Amazon SageMaker Model Monitor \(p. 635\)](#).

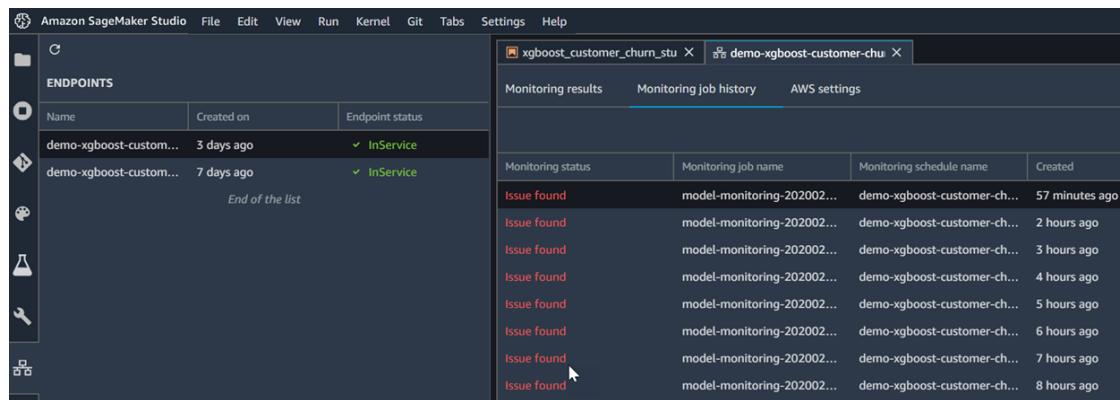
After you set up and schedule model monitoring, you can see the results in Amazon SageMaker Studio.

To monitor a deployed model

1. In Studio, choose the SageMaker Endpoint List icon ().



2. Choose the endpoint that you scheduled monitoring for.



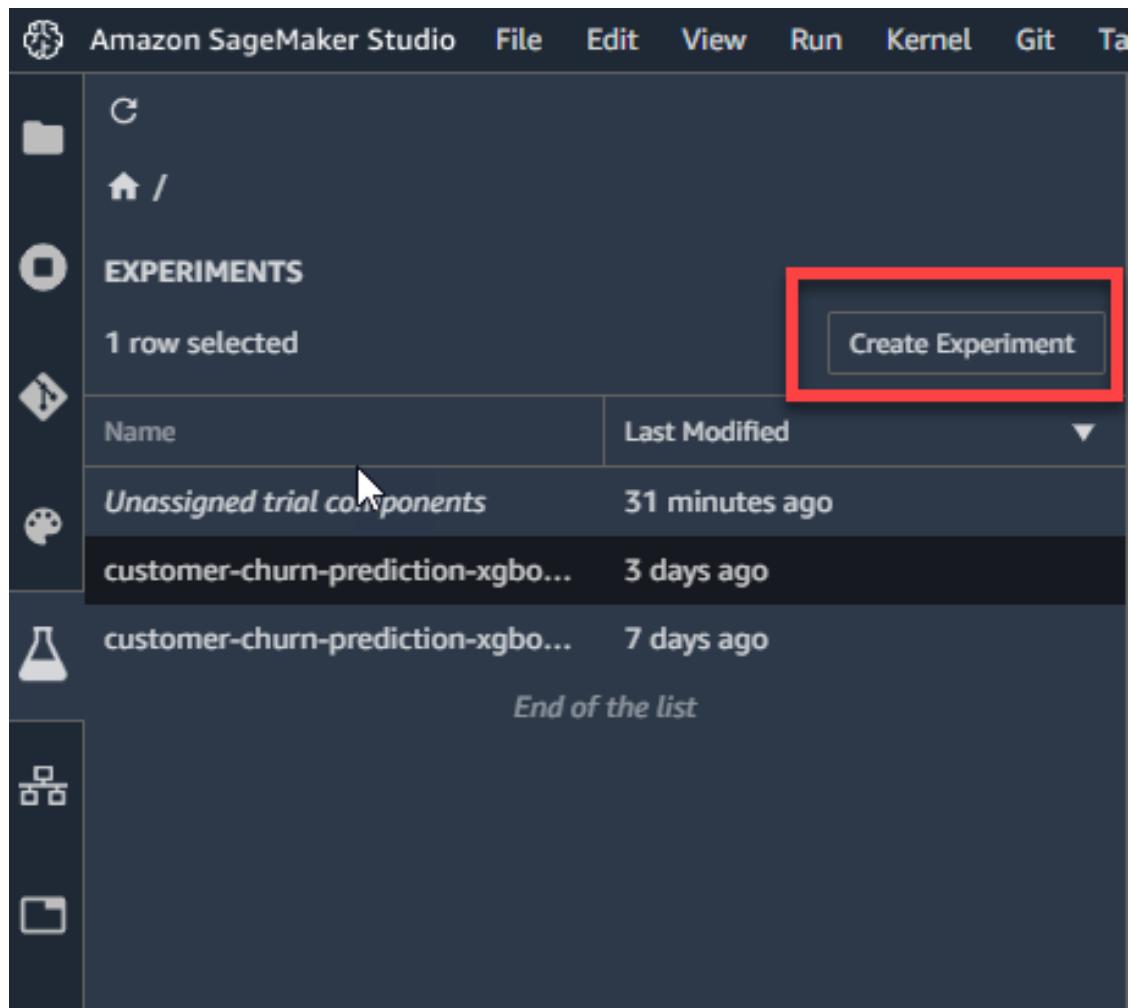
In the **Monitoring job history** list, you can see any issues that the monitoring jobs found. To see details about an issue, choose the issue.

Create an Amazon SageMaker Autopilot Experiment

If the dataset that you want to train on contains tabular data that is formatted as comma-separated values (CSV), you can use Amazon SageMaker Autopilot in Amazon SageMaker Studio to create models. Autopilot simplifies the machine learning process by helping you explore your data by trying different algorithms. It also automatically trains and tunes models on your behalf, to help you find the best algorithm. For more information about Amazon SageMaker Autopilot, see [Use Amazon SageMaker Autopilot to Automate Model Development \(p. 60\)](#).

To create an Autopilot experiment

1. In Studio, on the **Experiments** pane, choose **Create Experiment**.

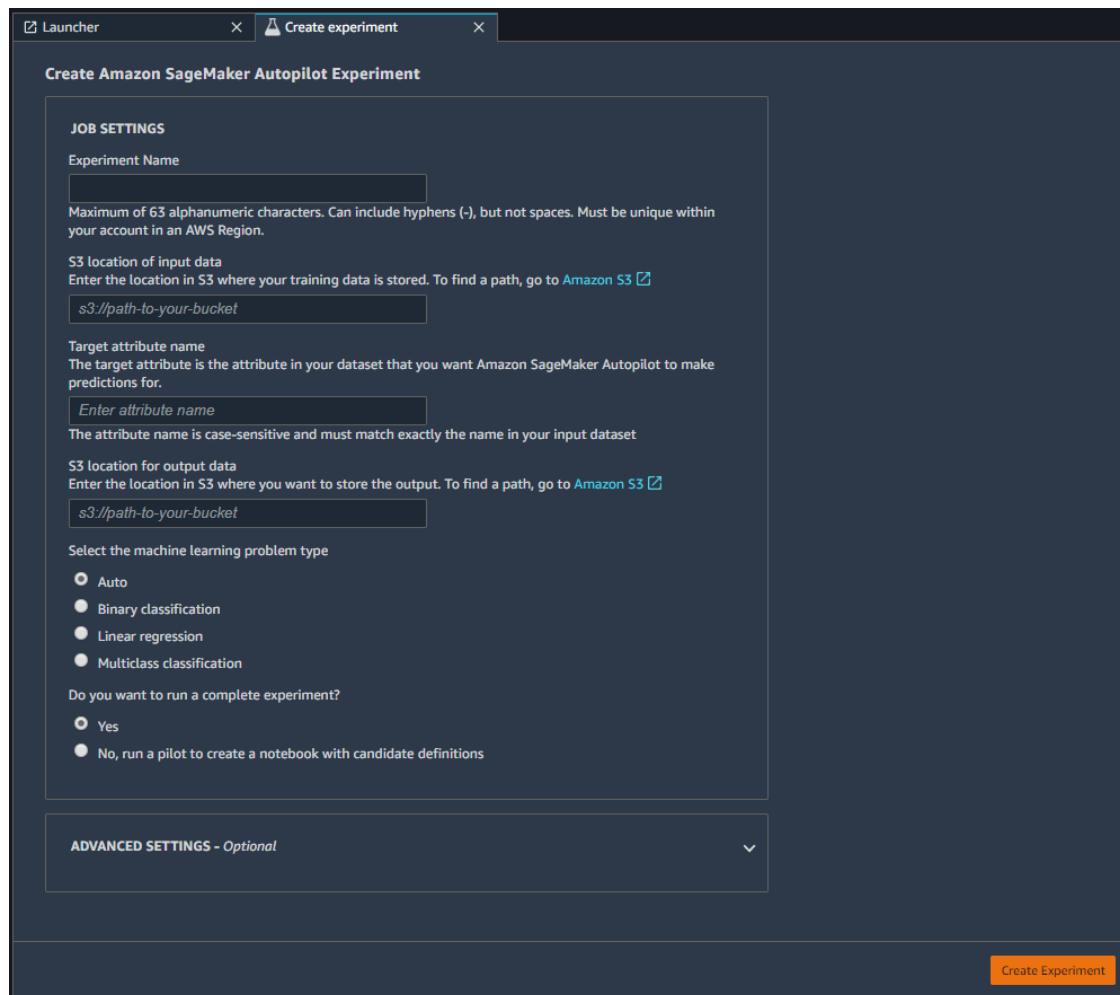


2. On the **Create experiment** page, provide the required information for the Autopilot experiment:
 - A name for the experiment.
 - The location in Amazon S3 where the input dataset is stored.

Note

The input data must be in CSV format and contain at least 1000 rows.

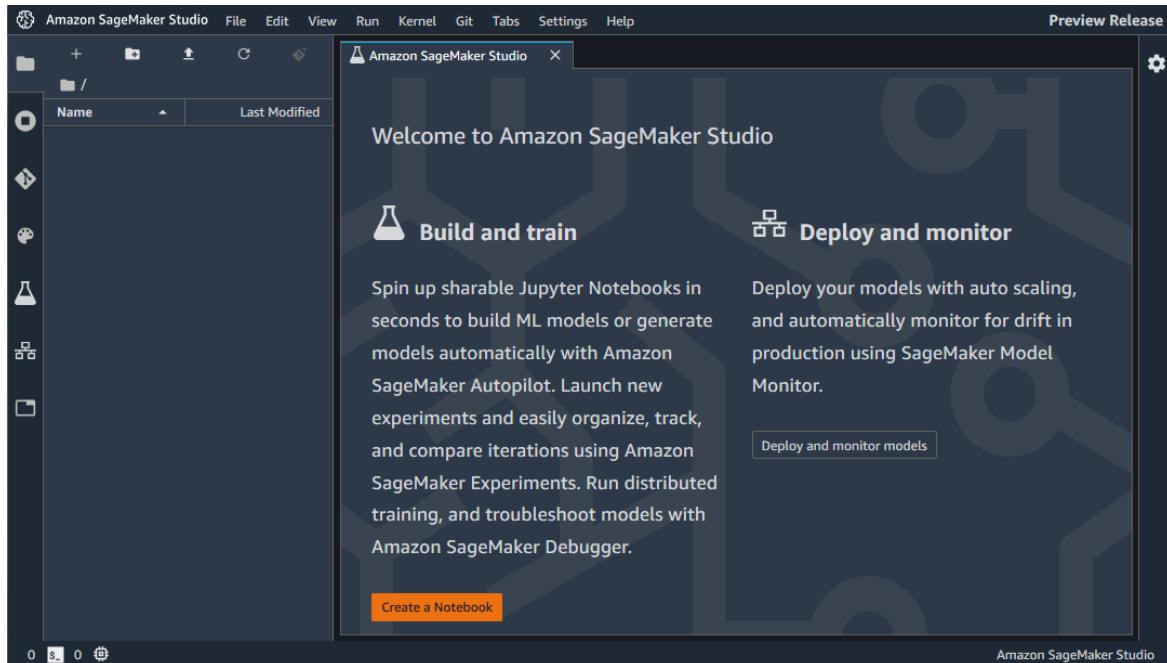
- The target attribute in the dataset that you want the model to predict.
- The location in Amazon S3 where you want to store the model output data.



Amazon SageMaker Studio UI Overview

Amazon SageMaker Studio extends the JupyterLab interface. Previous users of JupyterLab will notice the similarity of the user interface, including the workspace. Studio adds many additions to the interface. The most prominent additions are detailed in the following sections. For an overview of the basic JupyterLab interface, see [The JupyterLab Interface](#).

The following image shows the initial screen when you sign-on to Amazon SageMaker Studio.



At the top of the screen is the *menu bar*. At the left of the screen is the *left sidebar* which contains icons for different file and resource browsers. At the right of the screen is the *right sidebar*, represented by the gear icon, which displays contextual preference settings. At the bottom of the screen is the *status bar*.

The main work area is divided horizontally into two panes. The left pane is the *file and resource browser*. The right pane contains one or more tabs for resources such as notebooks, terminals, metrics, and graphs.

Topics

- [Left Sidebar \(p. 36\)](#)
- [SageMaker Experiment List \(p. 37\)](#)
- [SageMaker Endpoint List \(p. 37\)](#)
- [Using Notebooks \(p. 37\)](#)
- [View a Trial's Properties and Components \(p. 38\)](#)

Left Sidebar

The left sidebar includes the following icons. When you hover over an icon, a tooltip displays the icon name. When you select an icon, the file and resource browser displays the described functionality. For hierarchical entries, a selectable breadcrumb at the top of the browser shows your location.

Icon	Description
	File browser.
	Running terminals and kernels.

Icon	Description
	Git.
	Commands (Ctrl + Shift + C). The majority of the menu commands are available here.
	SageMaker Experiment List. Displays a list of experiments, trials, or trial components. Includes a button to create an Autopilot experiment.
	SageMaker Endpoint List. Selected when you choose Deploy and monitor models on the Studio landing page.
	Open tabs

SageMaker Experiment List

Displays a list of experiments, trials, or trial components, and allows you to create an Amazon SageMaker Autopilot experiment. The following primary actions are available:

- To create an Autopilot experiment, choose **Create Experiment**.
- To display the list of trials that are part of an experiment, double-click the experiment.
- To display the list of trial components that make up a trial, double-click the trial.
- To open a tab in the work area that describes a component, double-click the component.
- To compare experiments or trials, multi-select the items, right-click one of the selections, and then choose **Open in trial component list**.

A selectable breadcrumb above the header shows your position in the hierarchy and allows you to navigate to a higher level.

SageMaker Endpoint List

To open a tab that shows a description of an endpoint, double click the endpoint.

Using Notebooks

To open a notebook, follow these steps:

1. In the left sidebar, select the **File Browser** icon.
2. At the top of the file browser, select the **Up arrow** icon, and then a **File Upload** dialog opens. Browse to and select the notebook and any accompanying files, and then choose **Open**.
3. Double-click the uploaded notebook file to open the notebook in a new tab.

View a Trial's Properties and Components

To display the list of trials that are part of an experiment, choose the **SageMaker Experiment List** icon and then double-click the experiment.

There are two ways to view a trial's properties and components. Each method displays similar information in different formats, as well as unique information as noted in each item below.

- Right click one of the trials and then choose **Open in trial component list**. A new tab opens that displays a list of the trial's components.

Use this view to deploy a model.
- Double-click one of the trials and a list of the trial's components is displayed. Double-click one of the components in the list and a new tab opens that describes each component.

Amazon SageMaker Studio Features

Amazon SageMaker Studio includes the following features.

Topics

- [Amazon SageMaker Studio Notebooks \(p. 38\)](#)
- [Amazon SageMaker Experiments \(p. 38\)](#)
- [Amazon SageMaker Autopilot \(p. 39\)](#)
- [Amazon SageMaker Debugger \(p. 39\)](#)
- [Amazon SageMaker Model Monitor \(p. 39\)](#)

Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio Notebooks is the next generation of Amazon SageMaker notebooks. These notebooks include the following new features:

- AWS Single Sign-On (AWS SSO) integration
- Fast start-up times
- Ability to share notebooks with a single click

For more information, see [Use Amazon SageMaker Notebooks \(p. 247\)](#).

Note

Because Amazon SageMaker Studio Notebooks is in preview, visual elements of Amazon SageMaker Studio might be impacted.

Amazon SageMaker Experiments

Amazon SageMaker provides experiment management and tracking. Users can organize their experiments and artifacts in a centralized location using a structured organization scheme.

An experiment is a collection of machine learning iterations called trials. A trial is a set of steps called trial components. A trial takes a combination of inputs such as a dataset, an algorithm, and parameters, and produces specific outputs such as a model, metrics, and checkpoints.

Experiment tracking enables both Amazon SageMaker automated tracking of model training, tuning, and evaluation jobs, and API-enabled tracking of experiments done locally on Amazon SageMaker notebooks.

Customers can use the tracked data to reconstruct an experiment, incrementally build on experiments conducted by peers, and trace model lineage for compliance and audit verifications.

For more information, see [Manage Machine Learning with Amazon SageMaker Experiments \(p. 524\)](#).

Amazon SageMaker Autopilot

Amazon SageMaker Autopilot provides automatic machine learning that allows users without machine learning knowledge to quickly build classification and regression models. Users only need to provide a tabular dataset and select the target column to predict. Autopilot automatically explores machine learning solutions with different combinations of data preprocessors, algorithms, and algorithm parameters, to find the best model.

When a user runs an Autopilot job, Amazon SageMaker creates an experiment for the job and then creates a trial for each combination and stores all data and results. After the best model is determined, the user can drill down to view each trial and see which features had the most influence on the result.

For more information, see [Use Amazon SageMaker Autopilot to Automate Model Development \(p. 60\)](#).

Amazon SageMaker Debugger

Amazon SageMaker Debugger provides full visibility into the model training process by enabling the inspection of all the training parameters and data throughout the training process.

Debugger provides a visual interface to analyze debug data and visual indicators about potential anomalies in the data.

Debugger automatically detects and alerts users to commonly occurring errors such as parameter values getting too large or small. Users can extend Debugger to detect new classes of errors that are specific to their model.

For more information, see [Amazon SageMaker Debugger \(p. 536\)](#).

Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor is a tool for the monitoring and analysis of models in production (Amazon SageMaker endpoints). Model Monitor offers a framework-agnostic analysis.

Machine learning models are typically trained and evaluated using historical data. After they are deployed in production, the quality of their predictions can degrade over time due to model drift. Model drift is when the distribution of the data sent to the models for predictions varies from the distribution of data used during training.

Model Monitor continuously monitors and analyzes the prediction requests. Model Monitor can store this data and use built-in statistical rules to detect common issues such as outliers in data and data drift.

For more information, see [Amazon SageMaker Model Monitor \(p. 635\)](#).

Get Started with the Amazon SageMaker Console

The best way to learn how to use Amazon SageMaker is to create, train, and deploy a simple machine learning model. To do this, you need the following:

- A dataset. You use the MNIST (Modified National Institute of Standards and Technology database) dataset of images of handwritten, single digit numbers. This dataset provides a training set of 50,000

example images of handwritten single-digit numbers, a validation set of 10,000 images, and a test dataset of 10,000 images. You provide this dataset to the algorithm for model training. For more information about the MNIST dataset, see [MNIST Dataset](#).

- An algorithm. You use the XGBoost algorithm provided by Amazon SageMaker to train the model using the MNIST dataset. During model training, the algorithm assigns example data of handwritten numbers into 10 clusters: one for each number, 0 through 9. For more information about the algorithm, see [XGBoost Algorithm \(p. 445\)](#).

You also need a few resources for storing your data and running the code in this exercise:

- An Amazon Simple Storage Service (Amazon S3) bucket to store the training data and the model artifacts that Amazon SageMaker creates when it trains the model.
- An Amazon SageMaker notebook instance to prepare and process data and to train and deploy a machine learning model.
- A Jupyter notebook to use with the notebook instance to prepare your training data and train and deploy the model.

In this exercise, you learn how to create all of the resources that you need to create, train, and deploy a model.

Important

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

- The high-level Amazon SageMaker Python SDK
- The AWS SDK for Python (Boto 3)

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. This exercise provides code examples for both libraries. If you're a first-time Amazon SageMaker user, we recommend that you use the Amazon SageMaker Python SDK. For more information, see <https://sagemaker.readthedocs.io/en/stable/overview.html>.

If you're new to Amazon SageMaker, we recommend that you read [How Amazon SageMaker Works \(p. 3\)](#) before starting this exercise.

Topics

- [Step 1: Create an Amazon S3 Bucket \(p. 40\)](#)
- [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#)
- [Step 3: Create a Jupyter Notebook \(p. 42\)](#)
- [Step 4: Download, Explore, and Transform the Training Data \(p. 42\)](#)
- [Step 5: Train a Model \(p. 45\)](#)
- [Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#)
- [Step 7: Validate the Model \(p. 54\)](#)
- [Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications \(p. 58\)](#)
- [Step 9: Clean Up \(p. 58\)](#)

Step 1: Create an Amazon S3 Bucket

Training a model produces the following

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

You save these in an Amazon Simple Storage Service (Amazon S3) bucket: You can store datasets that you use as your training data and model artifacts that are the output of a training job in a single bucket or in two separate buckets. For this exercise and others in this guide, one bucket is sufficient. If you already have S3 buckets, you can use them, or you can create new ones.

To create a bucket, follow the instructions in [Create a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*. Include `sagemaker` in the bucket name. For example, `sagemaker-datetime`.

Note

Amazon SageMaker needs permission to access these buckets. You grant permission with an IAM role, which you create in the next step when you create an Amazon SageMaker notebook instance. This IAM role automatically gets permissions to access any bucket that has `sagemaker` in the name. It gets these permissions through the `AmazonSageMakerFullAccess` policy, which Amazon SageMaker attaches to the role. If you add a policy to the role that grants the SageMaker service principal `S3FullAccess` permission, the name of the bucket does not need to contain `sagemaker`.

Next Step

[Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#)

Step 2: Create an Amazon SageMaker Notebook Instance

An Amazon SageMaker notebook instance is a fully managed machine learning (ML) Amazon Elastic Compute Cloud (Amazon EC2) compute instance that runs the Jupyter Notebook App. You use the notebook instance to create and manage Jupyter notebooks that you can use to prepare and process data and to train and deploy machine learning models. For more information, see [Explore, Analyze, and Process Data \(p. 5\)](#).

Note

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

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information (if a field is not mentioned, leave the default values):
 - a. For **Notebook instance name**, type a name for your notebook instance.
 - b. For **Instance type**, choose `m1.t2.medium`. This is the least expensive instance type that notebook instances support, and it suffices for this exercise.
 - c. For **IAM role**, choose **Create a new role**, then choose **Create role**.
 - d. Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries.

Next Step

[Step 3: Create a Jupyter Notebook \(p. 42\)](#).

Step 3: Create a Jupyter Notebook

You can create a Jupyter notebook in the notebook instance you created in [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#), and create a cell that gets the IAM role that your notebook needs to run Amazon SageMaker APIs and specifies the name of the Amazon S3 bucket that you will use to store the datasets that you use for your training data and the model artifacts that a Amazon SageMaker training job outputs.

To create a Jupyter notebook

1. Open the notebook instance.
 - a. Sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 - b. Open the Notebook Instances, and then open the notebook instance you created by choosing either **Open Jupyter** for classic Jupyter view or **Open JupyterLab** for JupyterLab view next to the name of the notebook instance.

Note

If you see **Pending** to the right of the notebook instance in the **Status** column, your notebook is still being created. The status will change to **InService** when the notebook is ready for use.

2. Create a notebook.
 - a. If you opened the notebook in Jupyter classic view, on the **Files** tab, choose **New**, and **conda_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.
 - b. If you opened the notebook in JupyterLab view, on the **File** menu, choose **New**, and then choose **Notebook**. For **Select Kernel**, choose **conda_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.
3. In the Jupyter notebook, choose **File** and **Save as**, and name the notebook.

Next Step

[Step 4: Download, Explore, and Transform the Training Data \(p. 42\)](#)

Step 4: Download, Explore, and Transform the Training Data

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

You transform the data by changing its format from `numpy.array` to comma-separated values (CSV). The [XGBoost Algorithm \(p. 445\)](#) expects input in either the LIBSVM or CSV format. LIBSVM is an open source machine learning library. In this exercise , you use CSV format because it's simpler.

Topics

- [Step 4.1: Download the MNIST Dataset \(p. 42\)](#)
- [Step 4.2: Explore the Training Dataset \(p. 43\)](#)
- [Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 \(p. 44\)](#)

Step 4.1: Download the MNIST Dataset

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

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

# 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')
print(train_set[0].shape)
```

The code does the following:

1. Downloads the MNIST dataset (`mnist.pkl.gz`) from the MNIST Database website to your notebook.
2. Unzips the file and reads the following datasets into the notebook's memory:
 - `train_set` – You use these images of handwritten numbers to train a model.
 - `valid_set` – The [XGBoost Algorithm \(p. 445\)](#) uses these images to evaluate the progress of the model during training.
 - `test_set` – You use this set to get inferences to test the deployed model.

Next Step

[Step 4.2: Explore the Training Dataset \(p. 43\)](#)

Step 4.2: 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.

To explore the dataset

- Type the following code in a cell in your notebook and run the cell to display the first 10 images in `train_set`:

```
%matplotlib inline
import matplotlib.pyplot as plt

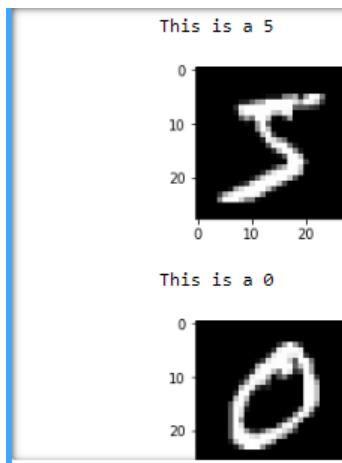
plt.rcParams["figure.figsize"] = (2,10)

for i in range(0, 10):
    img = train_set[0][i]
    label = train_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```

`train_set` contains the following structures:

- `train_set[0]` – Contains images.
- `train_set[1]` – Contains labels.

The code uses the `matplotlib` library to get and display the first 10 images from the training dataset.



Next Step

[Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 \(p. 44\)](#)

Step 4.3: Transform the Training Dataset and Upload It to Amazon S3

The [XGBoost Algorithm \(p. 445\)](#) expects comma-separated values (CSV) for its training input. The format of the training dataset is `numpy.array`. Transform the dataset from `numpy.array` format to the CSV format. Then upload it to the Amazon S3 bucket that you created in [Step 1: Create an Amazon S3 Bucket \(p. 40\)](#)

To convert the dataset to CSV format and upload it

- Type the following code into a cell in your notebook and then run the cell.

```
%time

import os
import boto3
import re
import copy
import time
import io
import struct
from time import gmtime, strftime
from sagemaker import get_execution_role

role = get_execution_role()

region = boto3.Session().region_name

bucket='myBucket' # Replace with your s3 bucket name
prefix = 'sagemaker/xgboost-mnist' # Used as part of the path in the bucket where you
    store data
bucket_path = 'https://s3-{}.amazonaws.com/{}'.format(region,bucket) # The URL to
access the bucket

def convert_data():
    data_partitions = [('train', train_set), ('validation', valid_set), ('test',
    test_set)]
```

```
for data_partition_name, data_partition in data_partitions:
    print('{{}}: {} {}'.format(data_partition_name, data_partition[0].shape,
data_partition[1].shape))
    labels = [t.tolist() for t in data_partition[1]]
    features = [t.tolist() for t in data_partition[0]]

    if data_partition_name != 'test':
        examples = np.insert(features, 0, labels, axis=1)
    else:
        examples = features
    #print(examples[50000,:])

    np.savetxt('data.csv', examples, delimiter=',')

key = "{}/{}/{}/examples".format(prefix,data_partition_name)
url = 's3://{}/{}/{}'.format(bucket, key)

boto3.Session().resource('s3').Bucket(bucket).Object(key).upload_file('data.csv')
print('Done writing to {}'.format(url))

convert_data()
```

After it converts the dataset to the CSV format, the code uploads the CSV file to the S3 bucket.

Next Step

[Step 5: Train a Model \(p. 45\)](#)

Step 5: Train a Model

To train, deploy, and validate a model in Amazon SageMaker, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). (You can also use the console, but for this exercise, you will use the notebook instance and one of the SDKs.) This exercise provides code examples for each library.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you're a first-time Amazon SageMaker user, we recommend that you use it to train, deploy, and validate the model. For more information, see <https://sagemaker.readthedocs.io/en/stable/overview.html>.

Topics

- [Choose the Training Algorithm \(p. 45\)](#)
- [Create and Run a Training Job \(Amazon SageMaker Python SDK\) \(p. 46\)](#)
- [Create and Run a Training Job \(AWS SDK for Python \(Boto 3\)\) \(p. 47\)](#)

Choose the Training Algorithm

To choose the right algorithm for your model, you typically follow an evaluation process. For this exercise, you use the [XGBoost Algorithm \(p. 445\)](#) provided by Amazon SageMaker, so no evaluation is required. For information about choosing algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).

Create and Run a Training Job (Amazon SageMaker Python SDK)

The Amazon SageMaker Python SDK includes the `sagemaker.estimator.Estimator` estimator. You can use this class, in the `sagemaker.estimator` module, with any algorithm. For more information, see <https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator>.

To run a model training job (Amazon SageMaker Python SDK)

1. Import the Amazon SageMaker Python SDK and get the XGBoost container.

```
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

2. Download the training and validation data from the Amazon S3 location where you uploaded it in [Step 4.3: Transform the Training Dataset and Upload It to Amazon S3 \(p. 44\)](#), and set the location where you store the training output.

```
train_data = 's3://{}//{}//{}'.format(bucket, prefix, 'train')
validation_data = 's3://{}//{}//{}'.format(bucket, prefix, 'validation')
s3_output_location = 's3://{}//{}//{}'.format(bucket, prefix, 'xgboost_model_sdk')
print(train_data)
```

3. Create an instance of the `sagemaker.estimator.Estimator` class.

```
xgb_model = sagemaker.estimator.Estimator(container,
                                             role,
                                             train_instance_count=1,
                                             train_instance_type='ml.m4.xlarge',
                                             train_volume_size = 5,
                                             output_path=s3_output_location,
                                             sagemaker_session=sagemaker.Session())
```

In the constructor, you specify the following parameters:

- `role` – The AWS Identity and Access Management (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). This is the role that you got in [Step 3: Create a Jupyter Notebook \(p. 42\)](#).
 - `train_instance_count` and `train_instance_type` – The type and number of ML compute instances to use for model training. For this exercise, you use only a single training instance.
 - `train_volume_size` – The size, in GB, of the Amazon Elastic Block Store (Amazon EBS) storage volume to attach to the training instance. This must be large enough to store training data if you use File mode (File mode is the default).
 - `output_path` – The path to the S3 bucket where Amazon SageMaker stores the training results.
 - `sagemaker_session` – The session object that manages interactions with Amazon SageMaker APIs and any other AWS service that the training job uses.
4. Set the hyperparameter values for the XGBoost training job by calling the `set_hyperparameters` method of the estimator. For a description of XGBoost hyperparameters, see [XGBoost Hyperparameters \(p. 448\)](#).

```
xgb_model.set_hyperparameters(max_depth = 5,
                               eta = .2,
```

```
gamma = 4,  
min_child_weight = 6,  
silent = 0,  
objective = "multi:softmax",  
num_class = 10,  
num_round = 10)
```

5. Create the training channels to use for the training job. For this example, we use both `train` and `validation` channels.

```
train_channel = sagemaker.session.s3_input(train_data, content_type='text/csv')  
valid_channel = sagemaker.session.s3_input(validation_data, content_type='text/csv')  
  
data_channels = {'train': train_channel, 'validation': valid_channel}
```

6. To start model training, call the estimator's `fit` method.

```
xgb_model.fit(inputs=data_channels, logs=True)
```

This is a synchronous operation. The method displays progress logs and waits until training completes before returning. For more information about model training, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

Model training for this exercise can take up to 15 minutes.

Next Step

[Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#)

Create and Run a Training Job (AWS SDK for Python (Boto 3))

To train a model, Amazon SageMaker uses the [CreateTrainingJob](#) API. The AWS SDK for Python (Boto 3) provides the corresponding `create_training_job` method.

When using this method, you provide the following information:

- The training algorithm – Specify the registry path of the Docker image that contains the training code. For the registry paths for the algorithms provided by Amazon SageMaker, see [Common Parameters for Built-In Algorithms \(p. 274\)](#).
- Algorithm-specific hyperparameters – Specify algorithm-specific hyperparameters to influence the final quality of the model. For information, see [XGBoost Hyperparameters \(p. 448\)](#).
- 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).

To run a model training job (AWS SDK for Python (Boto 3))

1. Import the `get_image_uri` utility function Amazon SageMaker Python SDK and get the location of the XGBoost container.

```
import sagemaker  
  
from sagemaker.amazon.amazon_estimator import get_image_uri  
  
container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

2. Set up the training information for the job. You pass this information when you call `create_training_job`. For more information about the information that you need to send to a training job, see [CreateTrainingJob](#).

```
#Ensure that the train and validation data folders generated above are reflected in the
#"InputDataConfig" parameter below.
common_training_params = \
{
    "AlgorithmSpecification": {
        "TrainingImage": container,
        "TrainingInputMode": "File"
    },
    "RoleArn": role,
    "OutputDataConfig": {
        "S3OutputPath": bucket_path + "/" + prefix + "/xgboost"
    },
    "ResourceConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m4.xlarge",
        "VolumeSizeInGB": 5
    },
    "HyperParameters": {
        "max_depth": "5",
        "eta": "0.2",
        "gamma": "4",
        "min_child_weight": "6",
        "silent": "0",
        "objective": "multi:softmax",
        "num_class": "10",
        "num_round": "10"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 86400
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": bucket_path + "/" + prefix + '/train/',
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "text/csv",
            "CompressionType": "None"
        },
        {
            "ChannelName": "validation",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": bucket_path + "/" + prefix + '/validation/',
                    "S3DataDistributionType": "FullyReplicated"
                }
            },
            "ContentType": "text/csv",
            "CompressionType": "None"
        }
    ]
}
```

3. Name your training job, and finish configuring the parameters that you send to it.

```
#training job params
training_job_name = 'xgboost-mnist' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Job name is:", training_job_name)

training_job_params = copy.deepcopy(common_training_params)
training_job_params['TrainingJobName'] = training_job_name
training_job_params['ResourceConfig']['InstanceCount'] = 1
```

4. Call `create_training_job` to start the training job, and wait for it to complete. If the training job fails, print the reason that it failed.

```
%%time

region = boto3.Session().region_name
sm = boto3.Session().client('sagemaker')

sm.create_training_job(**training_job_params)

status = sm.describe_training_job(TrainingJobName=training_job_name)
['TrainingJobStatus']
print(status)
sm.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=training_job_name)
status = sm.describe_training_job(TrainingJobName=training_job_name)
['TrainingJobStatus']
print("Training job ended with status: " + status)
if status == 'Failed':
    message = sm.describe_training_job(TrainingJobName=training_job_name)
    ['FailureReason']
    print('Training failed with the following error: {}'.format(message))
    raise Exception('Training job failed')
```

You now have a trained model. Amazon SageMaker stores the resulting artifacts in your S3 bucket.

Next Step

[Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#)

Step 6: Deploy the Model to Amazon SageMaker

To get predictions, deploy your model. The method you use depends on how you want to generate inferences:

- To get one inference at a time in real time, set up a persistent endpoint using Amazon SageMaker hosting services.
- To get inferences for an entire dataset, use Amazon SageMaker batch transform.

Topics

- [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#)
- [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#)

Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services

To deploy a model in Amazon SageMaker, hosting services, you can use either the Amazon SageMaker Python SDK or the AWS SDK for Python (Boto 3). This exercise provides code examples for both libraries.

The Amazon SageMaker Python SDK abstracts several implementation details, and is easy to use. If you're a first-time Amazon SageMaker user, we recommend that you use it. For more information, see <https://sagemaker.readthedocs.io/en/stable/overview.html>.

Topics

- [Deploy the Model to Amazon SageMaker Hosting Services \(Amazon SageMaker Python SDK\) \(p. 50\)](#)
- [Deploy the Model to Amazon SageMaker Hosting Services \(AWS SDK for Python \(Boto 3\)\). \(p. 50\)](#)

Deploy the Model to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)

Deploy the model that you trained in [Create and Run a Training Job \(Amazon SageMaker Python SDK\) \(p. 46\)](#) by calling the `deploy` method of the `sagemaker.estimator.Estimator` object. This is the same object that you used to train the model. When you call the `deploy` method, specify the number and type of ML instances that you want to use to host the endpoint.

```
xgb_predictor = xgb_model.deploy(initial_instance_count=1,
                                   content_type='text/csv',
                                   instance_type='ml.t2.medium'
)
```

The `deploy` method creates the deployable model, configures the Amazon SageMaker hosting services endpoint, and launches the endpoint to host the model. For more information, see <https://sagemaker.readthedocs.io/en/stable/estimators.html#sagemaker.estimator.Estimator.deploy>.

It also returns a `sagemaker.predictor.RealTimePredictor` object, which you can use to get inferences from the model. For information, see <https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor>.

Next Step

[Step 7: Validate the Model \(p. 54\)](#)

Deploy the Model to Amazon SageMaker Hosting Services (AWS SDK for Python (Boto 3)).

Deploying a model using the AWS SDK for Python (Boto 3) is a three-step process:

1. Create a model in Amazon SageMaker – Send a `CreateModel` request to provide information such as the location of the S3 bucket that contains your model artifacts and the registry path of the image that contains inference code.
2. Create an endpoint configuration – Send a `CreateEndpointConfig` request to provide the resource configuration for hosting. This includes the type and number of ML compute instances to launch to deploy the model.
3. Create an endpoint – Send a `CreateEndpoint` request to create an endpoint. Amazon SageMaker launches the ML compute instances and deploys the model. Amazon SageMaker returns an endpoint. Applications can send requests for inference to this endpoint.

To deploy the model (AWS SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in [Step 3: Create a Jupyter Notebook \(p. 42\)](#) and run the cell.

1. Create a deployable model by identifying the location of model artifacts and the Docker image that contains the inference code.

```
model_name = training_job_name + '-mod'

info = sm.describe_training_job(TrainingJobName=training_job_name)
model_data = info['ModelArtifacts']['S3ModelArtifacts']
print(model_data)

primary_container = {
    'Image': container,
    'ModelDataURL': model_data
}

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

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

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

```
endpoint_config_name = 'DEMO-XGBoostEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S",
    gmtime())
print(endpoint_config_name)
create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialVariantWeight':1,
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

3. Create an Amazon SageMaker endpoint.

```
%%time
import time

endpoint_name = 'DEMO-XGBoostEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_name)
create_endpoint_response = sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])

resp = sm.describe_endpoint(EndpointName=endpoint_name)
status = resp['EndpointStatus']
print("Status: " + status)

while status=='Creating':
    time.sleep(60)
    resp = sm.describe_endpoint(EndpointName=endpoint_name)
    status = resp['EndpointStatus']
```

```
print("Status: " + status)

print("Arn: " + resp['EndpointArn'])
print("Status: " + status)
```

This code continuously calls the `describe_endpoint` command in a `while` loop until the endpoint either fails or is in service, and then prints the status of the endpoint. When the status changes to `InService`, the endpoint is ready to serve inference requests.

Next Step

[Step 7: Validate the Model \(p. 54\)](#)

Step 6.2: Deploy the Model with Batch Transform

To get inference for an entire dataset, use batch transform. Amazon SageMaker stores the results in Amazon S3.

For information about batch transforms, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#). For an example that uses batch transform, see the batch transform sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_batch_transform/introduction_to_batch_transform.

Topics

- [Deploy a Model with Batch Transform \(Amazon SageMaker High-level Python Library\) \(p. 52\)](#)
- [Deploy a Model with Batch Transform \(SDK for Python \(Boto 3\)\) \(p. 53\)](#)

Deploy a Model with Batch Transform (Amazon SageMaker High-level Python Library)

The following code creates a `sagemaker.transformer.Transformer` object from the model that you trained in [Create and Run a Training Job \(Amazon SageMaker Python SDK\) \(p. 46\)](#). Then it calls that object's `transform` method to create a transform job. When you create the `sagemaker.transformer.Transformer` object, you specify the number and type of ML instances to use to perform the batch transform job, and the location in Amazon S3 where you want to store the inferences.

Paste the following code in a cell in the Jupyter notebook you created in [Step 3: Create a Jupyter Notebook \(p. 42\)](#) and run the cell.

```
# The location of the test dataset
batch_input = 's3://{{}/{}}/test/examples'.format(bucket, prefix)

# The location to store the results of the batch transform job
batch_output = 's3://{{}/{}}/batch-inference'.format(bucket, prefix)

transformer = xgb_model.transformer(instance_count=1, instance_type='ml.m4.xlarge',
output_path=batch_output)

transformer.transform(data=batch_input, data_type='S3Prefix', content_type='text/csv',
split_type='Line')

transformer.wait()
```

For more information, see <https://sagemaker.readthedocs.io/en/stable/transformer.html>.

Next Step

[Step 7: Validate the Model \(p. 54\)](#)

Deploy a Model with Batch Transform (SDK for Python (Boto 3))

To run a batch transform job, call the `create_transform_job` method using the model that you trained in [Create and Run a Training Job \(AWS SDK for Python \(Boto 3\)\) \(p. 47\)](#).

To create a batch transform job (SDK for Python (Boto 3))

For each of the following steps, paste the code in a cell in the Jupyter notebook you created in [Step 3: Create a Jupyter Notebook \(p. 42\)](#) and run the cell.

1. Name the batch transform job and specify where the input data (the test dataset) is stored and where to store the job's output.

```
batch_job_name = 'xgboost-mnist-batch' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())  
  
batch_input = 's3://{}/{}/test/examples'.format(bucket, prefix)  
print(batch_input)  
  
batch_output = 's3://{}/{}/batch-inference'.format(bucket, prefix)
```

2. Configure the parameters that you pass when you call the `create_transform_job` method.

```
request = \  
{  
    "TransformJobName": batch_job_name,  
    "ModelName": model_name,  
    "BatchStrategy": "MultiRecord",  
    "TransformOutput": {  
        "S3OutputPath": batch_output  
    },  
    "TransformInput": {  
        "DataSource": {  
            "S3DataSource": {  
                "S3DataType": "S3Prefix",  
                "S3Uri": batch_input  
            }  
        },  
        "ContentType": "text/csv",  
        "SplitType": "Line",  
        "CompressionType": "None"  
    },  
    "TransformResources": {  
        "InstanceType": "ml.m4.xlarge",  
        "InstanceCount": 1  
    }  
}
```

For more information about the parameters, see [CreateTransformJob](#).

3. Call the `create_transform_job` method, passing in the parameters that you configured in the previous step. Then call the `describe_transform_job` method in a loop until it completes.

Paste the following code in a cell in the Jupyter notebook you created in [Step 3: Create a Jupyter Notebook \(p. 42\)](#) and run the cell.

```
sm.create_transform_job(**request)  
  
while(True):  
    response = sm.describe_transform_job(TransformJobName=batch_job_name)  
    status = response['TransformJobStatus']
```

```
if status == 'Completed':
    print("Transform job ended with status: " + status)
    break
if status == 'Failed':
    message = response['FailureReason']
    print('Transform failed with the following error: {}'.format(message))
    raise Exception('Transform job failed')
print("Transform job is still in status: " + status)
time.sleep(30)
```

Next Step

[Step 7: Validate the Model \(p. 54\)](#)

Step 7: Validate the Model

Now that you have trained and deployed a model in Amazon SageMaker, validate it to ensure that it generates accurate predictions on new data. That is, on data that is different from the data that the model was trained on. For this, use the test dataset that you created in [Step 4: Download, Explore, and Transform the Training Data \(p. 42\)](#).

Topics

- [Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services \(p. 54\)](#)
- [Step 7.2: Validate a Model Deployed with Batch Transform \(p. 57\)](#)

Step 7.1: Validate a Model Deployed to Amazon SageMaker Hosting Services

If you deployed a model to Amazon SageMaker hosting services in [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#), you now have an endpoint that you can invoke to get inferences in real time. To validate the model, invoke the endpoint with example images from the test dataset and check whether the inferences you get match the actual labels of the images.

Topics

- [Validate a Model Deployed to Amazon SageMaker Hosting Services \(Amazon SageMaker Python SDK\) \(p. 54\)](#)
- [Validate a Model Deployed to Amazon SageMaker Hosting Services \(AWS SDK for Python \(Boto 3\)\) \(p. 55\)](#)

Validate a Model Deployed to Amazon SageMaker Hosting Services (Amazon SageMaker Python SDK)

To validate the model by using the Amazon SageMaker Python SDK, use the `sagemaker.predictor.RealTimePredictor` object that you created in [Deploy the Model to Amazon SageMaker Hosting Services \(Amazon SageMaker Python SDK\) \(p. 50\)](#). For information, see <https://sagemaker.readthedocs.io/en/stable/predictors.html#sagemaker.predictor.RealTimePredictor>.

To validate the model (Amazon SageMaker Python SDK)

1. Download the test data from Amazon S3.

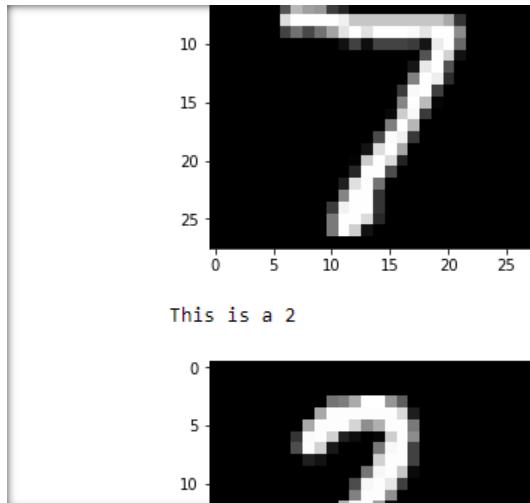
```
s3 = boto3.resource('s3')
```

```
test_key = "{}/test/examples".format(prefix)
s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```



3. To get inferences for the first 10 examples in the test dataset, call the `predict` method of the `sagemaker.predictor.RealTimePredictor` object.

```
with open('test_data', 'r') as f:
    for j in range(0,10):
        single_test = f.readline()
        result = xgb_predictor.predict(single_test)
        print(result)
```

To see if the model is making accurate predictions, check the output from this step against the numbers that you plotted in the previous step.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

[Step 9: Clean Up \(p. 58\)](#)

[Validate a Model Deployed to Amazon SageMaker Hosting Services \(AWS SDK for Python \(Boto 3\)\)](#)

To use the AWS SDK for Python (Boto 3) to validate the model, call the `invoke_endpoint` method. This method corresponds to the [InvokeEndpoint](#) API provided by the Amazon SageMaker runtime.

To validate the model (AWS SDK for Python (Boto 3))

1. Download the test data from Amazon S3.

```
s3 = boto3.resource('s3')

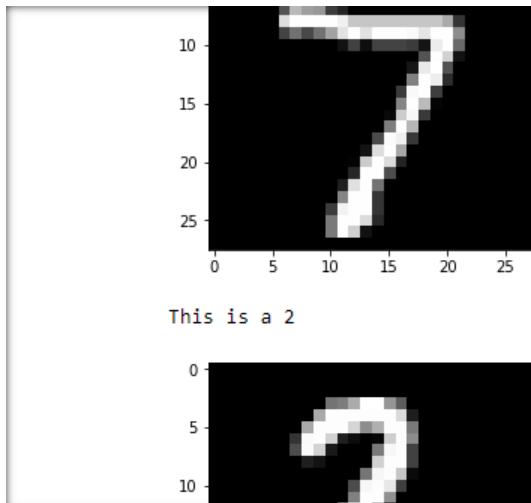
test_key = "{}/test/examples".format(prefix)

s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```



3. Get the Amazon SageMaker runtime client, which provides the `invoke_endpoint` method.

```
runtime_client = boto3.client('runtime.sagemaker')
```

4. Get inferences from the first 10 examples in the test dataset by calling `invoke_endpoint`.

```
with open('test_data', 'r') as f:

    for i in range(0,10):
        single_test = f.readline()
        response = runtime_client.invoke_endpoint(EndpointName = endpoint_name,
                                                    ContentType = 'text/csv',
                                                    Body = single_test)
        result = response['Body'].read().decode('ascii')
        print('Predicted label is {}'.format(result))
```

5. To see if the model is making accurate predictions, check the output from this step against the numbers you plotted in the previous step.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

[Step 9: Clean Up \(p. 58\)](#)

Step 7.2: Validate a Model Deployed with Batch Transform

You now have a file in Amazon S3 that contains inferences that you got by running a batch transform job in [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#). To validate the model, check a subset of the inferences from the file to see whether they match the actual numbers from the test dataset.

To validate the batch transform inferences

1. Download the test data from Amazon S3.

```
s3 = boto3.resource('s3')

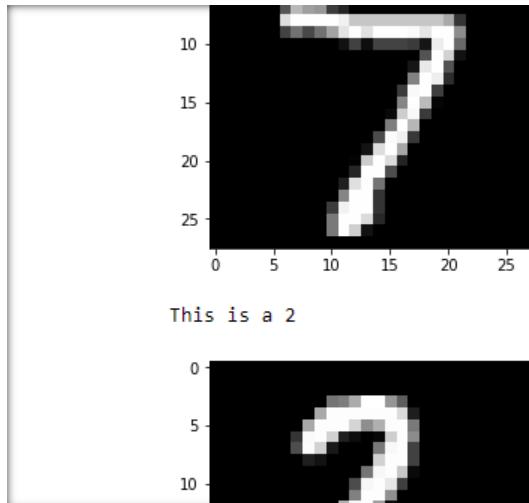
test_key = "{}/test/examples".format(prefix)

s3.Bucket(bucket).download_file(test_key, 'test_data')
```

2. Plot the first 10 images from the test dataset with their labels.

```
%matplotlib inline

for i in range (0, 10):
    img = test_set[0][i]
    label = test_set[1][i]
    img_reshape = img.reshape((28,28))
    imgplot = plt.imshow(img_reshape, cmap='gray')
    print('This is a {}'.format(label))
    plt.show()
```



3. Download the output from the batch transform job from Amazon S3 to a local file.

```
s3.Bucket(bucket).download_file(prefix + '/batch-inference/examples.out',
                                'batch_results')
```

4. Get the first 10 results from the batch transform job.

```
with open('batch_results') as f:  
    results = f.readlines()  
for j in range (0, 10):  
    print(results[j])
```

5. To see if the batch transform job made accurate predictions, check the output from this step against the numbers that you plotted from the test data.

You have now trained, deployed, and validated your first model in Amazon SageMaker.

Next Step

[Step 8: Integrating Amazon SageMaker Endpoints into Internet-facing Applications \(p. 58\)](#)

Step 8: 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.

For an example of how to use Amazon API Gateway and AWS Lambda to set up and deploy a web service that you can call from a client application, see [Call an Amazon SageMaker model endpoint using Amazon API Gateway and AWS Lambda](#) in the *AWS Machine Learning Blog*.

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](#).

Next Step

[Step 9: Clean Up \(p. 58\)](#)

Step 9: 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. Deleting the endpoint also deletes the ML compute instance or instances that support it.
 - The endpoint configuration.
 - The model.
 - The notebook instance. Before deleting the notebook instance, stop 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/.

Use Amazon SageMaker Autopilot to Automate Model Development

Amazon SageMaker Autopilot simplifies the machine learning experience by helping you explore your data and try different algorithms. It also automatically trains and tunes models on your behalf, to help you find the best algorithm. You simply upload tabular data in a file with comma-separated values (for example, from a spreadsheet or database), choose the target column to predict, and Autopilot builds a predictive model for you. These predictions can take the form of ordered numerical values (i.e., this is a regression model) or the form of categories (i.e., a classification model). Autopilot explores different combinations of data preprocessors, algorithms, and algorithm parameter settings to find an accurate model, similar to how a data scientist would, making it easier for novices to get started.

Topics

- [Get Started \(p. 60\)](#)
- [Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio \(p. 60\)](#)
- [SageMaker Autopilot Notebook Output \(p. 61\)](#)
- [SageMaker Autopilot Problem Types \(p. 62\)](#)

Get Started

Autopilot works with tabular data only. Autopilot is available in Amazon SageMaker Studio in the **Create Experiment** tab. It is recommended that you familiarize yourself with the features, and then try it out with your own data. The general flow is as follows:

1. Launch a new experiment, specify the location of the input data and where you want the outputs stored. The outputs include intermediate preprocessed data, the candidate generation notebook, the data exploration notebook, and the model artifacts.
2. Choose the column in your dataset that you want Autopilot to predict.
3. (Optional) Specify additional parameters, such as problem type, security configuration, or job completion criteria.
4. Launch Autopilot.

In addition to the visual experience in SageMaker Studio, there is an example notebook for Autopilot that can help you understand the flow using the AWS SDK.

Create an Amazon SageMaker Autopilot Experiment in SageMaker Studio

1. Open Amazon SageMaker Studio and login.
2. Choose the **Experiments** tab (it looks like a conical flask).
3. Choose **Create experiment**.
4. Enter the experiment's details in the **Job Settings** form.

- Name of the experiment - Must be unique to this experiment in the current AWS Region.
- Input dataset S3 location - An s3:// formatted URL where SageMaker has read permissions.

Note

The input data must be in CSV format and contain at least 1000 rows.

- Target attribute - This is the column of your data you want the model to target.
 - Output S3 location - An s3:// formatted URL where SageMaker has write permissions.
 - (Optional) Tell Autopilot whether you want a regression (e.g., house prices), binary classification (e.g., hotdog or not hotdog), or multi-class classification (e.g., cat, dog or bird) model. If you don't specify this, Autopilot infers it from the values of the attribute you want to predict. In some cases, Autopilot is unable to infer accurately, in which case you must provide the value for the job to succeed.
 - (Optional) Tell Autopilot the metric you want it to use to evaluate models. If you don't specify a metric, Autopilot makes the decision on your behalf based on the best metrics for the situation.
 - (Optional) Configure other job parameters. For example, security configuration or job completion criteria. In the latter case, if you are looking to limit the total wall clock time of the AutoML job, you can specify to limit the number of seconds/minutes the job is allowed to run.
 - (Optional) Specify **Generate Candidate Definitions Only**. In this case, instead of executing the entire AutoML workflow on autopilot, Autopilot stops execution after generating the notebooks for data exploration and candidate generation. This way, you can use the notebooks as a starting point to guide your own process of data exploration and model training/tuning. Both notebooks have highlighted sections that explain what kinds of changes are typical, such as changing instance type, cluster size, and so on.
5. Choose to run the training immediately, or only produce data exploration and candidate generation notebooks.

This is all you need to run a Autopilot experiment. The process will generate a model as well as statistics that you can view in real time while the experiment is running. After the experiment is completed, you can view the trials, sort by objective metric, and right-click to deploy the model for use in other environments.

SageMaker Autopilot Notebook Output

During the analysis phase of the AutoML job, two notebooks are created that describe the plan that Autopilot will follow to generate candidate models. First, there's a data exploration notebook, that describes what Autopilot learned about the data that you provided. Second, there's a candidate generation notebook, which uses the information about the data to generate candidates. Together, these notebooks describe the plan that would be executed on autopilot if you don't choose to stop with candidate generation only.

You can run both notebooks in SageMaker or locally if you have installed the SageMaker Python SDK. You can share the notebooks just like any other SageMaker Studio notebook. Modifications on the candidate generation notebook is encouraged. The notebooks are created for you to experiment. When you run the notebooks in your default instance you will incur baseline costs, but when you execute HPO jobs from the candidate notebook, these jobs use additional compute resources that will incur additional costs.

Candidate Generation Notebook

The candidate generation notebook contains each suggested preprocessing step, suggested algorithm, and suggested hyperparameter ranges. If you chose to only produce the notebook and not run the AutoML job, you can then decide which candidates to be trained and tuned. They optimize automatically

and a final, best candidate will be identified. If you ran the job directly without seeing the candidates first, the best candidate is displayed when you open the notebook after the job's completion.

Data Exploration Notebook

A second notebook is produced during the analysis phase of the AutoML job that helps you identify problems in your dataset. It identifies specific areas for investigation in order to help you identify upstream problems with your data that may result in a suboptimal model.

SageMaker Autopilot Problem Types

You have the option of focusing the Autopilot experiment on specific problem types, which in turn limits the kind of preprocessing and algorithms that are tried.

Your problem type options are as follows:

Topics

- [Linear regression \(p. 62\)](#)
- [Binary classification \(p. 62\)](#)
- [Multi-class classification \(p. 62\)](#)
- [Automatic problem type detection \(p. 63\)](#)

Each of the problem types require a tabular data input with the columns labelled. In a typical ML environment you set the feature that you are interested in predicting (objective) and you can also specify the objective type (classification or regression). However, with Autopilot these are optional. It can detect these settings for you.

Linear regression

One commonly used linear regression example is home prices prediction. Provided a dataset of home prices and other features like number of bathrooms and bedrooms, linear regression can create a model for you that takes one or more of these features as an input and then predicts a home price. When using your own data, it is possible to provide some missing observations (blank data), but you are notified when there's too much missing data for a good predictive model.

Binary classification

Binary classification models are trained using labeled examples of objects mixed with other examples that are not that object. For example, there is the Not Hotdog app whose primary function was to evaluate images and tell you if it contains a hotdog or not. While this provides a humorous look at an AI application, a real-world example is evaluating the Titanic dataset and making fatality predictions (alive or dead). Being alive or dead is a binary outcome that can be easily measured. The provided features such as cabin class, life boat number, or age could all be considered reasonable survivability factors for an ill-fated transatlantic sea voyage.

Multi-class classification

Multi-class refers to the range of possible predictions the model will make. For example, an emotion detection model might have a handful of possible classes: happy, sad, angry, or surprised. A model of this type would not only predict each class, it would return the percentage probability of each class. Another example is found in self-driving cars where they use models to identify objects like pedestrians, vehicles, stop signs, and green/yellow/red lights.

Automatic problem type detection

When setting a problem type with the AutoML API, you have the option of defining one, or letting Autopilot detect it on your behalf. When the job is finished, if you set a `ProblemType`, the `ResolvedAttribute`'s `ProblemType` will match the `ProblemType` you set. If you left it blank (or null), the `ProblemType` will be whatever Autopilot decides on your behalf.

Prepare and Label Data

Prepare and label data in Amazon SageMaker.

Topics

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [Using Amazon Augmented AI for Human Review \(p. 137\)](#)
- [Create and Manage Workforces \(p. 177\)](#)
- [HTML Elements Reference \(p. 189\)](#)

Use Amazon SageMaker Ground Truth for Labeling

To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models. With Ground Truth, you can use workers from either Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce along with machine learning to enable you to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

In order to automate labeling your training dataset, you can optionally use *automated data labeling*, a Ground Truth process that uses machine learning to decide which data needs to be labeled by humans. Automated data labeling may reduce the labeling time and manual effort required. For more information, see [Automate Data Labeling \(p. 91\)](#).

Use either pre-built or custom tools to assign the labeling tasks for your training dataset. A *labeling UI template* is a webpage that Ground Truth uses to present tasks and instructions to your workers. The Amazon SageMaker console provides built-in templates for labeling data. You can use these templates to get started, or you can build your own tasks and instructions by using our HTML 2.0 components. For more information, see [Creating Custom Labeling Workflows \(p. 114\)](#).

Use the workforce of your choice to label your dataset. You can choose your workforce from:

- The Amazon Mechanical Turk workforce of over 500,000 independent contractors worldwide.
- A private workforce that you create from your employees or contractors for handling data within your organization.
- A vendor company that you can find in the AWS Marketplace that specializes in data labeling services.

For more information, see [Create and Manage Workforces \(p. 177\)](#).

You store your datasets in Amazon S3 buckets. The buckets contain three things: The data to be labeled, an input manifest file that Ground Truth uses to read the data files, and an output manifest file. The output file contains the results of the labeling job. For more information, see [Use Input and Output Data \(p. 99\)](#).

Events from your labeling jobs appear in Amazon CloudWatch under the /aws/sagemaker/LabelingJobs group. CloudWatch uses the labeling job name as the name for the log stream.

Are You a First-time User of Ground Truth?

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

1. **Read [Getting started \(p. 65\)](#)**—This section walks you through setting up your first Ground Truth labeling job.
2. **Explore other topics**—Depending on your needs, do the following:
 - **Create instruction pages for your labeling jobs**—Create a custom instruction page that makes it easier for your workers to understand the requirements of the job. For more information, see [Creating Instruction Pages \(p. 112\)](#).
 - **Manage your labeling workforce**—Create new work teams and manage your existing workforce. For more information, see [Create and Manage Workforces \(p. 177\)](#).
 - **Create a custom UI**—Make it easier for your workers to quickly and correctly label your data by creating a custom UI for them to use. For more information, see [Creating Custom Labeling Workflows \(p. 114\)](#).
3. **See the [Reference](#)**—This section describes operations to automate Ground Truth operations.

Getting started

To get started using Amazon SageMaker Ground Truth, follow the instructions in the following sections. The sections here explain how to use the console to create a labeling job, assign a public or private workforce, and send the labeling job to your workforce. You can also learn how to monitor the progress of a labeling job.

If you want to create a custom labeling job, see [Creating Custom Labeling Workflows \(p. 114\)](#) for instructions.

Before you create a labeling job, you must upload your dataset to an Amazon S3 bucket. For more information, see [Use Input and Output Data \(p. 99\)](#).

Topics

- [Step 1: Before You Begin \(p. 65\)](#)
- [Step 2: Create a Labeling Job \(p. 66\)](#)
- [Step 3: Select Workers \(p. 67\)](#)
- [Step 4: Configure the Bounding Box Tool. \(p. 68\)](#)
- [Step 5: Monitoring Your Labeling Job \(p. 68\)](#)

Step 1: Before You Begin

Before you begin using the Amazon SageMaker console to create a labeling job, you must set up the dataset for use. Do this:

1. Save two images at publicly available HTTP URLs. The images are used when creating instructions for completing a labeling task. The images should have an aspect ratio of around 2:1. For this exercise, the content of the images is not important.
2. Create an Amazon S3 bucket to hold the input and output files. The bucket must be in the same Region where you are running Ground Truth. Make a note of the bucket name because you use it during step 2.
3. Place 5–10 PNG images in the bucket.
4. Create a manifest file for the dataset and store it in the S3 bucket. Use these steps:

- a. Using a text editor, create a new text file.
- b. Add a line similar to the following for each image file in your dataset:

```
{"source-ref": "s3://bucket/path/imageFile.png"}
```

Add one line for each PNG file in your S3 bucket.

- c. Save the file in the S3 bucket containing your source files. Record the name because you use it in step 2.

Note

It is not necessary to store the manifest file in the same bucket as the source file. You use the same bucket in this exercise because it is easier.

For more information, see [Input Data \(p. 99\)](#).

Assign the following permissions policy to the user that is creating the labeling job:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "sagemakergroundtruth",
            "Effect": "Allow",
            "Action": [
                "cognito-idp:CreateGroup",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolDomain",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:UpdateUserPool"
            ],
            "Resource": "*"
        }
    ]
}
```

Next

[Step 2: Create a Labeling Job \(p. 66\)](#)

Step 2: Create a Labeling Job

In this step you use the console to create a labeling job. You tell Amazon SageMaker Ground Truth the Amazon S3 bucket where the manifest file is stored and configure the parameters for the job. For more information about storing data in an Amazon S3 bucket, see [Use Input and Output Data \(p. 99\)](#).

To create a labeling job

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. From the left navigation, choose **Labeling jobs**.
3. Choose **Create labeling job** to start the job creation process.
4. In the **Job overview** section, provide the following information:

- **Job name** — Give the labeling job a name that describes the job. This name is shown in your job list. The name must be unique in your account in an AWS Region.
 - **Label attribute name** — Leave this unchecked as the default value is the best option for this introductory job.
 - **Input dataset location** — Enter the S3 location of the manifest file that you created in step 1.
 - **Output dataset location** — the location where your output data is written.
 - **IAM role** — Create or choose an IAM role with the SageMakerFullAccess IAM policy attached.
5. In the **Task type** section, for the **Dataset type** field, choose **Bounding box** as the task type.
 6. Choose **Next** to move on to configuring your labeling job.

Next

[Step 3: Select Workers \(p. 67\)](#)

Step 3: Select Workers

In this step you choose a workforce for labeling your dataset. You can create your own private workforce or you can use the Amazon Mechanical Turk workforce. If you create a private workforce in this step you won't be able to import your Amazon Cognito user pool later. For more information, see [Manage a Private Workforce \(p. 182\)](#). Use the Amazon Mechanical Turk workforce for this exercise instead.

You can create a private workforce to test Amazon SageMaker Ground Truth. Use email addresses to invite the members of your workforce.

To create a private workforce

1. In the **Workers** section, choose **Private**.
2. If this is your first time using a private workforce, in the **Email addresses** field, enter up to 100 email addresses. The addresses must be separated by a comma. You should include your own email address so that you are part of the workforce and can see data object labeling tasks.
3. In the **Organization name** field, enter the name of your organization. This information is used to customize the email sent to invite a person to your private workforce.
4. In the **Contact email** field enter an email address that members of the workforce use to report problems with the task.

If you choose to use the Amazon Mechanical Turk workforce to label the dataset, you are charged for labeling tasks completed on the dataset.

To use the Amazon Mechanical Turk workforce

1. In the **Workers** section, choose **Public**.
2. Choose **The dataset does not contain PII** to acknowledge that the dataset does not contain any personally identifiable information.
3. Choose **The dataset does not contain adult content**. to acknowledge that the sample dataset has no adult content.
4. Review and accept the statement that the dataset will be viewed by the public workforce.

Next

[Step 4: Configure the Bounding Box Tool. \(p. 68\)](#)

Step 4: Configure the Bounding Box Tool.

Finally you configure the bounding box tool to give instructions to your workers. You can configure a task title that describes the task and provides high-level instructions for the workers. You can provide both quick instructions and full instructions. Quick instructions are displayed next to the image to be labeled. Full instructions contain detailed instructions for completing the task. In this example, you only provide quick instructions. You can see an example of full instructions by choosing **Full instructions** at the bottom of the section.

To configure the bounding box tool

1. In the **Task description** field type in brief instructions for the task. For example:

Draw a box around any *objects* in the image.

Replace *objects* with the name of an object that appears in your images.

2. In the **Labels** field, type a category name for the objects that the worker should draw a bounding box around. For example, if you are asking the worker to draw boxes around football players, you could use "FootballPlayer" in this field.
3. The **Short instructions** section enables you to create instructions that are displayed on the page with the image that your workers are labeling. We suggest that you include an example of a correctly drawn bounding box and an example of an incorrectly drawn box. To create your own instructions, use these steps:

- a. Select the text between **GOOD EXAMPLE** and the image placeholder. Replace it with the following text:

Draw the box around the object with a small border.

- b. Select the first image placeholder and delete it.
- c. Choose the image button and then enter the HTTPS URL of one of the images that you created in step 1.
- d. Select the text between **BAD EXAMPLE** and the image placeholder. Replace it with the following text:

Don't make the bounding box too large or cut into the object.

- e. Select the second image placeholder and delete it.
- f. Choose the image button and then enter the HTTPS URL of the other image that you created in step 1.

Configuration of your labeling job is complete. To start your job, choose **Submit**.

Next

[Step 5: Monitoring Your Labeling Job \(p. 68\)](#)

Step 5: Monitoring Your Labeling Job

After you create your labeling job, you see a list of all the jobs that you have created. You can use this list to monitor the status of your labeling jobs. The list has the following fields:

- **Name**—The name that you assigned the job when you created it.
- **Status**—The completion status of the job. The status can be one of Complete, Failed, In progress, or Stopped.
- **Labeled objects/total**—Shows the total number of objects in the labeling job and how many of them have been labeled.

- **Creation time**—The date and time that you created the job.

You can also clone, chain, or stop a job. Select a job and then select one of the following from the **Actions** menu:

- **Clone**—Creates a new labeling job with the configuration copied from the selected job. You can clone a job when you want to change to the job and run it again. For example, you can clone a job that was sent to a private workforce so that you can send it to the Amazon Mechanical Turk workforce. Or you can clone a job to rerun it against a new dataset stored in the same location as the original job.
- **Chain**—Creates a new labeling job that can build upon the data and models (if any) of a stopped, failed, or completed job. For more information about the use cases and how to use it, see [Chaining Labeling Jobs \(p. 94\)](#).
- **Stop**—Stops a running job. You cannot restart a stopped job. You can clone a job to start over or chain the job to continue from where it left off. Labels for any already labeled objects are written to the output file location. For more information, see [Output Data \(p. 103\)](#).

Built in Task Types

Amazon SageMaker Ground Truth has eight built in task types. These are image- and text-based labeling jobs that use Amazon SageMaker built in algorithms. Ground Truth provides worker task templates in the console for built in task types. The following topics describe each built in task type and demo the worker task templates that are provided by Ground Truth in the console. To learn how to create a labeling job in the console using one of these task types, see [Getting started \(p. 65\)](#).

If you are creating a labeling job using the API, you will need to provide a custom worker task template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). For a repository of demo templates for a variety of labeling job task types, see [Amazon SageMaker Ground Truth Sample Task UIs](#).

Topics

- [Bounding Box \(p. 69\)](#)
- [Image Classification \(p. 71\)](#)
- [Image Classification \(Multi-label\) \(p. 72\)](#)
- [Semantic Segmentation \(p. 77\)](#)
- [Label Verification \(p. 79\)](#)
- [Text Classification \(p. 80\)](#)
- [Text Classification \(Multi-label\) \(p. 80\)](#)
- [Named Entity Recognition \(p. 85\)](#)

Bounding Box

The images used to train a machine learning model often contain more than one object. To classify and localize one or more objects within images, use the Amazon SageMaker Ground Truth bounding box labeling job task type. In this context, localization means the pixel-location of the bounding box. For example, the output manifest file of a successfully completed single-class bounding box task will contain the following:

```
[  
 {  
   "boundingBox": {  
     "boundingBoxes": [  
       {  
         "x": 100,  
         "y": 100,  
         "width": 200,  
         "height": 150  
       }  
     ]  
   }  
 }
```

```
{
    "height": 2833,
    "label": "bird",
    "left": 681,
    "top": 599,
    "width": 1364
},
],
"inputImageProperties": {
    "height": 3726,
    "width": 2662
}
}
]
```

The `boundingBoxes` parameter identifies the location of the bounding box drawn around an object identified as a "bird" relative to the top-left corner of the image which is taken to be the (0,0) pixel-coordinate. In the previous example, `left` and `top` identify the location of the pixel in the top-left corner of the bounding box relative to the top-left corner of the image. The dimensions of the bounding box are identified with `height` and `width`. The `inputImageProperties` parameter gives the pixel-dimensions of the original input image.

When you use the bounding box task type, you can create single- and multi-class bounding box labeling jobs. The output manifest file of a successfully completed multi-class bounding box will contain the following:

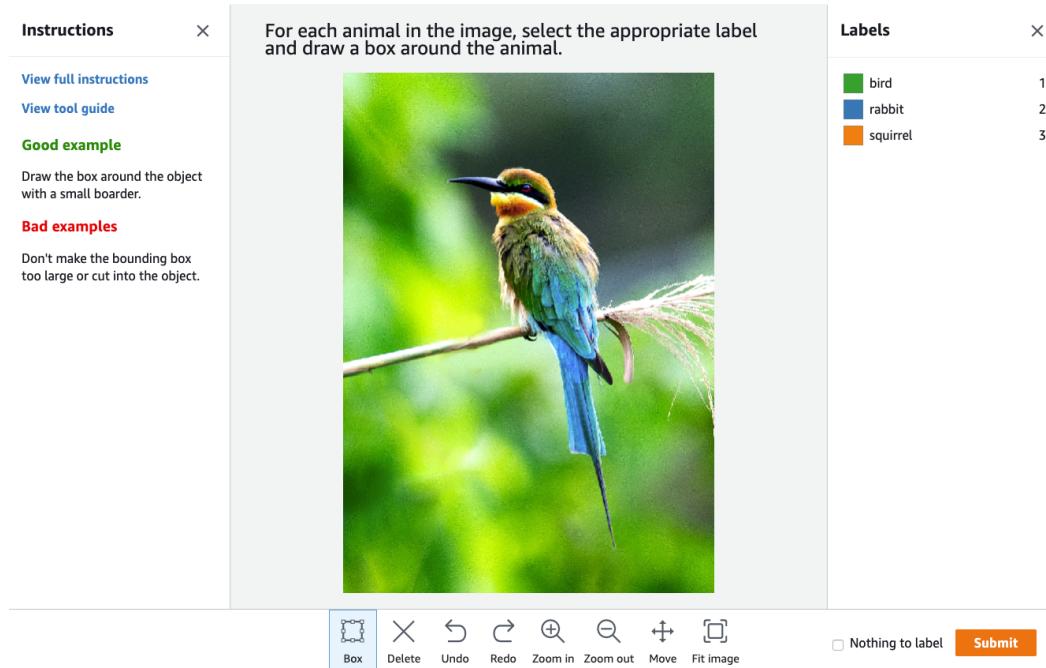
```
[
{
    "boundingBox": {
        "boundingBoxes": [
            {
                "height": 938,
                "label": "squirrel",
                "left": 316,
                "top": 218,
                "width": 785
            },
            {
                "height": 825,
                "label": "rabbit",
                "left": 1930,
                "top": 2265,
                "width": 540
            },
            {
                "height": 1174,
                "label": "bird",
                "left": 748,
                "top": 2113,
                "width": 927
            },
            {
                "height": 893,
                "label": "bird",
                "left": 1333,
                "top": 847,
                "width": 736
            }
        ],
        "inputImageProperties": {
            "height": 3726,
            "width": 2662
        }
    }
]
```

```
}
```

To learn more about the output manifest file that results from a bounding box labeling job, see [Bounding Box Job Output \(p. 107\)](#).

Creating a Bounding Box Labeling Job

You create a bounding box labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown in the following image. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for bounding box labeling job types, see this [GitHub Repository](#).



To learn how to start a bounding box labeling job in the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Image Classification

Use an Amazon SageMaker Ground Truth image classification labeling task when you need workers to classify images using predefined labels that you specify. Workers are shown images and are asked to choose one label for each image.

You create an image classification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for image classification job types, see this [Github Repository](#).

Instructions ×

[View full instructions](#)
[View tool guide](#)

You must select one label for each image. Once you have selected a label, click **Submit**.

Please identify the image by selecting the appropriate label on the right.



Select an option

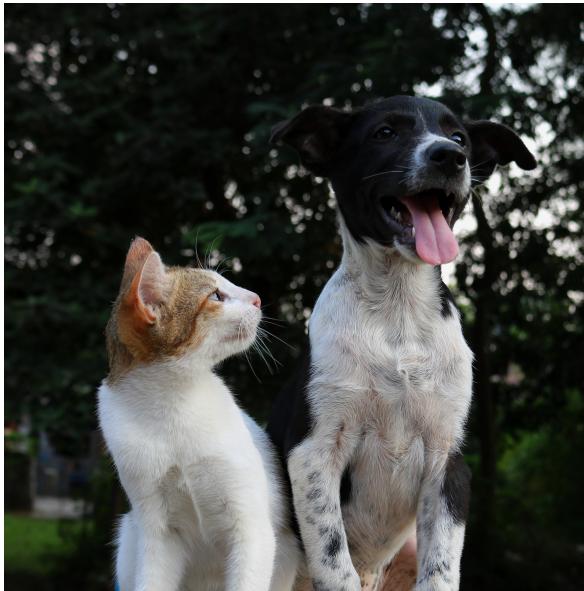
bird	1
squirrel	2
rabbit	3

⊕ ⊖ ✎ □ **Submit**

You can create an image classification labeling job using the Amazon SageMaker console or API. To learn how to start an image classification labeling job using the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Image Classification (Multi-label)

Use an Amazon SageMaker Ground Truth multi-label image classification labeling task when you need workers to classify multiple objects in an image. For example, the following image features a dog and a cat. You can use multi-label image classification to associate the labels "dog" and "cat" with this image.



When working on a multi-label image classification task, workers should choose all applicable labels, but must choose at least one. When creating a job using this task type, you can provide up to 50 label-categories.

When creating a labeling job in the console, Amazon SageMaker Ground Truth doesn't provide a "none" category for when none of the labels applies to an image. To provide this option to workers, include a label similar to "none" or "other" when you create a multi-label image classification job.

When you create a multi-label image classification job in the console, you will be provided with a default worker UI template that you can use to:

- Specify the labels that you want the worker to see
- Create worker instructions

See [Create a Multi-Label Image Classification Labeling Job \(Console\) \(p. 74\)](#) for an example of the template provided in the console.

When you create a multi-label image classification job using the Amazon SageMaker API or your preferred Amazon SageMaker SDK, you will need to provide a custom worker task template which will be used to generate your workers' UI. For more information, see [Create a Custom Template for Multi-label Image Classification \(p. 76\)](#).

To restrict workers to choosing a single label for each image, use the [Image Classification \(p. 71\)](#) task type.

Note

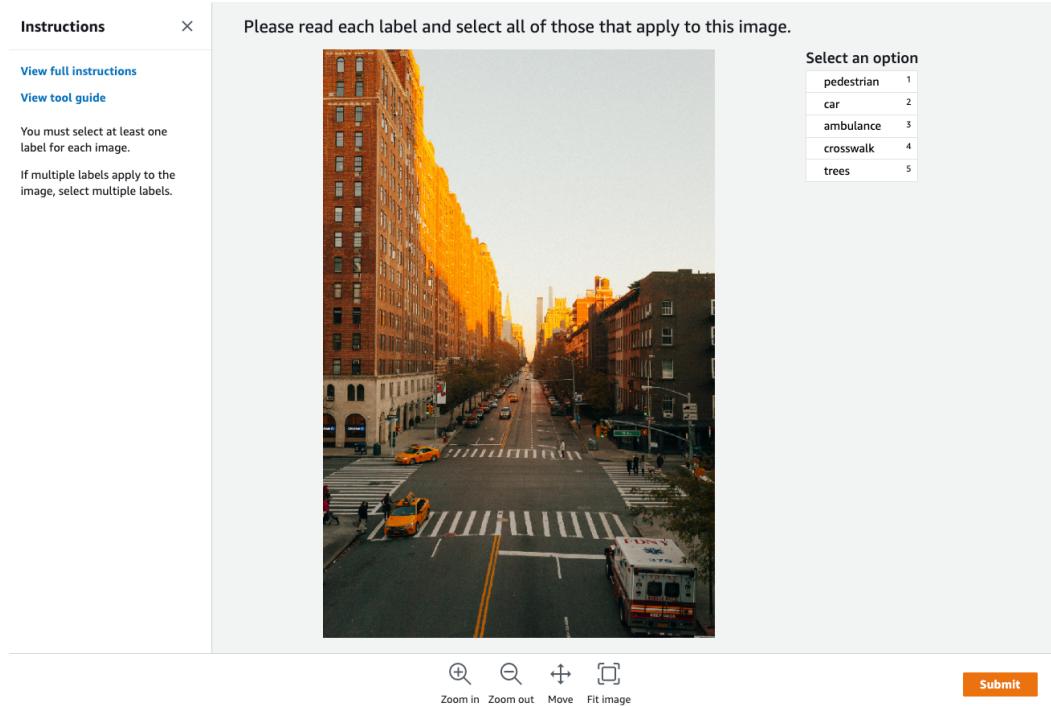
Automated labeling (auto-labeling) isn't supported for the multi-label image classification task type.

Topics

- [Create a Multi-Label Image Classification Labeling Job \(Console\) \(p. 74\)](#)
- [Create a Multi-Label Image Classification Labeling Job \(API\) \(p. 74\)](#)
- [Multi-label Image Classification Output Data \(p. 77\)](#)

Create a Multi-Label Image Classification Labeling Job (Console)

To create a multi-label image classification labeling job, use the Ground Truth section of the Amazon SageMaker console. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown.



To create a multi-label image classification labeling job, use the instructions in the [Getting started \(p. 65\)](#) guide. When following [Step 2: Create a Labeling Job \(p. 66\)](#), choose **Image classification (Multi-label)** for **Task type**.

Create a Multi-Label Image Classification Labeling Job (API)

To create a multi-image label classification labeling job using the Amazon SageMaker API, you use the `CreateLabelingJob` operation. To use this operation, you need:

- A custom worker task template. For information, see [Create a Custom Template for Multi-label Image Classification \(p. 76\)](#).
- At least one Amazon Simple Storage Service (Amazon S3) bucket to store your input and output data.
- An input manifest file that specifies your input data. For information about creating an input manifest, see [Input Data \(p. 99\)](#).
- An AWS Identity and Access Management (IAM) role with the `AmazonSageMakerFullAccess` IAM policy attached and with permissions to access your S3 buckets. For example, if the images specified in your manifest file are in an S3 bucket named `my_input_bucket`, and if you want the image-labeling data to be stored in a bucket named `my_output_bucket`, you would attach the following IAM policy to the role that is passed to the `CreateLabelingJob` operation.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:ListBucket"  
            ]  
        }  
    ]  
}
```

```

        "s3:GetObject"
    ],
    "Resource": [
        "arn:aws:s3:::my_input_bucket/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::my_output_bucket/*"
    ]
}
]
}

```

- A pre-annotation AWS Lambda function ARN to process your input data. Lambda functions are predefined in each AWS Region for multi-label image classification labeling jobs. Pre-annotation Lambda functions for this task type end with `PRE-ImageMultiClassMultiLabel`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- A work team Amazon Resource Name (ARN). To learn more about work teams and workforces, see [Create and Manage Workforces \(p. 177\)](#).

If you use the Amazon Mechanical Turk public workforce, use the `ContentClassifiers` parameter in `CreateLabelingJob` to declare that your content is free of personally identifiable information or adult content. If your content contains personally identifiable information or adult content, Amazon SageMaker might restrict the Amazon Mechanical Turk workers that can view your task.

- (Optional) To have multiple workers label a single image (by inputting a number greater than one for the `NumberOfHumanWorkersPerDataObject` parameter), include an annotation-consolidation Lambda ARN as an input to the `AnnotationConsolidationLambdaArn` parameter. These Lambda functions are predefined in each Region for multi-label image classification labeling jobs. Annotation-consolidation Lambda functions for this task type end with `ACS-ImageMultiClassMultiLabel`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [AWS Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region.

```

response = client.create_labeling_job(
    LabelingJobName='example-multi-label-image-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
)

```

```

StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
    'UiConfig': {
        'UiTemplateS3Uri': 's3://bucket/path/custom-worker-task-template.html'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
ImageMultiClassMultiLabel',
    'TaskKeywords': [
        'Image Classification',
    ],
    'TaskTitle': 'Multi-label image classification task',
    'TaskDescription': 'Select all labels that apply to the images shown',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-ImageMultiClassMultiLabel'
    },
    Tags:[
        {
            'Key': 'string',
            'Value': 'string'
        },
    ]
}

```

For more information about this operation, see [CreateLabelingJob](#). For information about how to use other language-specific SDKs, see [See Also](#) in the [CreateLabelingJobs](#) topic.

Create a Custom Template for Multi-label Image Classification

If you create a labeling job using the API, you must create and supply a custom worker task template. You can use the Amazon SageMaker `crowd-image-classifier-multi-select` HTML crowd element to create a template for a multi-label image classification job. The following example shows how to use this crowd element.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-image-classifier-multi-select
        name="animals"
        categories="['Cat', 'Dog', 'Horse', 'Pig', 'Bird']"
        src="https://images.unsplash.com/photo-1509205477838-a534e43a849f?
ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDD9&auto=format&fit=crop&w=1998&q=80"
        header="Please identify the animals in this image"
    >
        <full-instructions header="Classification Instructions">
            <p>If more than one label applies to the image, select multiple labels.</p>
            <p>If no labels apply, select <b>None of the above</b></p>
        </full-instructions>

        <short-instructions>
            <p>Read the task carefully and inspect the image.</p>
            <p>Choose the appropriate label(s) that best suit the image.</p>
        </short-instructions>
    </crowd-image-classifier-multi-select>
</crowd-form>

```

To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#).

Multi-label Image Classification Output Data

Once you have created a multi-label image classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 103\)](#).

To see an example of output manifest files for multi-label image classification labeling job, see [Multi-label Classification Job Output \(p. 106\)](#).

Semantic Segmentation

To identify the contents of an image at the pixel level, use an Amazon SageMaker Ground Truth semantic segmentation labeling task. When assigned a semantic segmentation labeling job, workers classify pixels in the image into a set of predefined labels or classes. Ground Truth supports single and multi-class semantic segmentation labeling jobs.

You create a semantic segmentation labeling job using the Ground Truth section of the Amazon SageMaker console or the `CreateLabelingJob` operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for semantic segmentation labeling job types, see this [Github Repository](#).

Instructions

X

For each animal in the image, the app...

[View full instructions](#)

[View tool guide](#)

[How to use the Auto-segmentation tool](#)

Good example

All pixels in the image that are part of an animal have been colored with the appropriate label color.

Bad example

Some animals in the image have not been colored in completely.

The color for a given animal extends beyond the boundaries of the animal.

Images that contain large numbers of objects that need to be segmented require more time. To help workers label these objects in less time and with greater accuracy, Ground Truth provides an AI-assisted auto-segmentation tool. For information, see [Auto-Segmentation Tool \(p. 86\)](#).

Note

The auto-segmentation tool is available in all segmentation tasks that are sent to a private workforce or vendor workforce. It isn't available for tasks sent to the public workforce (Amazon Mechanical Turk).

You can create a semantic segmentation labeling job using the Amazon SageMaker console or API. To learn how to start a semantic segmentation labeling job on the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Label Verification

Building a highly accurate training dataset for your machine learning (ML) algorithm is an iterative process. Typically, you review and continuously adjust your labels until you are satisfied that they accurately represent the ground truth, or what is directly observable in the real world.

You can use an Amazon SageMaker Ground Truth label verification task to direct workers to review a dataset's labels and improve label accuracy. Workers can indicate if the existing labels are correct or rate label quality. They can also add comments to explain their reasoning. Amazon SageMaker Ground Truth supports label verification for [Bounding Box \(p. 69\)](#) and [Semantic Segmentation \(p. 77\)](#) labels.

You create a label verification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for label verification labeling job types, see this [Github Repository](#).

Instructions

Review the existing labels on the objects and choose the appropriate option.

[View full instructions](#)

[View tool guide](#)

Existing labels

- bird
- rabbit
- squirrel

Instructions

Please review the labels selected and corresponding box(es) draw for each animal in the image. If the incorrect animal has been selected, or the box has been incorrectly drawn choose **reject**. Otherwise, choose **accept**.

About existing labels

Select the appropriate label to identify the animal and draw a box around the animal.



Select an option

accept	1
reject	2

[Add a comment](#)

Dimmer Zoom in Zoom out Move Fit image

Submit

You can create a label verification labeling job using the Amazon SageMaker console or API. To learn how to start a label verification job on the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Text Classification

To categorize articles and text into predefined categories, use text classification. For example, you can use text classification to identify the sentiment conveyed in a review or the emotion underlying a section of text. Use Amazon SageMaker Ground Truth text classification to have workers sort text into categories that you define.

You create a text classification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for text classification job types, see this [Github Repository](#).

The screenshot shows a labeling interface with the following components:

- Instructions**: A sidebar containing:
 - [View full instructions](#)
 - [View tool guide](#)
 - If you are unsure if the review is positive or negative, choose unclear.
 - You can only select one label for each review.
 - Click **Submit** when finished.
- Select the label that best describes the sentiment in the review below.**: A text input field containing the review: "The movie tells a lovely and wise story with honesty and has been acted out with unassuming grace."
- Select an option**: A table showing three options:

positive	1
negative	2
unclear	3
- Submit**: A button at the bottom right of the interface.

You can create a text classification labeling job using the Amazon SageMaker console or API. To learn how to start a text classification labeling job on the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Text Classification (Multi-label)

To categorize articles and text into multiple predefined categories, use the multi-label text classification task type. For example, you can use this task type to identify more than one emotion conveyed in text.

When working on a multi-label text classification task, workers should choose all applicable labels, but must choose at least one. When creating a job using this task type, you can provide up to 50 label categories.

Amazon SageMaker Ground Truth doesn't provide a "none" category for when none of the labels applies. To provide this option to workers, include a label similar to "none" or "other" when you create a multi-label text classification job.

When you create a multi-label text classification job in the console, you will be provided with a default worker UI template that you can use to:

- Specify the labels that you want the worker to see
- Create worker instructions

See [Create a Multi-Label Text Classification Labeling Job \(Console\) \(p. 81\)](#) for an example of the template provided in the console.

When you create a multi-label text classification job using the Amazon SageMaker API or your preferred Amazon SageMaker SDK, you will need to provide a custom worker task template which will be used to generate your workers' UI. For more information, see [Create a Custom Template for Multi-label Text Classification \(p. 84\)](#).

To restrict workers to choosing a single label for each document or text selection, use the [Text Classification \(p. 80\)](#) task type.

Note

Automated labeling (auto-labeling) isn't supported for the multi-label text classification task type.

Topics

- [Create a Multi-Label Text Classification Labeling Job \(Console\) \(p. 81\)](#)
- [Create a Multi-Label Text Classification Labeling Job \(API\) \(p. 82\)](#)
- [Multi-label Text Classification Output Data \(p. 85\)](#)

Create a Multi-Label Text Classification Labeling Job (Console)

To create a multi-label text classification labeling job, you use the Ground Truth section of the Amazon SageMaker console. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the contents that are shown.

Instructions ×

[View full instructions](#)

[View tool guide](#)

You must select at least one label.

If you think that multiple sentiments are expressed in this review, select multiple labels.

Read the review below and select the sentiments that are expressed by the writer.

'The movie tells a lovely and wise story with honestly that has been acted with unassuming grace. However, some of the scenes were slow.'

Select an option

Positive	1
Negative	2
Critical	3
Confused	4

Submit

To create a multi-label text classification labeling job, use the instructions in the [Getting started \(p. 65\)](#) guide: For [Step 2: Create a Labeling Job \(p. 66\)](#), choose **Text classification (Multi-label)** as the **Task type**.

Create a Multi-Label Text Classification Labeling Job (API)

To create a multi-label text classification labeling job using the Amazon SageMaker API, you use the `CreateLabelingJob` operation. To use this operation, you need:

- A custom worker task template. For information, see [Create a Custom Template for Multi-label Text Classification \(p. 84\)](#).
- At least one Amazon Simple Storage Service (Amazon S3) bucket to store your input and output data.
- An input manifest file that specifies your input data. For information about creating an input manifest, see [Input Data \(p. 99\)](#).
- An AWS Identity and Access Management (IAM) role with the `AmazonSageMakerFullAccess` IAM policy attached and with permissions to access your S3 buckets. For example, if the text or documents specified in your manifest file are in an S3 bucket named `my_input_bucket`, and if you want the label data to be stored in a bucket named `my_output_bucket`, you would attach the following IAM policy to the role that is passed to the `CreateLabelingJob` operation.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::my_output_bucket/*"  
        }  
    ]  
}
```

```

        "Action": [
            "s3:GetObject"
        ],
        "Resource": [
            "arn:aws:s3:::my_input_bucket/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::my_output_bucket/*"
        ]
    }
]
}

```

- The ARN for a pre-annotation AWS Lambda function that will be used to process your input data. These Lambda functions are predefined in each AWS Region for multi-label text classification labeling jobs. Pre-annotation Lambda functions for this task type end with `PRE-TextMultiClassMultiLabel`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- A work team ARN. To learn about work teams and workforces, see [Create and Manage Workforces \(p. 177\)](#).

If you use the Amazon Mechanical Turk public workforce, use the `ContentClassifiers` parameter in `CreateLabelingJob` to declare that your content is free of personally identifiable information or adult content. If your content contains personally identifiable information or adult content, Amazon SageMaker might restrict the Amazon Mechanical Turk workers that can view your task.

- (Optional) To have multiple workers label a single text passage (by inputting a number greater than one for the `NumberOfHumanWorkersPerDataObject` parameter), include an annotation-consolidation Lambda ARN as an input to the `AnnotationConsolidationLambdaArn` parameter. These Lambda functions are predefined in each AWS Region for multi-label text classification labeling jobs. Annotation-consolidation Lambda functions for this task type end with `ACS-TextMultiClassMultiLabel`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [AWS Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region.

```

response = client.create_labeling_job(
    LabelingJobName='example-multi-label-text-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
)

```

```

RoleArn='arn:aws:iam::*:role/*',
LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:region*:workteam/private-crowd/*',
    'UiConfig': {
        'UiTemplateS3Uri': 's3://bucket/path/custom-worker-task-template.html'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:function:PRE-TextMultiClassMultiLabel',
    'TaskKeywords': [
        'Text Classification',
    ],
    'TaskTitle': 'Multi-label text classification task',
    'TaskDescription': 'Select all labels that apply to the text shown',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-TextMultiClassMultiLabel'
    },
    Tags:[
        {
            'Key': 'string',
            'Value': 'string'
        },
    ]
}
)

```

For more information about this operation, see [CreateLabelingJob](#). For information about how to use other language-specific SDKs, see [See Also](#) in the [CreateLabelingJobs](#) topic.

Create a Custom Template for Multi-label Text Classification

If you create a labeling job using the API, you must create and supply a custom worker task template. You can use the Amazon SageMaker [crowd-classifier-multi-select](#) HTML crowd element to create a template for a multi-label text classification job. The following example shows how to use this crowd element.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-classifier-multi-select
        name="category"
        categories="['Positive', 'Negative', 'Neutral']"
        header="Select the relevant categories"
    >
        <classification-target>
            {{ task.input.taskObject }}
        </classification-target>

        <full-instructions header="Text Categorization Instructions">
            <p><strong>Positive</strong> sentiment include: joy, excitement, delight</p>
            <p><strong>Negative</strong> sentiment include: anger, sarcasm, anxiety</p>
            <p><strong>Neutral</strong>: neither positive or negative, such as stating a fact</p>
            <p><strong>N/A</strong>: when the text cannot be understood</p>
            <p>When the sentiment is mixed, such as both joy and sadness, use your judgment to
            choose the stronger emotion.</p>
        </full-instructions>

```

```
<short-instructions>
  Choose all categories that are expressed by the text.
</short-instructions>
</crowd-classifier-multi-select>
</crowd-form>
```

To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#).

Multi-label Text Classification Output Data

Once you have created a multi-label text classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 103\)](#).

To see an example of output manifest files for multi-label text classification labeling job, see [Multi-label Classification Job Output \(p. 106\)](#).

Named Entity Recognition

To extract information from unstructured text and classify it into predefined categories, use an Amazon SageMaker Ground Truth named entity recognition (NER) labeling task. Traditionally, NER involves sifting through text data to locate noun phrases, called *named entities*, and categorizing each with a label, such as "person," "organization," or "brand." You can broaden this task to label longer spans of text and categorize those sequences with predefined labels that you specify.

When tasked with a named entity recognition labeling job, workers apply your labels to specific words or phrases within a larger text block. They choose a label, then apply it by using the cursor to highlight the part of the text to which the label applies. Workers can't apply multiple labels to the same text, and labels can't overlap.

You create a named entity recognition labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation. Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. If you create a labeling job using the API, you must supply a custom-built template. To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 114\)](#). To see examples of custom templates that can be used for named entity recognition labeling job types, see this [Github Repository](#).

Instructions

[View full instructions](#)

[View tool guide](#)

Highlight every word that is associated with the quantity. The quantity may be more than one word. For example, highlight **3 1/2 cups**.

Highlight every word that is associated with the ingredient. The ingredient may be more than one word. For example, highlight **green apple**.

Select a label on the right and identify the quantity and ingredient in the text.

3 cups of flour

Labels

Quantity	1
Ingredient	2

No entities to label
Submit

You can create a named entity recognition labeling job using the Amazon SageMaker console or API. To learn how to start a named entity recognition labeling job using on the console, see [Getting started \(p. 65\)](#). To use the API, see [CreateLabelingJob](#).

Auto-Segmentation Tool

Image segmentation is the process of dividing an image into multiple segments, or sets of labeled pixels. In Amazon SageMaker Ground Truth, the process of identifying all pixels that fall under a given label involves applying a colored filler, or "mask", over those pixels. Some labeling job tasks contain images with a large numbers of objects that need to be segmented. To help workers label these objects in less time and with greater accuracy, Ground Truth provides an auto-segmentation tool for segmentation tasks assigned to private and vendor workforces. This tool uses a machine learning model to automatically segment individual objects in the image with minimal worker input. Workers can refine the mask generated by the auto-segmentation tool using other tools found in the worker console. This helps workers complete image segmentation tasks faster and more accurately, resulting in lower cost and higher label quality.

Note

The auto-segmentation tool is available for segmentation tasks that are sent to a private workforce or vendor workforce. It isn't available for tasks sent to the public workforce (Amazon Mechanical Turk).

Tool Preview

When workers are assigned a labeling job that provides the auto-segmentation tool, they are provided with detailed instructions on how to use the tool. For example, a worker might see the following in the worker console:

Instructions

X

For each animal in the image, the app...

[View full instructions](#)

[View tool guide](#)

[How to use the Auto-segment tool](#)

Good example

All pixels in the image that are part of an animal have been colored with the appropriate label color.

Bad example

Some animals in the image have not been colored in completely.

The color for a given animal extends beyond the boundaries of the animal.



Workers can use [View full instructions](#) to learn how to use the tool. Workers will need to place a point on four extreme-points (top-most, bottom-most, left-most, and right-most points) of the object of interest, and the tool will automatically generate a mask for the object. Workers can further-refine the mask using the other tools provided, or by using the auto-segment tool on smaller portions of the object that were missed.

Tool Availability

The auto-segmentation tool automatically appears in your workers' consoles if you create a semantic segmentation labeling job using the Amazon SageMaker console. While creating a semantic segmentation job in the Amazon SageMaker console, you will be able to preview the tool while creating worker instructions. To learn how to create a semantic segmentation labeling job in the Amazon SageMaker console, see [Getting started \(p. 65\)](#).

If you are creating a custom instance segmentation labeling job in the Amazon SageMaker console or creating an instance- or semantic-segmentation labeling job using the Ground Truth API, you need to create a custom task template to design your worker console and instructions. To include the auto-segmentation tool in your worker console, ensure that the following conditions are met in your custom task template:

- For semantic segmentation labeling jobs created using the API, the `<crowd-semantic-segmentation>` is present in the task template. For custom instance segmentation labeling jobs, the `<crowd-instance-segmentation>` tag is present in the task template.
- The task is assigned to a private workforce or vendor workforce.
- The images to be labeled are Amazon Simple Storage Service (Amazon S3) objects that have been pre-signed for the Worker so that they can access it. This is true if the task template includes the `grant_read_access` filter. For information about the `grant_read_access` filter, see [Adding automation with Liquid \(p. 118\)](#).

The following is an example of a custom task template for a custom instance segmentation labeling job, which includes the `<crowd-instance-segmentation>` tag and the `grant_read_access` Liquid filter.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-instance-segmentation
    name="crowd-instance-segmentation"
    src="{{ task.input.taskObject | grant_read_access }}"
    labels="['Car', 'Road']"
    <full-instructions header="Segmentation instructions">
      Segment each instance of each class of objects in the image.
    </full-instructions>

    <short-instructions>
      <p>Segment each instance of each class of objects in the image.</p>

      <h3 style="color: green">GOOD EXAMPLES</h3>
      
      <p>Good because A, B, C.</p>

      <h3 style="color: red">BAD EXAMPLES</h3>
      
      <p>Bad because X, Y, Z.</p>
    </short-instructions>
  </crowd-instance-segmentation>
</crowd-form>
```

Data Labeling

Amazon SageMaker Ground Truth manages sending your data objects to workers to be labeled. Labeling each data object is a *task*. Workers complete each task until the entire labeling job is complete. Ground Truth divides the total number of tasks into smaller *batches* that are sent to workers. A new batch is sent to workers when the previous one is finished.

Ground Truth provides two features that help improve the accuracy of your data labels and reduce the total cost of labeling your data:

- *Annotation consolidation* helps to improve the accuracy of your data object's labels. It combines the results of multiple worker's annotation tasks into one high-fidelity label.
- *Automated data labeling* uses machine learning to label portions of your data automatically without having to send them to human workers.

Topics

- [Batches for Labeling Tasks \(p. 89\)](#)
- [Consolidate Annotations \(p. 89\)](#)
- [Automate Data Labeling \(p. 91\)](#)
- [Chaining Labeling Jobs \(p. 94\)](#)
- [Verify and Adjust Labels \(p. 96\)](#)

Batches for Labeling Tasks

Amazon SageMaker Ground Truth sends data objects to your workers in batches. There are one or more tasks for each data object. For each task, a worker annotates one of your data objects. A batch does the following:

- Sets the number of data objects that are available to workers. After the objects are annotated, another batch is sent.
- Breaks the work into smaller chunks to avoid overloading your workforce.
- Provides chunks of data for the iterative training of automated data labeling models.

Ground Truth first sends a batch of 10 tasks to your workers. It uses this small batch to set up the labeling job and to make sure that the job is correctly configured.

Ground Truth then sends larger batches to your workers.

You configure batch size when you create the job using the [CreateLabelingJob](#) operation. If you use the Amazon SageMaker console to create the labeling job, Ground Truth automatically configures your job to use 1,000 tasks in each batch.

Consolidate Annotations

An *annotation* is the result of a single worker's labeling task. *Annotation consolidation* combines the annotations of two or more workers into a single label for your data objects. A label, which is assigned to each object in the dataset, is a probabilistic estimate of what the true label should be. Each object in the dataset typically has multiple annotations, but only one label or set of labels.

You can decide how many workers should annotate each object in your dataset. More workers can increase the accuracy of your labels, but also increases the cost of labeling. If you use the Amazon

SageMaker console to create a labeling job, the following are the defaults for the number of workers who can annotate objects:

- Text classification—3 workers
- Image classification—3 workers
- Bounding boxes—5 workers
- Semantic segmentation—3 workers
- Named entity recognition—3 workers

When you use the [CreateLabelingJob](#) operation, you set the number of workers to annotate each data object with the `NumberOfHumanWorkersPerDataObject` parameter. You can override the default number of workers that annotate a data object using the console or the [CreateLabelingJob](#) operation.

Ground Truth provides an annotation consolidation function for each of its predefined labeling tasks: bounding box, image classification, name entity recognition, semantic segmentation, and text classification. These are the functions:

- Multi-class annotation consolidation for image and text classification uses a variant of the [Expectation Maximization](#) approach to annotations. It estimates parameters for each worker and uses Bayesian inference to estimate the true class based on the class annotations from individual workers.
- Bounding box annotation consolidates bounding boxes from multiple workers. This function finds the most similar boxes from different workers based on the [Jaccard index](#), or intersection over union, of the boxes and averages them.
- Semantic segmentation annotation consolidation treats each pixel in a single image as a multi-class classification. This function treats the pixel annotations from workers as "votes," with more information from surrounding pixels incorporated by applying a smoothing function to the image.
- Named entity recognition clusters text selections by Jaccard similarity and calculates selection boundaries based on the mode, or median if the mode isn't clear. The label resolves to the most assigned entity label in the cluster, breaking ties by random selection.

You can use other algorithms to consolidate annotations. For information, see [Create Your Own Annotation Consolidation Function \(p. 90\)](#).

Create Your Own Annotation Consolidation Function

You can choose to use your own annotation consolidation function to determine the final labels for your labeled objects. There are many possible approaches for writing a function and the approach that you take depends on the nature of the annotations to consolidate. Broadly, consolidation functions look at the annotations from workers, measure the similarity between them, and then use some form of probabilistic judgment to determine what the most probable label should be.

If you want to use other algorithms to create annotation consolidations functions, you can find the worker responses in the `[project-name]/annotations/worker-response` folder of the Amazon S3 bucket where you direct the job output.

Assess Similarity

To assess the similarity between labels, you can use one of the following strategies, or you can use one that meets your data labeling needs:

- For label spaces that consist of discrete, mutually exclusive categories, such as multi-class classification, assessing similarity can be straightforward. Discrete labels either match or not.
- For label spaces that don't have discrete values, such as bounding box annotations, find a broad measure of similarity. For bounding boxes, one such measure is the Jaccard index. This measures

the ratio of the intersection of two boxes with the union of the boxes to assess how similar they are. For example, if there are three annotations, then there can be a function that determines which annotations represent the same object and should be consolidated.

Assess the Most Probable Label

With one of the above strategies in mind, make some sort of probabilistic judgment on what the consolidated label should be. In the case of discrete, mutually exclusive categories, this can be straightforward. One of the most common ways to do this is to take the results of a majority vote between the annotations. This weights the annotations equally.

Some approaches attempt to estimate the accuracy of different annotators and weight their annotations in proportion to the probability of correctness. An example of this is the Expectation Maximization method, which is used in the default Ground Truth consolidation function for multi-class annotations.

For more information about creating an annotation consolidation function, see [Step 3: Processing with AWS Lambda \(p. 132\)](#).

Automate Data Labeling

If you choose, Amazon SageMaker Ground Truth can use active learning to automate the labeling of your input data. *Active learning* is a machine learning technique that identifies data that should be labeled by your workers. In Ground Truth, this functionality is called automated data labeling. Automated data labeling helps to reduce the cost and time that it takes to label your dataset compared to using only humans. When you use automated labeling, you incur Amazon SageMaker training and inference costs.

We recommend using automated data labeling on large datasets because the neural networks used with active learning require a significant amount of data for every new dataset. Typically, as more data is provided, the potential for high accuracy predictions goes up. Data will only be auto-labeled if the neural network used in the auto-labeling model can achieve an acceptably high level of accuracy. Therefore, with larger datasets, there is more potential to automatically label the data because the neural network can achieve high enough accuracy for auto-labeling. Automated data labeling is most appropriate when you have thousands of data objects. The minimum number of objects allowed for automated data labeling is 1,250, but we strongly suggest providing a minimum of 5,000 objects.

You enable automated data labeling when you create a labeling job. This is how it works:

1. When Ground Truth starts an automated data labeling job, it selects a random sample of input data (objects) and sends it to human workers.
2. When the labeled data are returned, Ground Truth uses this data, the validation data, to validate the models trained for automated data labeling.
3. Ground Truth runs a batch transform job, using the validated model for inference on the validation data. Batch inference produces a confidence score and quality metric for each object in the validation data.
4. The auto labeling component will use these quality metrics and confidence scores to create a confidence score threshold that ensure quality labels.
5. Ground Truth runs a batch transform job on the unlabeled data in the dataset, using the same validated model for inference. This will produce a confidence score for each object.
6. The Ground Truth auto labeling component determines if the confidence score produced in step 5 for each object meets the required threshold determined in step 4. If the confidence score meets the threshold, the expected quality of automatically labeling exceeds the requested level of accuracy and that object will be considered auto-labeled.
7. Step 6 will produce a dataset of unlabeled data with confidence scores. Ground Truth will select data points with low confidence scores from this dataset and send them to human workers.
8. Ground Truth uses the existing human-labeled data and this additional labeled data from human workers to train a new model.

9. The process is repeated until the dataset is fully labeled or until another stopping condition is met. For example, auto labeling will stop if your human annotation budget is reached.

Automated data labeling is available only for the following Ground Truth built-in algorithms:

- Image classification
- Semantic segmentation
- Object detection (bounding box)
- Text classification

Input data quotas apply for automated data labeling jobs. See [Input Data Quotas \(p. 101\)](#) for information about dataset size, input data size and resolution limits for automated Semantic Segmentation, Object Detection, and Image Classification.

Note

Before you use an the automated-labeling model in production, you need to fine-tune or test the it, or both . You might fine-tune the model (or create and tune another supervised model of your choice) on the dataset produced by your labeling job to optimize the model's architecture and hyperparameters. If you decide to use the model for inference without fine-tuning it, we strongly recommend making sure that you evaluate its accuracy on a representative (for example, randomly selected) subset of the dataset labeled with Ground Truth and that it matches your expectations.

Create an Automated Data Labeling Job (Console)

Automated data labeling is available only for Ground Truth built-in algorithms.

To use the Amazon SageMaker console to create a labeling job that uses automated data labeling you need to know how to create a labeling job using the Amazon SageMaker console. To learn how to create a labeling job in the console using Ground Truth, see [Getting started \(p. 65\)](#).

To create an automated data labeling job (console)

1. Open the Ground Truth **Labeling jobs** section of the Amazon SageMaker console: <https://console.aws.amazon.com/sagemaker/groundtruth>.
2. Using [Getting started \(p. 65\)](#) as a guide, complete the **Job overview** and **Task type** sections. Note that auto labeling is not support for custom task types.
3. Under **Workers**, choose your workforce type.
4. In the same section, choose **Enable automated data labeling**.
5. Using [Step 4: Configure the Bounding Box Tool. \(p. 68\)](#) as a guide, create worker instructions in the section **Task Type labeling tool**. For example, if you chose **Semantic segmentation** as your labeling job type, this section will be called **Semantic segmentation labeling tool**.
6. To preview your worker instructions and dashboard, choose **Preview**.
7. Choose **Create**. This will create and start your labeling job and the auto labeling process.

You will see your labeling job appear in the **Labeling jobs** section of the Amazon SageMaker console. Your output data will appear in the Amazon S3 bucket that you specified when creating the labeling job. For more information about the format and file structure of your labeling job output data, see [Output Data \(p. 103\)](#).

Create an Automated Data Labeling Job (API)

To create an automated data labeling job using the the Amazon SageMaker API, use the `LabelingJobAlgorithmsConfig` parameter of the `CreateLabelingJob` operation.

Specify the Amazon Resource Name (ARN) of the algorithm that you are using for automated data labeling in the [LabelingJobAlgorithmSpecificationArn](#) parameter. Choose from one of the four Ground Truth built-in algorithms that are supported with automated labeling:

- Image classification
- Semantic segmentation
- Object detection (bounding box)
- Text classification

When an automated data labeling job finishes, Ground Truth returns the ARN of the model it used for the automated data labeling job. Use this model as the starting model for similar auto-labeling job types by providing the ARN, in string format, in the [InitialActiveLearningModelArn](#) parameter. To retrieve the model's ARN, use an AWS Command Line Interface (AWS CLI) command similar to the following.

```
# Fetch the mARN of the model trained in the final iteration of the previous labeling
# job.Ground Truth
pretrained_model_arn = sagemaker_client.describe_labeling_job(LabelingJobName=job_name)
['LabelingJobOutput']['FinalActiveLearningModelArn']
```

To encrypt data on the storage volume attached to the ML compute instance(s) that are used in automated labeling, include an AWS Key Management Service (AWS KMS) key in the [VolumeKmsKeyId](#) parameter. For information about AWS KMS keys see [What is AWS Key Management Service?AWS Key Management Service Developer Guide](#).

For an example that uses the [CreateLabelingJob](#) operation to create an automated data labeling job, see the [object_detection_tutorial](#) example in the [SageMaker Examples, Ground Truth Labeling Jobs](#) section of a Amazon SageMaker notebook instance. To learn how to create and open a notebook instance, see [Create a Notebook Instance \(p. 252\)](#). To learn how to access Amazon SageMaker example notebooks, see [Use Example Notebooks \(p. 260\)](#).

Amazon EC2 Instances Required for Automated Data Labeling

The following table lists the Amazon Elastic Compute Cloud (Amazon EC2) instances that you need to run automated data labeling for training and batch inference jobs.

Automated Data Labeling Job Type	Training Instance Type	Inference Instance Type
Image classification	ml.p3.2xlarge*	ml.c5.xlarge
Object detection (bounding box)	ml.p3.2xlarge*	ml.c5.4large
Text classification	ml.c5.2xlarge	ml.m4.xlarge
Semantic segmentation	ml.p3.2xlarge*	ml.p3.2xlarge*

* In the Asia Pacific (Mumbai) Region (ap-south-1) use ml.p2.8xlarge instead.

Automated data labeling incurs two separate charges: the per-item charge (for information, see [pricing](#)), and the charge for the Amazon EC2 instance required to run the model (see [Amazon EC2 pricing](#)).

Ground Truth manages the instances that you use for automated data labeling jobs. It creates, configures, and terminates the instances as needed to perform your job. These instances don't appear in your Amazon EC2 instance dashboard.

Chaining Labeling Jobs

Amazon SageMaker Ground Truth can reuse datasets from prior jobs in two ways: cloning and chaining.

Cloning copies the set-up of a prior labeling job and allows you to make additional changes, before setting it to run.

Chaining uses not only the setup of the prior job, but also the results. This allows you to continue an incomplete job, and add labels or data objects to a completed job. Chaining is a more complex operation.

For data processing:

- Cloning uses the prior job's *input* manifest, with optional modifications, as the new job's input manifest.
- Chaining uses the prior job's *output* manifest as the new job's input manifest.

Chaining is useful when you need to:

- Continue a labeling job that was manually stopped.
- Continue a labeling job that failed mid-job, after fixing issues.
- Switch to automated data labeling after manually labeling part of a job (or vice-versa).
- Add more data objects to a completed job and start the job from there.
- Add another annotation to a completed job. For example, you have a collection of phrases labeled for topic, then want to run the set again, categorizing them by the topic's implied audience.

In Amazon SageMaker Ground Truth you can configure a chained labeling job with either the console or the API.

Key Term: Label Attribute Name

The *label attribute name* (`LabelAttributeName` in the API) is a string used as the key for the key-value pair formed with the label that a worker assigns to the data object.

The following rules apply for the label attribute name:

- It can't end with `-metadata`.
- The names `source` and `source-ref` are reserved and can't be used.
- For semantic segmentation labeling jobs, it must end with `-ref`. For all other labeling jobs, it *can't* end with `-ref`. If you use the console to create the job, Amazon SageMaker Ground Truth automatically appends `-ref` to all label attribute names except for semantic segmentation jobs.
- For a chained labeling job, if you're using the same label attribute name from the originating job and you configure the chained job to use auto-labeling, then if it had been in auto-labeling mode at any point, Ground Truth uses the model from the originating job.

In an output manifest, the label attribute name appears similar to the following.

```
"source-ref": "<S3 URI>",
"<label attribute name>": {
  "annotations": [
    {
      "class_id": 0,
      "width": 99,
      "top": 87,
      "height": 62,
      "left": 175
    }
  ]
}
```

```
        }],
      "image_size": [
        {
          "width": 344,
          "depth": 3,
          "height": 234
        }
      ],
      "<label attribute name>-metadata": {
        "job-name": "<job name>",
        "class-map": {
          "0": "<label attribute name>"
        },
        "human-annotated": "yes",
        "objects": [
          {
            "confidence": 0.09
          },
          "creation-date": "<timestamp>",
          "type": "groundtruth/object-detection"
        }
      }
    }
```

If you're creating a job in the console and don't explicitly set the label attribute name value, Ground Truth uses the job name as the label attribute name for the job.

Start a Chained Job (Console)

Choose a stopped, failed, or completed labeling job from the list of your existing jobs. This enables the **Actions** menu.

From the **Actions** menu, choose **Chain**.

Job Overview Panel

In the **Job overview** panel, a new **Job name** is set based on the title of the job from which you are chaining this one. You can change it.

You may also specify a label attribute name different from the labeling job name.

If you're chaining from a completed job, the label attribute name uses the name of the new job you're configuring. To change the name, select the check box.

If you're chaining from a stopped or failed job, the label attribute name uses to the name of the job from which you're chaining. Its easy to see and edit the value because the name check box is checked.

Attribute label naming considerations

- **The default** uses the label attribute name Ground Truth has selected. All data objects without data connected to that label attribute name are labeled.
- **Using a label attribute name** not present in the manifest causes the job to process *all* the objects in the dataset.

The **input dataset location** in this case is automatically selected as the output manifest of the chained job. The input field is not available, so you cannot change it.

Adding data objects to a labeling job

You cannot specify an alternate manifest file. Manually edit the output manifest from the previous job to add new items before starting a chained job. The S3 URI helps you locate where you are storing the manifest in your S3 bucket. Download the manifest file from there, edit it locally on your computer, then upload the new version to replace it. Make sure you are not introducing errors during editing. We recommend you use JSON linter to check your JSON. Many popular text editors and IDEs have linter plugins available.

Start a Chained Job (API)

The procedure is almost the same as setting up a new labeling job with `CreateLabelingJob`, except for two primary differences:

- **Manifest location:** Rather than use your original manifest from the prior job, the value for the `ManifestS3Uri` in the `DataSource` should point to the S3 URI of the *output manifest* from the prior labeling job.
- **Label attribute name:** Setting the correct `LabelAttributeName` value is important here. As pointed out, this is the key portion of a key-value pair where labeling data is the value. Sample use cases include:
 - **Adding new or more-specific labels to a completed job** — set a new label attribute name.
 - **Labeling the unlabeled items from a prior job** — use the label attribute name from the prior job.

Use a Partially Labeled Dataset

You can get some chaining benefits if you use an augmented manifest that has already been partially labeled. Check the **Label attribute name** check box and set the name so that it matches the name in your manifest.

If you're using the API, the instructions are the same as starting a chained job. However, be sure to upload your manifest to an S3 bucket and use it instead of using the output manifest from a prior job.

The **Label attribute name** value in the manifest has to conform to the naming considerations discussed above.

Verify and Adjust Labels

When the labels on a dataset need to be validated, Amazon SageMaker Ground Truth provides functionality to have workers verify that labels are correct or to adjust previous labels.

These types of jobs fall into two distinct categories:

- **Label verification** – Workers indicate if the existing labels are correct, or rate their quality, and can add comments to explain their reasoning.
- **Label adjustment** – Workers adjust prior annotations to correct them.

You can use Ground Truth to adjust or verify only labels that resulted from bounding box and semantic segmentation labeling jobs. You can start a label verification and adjustment jobs using the Amazon SageMaker console or the API.

Create and Start a Label Verification Job (Console)

To start a label verification job (console)

1. Open the Amazon SageMaker console: <https://console.aws.amazon.com/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 94\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
3. Choose the **Label verification** task type and choose **Next**.
4. In the **Display existing labels** pane, the system shows the available label attribute names in your manifest. Choose the label attribute name for the labeling job that you want to verify.
5. Use the instructions areas of the tool designer to provide context about what the previous labelers were asked to do and what the current verifiers need to check.

6. Choose **See preview** to check that the tool is displaying the prior labels correctly and presents the label verification task clearly.
7. Select **Create**. This will create and start your labeling job.

Start an Label Adjustment Job (Console)

Use the Amazon SageMaker console to start a label verification or adjustment job.

To start a label adjustment job (console)

1. Open the Amazon SageMaker console: <https://console.aws.amazon.com/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 94\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
3. Choose the correct task type for your data and choose **Next**.
4. After choosing the workers, expand **Display existing labels**. If it isn't expanded, choose the arrow next to the title to expand it.
5. Check the box next to **I want to display existing labels from the dataset for this job**.
6. For **Label attribute name**, choose the name from your manifest that corresponds to the labels that you want to display for adjustment. Ground Truth tries to detect and populate these values by analyzing the manifest, but you might need to set the correct value.
7. Use the instructions areas of the tool designer to provide context about what the previous labelers were tasked with doing and what the current verifiers need to check and adjust.
8. Choose **See preview** to check that the tool shows the prior labels correctly and presents the task clearly.
9. Select **Create**. This will create and start your labeling job.

Start a Label Verification or Adjustment Job (API)

Start a label verification or adjustment job by chaining a successfully completed job or starting a new job from scratch using the [CreateLabelingJob](#) operation. To learn how to start a new labeling job by chaining a previous job, see [Start a Chained Job \(API\) \(p. 96\)](#).

To create a label verification or adjustment job, use the following guidelines to specify API attributes for the `CreateLabelingJob` operation:

- Use the **LabelAttributeName** parameter to specify the output label name that you want to use for verified or adjusted labels.
- If you are chaining the job, the labels from the previous labeling job to be adjusted or verified will be specified in the custom UI template. To learn how to create a custom template, see [Create Custom Worker Templates \(p. 163\)](#).

Identify the location of the UI template in the **UiTemplateS3Uri** parameter. Amazon SageMaker provides widgets that you can use in your custom template to display old labels. Use the **initial-value** attribute in one of the following crowd elements to extract the labels that need verification or adjustment and include them in your task template:

- [crowd-semantic-segmentation \(p. 225\)](#)—Use this crowd element in your custom UI task template to specify semantic segmentation labels that need to be verified or adjusted.
- [crowd-bounding-box \(p. 192\)](#)—Use this crowd element in your custom UI task template to specify bounding box labels that need to be verified or adjusted.
- The **LabelCategoryConfigS3Uri** parameter must contain the same label categories as the previous labeling job. Adding new label categories or adjusting label categories is not supported.

Ground Truth stores the output data from a label verification or adjustment job in the S3 bucket that you specified in the [S3OutputPath](#) parameter of the [CreateLabelingJob](#) operation. For more information about the output data from a label verification or adjustment labeling job, see [Label Verification and Adjustment Data in the Output Manifest \(p. 98\)](#).

Label Verification and Adjustment Data in the Output Manifest

Amazon SageMaker Ground Truth writes label verification data to the output manifest within the metadata for the label. It adds two properties to the metadata:

- A `type` property, with a value of "groundtruth/label-verification".
- A `worker-feedback` property, with an array of `comment` values. This property is added when the worker enters comments. If there are no comments, the field doesn't appear.

The following example output manifest shows how label verification data appears:

```
{  
    "source-ref": "S3 bucket location",  
    "verify-bounding-box": "1",  
    "verify-bounding-box-metadata":  
    {  
        "class-name": "bad",  
        "confidence": 0.93,  
        "type": "groundtruth/label-verification",  
        "job-name": "verify-bounding-boxes",  
        "human-annotated": "yes",  
        "creation-date": "2018-10-18T22:18:13.527256",  
        "worker-feedback": [  
            {"comment": "The bounding box on the bird is too wide on the right side."},  
            {"comment": "The bird on the upper right is not labeled."}  
        ]  
    }  
}
```

The worker output of adjustment tasks resembles the worker output of the original task, except that it contains the adjusted values and an `adjustment-status` property with the value of `adjusted` or `unadjusted` to indicate whether an adjustment was made.

For more examples of the output of different tasks, see [Output Data \(p. 103\)](#).

Cautions and Considerations

To get expected behavior when creating a label verification or adjustment job, carefully verify your input data.

- If you are using image data, verify that your manifest file contains hexadecimal RGB color information.
- To save money on processing costs, filter your data to ensure you are not including unwanted objects in your labeling job input manifest.
- Add required Amazon S3 permissions to ensure your input data is processed correctly.

Color Information Requirements for Semantic Segmentation Jobs

To properly reproduce color information in verification or adjustment tasks, the tool requires hexadecimal RGB color information in the manifest (for example, #FFFFFF for white). When you set up a Semantic Segmentation verification or adjustment job, the tool examines the manifest to determine if this information is present. If it can't find it, Amazon SageMaker Ground Truth displays an error message and the ends job setup.

In prior iterations of the Semantic Segmentation tool, category color information wasn't output in hexadecimal RGB format to the output manifest. That feature was introduced to the output manifest at the same time the verification and adjustment workflows were introduced. Therefore, older output manifests aren't compatible with this new workflow.

Filter Your Data Before Starting the Job

Amazon SageMaker Ground Truth processes all objects in your input manifest. If you have a partially labeled data set, you might want to create a custom manifest using an [Amazon S3 Select query](#) on your input manifest. Unlabeled objects individually fail, but they don't cause the job to fail, and they might incur processing costs. Filtering out objects you don't want verified reduces your costs.

If you create a verification job using the console, you can use the filtering tools provided there. If you create jobs using the API, make filtering your data part of your workflow where needed.

Security Considerations for Images

Due to browser security models, some image markup tasks—like keypoints, polygons, bounding boxes, and semantic segmentation—require you to add a cross-origin resource sharing (CORS) specification to the Amazon Simple Storage Service (Amazon S3) bucket where you store the images. You must apply the CORS specification prior to marking up the images.

Applying CORS to your bucket

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Select the bucket in which you are storing your images.
3. Select the **Permissions** tab, then **CORS configuration**.
4. Add the following block of XML and save.

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
</CORSRule>
</CORSConfiguration>
```

Use Input and Output Data

The input data that you provide to Amazon SageMaker Ground Truth is sent to your workers for labeling. You choose the data to send to your workers by creating a manifest file that defines the data that requires labeling.

The output data is the result of your labeling job. The output data file contains one entry for each object in the input dataset that specifies the label.

Topics

- [Input Data \(p. 99\)](#)
- [Output Data \(p. 103\)](#)

Input Data

The input data are the data objects that you send to your workforce to be labeled. Each object in the input data is described in a manifest file. Each line in the manifest is an entry containing an object to label. An entry can also contain labels from previous jobs.

Input data and the manifest file must be stored in Amazon Simple Storage Service (Amazon S3). Each has specific storage and access requirements, as follows:

- The S3 bucket that contains the input data must be in the same AWS Region in which you are running Amazon SageMaker Ground Truth. You must give Amazon SageMaker access to the data stored in the S3 bucket so that it can read it. For more information about S3 buckets, see [Working with Amazon S3 buckets](#).
- The manifest file must be in the same AWS Region as the data files, but it doesn't need to be in the same location as the data files. It can be stored in any S3 bucket that is accessible to the AWS Identity and Access Management (IAM) role that you assigned to Ground Truth when you created the labeling job.

The manifest is a UTF-8 encoded file where each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \r\n. Because each line must be a valid JSON object, you can't have unescaped line break characters. For more information about data format, see [JSON Lines](#).

Limits: Each JSON object in the manifest file can be no larger than 100 K characters. No single attribute within an object can be larger than 20,000 characters. Attribute names can't begin with \$ (dollar sign).

Each JSON object in the manifest file must contain one of the following keys: `source-ref` or `source`. The value of the keys are interpreted as follows:

- `source-ref`—The source of the object is the Amazon S3 object specified in the value. Use this value when the object is a binary object, such as an image, or when you have text in individual files. You also use the `source-ref` key for image files for a bounding box or semantic segmentation labeling job.
- `source`—The source of the object is the value. Use this value when the object is a text value.

The following is an example of a manifest file for files stored in an S3 bucket:

```
{"source-ref": "S3 bucket location 1"}  
{"source-ref": "S3 bucket location 2"}  
...  
{"source-ref": "S3 bucket location n"}
```

The following is an example of a manifest file with the input data stored in the manifest:

```
{"source": "Lorem ipsum dolor sit amet"}  
 {"source": "consectetur adipiscing elit"}  
 ...  
 {"source": "mollit anim id est laborum"}
```

You can include other key-value pairs in the manifest file. These pairs are passed to the output file unchanged. This is useful when you want to pass information between your applications. For more information, see [Output Data \(p. 103\)](#).

Create a Manifest File

You can create a manifest file for your labeling jobs in the Ground Truth console using images, text (.txt) files, and comma-separated value (.csv) files. Before using the following procedure, ensure that your input images or files are correctly formatted:

- **Image files** – Image files must comply with the size and resolution limits listed in the tables found in [Input File Size Quota \(p. 101\)](#).
- **Text files** – Text data can be stored in one or more .txt files. Each item that you want labeled must be separated by a standard line break.

- CSV files – Text data can be stored in one or more .csv files. Each item that you want labeled must be in a separate row.

To create a manifest file

1. Store the images or text files that you want to have labeled in an S3 bucket that is in the same Region as your labeling job. You must give access to Amazon SageMaker for the data to be read. For more information about Amazon S3 buckets, see [Working with Amazon S3 buckets](#).
2. Open the Ground Truth console: <https://console.aws.amazon.com/sagemaker/groundtruth>.
3. Choose **Labeling Job**.
4. In the **Job overview** section, under **Input dataset location**, choose **Create manifest file**.
5. In **Input dataset location**, enter the path to the S3 bucket where you data is stored (for example, `s3://bucket/path-to-your-objects`).
6. Choose the **Data type**, then choose **Create**. You see a loading screen while Ground Truth examines the S3 bucket and creates a `.manifest` file in the S3 location specified above. If you would like to visually inspect the auto-generated manifest file, convert this file into a human-readable format by downloading a copy of the file and changing the file suffix from `.manifest` to `.txt`.
7. For images or text object, a green box indicates the number of objects detected. If you're satisfied with the results, choose **Use this manifest**. If not, modify the image or text files in your S3 bucket and use this procedure to generate a new manifest file.

Input Data Quotas

Input datasets used in semantic segmentation labeling jobs have a quota of 20,000 items. For all other labeling job types, the dataset size quota is 100,000 items. To request an increase to the quota for labeling jobs other than semantic segmentation jobs, review the procedures in [AWS Service Quotas](#) to request a quota increase.

Input image data for active and non-active learning labeling jobs must not exceed size and resolution quotas. *Active learning* refers to labeling job that use [automated data labeling](#). *Non-active learning* refers to labeling jobs that don't use automated data labeling.

Input File Size Quota

Input files can't exceed the following size- quotas for both active and non-active learning labeling jobs.

Labeling Job	Input File Size Quota
Image classification	12 MB
Object detection (Bounding box)	12 MB
Semantic segmentation	6 MB
Image classification label adjustment	12 MB
Object detection label adjustment	12 MB
Semantic segmentation label adjustment	6 MB
Image classification label verification	12 MB
Object detection label verification	12 MB
Semantic segmentation label verification	6 MB

Input Image Resolution Quotas

Image file resolution refers to the number of pixels in an image, and determines the amount of detail an image holds. Image resolution quotas differ depending on the labeling job type and the Amazon SageMaker built-in algorithm used. The following table lists the resolution quotas for images used in active and non-active learning labeling jobs.

Labeling Job	Resolution Quota - Non Active Learning	Resolution Quota - Active Learning
Image classification	7680 × 4320 pixels (8 KB)	3840 x 2160 pixels (4 KB)
Object detection (Bounding box)	7680 × 4320 pixels (8 KB)	3840 x 2160 pixels (4 KB)
Semantic segmentation	3840 x 2160 pixels (4 KB)	1920 x 1080 pixels (1080 p)
Image classification label adjustment	7680 × 4320 pixels (8 KB)	3840 x 2160 pixels (4 KB)
Object detection label adjustment	7680 × 4320 pixels (8 KB)	3840 x 2160 pixels (4 KB)
Semantic segmentation label adjustment	3840 x 2160 pixels (4 KB)	1920 x 1080 pixels (1080 p)
Image classification label verification	7680 × 4320 pixels (8 KB)	Not available
Object detection label verification	7680 × 4320 pixels (8 KB)	Not available
Semantic segmentation label verification	3840 x 2160 pixels (4 KB)	Not available

Note

Active learning isn't available for label verification jobs.

Filter and Select Data for Labeling

You can use the Amazon SageMaker console to select a portion of your dataset for labeling. The data must be stored in an Amazon S3 bucket. You have three options:

- Use the full dataset.
- Choose a randomly selected sample of the dataset.
- Specify a subset of the dataset using a query.

The following options are available in the **Labeling jobs** section of the [Amazon SageMaker console](#) after selecting **Create labeling job**. See [Getting started \(p. 65\)](#) to learn how to Create a labeling job in the console. To configure the dataset that you use for labeling, in the **Job overview** section, select **Additional configuration**.

Use the Full Dataset

When you choose to use **Full dataset**, you must provide a manifest file for your data objects. You can provide the path of the S3 bucket that contains the manifest file or you can use the Amazon SageMaker console to create the file. See [Create a Manifest File \(p. 100\)](#) to learn how to create a manifest file using the console.

Choose a Random Sample

When you want to label a random subset of your data, select **Random sample**. The dataset is stored in the S3 bucket specified in the **Input dataset location** field.

After you have specified the percentage of data objects that you want to include in the sample, choose **Create subset**. Amazon SageMaker randomly picks the data objects for your labeling job. After the objects are selected, choose **Use this subset**.

Amazon SageMaker creates a manifest file for the selected data objects. It also modifies the value in the **Input dataset location** field to point to the new manifest file.

Specify a Subset

You can specify a subset of your data objects using an Amazon Simple Storage Service (Amazon S3) **SELECT** query on the object file names.

The **SELECT** statement of the SQL query is defined for you. You provide the **WHERE** clause to specify which data objects should be returned.

For more information about the Amazon S3 **SELECT** statement, see [Selecting Content from Objects](#).

Choose **Create subset** to start the selection, and then choose **Use this subset** to use the selected data.

Amazon SageMaker creates a manifest file for the selected data objects. It also updates the value in the **Input dataset location** field to point to the new manifest file.

Output Data

The output from a labeling job is placed in the location that you specified in the console or in the call to the [CreateLabelingJob](#) operation.

Each line in the output data file is identical to the manifest file with the addition of an attribute and value for the label assigned to the input object. The attribute name for the value is defined in the console or in the call to the [CreateLabelingJob](#) operation. You can't use **-metadata** in the label attribute name. If you are running a semantic segmentation job, the label attribute must end with **-ref**. For any other type of job, the attribute name can't end with **-ref**.

The output of the labeling job is the value of the key/value pair with the label. The label and the value overwrites any existing JSON data in the input file with the new value.

For example, the following is the output from an image classification labeling job where the input data files were stored in an Amazon S3 bucket and the label attribute name was defined as "sport". In this example the JSON object is formatted for readability, in the actual output file the JSON object is on a single line. For more information about the data format, see [JSON Lines](#).

```
{  
    "source-ref": "S3 bucket location",  
    "sport":0,  
    "sport-metadata":  
    {  
        "class-name": "football",  
        "confidence": 0.8,  
        "type":"groundtruth/image-classification",  
        "job-name": "identify-sport",  
        "human-annotated": "yes",  
        "creation-date": "2018-10-18T22:18:13.527256"  
    }  
}
```

The value of the label can be any valid JSON. In this case the label's value is the index of the class in the classification list. Other job types, such as bounding box, have more complex values.

Any key-value pair in the input manifest file other than the label attribute is unchanged in the output file. You can use this to pass data to your application.

The output from a labeling job can be used as the input to another labeling job. You can use this when you are chaining together labeling jobs. For example, you can send one labeling job to determine the sport that is being played. Then you send another using the same data to determine if the sport is being played indoors or outdoors. By using the output data from the first job as the manifest for the second job, you can consolidate the results of the two jobs into one output file for easier processing by your applications.

The output data file is written to the output location periodically while the job is in progress. These intermediate files contain one line for each line in the manifest file. If an object is labeled, the label is included, if the object has not been labeled it is written to the intermediate output file identically to the manifest file.

Output Directories

Ground Truth creates several directories in your Amazon S3 output path. These directories contain the results of your labeling job and other artifacts of the job. The top-level directory for a labeling job is given the same name as your labeling job, the output directories are placed beneath it. For example, if you named your labeling job **find-people** you output would be in the following directories:

```
s3://bucket/find-people/activelearning  
s3://bucket/find-people/annotations  
s3://bucket/find-people/inference  
s3://bucket/find-people/manifests  
s3://bucket/find-people/training
```

Each directory contains the following output:

Active Learning Directory

The `activelearning` directory is only present when you are using automated data labeling. It contains the input and output validation set for automated data labeling, and the input and output folder for automatically labeled data.

Annotations Directory

The `annotations` directory contains all of the annotations made by the workforce. These are the responses from individual workers that have not been consolidated into a single label for the data object.

There are three subdirectories in the `annotations` directory.

The first, `worker-response` contains the responses from individual workers. This contains a subdirectory for each iteration, which in turn contains a subdirectory for each data object in that iteration. The annotation for each data object is stored in a timestamped `.json` file. There may be more than one annotation for each data object in this directory, depending on how many workers you want to annotate each object.

The second, `consolidated-annotation` contains information required to consolidate the annotations in the current batch into labels for your data objects.

The third, `intermediate`, contains the output manifest for the current batch with any completed labels. This file is updated as the label for each data object is completed.

Inference Directory

The `inference` directory is only present when you are using automated data labeling. This directory contains the input and output files for the Amazon SageMaker batch transform used while labeling data objects.

Manifest Directory

The `manifest` directory contains the output manifest from your labeling job. There is one subdirectory in the `manifest` directory, `output`. The `output` directory contains the output manifest file for your labeling job. The file is named `output.manifest`.

Training Directory

The `training` directory is only present when you are using automated data labeling. This directory contains the input and output files used to train the automated data labeling model.

Confidence Score

Ground Truth calculates a confidence score for each label. A *confidence score* is a number between 0 and 1 that indicates how confident Ground Truth is in the label. You can use the confidence score to compare labeled data objects to each other, and to identify the least or most confident labels.

You should not interpret the value of the confidence scores as an absolute value, or compare them across labeling jobs. For example, if all of the confidence scores are between 0.98 and 0.998, you should only compare the data objects with each other and not rely on the high confidence scores.

You should not compare the confidence scores of human-labeled data objects and auto-labeled data objects. The confidence scores for humans are calculated using the annotation consolidation function for the task, the confidence scores for automated labeling are calculated using a model that incorporates object features. The two models generally have different scales and average confidence.

For a bounding box labeling job, Ground Truth calculates a confidence score per box. You can compare confidence scores within one image or across images for the same labeling type (human or auto). You can't compare confidence scores across labeling jobs.

Output Metadata

The output from each job contains metadata about the label assigned to data objects. These elements are the same for all jobs with minor variations. The following example shows the metadata elements.

```
"confidence": 0.93,  
"type": "groundtruth/image-classification",  
"job-name": "identify-animal-species",  
"human-annotated": "yes",  
"creation-date": "2018-10-18T22:18:13.527256"
```

The elements have the following meaning:

- **confidence** – The confidence that Ground Truth has that the label is correct. For more information, see [Confidence Score \(p. 105\)](#).
- **type** – The type of classification job. For job types, see [Built in Task Types \(p. 69\)](#).
- **job-name** – The name assigned to the job when it was created.
- **human-annotated** – Indicates whether the data object was labeled by a human or by automated data labeling. For more information, see [Automate Data Labeling \(p. 91\)](#).
- **creation-date** – The date and time that the label was created.

Classification Job Output

The following are sample outputs (output manifest files) from an image classification job and a text classification job. They includes the label that Ground Truth assigned to the data object, the value for the label, and metadata that describes the label.

In addition to the standard metadata elements, the metadata for a classification job includes the text value of the label's class. For more information, see [Image Classification Algorithm \(p. 324\)](#).

```
{
  "source-ref": "S3 bucket location",
  "species": "0",
  "species-metadata":
  {
    "class-name": "dog",
    "confidence": 0.93,
    "type": "groundtruth/image-classification",
    "job-name": "identify-animal-species",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
}
```

```
{
  "source": "a bad day",
  "mood": "1",
  "mood-metadata":
  {
    "class-name": "sad",
    "confidence": 0.8,
    "type": "groundtruth/text-classification",
    "job-name": "label-mood",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
}
```

Multi-label Classification Job Output

The following are example output manifest files from a multi-label image classification job and a multi-label text classification job. They include the labels that Ground Truth assigned to the data object (for example the image or piece of text) and metadata that describes the labels the the worker saw when completing the labeling task.

The label attribute name parameter (for example, `image-label-attribute-name`) contains an array of all of the labels that were selected by at least one of the workers who completed this task. This array contains integer keys (for example, `[1, 0, 8]`) that correspond to the labels found in `class-map`. In the multi-label image classification example, `bicycle`, `person`, and `clothing` were selected by at least one of the workers who completed the labeling task for the image found in `source-ref`.

The `confidence-map` shows the confidence-score that Ground Truth assigned to each label. It assigns a confidence score to all labels, regardless of whether they were selected by a worker. To learn more about Ground Truth confidence scores, see [Confidence Score \(p. 105\)](#).

The following is an example of a multi-label image classification output manifest file.

```
{
  "source-ref": "S3 bucket location",
  "image-label-attribute-name": [1, 0, 8],
  "image-label-attribute-name-metadata":
```

```
{
    "job-name": "labeling-job/image-label-attribute-name",
    "class-map":
    {

"1": "bicycle", "0": "person", "3": "woman", "2": "dog", "5": "book", "4": "table", "7": "man", "6": "chair", "8": "clothing"
    },
    "human-annotated": "yes",
    "creation-date": "2020-02-27T21:36:25.000201",
    "confidence-map":
    {

"1": 0.95, "0": 0.77, "3": 0.05, "2": 0.05, "5": 0.05, "4": 0.05, "7": 0.05, "6": 0.05, "8": 0.2
    },
    "type": "groundtruth/image-classification-multilabel"
}
}
```

The following is an example of a multi-label text classification output manifest file.

```
{
    "source-ref": "S3 bucket location",
    "text-label-attribute-name": [1, 0, 4],
    "text-label-attribute-name-metadata":
    {
        "job-name": "labeling-job/text-label-attribute-name",
        "class-map":
        {
            "1": "approving", "0": "sad", "3": "confused", "2": "angry", "4": "critical"
        },
        "human-annotated": "yes",
        "creation-date": "2020-02-20T21:36:25.000201",
        "confidence-map":
        {
            "1": 0.95, "0": 0.77, "3": 0.05, "2": 0.05, "4": 0.2
        },
        "type": "groundtruth/text-classification-multilabel"
    }
}
```

Bounding Box Job Output

The following is sample output (output manifest file) from a bounding box job. For this task, three bounding boxes are returned. The label value contains information about the size of the image, and the location of the bounding boxes.

The `class_id` element is the index of the box's class in the list of available classes for the task. The `class-map` metadata element contains the text of the class.

The metadata has a separate confidence score for each bounding box. The metadata also includes the `class-map` element that maps the `class_id` to the text value of the class. For more information, see [Object Detection Algorithm \(p. 402\)](#).

```
{
    "source-ref": "S3 bucket location",
    "bounding-box":
    {
        "image_size": [{ "width": 500, "height": 400, "depth": 3}],
        "annotations":
        [
            {"class_id": 0, "left": 111, "top": 134,
                "width": 61, "height": 128},
            {"class_id": 1, "left": 111, "top": 134,
                "width": 61, "height": 128}
        ]
    }
}
```

```

        {"class_id": 5, "left": 161, "top": 250,
         "width": 30, "height": 30},
        {"class_id": 5, "left": 20, "top": 20,
         "width": 30, "height": 30}
    ]
},
"bounding-box-metadata":
{
    "objects":
    [
        {"confidence": 0.8},
        {"confidence": 0.9},
        {"confidence": 0.9}
    ],
    "class-map":
    {
        "0": "dog",
        "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "identify-dogs-and-toys"
}
}
}

```

The output of a bounding box adjustment job looks like the following JSON. Note that the original JSON is kept intact and two new jobs are listed, each with “adjust-” prepended to the original attribute’s name.

```

{
    "source-ref": "S3 bucket location",
    "bounding-box":
    {
        "image_size": [{ "width": 500, "height": 400, "depth":3}],
        "annotations":
        [
            {"class_id": 0, "left": 111, "top": 134,
             "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
             "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
             "width": 30, "height": 30}
        ]
    },
    "bounding-box-metadata":
    {
        "objects":
        [
            {"confidence": 0.8},
            {"confidence": 0.9},
            {"confidence": 0.9}
        ],
        "class-map":
        {
            "0": "dog",
            "5": "bone"
        },
        "type": "groundtruth/object-detection",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256",
        "job-name": "identify-dogs-and-toys"
    },
    "adjusted-bounding-box":
    {

```

```

    "image_size": [{ "width": 500, "height": 400, "depth":3}],
    "annotations":
    [
        {"class_id": 0, "left": 110, "top": 135,
         "width": 61, "height": 128},
        {"class_id": 5, "left": 161, "top": 250,
         "width": 30, "height": 30},
        {"class_id": 5, "left": 10, "top": 10,
         "width": 30, "height": 30}
    ]
},
"adjusted-bounding-box-metadata":
{
    "objects":
    [
        {"confidence": 0.8},
        {"confidence": 0.9},
        {"confidence": 0.9}
    ],
    "class-map":
    {
        "0": "dog",
        "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "job-name": "adjust-bounding-boxes-on-dogs-and-toys",
    "adjustment-status": "adjusted"
}
}
}

```

In this output, the job's type doesn't change, but an adjustment-status field is added. This field has the value of adjusted or unadjusted. If multiple workers have reviewed the object and at least one adjusted the label, the status is adjusted.

Label Verification Job Output

The output (output manifest file) of a bounding box verification job looks much different than the output of a bounding box annotation job. That's because the workers having a different type of task. They're not labeling objects, but evaluating the accuracy of prior labeling, making a judgment, and then providing that judgment and perhaps some comments.

If you are having human workers verify or adjust prior bounding box labels, the output of a verification job would look like the following JSON.

```

{
    "source-ref": "S3 bucket location",
    "bounding-box":
    {
        "image_size": [{ "width": 500, "height": 400, "depth":3}],
        "annotations":
        [
            {"class_id": 0, "left": 111, "top": 134,
             "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
             "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
             "width": 30, "height": 30}
        ]
    },
    "bounding-box-metadata":
    {
        "objects":

```

```
[{"confidence": 0.8}, {"confidence": 0.9}, {"confidence": 0.9}], "class-map": { "0": "dog", "5": "bone" }, "type": "groundtruth/object-detection", "human-annotated": "yes", "creation-date": "2018-10-18T22:18:13.527256", "job-name": "identify-dogs-and-toys"}, "verify-bounding-box": "1", "verify-bounding-box-metadata": { "class-name": "bad", "confidence": 0.93, "type": "groundtruth/label-verification", "job-name": "verify-bounding-boxes", "human-annotated": "yes", "creation-date": "2018-11-20T22:18:13.527256", "worker-feedback": [ {"comment": "The bounding box on the bird is too wide on the right side."}, {"comment": "The bird on the upper right is not labeled."} ] } }
```

Although the type on the original bounding box output was groundtruth/object-detection, the new type is groundtruth/label-verification. Also note that the worker-feedback array provides worker comments. If the worker doesn't provide comments, the empty fields are excluded during consolidation.

Semantic Segmentation Job Output

The following is the output manifest file from a semantic segmentation labeling job. The value of the label for this job is a reference to a PNG file in an S3 bucket.

In addition to the standard elements, the metadata for the label includes a color map that defines which color was used to label the image, the class name associated with the color, and the confidence score for each color. For more information, see [Semantic Segmentation Algorithm \(p. 424\)](#).

```
{ "source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-streets-metadata": { "internal-color-map": { "0": { "class-name": "BACKGROUND", "confidence": 0.9, "hex-color": "#ffffffff" }, "1": { "class-name": "buildings", "confidence": 0.9, "hex-color": "#2acf59" }, "2": { "class-name": "road", "confidence": 0.9, "hex-color": "#f28333" } } }
```

```

        },
    },
    "type": "groundtruth/semantic-segmentation",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "label-city-streets",
},
"verify-city-streets": "1",
"verify-city-streets-metadata": {
    "class-name": "bad",
    "confidence": 0.93,
    "type": "groundtruth/label-verification",
    "job-name": "verify-city-streets",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "worker-feedback": [
        {"comment": "The mask on the leftmost building is assigned the wrong side of
the road."},
        {"comment": "The curb of the road is not labeled but the instructions say
otherwise."}
    ]
}
}
}

```

Confidence is scored on a per-image basis. Confidence scores are the same across all classes within an image.

The output of a semantic segmentation adjustment job looks similar to the following JSON.

```
{
    "source-ref": "S3 bucket location",
    "city-streets-ref": "S3 bucket location",
    "city-streets-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "confidence": 0.9,
                "hex-color": "#ffffff"
            },
            "1": {
                "class-name": "buildings",
                "confidence": 0.9,
                "hex-color": "#2acf59"
            },
            "2": {
                "class-name": "road",
                "confidence": 0.9,
                "hex-color": "#f28333"
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256",
        "job-name": "label-city-streets",
    },
    "adjusted-city-streets-ref": "S3 bucket location",
    "adjusted-city-streets-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "confidence": 0.9,
                "hex-color": "#ffffff"
            },

```

```
"1": {
    "class-name": "buildings",
    "confidence": 0.9,
    "hex-color": "#2acf59"
},
"2": {
    "class-name": "road",
    "confidence": 0.9,
    "hex-color": "#f28333"
}
},
"type": "groundtruth/semantic-segmentation",
"human-annotated": "yes",
"creation-date": "2018-11-20T22:18:13.527256",
"job-name": "adjust-label-city-streets",
}
}
```

After you create an augmented manifest file, you can use it in a training job. See [object_detection_augmented_manifest_training.ipynb](#) for a demonstration of using of an "augmented manifest" to train an object detection machine learning model with AWS SageMaker. For more information, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Creating Instruction Pages

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console or you can create your own. The instructions are shown to the worker on the page where they complete their labeling task.

There are two kinds of instructions:

- *Short instructions*—instructions that are shown on the same webpage where the worker completes their task. These instructions should provide an easy reference to show the worker the correct way to label an object.
- *Full instructions*—instructions that are shown on a dialog box that overlays the page where the worker completes their task. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

Create instructions in the console when you are creating your labeling job. Start with the existing instructions for the task and use the editor to modify them to suit your labeling job.

Note

Once you create your labeling job, it will automatically start and you will not be able to modify your worker instructions. If you need to change your worker instructions, stop the labeling job that you created, clone it, and modify your worker instructions before creating a new job.

You can clone a labeling job in the console by selecting the labeling job and then selecting

Clone in the **Actions** menu.

To clone a labeling job using the Amazon SageMaker API or your preferred Amazon SageMaker SDK, make a new request to the `CreateLabelingJob` operation with the same specifications as your original job after modifying your worker instructions.

Short Instructions

Short instructions appear on the same web page that workers use to label your data object. For example, the following is the editing page for a bounding box task. The short instructions panel is on the left.

Bounding box labeling tool

Provide labeling instructions with examples below for workers. Workers will be viewing these instructions when they perform your tasks. Make sure the pop-up blocker of the browser is disabled before generating the preview

Preview

GOOD EXAMPLE
Enter description of a correct bounding box label

Upload image

Add a good example

Enter a brief description of the task

Label
Add a label name

BAD EXAMPLE
Enter description of an incorrect bounding box label

Upload image

Add a bad example

< >

► Additional instructions - *Optional*

Keep in mind that a worker will only spend seconds looking at the short instructions. Workers must be able to scan and understand your information quickly. In all cases it should take less time to understand the instructions than it takes to complete the task. Keep these points in mind:

- Your instructions should be clear and simple.
- Pictures are better than words. Create a simple illustration of your task that your workers can immediately understand.
- If you must use words, use short, concise examples.
- Your short instructions are more important than your full instructions.

The Amazon SageMaker Ground Truth console provides an editor so that you can create your short instructions. Replace the placeholder text and images with instructions for your task. Preview the worker's task page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

Full Instructions

You can provide additional instructions for your workers in a dialog box that overlays the page where workers label your data objects. Use full instructions to explain more complex tasks and to show workers the proper way to label edge cases or other difficult objects.

You can create full instructions using an editor in the Ground Truth console. As with quick instructions, keep the following in mind:

- Workers will want detailed instruction the first few times that they complete your task. Any information that they *must* have should be in the quick instructions.
- Pictures are more important than words.
- Text should be concise.
- Full instructions should supplement the short instructions. Don't repeat information that appears in the short instructions.

The Ground Truth console provides an editor so that you can create your full instructions. Replace the placeholder text and images with instructions for your task. Preview the full instruction page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

Add example images to your instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions:

- Place the cursor where the image should go in the instructions editor.
- Click the image icon in the editor toolbar.
- Enter the URL of your image.

If your instruction image in Amazon S3 is not publicly accessible:

- As the image URL, enter: `{} 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access {}`.
- This renders the image URL with a short-lived, one-time access code appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview.

Creating Custom Labeling Workflows

This document guides you through the process of setting up a workflow with a custom labeling template. For more information about starting a labeling job, see [Getting started \(p. 65\)](#). In that section, when you choose the **Task type**, select **Custom labeling task**, and then follow this section's instructions to configure it.

Next

[Step 1: Setting up your workforce \(p. 114\)](#)

Step 1: Setting up your workforce

In this step you use the console to establish which worker type to use and make the necessary sub-selections for the worker type. It assumes you have already completed the steps up to this point in the [Getting started \(p. 65\)](#) section and have chosen the **Custom labeling task** as the **Task type**.

To configure your workforce.

1. First choose an option from the **Worker types**. There are three types currently available:

- **Public** uses an on-demand workforce of independent contractors, powered by Amazon Mechanical Turk. They are paid on a per-task basis.
 - **Private** uses your employees or contractors for handling data that needs to stay within your organization.
 - **Vendor** uses third party vendors that specialize in providing data labeling services, available via the AWS Marketplace.
2. If you choose the **Public** option, you are asked to set the **number of workers per dataset object**. Having more than one worker perform the same task on the same object can help increase the accuracy of your results. The default is three. You can raise or lower that depending on the accuracy you need.

You are also asked to set a **price per task** by using a drop-down menu. The menu recommends price points based on how long it will take to complete the task.

The recommended method to determine this is to first run a short test of your task with a **private** workforce. The test provides a realistic estimate of how long the task takes to complete. You can then select the range your estimate falls within on the **Price per task** menu. If your average time is more than 5 minutes, consider breaking your task into smaller units.

Next

[Step 2: Creating your custom labeling task template \(p. 115\)](#)

Step 2: Creating your custom labeling task template

Topics

- [Starting with a base template \(p. 115\)](#)
- [Developing templates locally \(p. 115\)](#)
- [Using External Assets \(p. 116\)](#)
- [Track your variables \(p. 116\)](#)
- [A simple sample \(p. 116\)](#)
- [Adding automation with Liquid \(p. 118\)](#)
- [End-to-end demos \(p. 120\)](#)
- [Next \(p. 121\)](#)

Starting with a base template

To get you started, the **Task type** starts with a drop-down menu listing a number of our more common task types, plus a custom type. Choose one and the code editor area will be filled with a sample template for that task type. If you prefer not to start with a sample, choose **Custom HTML** for a minimal template skeleton.

If you've already created a template, upload the file directly using the **Upload file** button in the upper right of the task setup area or paste your template code into the editor area. For a repository of demo templates for a variety of labeling job task types, see [Amazon SageMaker Ground Truth Sample Task UIs](#).

Developing templates locally

While you need to be in the console to test how your template will process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding this code to the top of your HTML file.

Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you want to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this will not parse your variables. You may want to replace them with sample content while developing locally.

Using External Assets

Amazon SageMaker Ground Truth custom templates allow external scripts and style sheets to be embedded.

Example

```
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-
styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding types for remote scripts: `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets: `text/css;CHARSET=UTF-8`.

Track your variables

In the process of building the sample below, there will be a step that adds variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you will need to make sure you're aware of the variables it already uses. When you create your pre-annotation AWS Lambda script, its output will need to contain values for any of those variables you choose to keep.

The values you use for the variables can come from your manifest file. All the key-value pairs in your data object are provided to your pre-annotation Lambda. If it's a simple pass-through script, matching keys for values in your data object to variable names in your template is the easiest way to pass those values through to the tasks forms your workers see.

A simple sample

All tasks begin and end with the `<crowd-form> </crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them.

For a simple tweet-analysis task, use the `<crowd-classifier>` element. It requires the following attributes:

- `name` - the variable name to use for the result in the form output.
- `categories` - a JSON formatted array of the possible answers.
- `header` - a title for the annotation tool

As children of the `<crowd-classifier>` element, you must have three regions.

- *<classification-target>* - the text the worker will classify based on the options specified in the categories attribute above.
- *<full-instructions>* - instructions that are available from the "View full instructions" link in the tool. This can be left blank, but it is recommended that you give good instructions to get better results.
- *<short-instructions>* - a more brief description of the task that appears in the tool's sidebar. This can be left blank, but it is recommended that you give good instructions to get better results.

A simple version of this tool would look like this.

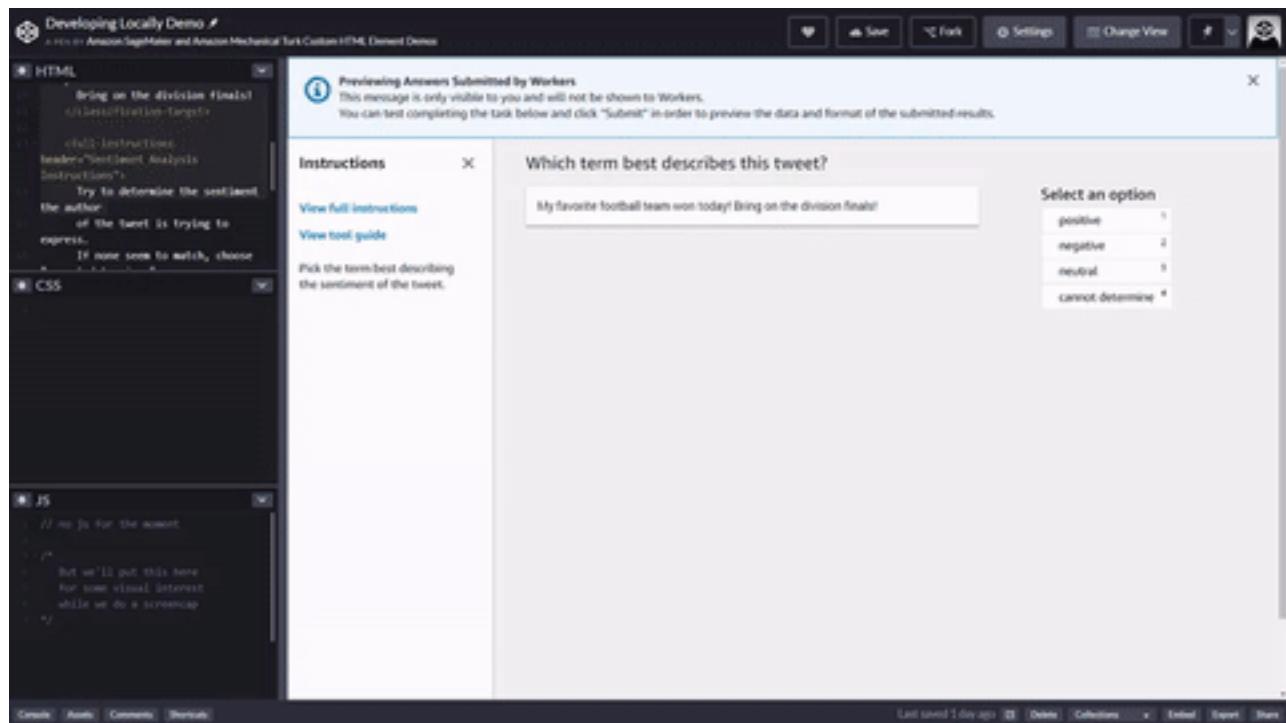
Example of using `crowd-classifier`

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive','negative','neutral', 'unclear']"
    header="Which term best describes this tweet?">
    <classification-target>
      My favorite football team won today!
      Bring on the division finals!
    </classification-target>

    <full-instructions header="Sentiment Analysis Instructions">
      Try to determine the sentiment the author
      of the tweet is trying to express.
      If none seem to match, choose "cannot determine."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

You can copy and paste the code into the editor in the Ground Truth labeling job creation workflow to preview the tool, or try out a [demo of this code on CodePen](#).



Adding automation with Liquid

Our custom template system uses [Liquid](#) for automation. It is an open source inline markup language. For more information and documentation, visit the [Liquid homepage](#).

The most common use of Liquid will be to parse the data coming from your [pre-annotation Lambda](#) and pull out the relevant variables to create the task. In Liquid, the text between single curly braces and percent symbols is an instruction or "tag" that creates control flow. Text between double curly braces is a variable or "object" which outputs its value.

The `taskInput` object returned by your [Pre-annotation Lambda \(p. 132\)](#) will be available as the `task.input` object in your templates.

The properties in your manifest's data objects are passed into your [Pre-annotation Lambda \(p. 132\)](#) as the `event.dataObject`. A simple pass-through script simply returns that object as the `taskInput` object. You would represent values from your manifest as variables as follows.

Example Manifest data object

```
{
  "source": "This is a sample text for classification",
  "labels": [ "angry" , "sad" , "happy" , "inconclusive" ],
  "header": "What emotion is the speaker feeling?"
}
```

Example Sample HTML using variables

```
<crowd-classifier
  name='tweetFeeling'
  categories='{{ task.input.labels | to_json }}'
  header='{{ task.input.header }}' >
```

```
<classification-target>
  {{ task.input.source }}
</classification-target>
```

Note the addition of " | to_json" to the labels property above. That's a filter to turn the array into a JSON representation of the array. Variable filters are explained next.

Variable filters

In addition to the standard Liquid filters and actions, Ground Truth offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, then specifying a filter name. Filters can be chained in the form of:

Example

```
{{ <content> | <filter> | <filter> }}
```

Autoescape and explicit escape

By default, inputs will be HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the escape filter to make it more obvious to someone reading the source of your template that the escaping is being done.

escape_once

escape_once ensures that if you've already escaped your code, it doesn't get re-escaped on top of that. For example, so that & doesn't become &&.

skip_autoescape

skip_autoescape is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

Use skip_autoescape sparingly

The best practice in templates is to avoid passing in functional code or markup with skip_autoescape unless you are absolutely sure you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a Cross Site Scripting attack.

to_json

to_json will encode what you feed it to JSON (JavaScript Object Notation). If you feed it an object, it will serialize it.

grant_read_access

grant_read_access takes an S3 URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible.

Example of the filters

Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?" }}
explicit escape: {{ "Have you read 'James & the Giant Peach'?" | escape }}
```

```
explicit escape_once: {{ "Have you read 'James & the Giant Peach'?" | escape_once }}
skip_autoescape: {{ "Have you read 'James & the Giant Peach'?" | skip_autoescape }}
to_json: {{ jsObject | to_json }}
grant_read_access: {{ "s3://mybucket/myphoto.png" | grant_read_access }}
```

Example

Output

```
auto-escape: Have you read 'James & the Giant Peach'?
explicit escape: Have you read 'James & the Giant Peach'?
explicit escape_once: Have you read 'James & the Giant Peach'?
skip_autoescape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?<access token and other
params>
```

Example of an automated classification template.

To automate the simple text classification sample, replace the tweet text with a variable.

The text classification template is below with automation added. The changes/additions are highlighted in bold.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'cannot determine']"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seem to match, choose "other."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

The tweet text that was in the prior sample is now replaced with an object. The `entry.taskInput` object uses `source` (or another name you specify in your pre-annotation Lambda) as the property name for the text and it is inserted directly in the HTML by virtue of being between double curly braces.

End-to-end demos

You can view the following end-to-end demos which include sample Lambdas:

- [Demo Template: Annotation of Images with `crowd-bounding-box` \(p. 121\)](#)
- [Demo Template: Labeling Intents with `crowd-classifier` \(p. 125\)](#)

Next

[Step 3: Processing with AWS Lambda \(p. 132\)](#)

Demo Template: Annotation of Images with crowd-bounding-box

When you chose to use a custom template as your task type in the Amazon SageMaker Ground Truth console, you reach the **Custom labeling task panel**. There you can choose from multiple base templates. The templates represent some of the most common tasks and provide a sample to work from as you create your customized labeling task's template. If you are not using the console, or as an additional recourse, see [Amazon SageMaker Ground Truth Sample Task UIs](#) for a repository of demo templates for a variety of labeling job task types.

This demonstration works with the **BoundingBox** template. The demonstration also works with the AWS Lambda functions needed for processing your data before and after the task. In the Github repository above, to find templates that work with AWS Lambda functions, look for {{ task.input.<property name> }} in the template.

Topics

- [Starter Bounding Box custom template \(p. 121\)](#)
- [Your own Bounding Box custom template \(p. 122\)](#)
- [Your manifest file \(p. 123\)](#)
- [Your pre-annotation Lambda function \(p. 123\)](#)
- [Your post-annotation Lambda function \(p. 124\)](#)
- [The output of your labeling job \(p. 125\)](#)

Starter Bounding Box custom template

This is the starter bounding box template that is provided.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-bounding-box
    name="boundingBox"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="{{ task.input.header }}"
    labels="{{ task.input.labels | to_json | escape }}"
  >

  <!-- The <full-instructions> tag is where you will define the full instructions of your task. -->
  <full-instructions header="Bounding Box Instructions" >
    <p>Use the bounding box tool to draw boxes around the requested target of interest:</p>
    <ol>
      <li>Draw a rectangle using your mouse over each instance of the target.</li>
      <li>Make sure the box does not cut into the target, leave a 2 - 3 pixel margin</li>
      <li>
        When targets are overlapping, draw a box around each object,
        include all contiguous parts of the target in the box.
        Do not include parts that are completely overlapped by another object.
      </li>
      <li>
        Do not include parts of the target that cannot be seen,
        even though you think you can interpolate the whole shape of the target.
      </li>
    </ol>
  </full-instructions>
</crowd-form>
```

```

</li>
<li>Avoid shadows, they're not considered as a part of the target.</li>
<li>If the target goes off the screen, label up to the edge of the image.</li>
</ol>
</full-instructions>

<!-- The <short-instructions> tag allows you to specify instructions that are displayed
in the left hand side of the task interface.
It is a best practice to provide good and bad examples in this section for quick
reference. -->
<short-instructions>
    Use the bounding box tool to draw boxes around the requested target of interest.
</short-instructions>
</crowd-bounding-box>
</crowd-form>
```

The custom templates use the [Liquid template language](#), and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{ task.input.<property name> }` in your template.

Your own Bounding Box custom template

As an example, assume you have a large collection of animal photos in which you know the kind of animal in an image from a prior image-classification job. Now you want to have a bounding box drawn around it.

In the starter sample, there are three variables: `taskObject`, `header`, and `labels`.

Each of these would be represented in different parts of the bounding box.

- `taskObject` is an HTTP(S) URL or S3 URI for the photo to be annotated. The added `| grant_read_access` is a filter that will convert an S3 URI to an HTTPS URL with short-lived access to that resource. If you're using an HTTP(S) URL, it's not needed.
- `header` is the text above the photo to be labeled, something like "Draw a box around the bird in the photo."
- `labels` is an array, represented as `['item1', 'item2', ...]`. These are labels that can be assigned by the worker to the different boxes they draw. You can have one or many.

Each of the variable names come from the JSON object in the response from your pre-annotation Lambda. The names above are merely suggested. Use whatever variable names make sense to you and will promote code readability among your team.

Only use variables when necessary

If a field will not change, you can remove that variable from the template and replace it with that text, otherwise you have to repeat that text as a value in each object in your manifest or code it into your pre-annotation Lambda function.

Example : Final Customized Bounding Box Template

To keep things simple, this template will have one variable, one label, and very basic instructions. Assuming your manifest has an "animal" property in each data object, that value can be re-used in two parts of the template.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
    <crowd-bounding-box
        name="boundingBox"
```

```

    labels="["{{ task.input.animal }}"]"
    src="{{ task.input.source-ref | grant_read_access }}"
    header="Draw a box around the {{ task.input.animal }}."
  >
  <full-instructions header="Bounding Box Instructions" >
    <p>Draw a bounding box around the {{ task.input.animal }} in the image. If there is more than one {{ task.input.animal }} per image, draw a bounding box around the largest one.</p>
    <p>The box should be tight around the {{ task.input.animal }} with no more than a couple of pixels of buffer around the edges.</p>
    <p>If the image does not contain a {{ task.input.animal }}, check the <strong> Nothing to label</strong> box.</p>
  </full-instructions>
  <short-instructions>
    <p>Draw a bounding box around the {{ task.input.animal }} in each image. If there is more than one {{ task.input.animal }} per image, draw a bounding box around the largest one.</p>
  </short-instructions>
</crowd-bounding-box>
</crowd-form>

```

Note the re-use of {{ task.input.animal }} throughout the template. If your manifest had all of the animal names beginning with a capital letter, you could use {{ task.input.animal | downcase }}, incorporating one of Liquid's built-in filters in sentences where it needed to be presented lowercase.

Your manifest file

Your manifest file should provide the variable values you're using in your template. You can do some transformation of your manifest data in your pre-annotation Lambda, but if you don't need to, you maintain a lower risk of errors and your Lambda will run faster. Here's a sample manifest file for the template.

```
{"source-ref": "<S3 image URI>", "animal": "horse"}
{"source-ref": "<S3 image URI>", "animal" : "bird"}
 {"source-ref": "<S3 image URI>", "animal" : "dog"}
 {"source-ref": "<S3 image URI>", "animal" : "cat"}
```

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process your manifest entries and pass them to the template engine.

Naming your Lambda function

The best practice in naming your function is to use one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation functions.

When you're using the console, if you have AWS Lambda functions that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic example, you're just passing through the information from the manifest without doing any additional processing on it. This sample pre-annotation function is written for Python 3.7.

```

import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }

```

}

The JSON object from your manifest will be provided as a child of the event object. The properties inside the taskInput object will be available as variables to your template, so simply setting the value of taskInput to event['dataObject'] will pass all the values from your manifest object to your template without having to copy them individually. If you wish to send more values to the template, you can add them to the taskInput object.

Your post-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer consolidation and scoring as it comes in, you can apply the scoring and/or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

Provide permissions to your post-annotation Lambda

The annotation data will be in a file designated by the s3Uri string in the payload object. To process the annotations as they come in, even for a simple pass through function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is in Python 2.7.

```
import json
import boto3
from urlparse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
    filecont = textFile['Body'].read()
    annotations = json.loads(filecont);

    for dataset in annotations:
        for annotation in dataset['annotations']:
            new_annotation = json.loads(annotation['annotationData']['content'])
            label = {
                'datasetObjectId': dataset['datasetObjectId'],
                'consolidatedAnnotation' : {
                    'content': {
                        event['labelAttributeName']: {
                            'workerId': annotation['workerId'],
                            'boxesInfo': new_annotation,
                            'imageSource': dataset['dataObject']
                        }
                    }
                }
            }
            consolidated_labels.append(label)

    return consolidated_labels
```

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through. What you send back will be an object meeting the [API contract \(p. 132\)](#).

The output of your labeling job

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For a bounding box task, the output you find in the output manifest will look a bit like the demo below. The example has been cleaned up for printing. The actual output will be a single line per record.

Example : JSON in your output manifest

```
{  
    "source-ref": "<URL>",  
    "<label attribute name>":  
    {  
        "workerId": "<URL>",  
        "imageSource": "<image URL>",  
        "boxesInfo": "{\"boundingBox\": {\"boundingBoxes\": [{\"height\": 878, \"label\": \"bird\", \"left\": 208, \"top\": 6, \"width\": 809}], \"inputImageProperties\": {\"height\": 924, \"width\": 1280}}}",  
        "<label attribute name>-metadata":  
        {  
            "type": "groundTruth/custom",  
            "job_name": "<Labeling job name>",  
            "human-annotated": "yes"  
        },  
        "animal": "bird"  
    }  
}
```

Note how the additional `animal` attribute from your original manifest is passed to the output manifest on the same level as the `source-ref` and labeling data. Any properties from your input manifest, whether they were used in your template or not, will be passed to the output manifest.

This should help you create your own custom template.

Demo Template: Labeling Intents with `crowd-classifier`

If you choose a custom template, you'll reach the **Custom labeling task panel**. There you can select from multiple starter templates that represent some of the more common tasks. The templates provide a starting point to work from in building your customized labeling task's template.

In this demonstration, you work with the **Intent Detection** template, which uses the [crowd-classifier \(p. 198\)](#) element, and the AWS Lambda functions needed for processing your data before and after the task.

Topics

- [Starter Intent Detection custom template \(p. 125\)](#)
- [Your Intent Detection custom template \(p. 126\)](#)
- [Your pre-annotation Lambda function \(p. 129\)](#)
- [Your post-annotation Lambda function \(p. 130\)](#)
- [Your labeling job output \(p. 131\)](#)

Starter Intent Detection custom template

This is the intent detection template that is provided as a starting point.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

```

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="{{ task.input.labels | to_json | escape }}"
    header="Pick the most relevant intention expressed by the below text"
  >
    <classification-target>
      {{ task.input.utterance }}
    </classification-target>

    <full-instructions header="Intent Detection Instructions">
      <p>Select the most relevant intention expressed by the text.</p>
      <div>
        <p><strong>Example: </strong>I would like to return a pair of shoes</p>
        <p><strong>Intent: </strong>Return</p>
      </div>
    </full-instructions>

    <short-instructions>
      Pick the most relevant intention expressed by the text
    </short-instructions>
  </crowd-classifier>
</crowd-form>

```

The custom templates use the [Liquid template language](#), and each of the items between double curly braces is a variable. The pre-annotation AWS Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

Your Intent Detection custom template

In the starter template, there are two variables: the `task.input.labels` property in the `crowd-classifier` element opening tag and the `task.input.utterance` in the `classification-target` region's content.

Unless you need to offer different sets of labels with different utterances, avoiding a variable and just using text will save processing time and creates less possibility of error. The template used in this demonstration will remove that variable, but variables and filters like `to_json` are explained in more detail in the [crowd-bounding-box demonstration](#) article.

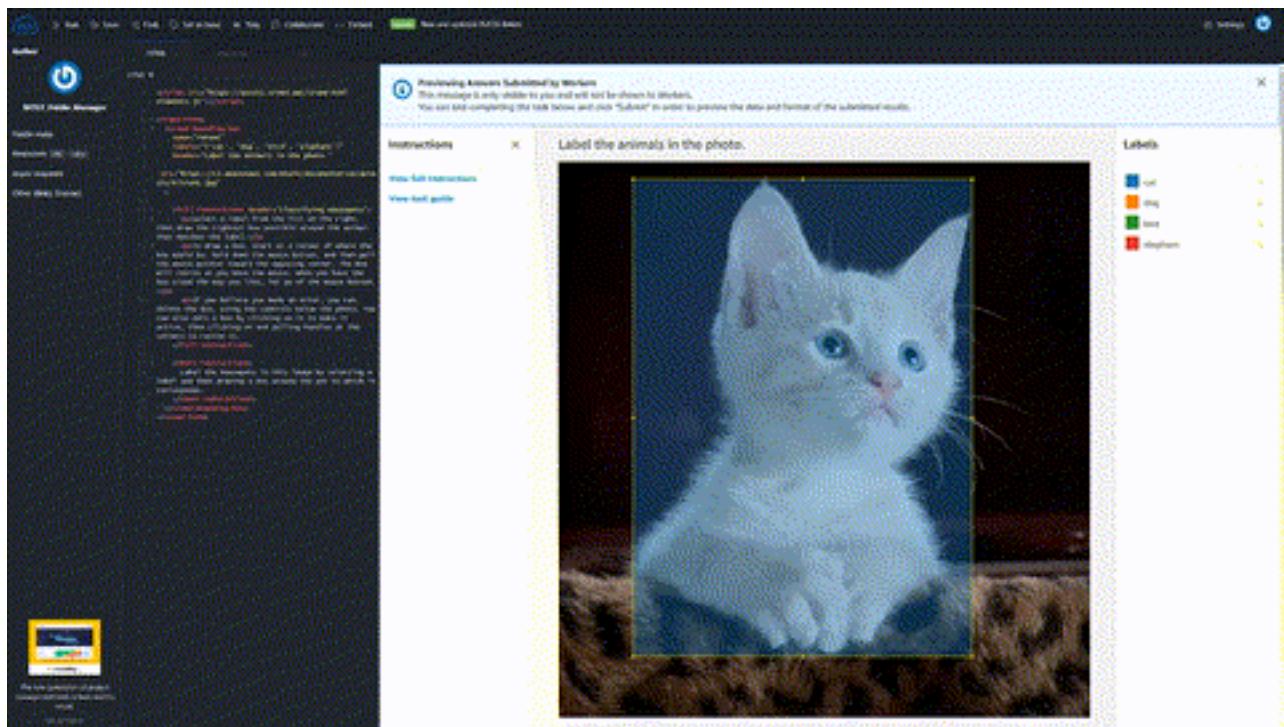
Styling Your Elements

Two parts of these custom elements that sometimes get overlooked are the `<full-instructions>` and `<short-instructions>` regions. Good instructions generate good results.

In the elements that include these regions, the `<short-instructions>` appear automatically in the "Instructions" pane on the left of the worker's screen. The `<full-instructions>` are linked from the "View full instructions" link near the top of that pane. Clicking the link opens a modal pane with more detailed instructions.

You can not only use HTML, CSS, and JavaScript in these sections, you are encouraged to if you believe you can provide a strong set of instructions and examples that will help workers complete your tasks with better speed and accuracy.

Example Try out a sample with JSFiddle



Try out an [example <crowd-classifier> task](#). The example is rendered by JSFiddle, therefore all the template variables are replaced with hard-coded values. Click the "View full instructions" link to see a set of examples with extended CSS styling. You can fork the project to experiment with your own changes to the CSS, adding sample images, or adding extended JavaScript functionality.

Example : Final Customized Intent Detection Template

This uses the [example <crowd-classifier> task](#), but with a variable for the <classification-target>. If you are trying to keep a consistent CSS design among a series of different labeling jobs, you can include an external stylesheet using a <link rel...> element the same way you'd do in any other HTML document.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="['buy', 'eat', 'watch', 'browse', 'leave']"
    header="Pick the most relevant intent expressed by the text below"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Emotion Classification Instructions">
      <p>In the statements and questions provided in this exercise, what category of action is the speaker interested in doing?</p>
      <table>
        <tr>
          <th>Example Utterance</th>
          <th>Good Choice</th>
        </tr>
        <tr>
```

```

<td>When is the Seahawks game on?</td>
<td>
    eat<br>
    <greenbg>watch</greenbg>
    <botchoice>browse</botchoice>
</td>
</tr>
<tr>
    <th>Example Utterance</th>
    <th>Bad Choice</th>
</tr>
<tr>
    <td>When is the Seahawks game on?</td>
    <td>
        buy<br>
        <greenbg>eat</greenbg>
        <botchoice>watch</botchoice>
    </td>
</tr>
</table>
</full-instructions>

<short-instructions>
    What is the speaker expressing they would like to do next?
</short-instructions>
</crowd-classifier>
</crowd-form>
<style>
    greenbg {
        background: #feee23;
        display: block;
    }

    table {
        *border-collapse: collapse; /* IE7 and lower */
        border-spacing: 0;
    }

    th, tfoot, .fakehead {
        background-color: #8888ee;
        color: #f3f3f3;
        font-weight: 700;
    }

    th, td, tfoot {
        border: 1px solid blue;
    }

    th:first-child {
        border-radius: 6px 0 0 0;
    }

    th:last-child {
        border-radius: 0 6px 0 0;
    }

    th:only-child{
        border-radius: 6px 6px 0 0;
    }

    tfoot:first-child {
        border-radius: 0 0 6px 0;
    }

    tfoot:last-child {
        border-radius: 0 0 0 6px;
    }
</style>

```

```

        }

tfoot:only-child{
    border-radius: 6px 6px;
}

td {
    padding-left: 15px ;
    padding-right: 15px ;
}

botchoice {
    display: block;
    height: 17px;
    width: 490px;
    overflow: hidden;
    position: relative;
    background: #fff;
    padding-bottom: 20px;
}

botchoice:after {
    position: absolute;
    bottom: 0;
    left: 0;
    height: 100%;
    width: 100%;
    content: "";
    background: linear-gradient(to top,
        rgba(255,255,255, 1) 55%,
        rgba(255,255,255, 0) 100%
    );
    pointer-events: none; /* so the text is still selectable */
}
</style>
```

Example : Your manifest file

If you are preparing your manifest file manually for a text-classification task like this, have your data formatted in the following manner.

```
{"source": "Roses are red"}
{"source": "Violets are Blue"}
 {"source": "Ground Truth is the best"}
 {"source": "And so are you"}
```

This differs from the manifest file used for the "[Demo Template: Annotation of Images with crowd-bounding-box \(p. 121\)](#)" demonstration in that `source-ref` was used as the property name instead of `source`. The use of `source-ref` designates S3 URLs for images or other files that must be converted to HTTP. Otherwise, `source` should be used like it is with the text strings above.

Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an AWS Lambda that can be called to process your manifest entries and pass them to the template engine.

This Lambda function is required to have one of the following four strings as part of the function name: `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`.

This applies to both your pre-annotation and post-annotation Lambdas.

When you're using the console, if you have Lambdas that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic sample, where you have only one variable, it's primarily a pass-through function. Here's a sample pre-labeling Lambda using Python 3.7.

```
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

The `dataObject` property of the event contains the properties from a data object in your manifest.

In this demonstration, which is a simple pass through, you just pass that straight through as the `taskInput` value. If you add properties with those values to the `event['dataObject']` object, they will be available to your HTML template as Liquid variables with the format `{% task.input.<property name> %}`.

Your post-annotation Lambda function

As part of the job set up, provide the ARN of an Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer-consolidation and scoring as data comes in, you can apply the scoring or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

Set permissions for your post-annotation Lambda function

The annotation data will be in a file designated by the `s3Uri` string in the payload object. To process the annotations as they come in, even for a simple pass through function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is for Python 3.7.

```
import json
import boto3
from urllib.parse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
    filecont = textFile['Body'].read()
    annotations = json.loads(filecont);

    for dataset in annotations:
        for annotation in dataset['annotations']:
            new_annotation = json.loads(annotation['annotationData']['content'])
            label = {
                'datasetObjectId': dataset['datasetObjectId'],
                'consolidatedAnnotation' : {
                    'content': {
                        event['labelAttributeName']: {
                            'workerId': annotation['workerId'],
                            'result': new_annotation,
                            'labeledContent': dataset['dataObject']
                        }
                    }
                }
            }
```

```

        }
        consolidated_labels.append(label)

    return consolidated_labels

```

Your labeling job output

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through.

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For an intent detection task, the output in the output manifest will look a bit like the demo below. The example has been cleaned up and spaced out to be easier for humans to read. The actual output will be more compressed for machine reading.

Example : JSON in your output manifest

```

[
  {
    "datasetObjectId": "<Number representing item's place in the manifest>",
    "consolidatedAnnotation":
    {
      "content":
      {
        "<name of labeling job>":
        {
          "workerId": "private.us-east-1.XXXXXXXXXXXXXXXXXXXXXX",
          "result":
          {
            "intent":
            {
              "label": "<label chosen by worker>"
            }
          },
          "labeledContent":
          {
            "content": "<text content that was labeled>"
          }
        }
      }
    }
  },
  "datasetObjectId": "<Number representing item's place in the manifest>",
  "consolidatedAnnotation":
  {
    "content":
    {
      "<name of labeling job>":
      {
        "workerId": "private.us-east-1.6UDLPKQZHYZWJOSCA4MBJBB7FWE",
        "result":
        {
          "intent":
          {
            "label": "<label chosen by worker>"
          }
        },
        "labeledContent":
        {
          "content": "<text content that was labeled>"
        }
      }
    }
  }
]

```

```

        }
    },
    ...
    ...
]

```

This should help you create and use your own custom template.

Step 3: Processing with AWS Lambda

In this step, you set which AWS Lambda functions to trigger on each dataset object prior to sending it to workers and which function will be used to process the results once the task is submitted. These functions are required.

You will first need to visit the AWS Lambda console or use AWS Lambda's APIs to create your functions. The AmazonSageMakerFullAccess policy is restricted to invoking AWS Lambda functions with one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation Lambdas. If you choose to use names without those strings, you must explicitly provide lambda:InvokeFunction permission to the IAM role used for creating the labeling job.

Select your lambdas from the **Lambda functions** section that comes after the code editor for your custom HTML in the Ground Truth console.

If you need an example, there is an end-to-end demo, including Python code for the Lambdas, in the "[Demo Template: Annotation of Images with crowd-bounding-box \(p. 121\)](#)" document.

Pre-annotation Lambda

Before a labeling task is sent to the worker, your AWS Lambda function will be sent a JSON formatted request to provide details.

Example of a Pre-annotation request

```
{
  "version": "2018-10-16",
  "labelingJobArn": <labelingJobArn>
  "dataObject" : {
    "source-ref": "s3://mybucket/myimage.png"
  }
}
```

The `dataObject` will contain the JSON formatted properties from your manifest's data object. For a very basic image annotation job, it might just be a `source-ref` property specifying the image to be annotated. The JSON line objects in your manifest can be up to 100 kilobytes in size and contain a variety of data.

In return, Ground Truth will require a response formatted like this:

Example of expected return data

```
{
  "taskInput": <json object>,
  "isHumanAnnotationRequired": <boolean> # Optional
}
```

In the previous example, the `<json object>` needs to contain *all* the data your custom form will need. If you're doing a bounding box task where the instructions stay the same all the time, it may just be the

HTTP(S) or S3 resource for your image file. If it's a sentiment analysis task and different objects may have different choices, it would be the object reference as a string and the choices as an array of strings.

Implications of `isHumanAnnotationRequired`

This value is optional because it will default to `true`. The primary use case for explicitly setting it is when you want to exclude this data object from being labeled by human workers.

If you have a mix of objects in your manifest, with some requiring human annotation and some not needing it, you can include a `isHumanAnnotationRequired` value in each data object. You can then use code in your pre-annotation Lambda to read the value from the data object and set the value in your Lambda output.

The pre-annotation Lambda runs first

Before any tasks are available to workers, your entire manifest will be processed into an intermediate form, using your Lambda. This means you won't be able to change your Lambda part of the way through a labeling job and see that have an impact on the remaining tasks.

Post-annotation Lambda

When all workers have annotated the data object or when `TaskAvailabilityLifetimeInSeconds` has been reached, whichever comes first, Ground Truth will send those annotations to your Post-annotation Lambda. This Lambda is generally used for [Consolidate Annotations \(p. 89\)](#). The request object will come in like this:

Example of a post-labeling task request

```
{
  "version": "2018-10-16",
  "labelingJobArn": <labelingJobArn>,
  "labelCategories": [<string>],
  "labelAttributeName": <string>,
  "roleArn" : "string",
  "payload": {
    "s3Uri": <string>
  }
}
```

Note

If no worker work on the data object and `TaskAvailabilityLifetimeInSeconds` has been reached, data object will be marked as failed and not included as part of post annotation lambda invocation.

Post-labeling task Lambda permissions

The actual annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign the necessary permissions to your Lambda to read files from your S3 bucket.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

Example of an annotation data file

```
[
  {
    "datasetObjectId": <string>,
    "dataObject": {
      "s3Uri": <string>,
      "content": <string>
    },
  },
```

```

    "annotations": [
        {
            "workerId": <string>,
            "annotationData": {
                "content": <string>,
                "s3Uri": <string>
            }
        }
    ]
]

```

Essentially, all the fields from your form will be in the `content` object. At this point you can start running data consolidation algorithms on the data, using an AWS database service to store results. Or you can pass some processed/optimized results back to Ground Truth for storage in your consolidated annotation manifests in the S3 bucket you specify for output during the configuration of the labeling job.

In return, Ground Truth will require a response formatted like this:

Example of expected return data

```

[
{
    "datasetObjectId": <string>,
    "consolidatedAnnotation": {
        "content": {
            "<labelattributename>": {
                # ... label content
            }
        }
    }
},
{
    "datasetObjectId": <string>,
    "consolidatedAnnotation": {
        "content": {
            "<labelattributename>": {
                # ... label content
            }
        }
    }
}
.
.
.
]

```

At this point, all the data you're sending to your S3 bucket, other than the `datasetObjectId` will be in the `content` object.

That will result in an entry in your job's consolidation manifest like this:

Example of label format in output manifest

```

{
    "source-ref"/"source" : "<s3uri or content>",
    "<labelAttributeName>": {
        # ... label content from you
    },
    "<labelAttributeName>-metadata": { # This will be added by Ground Truth
        "job_name": <labelingJobName>,
        "type": "groundTruth/custom",
        "human-annotated": "yes",
        "creation_date": <date> # Timestamp of when received from Post-labeling Lambda
    }
}

```

}

Because of the potentially complex nature of a custom template and the data it collects, Ground Truth does not offer further processing of the data or insights into it.

Next

[Custom Workflows via the API \(p. 135\)](#)

Custom Workflows via the API

When you have created your custom UI template (Step 2) and processing Lambda functions (Step 3), you should place the template in an Amazon S3 bucket with a file name format of: <FileName>.liquid.html.

Use the `CreateLabelingJob` action to configure your task. You'll use the location of a custom template ([Step 2: Creating your custom labeling task template \(p. 115\)](#)) stored in a <`filename`>.liquid.html file on S3 as the value for the `UiTemplateS3Uri` field in the `UiConfig` object within the `HumanTaskConfig` object.

For the AWS Lambda tasks described in [Step 3: Processing with AWS Lambda \(p. 132\)](#), the post-annotation task's ARN will be used as the value for the `AnnotationConsolidationLambdaArn` field, and the pre-annotation task will be used as the value for the `PreHumanTaskLambdaArn`.

Monitor Labeling Job Status

To monitor the status of your labeling jobs, you can set up an [Amazon CloudWatch Events](#) (CloudWatch Events) rule for Amazon SageMaker Ground Truth (Ground Truth) to send an event to CloudWatch Events when a labeling job status changes to `Completed`, `Failed`, or `Stopped`.

Once you create a rule, you can add a *target* to it. CloudWatch Events uses this target to invoke another AWS service to process the event. For example, you can create a target using a Amazon Simple Notification Service (Amazon SNS) topic to send a notification to your email when a labeling job status changes.

Prerequisites:

To create a CloudWatch Events rule, you will need an AWS Identity and Access Management (IAM) role with an `events.amazonaws.com` trust policy attached. The following is an example of an `events.amazonaws.com` trust policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "events.amazonaws.com"  
                ]  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

Topics

- [Send Events to CloudWatch Events \(p. 136\)](#)

- [Set Up a Target to Process Events \(p. 136\)](#)

Send Events to CloudWatch Events

To configure a CloudWatch Events rule to get status updates, or *events*, for your Ground Truth labeling jobs, use the AWS Command Line Interface (AWS CLI) [put-rule](#) command. You can filter events that are sent to your rule by status change. For example, you can create a rule that notifies you only if a labeling job status changes to Completed. When using the put-rule command, specify the following to receive labeling job statuses:

- `\\"source\\":[\"aws.sagemaker\"]`
- `\\"detail-type\\":[\"SageMaker Ground Truth Labeling Job State Change\"]`

To configure a CloudWatch Events rule to watch for all status changes, use the following command and replace the placeholder text. For example, replace `"GTLabelingJobStateChanges"` with a unique CloudWatch Events rule name and `"arn:aws:iam::111122223333:role/MyRoleForThisRule"` with the Amazon Resource Number (ARN) of an IAM role with an `events.amazonaws.com` trust policy attached.

```
aws events put-rule --name "GTLabelingJobStateChanges"
  --event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
  Truth Labeling Job State Change\"]}"
  --role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
  --region "region"
```

To filter by job status, use the `\\"detail\\":{\"LabelingJobStatus\":[\"Status\"]}}` syntax. Valid values for *Status* are Completed, Failed, and Stopped.

The following example creates a CloudWatch Events rule that notifies you when a labeling job in us-west-2 (Oregon) changes to Completed.

```
aws events put-rule --name "LabelingJobCompleted"
  --event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
  Truth Labeling Job State Change\"], \"detail\":{\"LabelingJobStatus\":[\"Completed\"]}}"
  --role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
  --region us-west-2
```

The following example creates a CloudWatch Events rule that notifies you when a labeling job in us-east-1 (Virginia) changes to Completed or Failed.

```
aws events put-rule --name "LabelingJobCompletedOrFailed"
  --event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
  Truth Labeling Job State Change\"], \"detail\":{\"LabelingJobStatus\":[\"Completed\",
  \"Failed\"]}}"
  --role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
  --region us-east-1
```

To learn more about the `put-rule` request, see [Event Patterns in CloudWatch Events](#) in the *Amazon CloudWatch Events User Guide*.

Set Up a Target to Process Events

After you have created a rule, events similar to the following are sent to CloudWatch Events. In this example, the labeling job `test-labeling-job`'s status changed to Completed.

```
{
```

```
"version": "0",
"id": "111e1111-11d1-111f-b111-1111b11dc11",
"detail-type": "SageMaker Ground Truth Labeling Job State Change",
"source": "aws.sagemaker",
"account": "123456789012",
"time": "2018-10-06T12:26:13Z",
"region": "us-east-1",
"resources": [
    "arn:aws:sagemaker:us-east-1:111122223333:labeling-job/test-labeling-job"
],
"detail": {
    "LabelingJobStatus": "Completed"
}
}
```

To process events, you need to set up a target. For example, if you want to receive an email when your labeling job status changes, use a procedure in [Setting Up Amazon SNS Notifications](#) in the *Amazon CloudWatch User Guide* to set up an Amazon SNS topic and subscribe your email to it. Once you have created a topic, you can use it to create a target.

To add a target to your CloudWatch Events rule

1. Open the CloudWatch console: <https://console.aws.amazon.com/cloudwatch/home>
2. In the navigation pane, choose **Rules**.
3. Choose the rule that you want to add a target to.
4. Choose **Actions**, and then choose **Edit**.
5. Under **Targets**, choose **Add Target** and choose the AWS service you want to act when a labeling job status change event is detected.
6. Configure your target. For instructions, see the topic for configuring a target in the [AWS documentation for that service](#).
7. Choose **Configure details**.
8. For **Name**, enter a name and, optionally, provide details about the purpose of the rule in **Description**.
9. Make sure that the check box next to **State** is selected so that your rule is listed as **Enabled**.
10. Choose **Update rule**.

Using Amazon Augmented AI for Human Review

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

When you use AI applications such as Amazon Rekognition, Amazon Textract, or your custom machine learning (ML) models you can use Amazon Augmented AI to get human review of low confidence or a random sample of predictions.

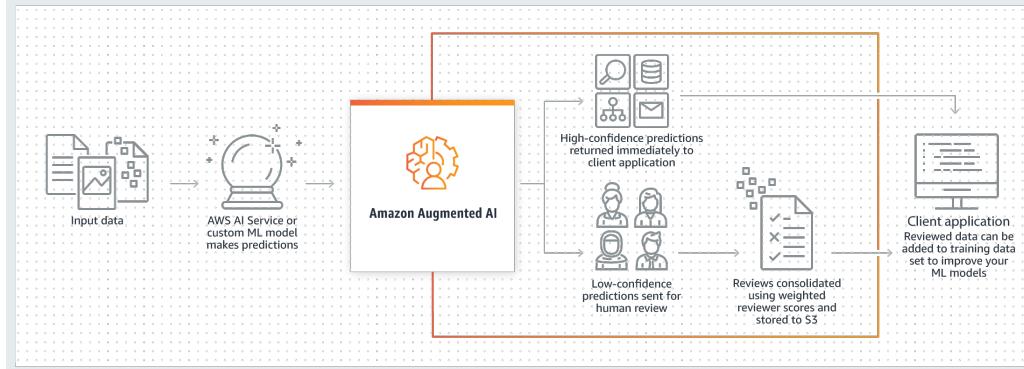
What is Amazon Augmented AI?

Amazon Augmented AI (Amazon A2I) makes it easy to build the workflows required for human review of ML predictions. Amazon A2I brings human review to all developers, removing the undifferentiated heavy lifting associated with building human review systems or managing large numbers of human reviewers.

Many machine learning applications require humans to review low-confidence predictions to ensure the results are correct. For example, extracting information from scanned mortgage application forms

can require human review in some cases due to low-quality scans or poor handwriting. But building human review systems can be time-consuming and expensive because it involves implementing complex processes or *workflows*, writing custom software to manage review tasks and results, and in many cases, managing large groups of reviewers.

Amazon A2I makes it easy to build and manage human reviews for machine learning applications. Amazon A2I provides built-in human review workflows for common machine learning use cases, such as content moderation and text extraction from documents, which allows predictions from Amazon Rekognition and Amazon Textract to be reviewed easily. You can also create your own workflows for ML models built on Amazon SageMaker or any other tools. Using Amazon A2I, you can allow human reviewers to step in when a model is unable to make a high-confidence prediction or to audit its predictions on an ongoing basis.



Topics

- [Get Started with Amazon Augmented AI \(p. 138\)](#)
- [Use Task Types \(p. 140\)](#)
- [Create a Flow Definition \(p. 143\)](#)
- [Create and Start a Human Loop \(p. 159\)](#)
- [Create a Worker UI \(p. 161\)](#)
- [Monitor and Manage Your Human Loop \(p. 170\)](#)
- [Permissions and Security in Amazon Augmented AI \(p. 171\)](#)
- [Use Amazon CloudWatch Events in Amazon Augmented AI \(p. 175\)](#)
- [Use APIs in Amazon Augmented AI \(p. 176\)](#)

Get Started with Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Augmented AI (Amazon A2I) helps you integrate human judgment into AI/ML workflows. With Amazon A2I, you can let AI handle straight-forward data and invoke human reviewers only when their skills are needed.

The AI/ML workflow that you integrate Amazon A2I into defines an Amazon A2I *task type*. Amazon A2I supports two *built-in task types*: Amazon Textract and Amazon Rekognition, and a *custom task type*. The built-in task types integrate Amazon A2I with [Amazon Textract's AnalyzeDocument API](#) operation and [Amazon Rekognition's DetectModerationLabels API](#) operation to incorporate a human review workflow when inference confidence is low for a given object (for example, for an image or

text in a document). The custom task type allows you to use Amazon A2I in custom machine learning applications. For more information about task types, see [Use Task Types \(p. 140\)](#).

To incorporate Amazon A2I into your data labeling workflow for all task types, you need three resources:

- A *worker task template* to create a worker UI. The worker UI displays your input data, such as documents or images, and instructions to workers. It also provides interactive tools that the worker uses to complete your tasks. For more information, see [Create a Worker UI \(p. 161\)](#).
- A *human review workflow*, also referred to as a *flow definition*. You use the flow definition to configure your human workforce and provide information about how to accomplish the labeling task. For built-in task types, you also use the flow definition to identify the conditions under which a review human loop is triggered. For example, Amazon Rekognition can perform image content moderation using machine learning. You can use the flow definition to specify that an image will be sent to a human for content moderation review if Amazon Rekognition's confidence is too low. You can create a flow definition in the Amazon SageMaker console or with the Amazon SageMaker API. To learn more about both of these options, see [Create a Flow Definition \(p. 143\)](#).
- A *human loop* to start your human review workflow. When you use one of the built-in task types, the corresponding AWS service creates and starts a human loop on your behalf when the conditions specified in your flow definition are met or for each object if no conditions were specified. When a human loop is triggered, human review tasks are sent to the workers as specified in the flow definition.

When using a custom task type, you start a human loop using the [Amazon Augmented AI Runtime API](#). When you call `StartHumanLoop` in your custom application, a task is sent to human reviewers.

To learn how to create and start a human loop, see [Create and Start a Human Loop \(p. 159\)](#).

To generate these resources and create a human review workflow, Amazon A2I integrates multiple APIs including the Amazon Augmented AI Runtime Model, the Amazon SageMaker APIs, and APIs associated with your task type. To learn more, see [Use APIs in Amazon Augmented AI \(p. 176\)](#).

Prerequisites

You can create a Amazon A2I human review workflow using both the Amazon SageMaker console and an API. To create a human review workflow, you need the following:

- One or more Amazon S3 buckets in the same AWS Region as the workflow for your input and output data. To create a bucket, follow the instructions in [Create a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
- An IAM role with required permissions to create a human review workflow and an IAM user or role with permission to access Augmented AI. For more information, see [Permissions and Security in Amazon Augmented AI \(p. 171\)](#).
- You're prompted to choose a public, private, or vendor workforce for your human review workflows. If you plan to use a private workforce, you need to set one up ahead of time in the same AWS Region as your Amazon A2I workflow. To learn more about these workforce types, see [Create and Manage Workforces \(p. 177\)](#).

Important

Use the Amazon Mechanical Turk workforce only to process data that is public or has been stripped of all sensitive information. Do not share confidential information, personal information, or protected health information with this workforce. Do not use the Amazon Mechanical Turk workforce when you use Amazon Augmented AI in conjunction with other AWS HIPAA-eligible services, such as Amazon Textract and Amazon Rekognition.

Next Steps

If you're new to Amazon A2I and are integrating a human review workflow with an Amazon Rekognition or Amazon Textract task, we recommend that you start by creating a human review workflow using the console. For more information, see [Create a Flow Definition \(Console\) \(p. 144\)](#).

If you are creating a human review workflow for a custom machine learning task, start by following the steps in [Use Amazon Augmented AI with Custom Task Types \(p. 142\)](#).

Use Task Types

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use Augmented AI to incorporate a human review into your workflow for *built in task types*, Amazon Textract and Amazon Rekognition, or your own custom tasks.

When you create a flow definition using one of the built in task types, you will be able to specify conditions, such as confidence thresholds, that will trigger a human review. For example, you can specify confidence thresholds for form keys in a document, or inference confidence for an image. These services create a human loop on your behalf using Amazon Augmented AI Runtime API when these conditions are met, and supply your input data directly to Amazon A2I to send to human reviewers.

When you use a custom task type, you create and start a human loop using the Amazon Augmented AI Runtime API. For more details, see [Use Amazon Augmented AI with Custom Task Types \(p. 142\)](#). Use the custom task type to incorporate a human review workflow with other AWS service, or your own custom ML application.

The topics in this section provide an overview of these three task types and examples of worker task templates that Amazon A2I provides for Amazon Textract and Amazon Rekognition task types. You specify your task type when creating a flow definition.

Topics

- [Use Amazon Augmented AI with Amazon Textract \(p. 140\)](#)
- [Use Amazon Augmented AI with Amazon Rekognition \(p. 141\)](#)
- [Use Amazon Augmented AI with Custom Task Types \(p. 142\)](#)

Use Amazon Augmented AI with Amazon Textract

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Textract enables you to add document text detection and analysis to your applications. Amazon Augmented AI (Amazon A2I) directly integrated with Amazon Textract's `AnalyzeDocument` API operation. You can use `AnalyzeDocument` to analyze a document for relationships between detected items. When you add an Amazon A2I human review look to an `AnalyzeDocument` request, Amazon A2I will monitor the Amazon Textract results and send a document to a human when the inference confidence for key-value pairs is low. For example, if you want a human to review a specific key like "Full name:" and their associated input-values you can create a trigger to start a human review anytime the key "Full name:" is detected or when the inference confidence for that key is low.

You can specify when Amazon Textract sends a task to a human worker for review when creating a human review workflow, or flow definition. Within your flow definition, you can specify thresholds

for Identification confidence and Qualification confidence for specific form keys and for all form keys detected with low confidence.

- **Identification confidence**—The confidence score for key-value pairs detected within a form.
- **Qualification confidence**—The confidence score for text contained within key and value respectively in a form.

In the image in the following section, **Full Name: Jane Doe** is the key-value pair, and **Full Name** and **Jane Doe** are the key and value respectively.

A2I Textract Worker Console Preview

When they're assigned a review task in an Amazon Textract workflow, workers might see UI similar to the following:

The screenshot shows a worker review interface for an "Employment Application". On the left, there's an "Instructions" panel with links to "View full instructions" and "View tool guide". It also contains a "Key-value pair" detection tool where a "Jane Doe" entry is highlighted with a red box. The main area displays the document content with several fields: "Full Name: Jane Doe", "Phone number: 550-0100", "Home address: 123 Any Street, Any Town, USA", and "Mail address: same as home address". To the right, there's a "Key-value pairs to review" panel with two entries. The first entry for "Full name: Jane Done" has a "Yes" radio button selected. The second entry for "Phone number: 550-0100" has a "No" radio button selected. At the bottom, there are "Zoom in", "Zoom out", "Move", and "Fit image" buttons, along with a "Submit" button and a "No adjustment needed" checkbox. A large watermark "Sample" is visible across the interface.

You can customize this UI in the Amazon SageMaker console when you create your human review definition, or by creating and using a custom template. To learn more, see [Create a Worker UI \(p. 161\)](#).

Integrate a Human Review into Amazon Textract

To integrate a human review into an Amazon Textract text detection and analysis job, you need to create a flow definition, and then use the Amazon Textract API to integrate that flow definition into your workflow. To learn how to create a flow definition using the Amazon SageMaker console or Augmented AI API, see the following topics:

- [Create a Flow Definition \(Console\) \(p. 144\)](#)
- [Create a Flow Definition \(API\) \(p. 146\)](#)

After you've created your flow definition, see [Using Augmented AI with Amazon Textract](#) to learn how to integrate your flow definition into your Amazon Textract task.

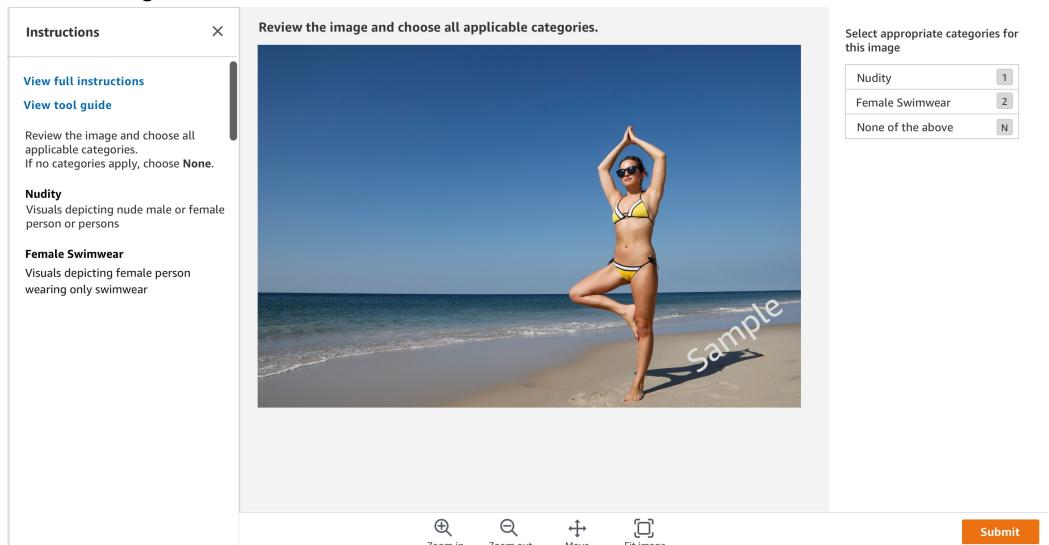
Use Amazon Augmented AI with Amazon Rekognition

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Amazon Rekognition makes it easy to add image analysis to your applications. The Amazon Rekognition `DetectModerationLabels` API operation is directly integrated with Amazon A2I so that you can easily create a human loop to review unsafe images, such as explicit adult or violent content. You can use `DetectModerationLabels` to configure a human loop using a flow definition ARN. This enables Amazon A2I to send low-confidence predictions from Amazon Rekognition to a human for review.

A2I Rekognition Worker Console Preview

When they're assigned a review task in an Amazon Rekognition workflow, workers might see UI similar to the following:



You can customize this interface in the Amazon SageMaker console when you create your human review definition, or by creating and using a custom template. To learn more, see [Create a Worker UI \(p. 161\)](#).

Integrating a Human Review into Amazon Rekognition

To integrate a human review into an Amazon Rekognition, see the following topics:

- [Create a Flow Definition \(Console\) \(p. 144\)](#)
- [Create a Flow Definition \(API\) \(p. 146\)](#)

After you've created your flow definition, see [Using Augmented AI with Amazon Rekognition](#) to learn how to integrate your flow definition into your Amazon Rekognition task.

Use Amazon Augmented AI with Custom Task Types

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use Amazon Augmented AI (Amazon A2I) to incorporate a human review (human loop) into your custom machine learning workflow. This is known as a *custom task type*.

To create a human loop using a flow definition, integrate it into your application, and monitor the results

1. Complete the Amazon A2I [Prerequisites \(p. 139\)](#). Note the following:

- The path the Amazon Simple Storage Service (Amazon S3) bucket or buckets where you will store your input and output data.
 - The Amazon Resource Name (ARN) of an AWS Identity and Access Management (IAM) role with required permissions attached.
 - (Optional) If you plan to use a private workforce, the ARN of your workforce.
2. Using HTML elements, create a custom worker template which Amazon A2I uses to generate your worker task UI. To learn how to create a custom template, see [Create Custom Worker Templates \(p. 163\)](#).
 3. Use the custom worker template from Step 2 to generate a worker task template in the Amazon SageMaker console. To learn how, see [Create a Custom Worker Template \(Console\) \(p. 162\)](#).

In the next Step you will create a flow definition:

- If you want to create a flow definition using the Amazon SageMaker API, note the ARN of this worker task template for the next step.
 - If you are creating a flow definition using the console, your template will automatically appear in **Worker task template** section when you choose **Create human review workflow**.
4. When creating your flow definition, provide the path to your S3 buckets, your IAM role ARN, and your worker template.
 - Learn how to create a flow definition using the Amazon SageMaker `CreateFlowDefinition` API: [Create a Flow Definition \(API\) \(p. 146\)](#).
 - Learn how to create a flow definition using the Amazon SageMaker console: [Create a Flow Definition \(Console\) \(p. 144\)](#).
 5. Configure your human loop using the [Amazon A2I Runtime API](#). To learn how, see [Create and Start a Human Loop \(p. 159\)](#).
 6. To control when human reviews are initiated in your application, specify conditions under which `StartHumanLoop` is called in your application. Human loop activation conditions, such as confidence thresholds that trigger the human loop, are not available when using Amazon A2I with custom task types. Every `StartHumanLoop` invocation results in a human review.

Once you have started a human loop, you can manage and monitor your loops using the Amazon Augmented AI Runtime API and Amazon EventBridge (also known as Amazon CloudWatch Events). To learn more, see [Monitor and Manage Your Human Loop \(p. 170\)](#).

Create a Flow Definition

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Use an Amazon Augmented AI (Amazon A2I) *human review workflow*, or *flow definition*, to specify the following:

- For the Amazon Textract and Amazon Rekognition built in task types, the conditions under which your human loop will be called.
- The workforce that your tasks will be sent to.
- The instructions that your workforce will receive. This is called a *worker task template*.
- The configuration of your worker tasks, including the number of workers that receive a task and time limits to complete tasks.
- Where your output data will be stored.

You can create a flow definition in the Amazon SageMaker console or using the Amazon SageMaker [CreateFlowDefinition](#) operation. You can build a worker task template using the console for Amazon Textract and Amazon Rekognition task types while creating your flow definition.

Important

Human loop activation conditions, which trigger the human loop—for example, confidence thresholds—are not available for Amazon A2I custom task types. When using the console to create a flow definition for a custom task type, you can't specify activation conditions. When using the Amazon A2I API to create a flow definition for a custom task type, you can't set the `HumanLoopActivationConditions` attribute of the `HumanLoopActivationConditionsConfig` parameter. To control when human reviews are initiated, specify conditions under which `StartHumanLoop` is called in your custom application. In this case, every `StartHumanLoop` invocation results in a human review. For more information, see [Use Amazon Augmented AI with Custom Task Types \(p. 142\)](#).

Prerequisites

To create a flow definition, you must have completed the prerequisites described in [Prerequisites \(p. 139\)](#).

If you use the API to create a flow definition for any task type, or if you use a custom task type when creating a flow definition in the console, first you will need to create a worker task template. For more information, see [Create a Worker UI \(p. 161\)](#).

If you want to preview your worker task template while creating a flow definition for a built in task type in the console, ensure that you grant the role that you use to create the flow definition permission to access the Amazon S3 bucket that contains your template artifacts using a policy like the one described in [Enable Worker Task Template Previews \(p. 174\)](#).

Topics

- [Create a Flow Definition \(Console\) \(p. 144\)](#)
- [Create a Flow Definition \(API\) \(p. 146\)](#)
- [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 147\)](#)

Create a Flow Definition (Console)

Use this procedure to create a Amazon Augmented AI (Amazon A2I) human review workflow using the Amazon SageMaker console. If you are new to Amazon A2I, we suggest that you create a private work team using people in your organization, and use this work team's ARN when creating your flow definition. To learn how to set up a private workforce and create a work team, see [Create a Private Workforce \(Console\) \(p. 180\)](#). If you have already set up a private workforce, see [Create a Work Team Using the Amazon SageMaker Console \(p. 183\)](#) to learn how to add a work team to that workforce.

If you are using Amazon A2I with one of the built in task types, you can create worker instructions using a default worker task template provided by Augmented AI while creating a human review workflow in the console. To see samples of the default templates provided by Augmented AI, see the built in task types in [Use Task Types \(p. 140\)](#).

To create flow definition (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, under the **Augmented AI** section, choose **Human review workflows** and then choose **Create human review workflow**.
3. In **Overview**, do the following:
 - a. For **Name**, enter a unique workflow name. The name must be lowercase, unique within the Region in your account, and can have up to 63 characters. Valid characters include: a-z, 0-9, and - (hyphen).

- b. For **S3 location for output**, enter the S3 bucket where you want to store the human review results. The bucket must be located in the same AWS Region as the workflow.
 - c. For **IAM role**, choose the role that has the required permissions. If you choose a built in task type and want to preview your worker template in the console, provide a role with the type of policy described in [Enable Worker Task Template Previews \(p. 174\)](#) attached.
4. For **Task type**, choose the task type that you want the human worker to perform.
 5. If you chose the Amazon Rekognition or Amazon Textract task type, specify the conditions that will invoke human review.
 - For Amazon Rekognition image moderation tasks, choose an inference confidence score threshold interval that triggers human review.
 - For Amazon Textract tasks, you can trigger a human review when specific form keys are missing or when form key detection confidence is low. You can also trigger a human review if, after evaluating all of the form keys in the text, confidence is lower than your required threshold for any form key. You will see two variables that you can use to specify your confidence thresholds: **Identification confidence** and **Qualification confidence**. To learn more about these variables, see [Use Amazon Augmented AI with Amazon Textract \(p. 140\)](#).
 6. Configure and specify your worker task template:
 - a. If you are using the Amazon Rekognition or Amazon Textract task type:
 - In the **Create template** section:
 - To create instructions for your workers using the Amazon A2I default template for Amazon Rekognition and Amazon Textract task types, choose **Build from a default template**.
 - If you choose **Build from a default template**, create your instructions under **Worker task design**:
 - Provide a **Template name** that is unique in the AWS Region you are in.
 - In the **Instructions** section, provide detailed instructions on how to complete your task. To help workers achieve greater accuracy, provide good and bad examples.
 - (Optional) In **Additional instructions**, provide your workers with additional information and instructions.
 - For information on creating effective instructions, see [Creating Good Worker Instructions \(p. 170\)](#).
 - To select a custom template that you've created, choose it from the **Template** menu and provide a **Task description** to briefly describe the task for your workers. To learn how to create a custom template, see [Create a Custom Worker Template \(Console\) \(p. 162\)](#).
 - b. If you are using the custom task type:
 - In the **Worker task template** section, select your template from the drop down menu. This menu will list all of the templates that you have created in the Amazon SageMaker console. To learn how to create a template for custom task types, see [Create a Worker UI \(p. 161\)](#).
7. (Optional) Preview your worker template:

For Amazon Rekognition and Amazon Textract task types, you have the option to choose **See a sample worker task** to preview your worker task UI.

If you are creating a flow definition for a custom task type, you can preview your worker task UI using the `RenderUiTemplate` operation. For more information, see [Preview a Worker Task Template \(p. 169\)](#).
8. For **Workers**, choose a workforce type.

9. Choose **Create**.

Next Steps:

After you've created a human review workflow, it appears in the console under **Human review workflows**. To see your flow definition's Amazon Resource Name (ARN) and configuration details, choose the workflow by selecting its name.

If you are using a built in task type, you can use the flow definition ARN to start a human loop using that AWS service's API (i.e. the Amazon Textract API). For custom task types, you can use the ARN to start a human loop using the Amazon Augmented AI Runtime API. To learn more about both options, see [Create and Start a Human Loop \(p. 159\)](#).

Create a Flow Definition (API)

To create a flow definition using the Amazon SageMaker API, you use the `CreateFlowDefinition` operation. For an overview of the `CreateFlowDefinition` operation, and details about each parameter, see [CreateFlowDefinition](#).

To create a flow definition (API)

1. For `FlowDefinitionName`, enter a unique name. The name must be unique within the Region in your account, and can have up to 63 characters. Valid characters include: a-z, 0-9, and - (hyphen).
2. For `RoleArn`, enter the ARN of the role that you configured to grant access to your data sources.
3. For `HumanLoopConfig`, enter information about the workers and what they should see. For information about each parameter in `HumanLoopConfig`, see [HumanLoopConfig](#).
4. (Optional) If you are using a built in task type, provide conditions that trigger a human loop in `HumanLoopActivationConfig`. To learn how to create the input required for the `HumanLoopActivationConfig` parameter, see [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 147\)](#). If you do not specify conditions here, when you provide a flow definition to the AWS service associated with a built in task type (i.e. Amazon Textract or Amazon Rekognition), that service will send every task to a human worker for review.

If you are using a custom task type, `HumanLoopActivationConfig` is disabled. To learn how to control when tasks are sent to human workers using a custom task type, see [Use Amazon Augmented AI with Custom Task Types \(p. 142\)](#).

5. For `OutputConfig`, indicate where in Amazon Simple Storage Service (Amazon S3) to store the output of the human loop.
6. (Optional) Use `Tags` to enter key value pairs to help you categorize and organize a flow definition. Each tag consists of a key and a value, both of which you define.

The following is an example of a request to create an Amazon Rekognition human loop using the AWS Python SDK (Boto3). Replace '`AWS/Rekognition/DetectModerationLabels/Image/V3`' with '`AWS/Texttract/AnalyzeDocument/Forms/V1`' to create a Amazon Textract human loop. For more information, see the [Boto 3 Augmented AI Runtime documentation](#).

```
response = client.create_flow_definition(  
    FlowDefinitionName='string',  
    HumanLoopActivationConfig={  
        'HumanLoopRequestSource': {  
            'AwsManagedHumanLoopRequestSource': 'AWS/Rekognition/DetectModerationLabels/  
Image/V3'  
        },  
        'HumanLoopActivationConditionsConfig': {
```

```
        'HumanLoopActivationConditions': 'string'
    },
},
HumanLoopConfig={
    'WorkteamArn': 'string',
    'HumanTaskUiArn': 'string',
    'TaskTitle': 'string',
    'TaskDescription': 'string',
    'TaskCount': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskKeywords': [
        'string',
    ],
    'PublicWorkforceTaskPrice': {
        'AmountInUsd': {
            'Dollars': 123,
            'Cents': 123,
            'TenthFractionsOfACent': 123
        }
    }
},
OutputConfig={
    'S3OutputPath': 'string',
    'KmsKeyId': 'string'
},
RoleArn='string',
Tags:[
    {
        'Key': 'string',
        'Value': 'string'
    },
]
)
```

Next Steps:

The return value of a successful call of the `CreateFlowDefinition` API operation is a flow definition Amazon Resource Name (ARN).

If you are using a built in task type, you can use the flow definition ARN to start a human loop using that AWS service's API (i.e. the Amazon Textract API). For custom task types, you can use the ARN to start a human loop using the Amazon Augmented AI Runtime API. To learn more about both of these options, see [Create and Start a Human Loop \(p. 159\)](#).

JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

The `HumanLoopActivationConditions` is an input parameter of the `CreateFlowDefinition` API. This parameter is a JSON-formatted string. The JSON models the conditions under which a human loop is created, when those conditions are evaluated against the response from an integrating AI service API (such as `Rekognition.DetectModerationLabels` or `Textract.AnalyzeDocument`). This response is referred to as an *inference*. For example, Amazon Rekognition sends an inference of a moderation label with an associated confidence score. In this example, the inference is the model's best estimate of the appropriate label for an image. For Amazon Textract, inference is made on the association between

blocks of text (*key-value pairs*), such as the association between `Name:` and `Sue` in a form as well as content within a block of text, or *word block*, such as '`Name`'.

The following is the schema for the JSON. At the top level, the `HumanLoopActivationConditions` has a JSON array, `Conditions`. Each member of this array is an independent condition that, if evaluated to true, will result in Amazon A2I creating a human loop. Each such independent condition can be a primitive condition or a complex condition. A simple condition has the following attributes:

- `ConditionType`: This attribute identifies the type of condition. Each AWS AI service API that integrates with Amazon A2I defines its own set of allowed `ConditionType`s.
 - `Rekognition DetectModerationLabels` – This API supports the `ModerationLabelConfidenceCheck` and `Sampling ConditionType` values.
 - `Texttract AnalyzeDocument` – This API supports the `ImportantFormKeyConfidenceCheck` and `Sampling ConditionType` values.
- `ConditionParameters` – This is a JSON object that parameterizes the condition. The set of allowed attributes of this object is dependent on the value of the `ConditionType`. Each `ConditionType` defines its own set of `ConditionParameters`.

A member of the `Conditions` array can model a complex condition. This is accomplished by logically connecting primitive conditions using the `And` and `Or` logical operators and nesting the underlying primitive conditions. Up to two levels of nesting are supported.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "definitions": {
    "Condition": {
      "type": "object",
      "properties": {
        "ConditionType": {
          "type": "string"
        },
        "ConditionParameters": {
          "type": "object"
        }
      },
      "required": [
        "ConditionType"
      ]
    },
    "OrConditionArray": {
      "type": "object",
      "properties": {
        "Or": {
          "type": "array",
          "minItems": 2,
          "items": {
            "$ref": "#/definitions/ComplexCondition"
          }
        }
      }
    },
    "AndConditionArray": {
      "type": "object",
      "properties": {
        "And": {
          "type": "array",
          "minItems": 2,
          "items": {
            "$ref": "#/definitions/ComplexCondition"
          }
        }
      }
    }
  }
}
```

```
        }
    },
    "ComplexCondition": {
        "anyOf": [
            {
                "$ref": "#/definitions/Condition"
            },
            {
                "$ref": "#/definitions/OrConditionArray"
            },
            {
                "$ref": "#/definitions/AndConditionArray"
            }
        ]
    }
},
"type": "object",
"properties": {
    "Conditions": {
        "type": "array",
        "items": {
            "$ref": "#/definitions/ComplexCondition"
        }
    }
}
}
```

Note

Human loop activation conditions aren't available for human review workflows that are integrated with custom task types. The `HumanLoopActivationConditions` parameter is disabled for custom task types.

Topics

- [Use Human Loop Activation Conditions JSON Schema with Amazon Textract \(p. 149\)](#)
- [Use Human Loop Activation Conditions JSON Schema with Amazon Rekognition \(p. 154\)](#)

Use Human Loop Activation Conditions JSON Schema with Amazon Textract

When used with Amazon A2I, the `AnalyzeDocument` operation supports the following inputs in the `ConditionType` parameter:

- `ImportantFormKeyConfidenceCheck` – Use this condition to create a human loop when inference confidence is within a specified range for document form keys and word blocks. A *form key* is any word in a document that is associated with an input. The input is called a *value*. Together, form keys and values are referred to as *key-value pairs*. A *word block* refers to the words that Amazon Textract recognizes inside of a detected block of text. To learn more about Amazon Textract document blocks, see [Documents and Block Objects](#) in the *Amazon Textract Developer Guide*.
- `Sampling` – Use this condition to specify a percentage of inferences to send to humans for review, regardless of confidence scores. Use this condition to:
 - Audit your ML model by randomly sampling all of your model's inferences and sending a specified percentage to humans for review.
 - (With the `ImportantFormKeyConfidenceCheck` condition) Randomly sample a percentage of the inferences that met the conditions specified in `ImportantFormKeyConfidenceCheck` to start a human loop and send only the specified percentage to humans for review.

Note

If you send the same request to `AnalyzeDocument` multiple times, the result of `Sampling` will not change for the inference of that input. For example, if you make an `AnalyzeDocument`

request once, and Sampling doesn't trigger a HumanLoop, subsequent requests to AnalyzeDocument with the same configuration will not trigger a human loop.

ImportantFormKey Inputs and Results

The ImportantFormKeyConfidenceCheck ConditionType supports the following ConditionParameters:

- ImportantFormKey – A string representing a key in a key-value pair detected by Amazon Textract that needs to be reviewed by human workers. If the value of this parameter is the special catch-all value (*), then all keys are considered to be matched to the condition. You can use this to model the case where any key-value pair satisfying certain confidence thresholds needs human review.
- ImportantFormKeyAliases – An array that represents alternate spellings or logical equivalents for the important form key.
- KeyValueBlockConfidenceEquals
- KeyValueBlockConfidenceLessThan
- KeyValueBlockConfidenceLessThanEquals
- KeyValueBlockConfidenceGreaterThan
- KeyValueBlockConfidenceGreaterThanOrEqual
- WordBlockConfidenceEquals
- WordBlockConfidenceLessThan
- WordBlockConfidenceLessThanOrEqual
- WordBlockConfidenceGreaterThan
- WordBlockConfidenceGreaterThanOrEqual

When you use the ImportantFormKeyConfidenceCheck ConditionType, Amazon A2I sends the key-value block and word block inferences of the key-value blocks and associated aliases that you specified in ImportantFormKey and ImportantFormKeyAliases for human review.

If you use the default worker task template that is provided in the **Human review workflows** section of the Amazon SageMaker console, these inferences are included in the worker UI when a worker opens your task. If you use a custom worker task template, you need to include the <task.input.selectedAiServiceResponse.blocks> custom HTML element to access these inferences. For an example of a custom template that uses this HTML element, see [Custom Template Example for Amazon Textract \(p. 164\)](#).

Sampling Inputs and Results

The Sampling ConditionType supports the RandomSamplingPercentage ConditionParameters. The input for RandomSamplingPercentage must be real number between 0.01 and 100. This number represents the percentage of data that qualifies for a human review and will be sent to humans for review. If you use the Sampling condition without any other conditions, this number represents the percentage of all resulting inferences made by the AnalyzeDocument operation from a single request that will be sent to humans for review.

If you specify the Sampling condition without any other condition type, all key-value and block inferences are sent to workers for review.

If you use the default worker task template that is provided in the **Human review workflows** section of the Amazon SageMaker console, inferences are included in the worker UI when a worker opens your task. If you use a custom worker task template, you need to include the <task.input.selectedAiServiceResponse.blocks> custom HTML element to access these inferences. For an example of a custom template that uses this HTML element, see [Custom Template Example for Amazon Textract \(p. 164\)](#).

Examples

While only one condition needs to evaluate to true to trigger a human loop, Amazon A2I will evaluate all conditions for each object analyzed by Amazon Textract. The human reviewers are asked to review the important form keys for all the conditions that evaluated to true.

Example 1: Detect important form keys with confidence scores in a specified range trigger HumanLoop

Following is an example of a `HumanLoopActivationConditions` JSON that triggers a `HumanLoop` if any one of the following three conditions is met:

- Textract AnalyzeDocument API returns a key-value pair whose key is one of `Employee Name`, `Name`, or `EmployeeName`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.
- Textract AnalyzeDocument API returns a key-value pair whose key is one of `Pay Date`, `PayDate`, `DateOfPay`, or `pay-date`, with the confidence of the key-value block being less than 65 and the confidences of each of the Word blocks making up the key and value being less than 85.
- Textract AnalyzeDocument API returns a key-value pair whose key is one of `Gross Pay`, `GrossPay`, or `GrossAmount`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.

```
{  
    "Conditions": [  
        {  
            "ConditionType": "ImportantFormKeyConfidenceCheck",  
            "ConditionParameters": {  
                "ImportantFormKey": "Employee Name",  
                "ImportantFormKeyAliases": [  
                    "Name",  
                    "EmployeeName"  
                ],  
                "KeyValueBlockConfidenceLessThan": 60,  
                "WordBlockConfidenceLessThan": 85  
            }  
        },  
        {  
            "ConditionType": "ImportantFormKeyConfidenceCheck",  
            "ConditionParameters": {  
                "ImportantFormKey": "Pay Date",  
                "ImportantFormKeyAliases": [  
                    "PayDate",  
                    "DateOfPay",  
                    "pay-date"  
                ],  
                "KeyValueBlockConfidenceLessThan": 65,  
                "WordBlockConfidenceLessThan": 85  
            }  
        },  
        {  
            "ConditionType": "ImportantFormKeyConfidenceCheck",  
            "ConditionParameters": {  
                "ImportantFormKey": "Gross Pay",  
                "ImportantFormKeyAliases": [  
                    "GrossPay",  
                    "GrossAmount"  
                ],  
                "KeyValueBlockConfidenceLessThan": 60,  
                "WordBlockConfidenceLessThan": 85  
            }  
        }  
    ]  
}
```

```
    ]  
}
```

Example 2: Use ImportantFormKeyConfidenceCheck

In the following example, if Amazon Textract detects a key-value pair whose confidence for the key-value block is less than 60 and is less than 90 for any underlying word blocks, a HumanLoop is created. The human reviewers are asked to review all the form key-value pairs that matched the confidence value comparisons.

```
{  
    "Conditions": [  
        {  
            "ConditionType": "ImportantFormKeyConfidenceCheck",  
            "ConditionParameters": {  
                "ImportantFormKey": "*"  
                "KeyValueBlockConfidenceLessThan": 60,  
                "WordBlockConfidenceLessThan": 90  
            }  
        }  
    ]  
}
```

Example 3: Use Sampling

In the following example, 5.2% of inferences resulting from an Amazon Textract AnalyzeDocument request will be sent to human workers for review. All detected key-value pairs returned by Amazon Textract are sent to workers for review.

```
{  
    "Conditions": [  
        {  
            "ConditionType": "Sampling",  
            "ConditionParameters": {  
                "RandomSamplingPercentage": 5.2  
            }  
        }  
    ]  
}
```

Example 4: Use Sampling and ImportantFormKeyConfidenceCheck with the And operator

In this example, 5.2% of key-value pairs detected by Amazon Textract whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with the confidence of the key-value block less than 65 and the confidences of each of the word blocks making up the key and value less than 85 are sent to workers for review.

```
{  
    "Conditions": [  
        {  
            "And": [  
                {  
                    "ConditionType": "Sampling",  
                    "ConditionParameters": {  
                        "RandomSamplingPercentage": 5.2  
                    }  
                },  
                {  
                    "ConditionType": "ImportantFormKeyConfidenceCheck",  
                    "ConditionParameters": {  
                        "ImportantFormKey": "Pay Date",  
                        "KeyValueBlockConfidenceLessThan": 65,  
                        "WordBlockConfidenceLessThan": 85  
                    }  
                }  
            ]  
        }  
    ]  
}
```

```
        "ImportantFormKey": "Pay Date",
        "ImportantFormKeyAliases": [
            "PayDate",
            "DateOfPay",
            "pay-date"
        ],
        "KeyValueBlockConfidenceLessThan": 65,
        "WordBlockConfidenceLessThan": 85
    }
}
]
```

Example 5: Use Sampling and ImportantFormKeyConfidenceCheck with the And operator

Use this example to configure your human review workflow to always send low confidence inferences of a specified key-value pair for human review and sample high confidence inference of a key-value pair at a specified rate.

In the following example, a human review is triggered in one of the following ways:

- Key-value pairs detected whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with key-value and word block confidences less than 60 will be sent for human review. Only the Pay Date form key (and its aliases) and associated values are sent to workers to review.
 - 5.2% of key-value pairs detected whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with key-value and word block confidences greater than 90 will be sent for human review. Only the Pay Date form key (and its aliases) and associated values are sent to workers to review.

```
{
    "Conditions": [
        {
            "Or": [
                {
                    "ConditionType": "ImportantFormKeyConfidenceCheck",
                    "ConditionParameters": {
                        "ImportantFormKey": "Pay Date",
                        "ImportantFormKeyAliases": [
                            "PayDate",
                            "DateOfPay",
                            "pay-date"
                        ],
                        "KeyValueBlockConfidenceLessThan": 60,
                        "WordBlockConfidenceLessThan": 60
                    }
                },
                {
                    "And": [
                        {
                            "ConditionType": "Sampling",
                            "ConditionParameters": {
                                "RandomSamplingPercentage": 5.2
                            }
                        },
                        {
                            "ConditionType": "ImportantFormKeyConfidenceCheck",
                            "ConditionParameters": {
                                "ImportantFormKey": "Pay Date",
                                "ImportantFormKeyAliases": [
                                    "PayDate",
                                    "DateOfPay",
                                    "pay-date"
                                ]
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

Example 6: Use Sampling and ImportantFormKeyConfidenceCheck with the Or operator

In the following example, the Amazon Textract AnalyzeDocument operation returns a key-value pair whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with the confidence of the key-value block less than 65 and the confidences of each of the word blocks making up the key and value less than 85. Additionally, 5.2% of all other key-value pairs detected are randomly chosen to start a human loop.

```
{
  "Conditions": [
    {
      "Or": [
        {
          "ConditionType": "Sampling",
          "ConditionParameters": {
            "RandomSamplingPercentage": 5.2
          }
        },
        {
          "ConditionType": "ImportantFormKeyConfidenceCheck",
          "ConditionParameters": {
            "ImportantFormKey": "Pay Date",
            "ImportantFormKeyAliases": [
              "PayDate",
              "DateOfPay",
              "pay-date"
            ],
            "KeyValueBlockConfidenceLessThan": 65,
            "WordBlockConfidenceLessThan": 85
          }
        }
      ]
    }
  ]
}
```

[Use Human Loop Activation Conditions JSON Schema with Amazon Rekognition](#)

When used with Amazon A2I, the `Amazonrekognition.DetectModerationLabels` operation supports the following inputs in the `ConditionType` parameters:

- **ModerationLabelConfidenceCheck** – Use this condition type to create a human loop when inference confidence is low for one or more specified labels.
 - **Sampling** – Use this condition to specify a percentage of all inferences to send to humans for review. Use this condition to:
 - Audit your ML model by randomly sampling all of your model's inferences and sending a specified percentage to humans for review.

- (With the `ModerationLabelConfidenceCheck` condition) Randomly sample a percentage of the inferences that met the conditions specified in `ModerationLabelConfidenceCheck` to start a human loop and send only the specified percentage to humans for review.

Note

If you send the same request to `DetectModerationLabels` multiple times, the result of `Sampling` will not change for the inference of that input. For example, if you make a `DetectModerationLabels` request once, and `Sampling` does not trigger a `HumanLoop`, subsequent requests to `DetectModerationLabels` with the same configuration won't trigger a human loop.

[ModerationLabelConfidenceCheck Inputs](#)

For the `ModerationLabelConfidenceCheck` `ConditionType`, the following `ConditionParameters` are supported:

- `ModerationLabelName` – The exact (case-sensitive) name of a [ModerationLabel](#) detected by the Amazon Rekognition `DetectModerationLabels` operation. You can specify the special catch-all value (*) to denote any moderation label.
- `ConfidenceEquals`
- `ConfidenceLessThan`
- `ConfidenceLessThanEquals`
- `ConfidenceGreaterThanOrEqual`
- `ConfidenceGreaterThanOrEqual`

When you use the `ModerationLabelConfidenceCheck` `ConditionType`, Amazon A2I sends label inferences for the labels that you specified in `ModerationLabelName` for human review.

If you use the default worker task template that is provided in the [Human review workflows](#) section of the Amazon SageMaker console, these inferences are included in the worker UI when a worker opens your task. If you use a custom worker task template, you need to include the `<task.input.selectedAiServiceResponse.moderationLabels>` HTML element to access these inferences. For an example of a custom template that uses this HTML element, see [Custom Template Example for Amazon Rekognition \(p. 165\)](#).

[Sampling Inputs](#)

The `Sampling` `ConditionType` supports the `RandomSamplingPercentage` `ConditionParameters`. The input for the `RandomSamplingPercentage` parameter should be real number between 0.01 and 100. This number represents the percentage of inferences that qualifies for a human review that are sent to humans for review. If you use the `Sampling` condition without any other conditions, this number represents the percentage of all inferences that result from a single `DetectModerationLabel` request that are sent to humans for review.

When you use `Sampling` without any other condition types, all label inferences are sent to workers for review. If you use the default worker task template that is provided in the [Human review workflows](#) section of the Amazon SageMaker console, these inferences are included in the worker UI when a worker opens your task. If you use a custom worker task template, you need to include the `<task.input.selectedAiServiceResponse.moderationLabels>` HTML element to access these inferences. For an example of a custom template that uses this HTML element, see [Custom Template Example for Amazon Rekognition \(p. 165\)](#).

[Examples](#)

Example 1: Use `ModerationLabelConfidenceCheck` with the `And` operator

The following example of a `HumanLoopActivationConditions` condition triggers a `HumanLoop` when one or more of the following conditions are met:

- Amazon Rekognition detects the `Graphic Male Nudity` moderation label with a confidence between 90 and 99.
- Amazon Rekognition detects the `Graphic Female Nudity` moderation label with a confidence between 80 and 99.

Note the use of the `Or` and `And` logical operators to model this logic.

Although only one of the two conditions under the `Or` operator need to evaluate to true for a `HumanLoop` to be created, Amazon Augmented AI evaluates all conditions. Human reviewers are asked to review the moderation labels for all the conditions that evaluated to true.

```
{
    "Conditions": [
        {
            "Or": [
                {
                    "And": [
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Male Nudity",
                                "ConfidenceLessThanOrEqual": 99
                            }
                        },
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Male Nudity",
                                "ConfidenceGreaterThanOrEqual": 90
                            }
                        }
                    ]
                },
                {
                    "And": [
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Female Nudity",
                                "ConfidenceLessThanOrEqual": 99
                            }
                        },
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Female Nudity",
                                "ConfidenceGreaterThanOrEqual": 80
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

Example 2: Use `ModerationLabelConfidenceCheck` with the catch-all value (*)

In the following example, if any moderation label with a confidence greater than or equal to 75 is detected, a `HumanLoop` is triggered. Human reviewers are asked to review all moderation labels with confidence scores greater than or equal to 75.

```
{
```

```

    "Conditions": [
        {
            "ConditionType": "ModerationLabelConfidenceCheck",
            "ConditionParameters": {
                "ModerationLabelName": "*",
                "ConfidenceGreaterThanOrEqual": 75
            }
        }
    ]
}

```

Example 3: Use Sampling

In the following example, 5.2% of Amazon Rekognition inferences from a `DetectModerationLabels` request will be sent to human workers. When using the default worker task template provided in the Amazon SageMaker console, all moderation labels returned by Amazon Rekognition are sent to workers for review.

```

{
    "Conditions": [
        {
            "ConditionType": "Sampling",
            "ConditionParameters": {
                "RandomSamplingPercentage": 5.2
            }
        }
    ]
}

```

Example 4: Use Sampling and ModerationLabelConfidenceCheck with the And operator

In this example, 5.2% of Amazon Rekognition inferences of the `Graphic Male Nudity` moderation label with a confidence greater than 50 will be sent workers for review. When using the default worker task template provided in the Amazon SageMaker console, only the `Graphic Male Nudity` label will be sent to workers for review.

```

{
    "Conditions": [
        {
            "And": [
                {
                    "ConditionType": "Sampling",
                    "ConditionParameters": {
                        "RandomSamplingPercentage": 5.2
                    }
                },
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Graphic Male Nudity",
                        "ConfidenceGreaterThanOrEqual": 50
                    }
                }
            ]
        }
    ]
}

```

Example 5: Use Sampling and ModerationLabelConfidenceCheck with the And operator

Use this example to configure your human review workflow to always send low confidence inferences of a specified label for human review and sample high confidence inference of a label at a specified rate.

In the following example, a human review is triggered in one of the following ways:

- Inferences for the `Graphic Male Nudity` moderation label with confidence scores less than 60 are always sent for human review. Only the `Graphic Male Nudity` label is sent to workers to review.
- 5.2% of all inferences for the `Graphic Male Nudity` moderation label with confidence scores greater than 90 will be sent for human review. Only the `Graphic Male Nudity` label is sent to workers to review.

```
{
  "Conditions": [
    {
      "Or": [
        {
          "And": [
            {
              "ConditionType": "Sampling",
              "ConditionParameters": {
                "RandomSamplingPercentage": 5.2
              }
            },
            {
              "ConditionType": "ModerationLabelConfidenceCheck",
              "ConditionParameters": {
                "ModerationLabelName": "Graphic Male Nudity",
                "ConfidenceGreaterThan": 90
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Example 6: Use Sampling and ModerationLabelConfidenceCheck with the Or operator

In the following example, a human loop is created if the Amazon Rekognition inference response contains the '`Graphic Male Nudity`' label with inference confidence greater than than 50. Additionally, 5.2% of all other inferences will trigger a human loop.

```
{
  "Conditions": [
    {
      "Or": [
        {
          "ConditionType": "Sampling",
          "ConditionParameters": {
            "RandomSamplingPercentage": 5.2
          }
        },
        {
          "ConditionType": "ModerationLabelConfidenceCheck",
          "ConditionParameters": {
            "ModerationLabelName": "Graphic Male Nudity",
            "ConfidenceGreaterThan": 50
          }
        }
      ]
    }
  ]
}
```

```
        "ConfidenceGreaterThanOrEqual": 50
    }
}
]
```

Create and Start a Human Loop

A *human loop* starts your human review workflow and sends data review tasks to human workers. When you use one of the Amazon A2I built-in task types, the corresponding AWS service creates and starts a human loop on your behalf when the conditions specified in your flow definition are met. If no conditions were specified in your flow definition, a human loop is created for each object. When using Amazon A2I for a custom task, a human loops starts when `StartHumanLoop` is called in your application.

Use the following instructions to configure a human loop with Amazon Rekognition or Amazon Textract built-in task types and custom task types.

Prerequisites

To create and start a human loop, the `AmazonAugmentedAIFullAccess` policy must be attached to the AWS Identity and Access Management (IAM) user or role that configures or starts the human loop. When using a built-in task type, this will be the user or role that you use to configure the human loop using `HumanLoopConfig`. For custom task types, this will be the user or role that you use to call `StartHumanLoop`.

You will need a flow definition ARN to create and start a human loop. To learn how to create a flow definition (or human review workflow), see [Create a Flow Definition \(p. 143\)](#).

Create and Start a Human Loop for a Built-in Task Type

To start a human loop for a built-in task type jobs use the corresponding service's API to provide your input data and to configure the human loop. For Amazon Textract, you use the `AnalyzeDocument` API operation. For Amazon Rekognition, you use the `DetectModerationLabels` API operation. You can use the AWS CLI, or a language-specific SDK to create requests using these API operations.

After you start your ML job using your built-in task type's AWS service API, Amazon A2I monitors the inference-results of that service. For example, when running a job with Amazon Rekognition, Amazon A2I checks the inference confidence score for each image and compares it to the confidence-thresholds specified in your flow definition. If the conditions to start a human review task are satisfied, or if you didn't specify conditions in your flow definition, a human review task is sent to workers.

Create an Amazon Textract Human Loop

Amazon A2I integrates with Amazon Textract so that you can configure and start a human loop using the Amazon Textract API. To send a document file to Amazon Textract for text analysis, you use the Amazon Textract [AnalyzeDocument API operation](#). To configure a human loop, set the `HumanLoopConfig` parameter when you configure `AnalyzeDocument`. To learn how, see Step 3 in [Running AnalyzeDocument with Amazon A2I \(Preview\)](#) in the *Amazon Textract Developer Guide*.

After you run the `AnalyzeDocument` with a human loop configured, Amazon A2I monitors the results from `AnalyzeDocument` and checks it against the flow definition's activation conditions. If Amazon Textract inference confidence score for one or more key value pairs meets the conditions for review, Amazon A2I starts a human review loop and includes the `HumanLoopActivationOutput` object in the `AnalyzeDocument` response.

Create an Amazon Rekognition Human Loop

Amazon A2I integrates with Amazon Rekognition so that you can configure and start a human loop using the Amazon Rekognition API. To send images to Amazon Rekognition for content moderation, you use the Amazon Rekognition [DetectModerationLabels API operation](#). To configure a human loop, set the `HumanLoopConfig` parameter when you configure `DetectModerationLabels`. To learn how, see Step 3 in [Running DetectModerationLabels with Amazon A2I](#) in the *Amazon Rekognition Developer Guide*.

After you run the `DetectModerationLabels` with a human loop configured, Amazon A2I monitors the results from `DetectModerationLabels` and checks it against the flow definition's activation conditions. If the Amazon Rekognition inference confidence score for an image meets the conditions for review, Amazon A2I starts a human review loop and includes the response element `HumanLoopActivationOutput` response object in the `DetectModerationLabels` response.

Create and Start a Human Loop for a Custom Task Type

To configure a human loop for a custom human review task, use the `StartHumanLoop` operation within your application. This section provides an example of a human loop request using the AWS SDK for Python (Boto 3) and the AWS Command Line Interface (AWS CLI). For documentation on other language specific SDK's that support `StartHumanLoop`, use the [See Also](#) section of `StartHumanLoop` in the Amazon Augmented AI Runtime API documentation.

Prerequisites

To complete this procedure, you need:

- Input data formatted as a string representation of a JSON-formatted file.
- The Amazon Resource Name (ARN) of your flow definition
- A flow definition ARN.

To configure the human loop

1. For `DataAttributes`, specify a set of `ContentClassifiers` related to the input provided to the `StartHumanLoop` operation. Use content classifiers to declare that your content is free of personally identifiable information or adult content.

To use Amazon Mechanical Turk, ensure your data is free of Personally Identifiable Information and include the `FreeOfPersonallyIdentifiableInformation` content classifier. If your data is free of adult content, also include the '`FreeOfAdultContent`' classifier. If you do not use these content classifiers, Amazon SageMaker may restrict the Mechanical Turk workers that can view your task.

2. For `FlowDefinitionArn`, enter the Amazon Resource Name (ARN) of your flow definition.
3. For `HumanLoopInput`, enter your input data as a string representation of a JSON-formatted file. Structure your input data and custom worker task template so that your input data is properly displayed to human workers when you start your human loop. See to learn how to preview your custom worker task template.
4. For `HumanLoopName`, enter a name for the human loop. The name must be unique within the Region in your account, and can have up to 63 characters. Valid characters: a-z, 0-9, and - (hyphen).

To start a human loop

- To start a human loop, submit a request similar to the following examples using your preferred language specific SDK.

AWS SDK for Python (Boto 3)

The following request example uses the SDK for Python (Boto 3). For more information, see [Boto 3 Augmented AI Runtime](#) in the *AWS SDK for Python (Boto) API Reference*.

```
response = client.start_human_loop(
    HumanLoopName='string',
    FlowDefinitionArn='string',
    HumanLoopInput={
        'InputContent': 'string'
    },
    DataAttributes={
        'ContentClassifiers': [
            'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
        ]
    }
)
```

AWS CLI

The following request example uses the AWS CLI. For more information, see [start-human-loop](#) in the *AWS CLI Command Reference*.

```
$ InputContent=string
ContentClassifiers="FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"
start-human-loop --human-loop-name example-humanloop --flow-definition-
arn arn:aws:sagemaker:us-west-2:111111111111:flow-definition/flowdef-nov-12 --human-
loop-input InputContent --data-attributes ContentClassifiers
```

When you successfully start a human loop by invoking `StartHumanLoop` directly, the response will include a `HumanLoopARN` and a `HumanLoopActivationResults` object which will be set to `NULL`. You can use this the human loop name to monitor and manage your human loop.

Next Steps:

After starting a human loop, you can manage and monitor it with the Amazon Augmented AI Runtime API and Amazon CloudWatch Events. To learn more, see [Monitor and Manage Your Human Loop \(p. 170\)](#).

Create a Worker UI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can create a task UI for your workers by creating a *worker task template* with detailed instructions on how to complete your task. Depending on your task type, this can be done directly in the Amazon SageMaker console or by using a custom template.

- If you are creating a human review workflow for Amazon Textract or Amazon Rekognition tasks, follow the instructions in [Create a Flow Definition \(Console\) \(p. 144\)](#) to create your worker template using the default templates provided by Amazon A2I.
- If you are adding a human review workflow to a custom task type, or want to create a custom worker UI using HTML and CSS elements:
 1. Create a custom worker template using the instructions found in [Create Custom Worker Templates \(p. 163\)](#).

2. Generate an Amazon Resource Name (ARN) to use this template in your flow definition. Generate a worker task template ARN using the instructions found in [Create a Custom Worker Template \(Console\) \(p. 162\)](#) or by using the [CreateHumanTaskUi](#) API operation.

Topics

- [Create a Custom Worker Template \(Console\) \(p. 162\)](#)
- [Create Custom Worker Templates \(p. 163\)](#)
- [Creating Good Worker Instructions \(p. 170\)](#)

Create a Custom Worker Template (Console)

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can use a worker template to customize the interface and instructions that your workers see when working on your tasks. Use the procedures in this topic to create a worker task template in the Amazon SageMaker console. A starter template is provided for Amazon Textract and Amazon Rekognition tasks. Using this procedures to create a worker template will allow you to choose this template when creating a flow definition in the console. The procedure will also produce a worker template ARN that can be used to create a flow definition using the API operation [CreateFlowDefinition](#).

If you are creating a template for an Amazon Textract or Amazon Rekognition task type, and you want to preview your worker template, you will need to attach the policy described in [Enable Worker Task Template Previews \(p. 174\)](#) to the IAM role that you use in this procedure.

To create a worker task template in the Amazon SageMaker console

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/>.
2. In Amazon Augmented AI in the left navigation pane, choose **Worker task templates**.
3. Choose **Create template**.
4. In **Template name**, enter a unique name.
5. (Optional) Enter an **IAM role** that grants A2I the permissions necessary to call services on your behalf.
6. In **Template type**, choose a template type from the drop-down menu. If you are creating a template for a **Textract-form extraction** or **Rekognition-image moderation** task, choose the appropriate option.
7. Enter your custom template elements as follows:
 - If you selected the Amazon Textract or Amazon Rekognition task template, the **Template editor** autopopulates with a default template that you can customize.
 - If you are using a custom template, enter your predefined template in the editor.
8. (Optional) To complete this step, you must provided an IAM role ARN with permission to read Amazon S3 objects that get rendered on your user interface in **Step 5**.

You can only preview your template if you are creating templates for Amazon Textract or Amazon Rekognition.

Choose **See preview** to preview the interface and instructions that workers will see. This is an interactive preview. After you complete the sample task and choose **Submit**, you see the resulting output from the task that you just performed.

If you are creating a worker task template for a custom task type, you can preview your worker task UI using `RenderUiTemplate`. For more information, see [Preview a Worker Task Template \(p. 169\)](#).

- When you're satisfied with your template, choose **Create**.

After you've created your template, you can select that template when you create a human review workflow in the console. Your template also appears in the Amazon Augmented AI section of the Amazon SageMaker console under **Worker task templates**. Choose your template to view its ARN.

Create Custom Worker Templates

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Crowd HTML elements are web components that provide a number of task widgets and design elements that you can tailor to the question you want to ask. You can use these crowd elements to create a custom worker template and integrate it with an Amazon Augmented AI (Amazon A2I) human review workflow to customize the worker console and instructions.

For a list of all HTML crowd elements available to Amazon A2I users, see [HTML Elements Reference \(p. 189\)](#). To see examples of templates, see the [AWS Github repository](#), which contains over 60 sample custom task templates.

Develop Templates Locally

When in the console to test how your template process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding the following code to the top of your HTML file.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this code if you want to develop your template's look and feel in your preferred editor instead of in the console.

This code won't parse your variables. You might want to replace them with sample content while developing locally.

Use External Assets

Amazon Augmented AI custom templates enable you to embed external scripts and style sheets. For example, the following header embeds a `text/css` style sheet name `stylesheet` located at `https://www.example.com/my-enhancement-styles.css` into the custom template.

Example

```
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding types for remote scripts is `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets is `text/css; CHARSET=UTF-8`.

Track Your Variables

When building a custom template, you must add variables to it to represent the pieces of data that might change from task to task, or worker to worker. If you're starting with one of the sample templates, you need to make sure you're aware of the variables it already uses.

For example, for a custom template that integrates an Augmented AI human review loop with a Amazon Textract text review task, `{{ task.input.selectedAiServiceResponse.blocks }}` is used for initial-value input data. For Amazon Augmented AI (Amazon A2I) integration with Amazon Rekognition , `{{ task.input.selectedAiServiceResponse.moderationLabels }}` is used. For a custom task type, you need to determine the input parameter for your task type. Use `{{ task.input.customInputValuesForStartHumanLoop}}` where you specify `customInputValuesForStartHumanLoop`.

Custom Template Example for Amazon Textract

All custom templates begin and end with the `<crowd-form>` `</crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between these elements.

For a Amazon Textract document analysis task, use the `<crowd-textract-document-analysis>` element. It uses the following attributes:

- `src` – Specifies the URL of the image file to be annotated.
- `initialValue` – Sets initial values for attributes found in the worker UI.
- `blockTypes (required)` – Determines the kind of analysis that the workers can do. Only `KEY_VALUE_SET` is currently supported.
- `keys (required)` – Specifies new keys and the associated text value that the worker can add.
- `no-key-edit (required)` – Prevents the workers from editing the keys of annotations passed through `initialValue`.
- `no-geometry-edit` – Prevents workers from editing the polygons of annotations passed through `initialValue`.

For children of the `<crowd-textract-document-analysis>` element, you must have two regions. You can use arbitrary HTML and CSS elements in these regions.

- `<full-instructions>` – Instructions that are available from the **View full instructions** link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- `<short-instructions>` – A brief description of the task that appears in the tool's sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

An Amazon Textract template would look similar to the following.

Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<% capture s3_arn %>http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.document.s3Object.bucket }}/
{{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}

<crowd-form>
<crowd-textract-analyze-document
    src="{{ s3_arn | grant_read_access }}"
```

```

initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
header="Review the key-value pairs listed on the right and correct them if they don't
match the following document."
no-key-edit
no-geometry-edit
keys="{{ task.input.humanLoopContext.importantFormKeys }}"
block-types=['KEY_VALUE_SET']
>
<short-instructions header="Instructions">
  <style>
    .instructions {
      white-space: pre-wrap;
    }
    .instructionsImage {
      display: inline-block;
      max-width: 100%;
    }
  </style>
  <p class='instructions'>Choose a key-value block to highlight the corresponding key-
value pair in the document.

If it is a valid key-value pair, review the content for the value. If the content is
incorrect, correct it.

The text of the value is incorrect, correct it.


A wrong value is identified, correct it.


If it is not a valid key-value relationship, choose No.


If you can't find the key in the document, choose Key not found.


If the content of a field is empty, choose Value is blank.


<b>Examples</b>
Key and value are often displayed next to or below to each other.

Key and value displayed in one line.


Key and value displayed in two lines.


If the content of the value has multiple lines, enter all the text without a line break.
Include all value text even if it extends beyond the highlight box.
</p>
  </short-instructions>

  <full-instructions header="Instructions"></full-instructions>
</crowd-extract-analyze-document>
</crowd-form>

```

Custom Template Example for Amazon Rekognition

All custom templates begin and end with the `<crowd-form>` `</crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between these elements. For an Amazon Rekognition custom task template, use the `<crowd-rekognition-detect-moderation-labels>` element. This element supports the following attributes:

- **categories** – An array of strings or an array of objects where each object has a `name` field.
 - If the categories come in as objects, the following applies:
 - The displayed categories are the value of the `name` field.
 - The returned answer contains the *full* objects of any selected categories.
 - If the categories come in as strings, the following applies:
 - The returned answer is an array of all the strings that were selected.
- **exclusion-category** – By setting this attribute, you create a button underneath the categories in the UI. When a user presses the button, all categories are deselected and disabled. If the worker presses the button again, you re-enable users to choose categories. If the worker submits the task by selecting the Submit button after you pressing the button, that task will return an empty array.

For children of the `<crowd-textract-document-analysis>` element, you must have three regions.

- `<full-instructions>` – Instructions that are available from the **View full instructions** link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- `<short-instructions>` – Brief description of the task that appears in the tool's sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

A template using these elements would look similar to the following.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{# capture s3_arn #}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/
{{ task.input.aiServiceRequest.image.s3Object.name }}{# endcapture #-}

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
    categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
        {
          name: "{{ label.name }}",
          parentName: "{{ label.parentName }}",
        },
        {% endfor %}
      ]'
    src="{{ s3_arn | grant_read_access }}"
    header="Review the image and choose all applicable categories."
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
      </style>
      <p class='instructions'>Review the image and choose all applicable categories.
      If no categories apply, choose None.

      <b>Nudity</b>
      Visuals depicting nude male or female person or persons

      <b>Graphic Male Nudity</b>
      Visuals depicting full frontal male nudity, often close ups

      <b>Graphic Female Nudity</b>
      Visuals depicting full frontal female nudity, often close ups

      <b>Sexual Activity</b>
      Visuals depicting various types of explicit sexual activities and pornography
    </short-instructions>
  </crowd-form>

```

```
<b>Illustrated Nudity or Sexual Activity</b>
Visuals depicting animated or drawn sexual activity, nudity, or pornography

<b>Adult Toys</b>
Visuals depicting adult toys, often in a marketing context

<b>Female Swimwear or Underwear</b>
Visuals depicting female person wearing only swimwear or underwear

<b>Male Swimwear Or Underwear</b>
Visuals depicting male person wearing only swimwear or underwear

<b>Partial Nudity</b>
Visuals depicting covered up nudity, for example using hands or pose

<b>Revealing Clothes</b>
Visuals depicting revealing clothes and poses, such as deep cut dresses

<b>Graphic Violence or Gore</b>
Visuals depicting prominent blood or bloody injuries

<b>Physical Violence</b>
Visuals depicting violent physical assault, such as kicking or punching

<b>Weapon Violence</b>
Visuals depicting violence using weapons like firearms or blades, such as shooting

<b>Weapons</b>
Visuals depicting weapons like firearms and blades

<b>Self Injury</b>
Visuals depicting self-inflicted cutting on the body, typically in distinctive patterns
using sharp objects

<b>Emaciated Bodies</b>
Visuals depicting extremely malnourished human bodies

<b>Corpses</b>
Visuals depicting human dead bodies

<b>Hanging</b>
Visuals depicting death by hanging</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>
```

Add Automation with Liquid

The custom template system uses [Liquid](#) for automation. *Liquid* is an open-source inline markup language. For more information and documentation, see the [Liquid homepage](#).

In Liquid, the text between single curly braces and percent symbols is an instruction or *tag* that creates control flow. Text between double curly braces is a variable or *object* that outputs its value.

Use Variable Filters

In addition to the standard Liquid filters and actions, Amazon Augmented AI (Amazon A2I) offers a few additional filters. You apply filters by placing a pipe (|) character after the variable name, and then specifying a filter name. To chain filters use the following format.

Example

```
{} <content> | <filter> | <filter> {}
```

Autoescape and Explicit Escape

By default, inputs are HTML-escaped to prevent confusion between your variable text and HTML. You can explicitly add the `escape` filter to make it more obvious to someone reading the source of your template that escaping is being done.

`escape_once`

`escape_once` ensures that if you've already escaped your code, it doesn't get re-escaped again. For example, to ensure that `&` doesn't become `&`:

`skip_autoescape`

`skip_autoescape` is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

Use `skip_autoescape` sparingly. As a best practice for templates, avoid passing in functional code or markup with `skip_autoescape` unless you are absolutely sure that you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a cross-site scripting attack.

`to_json`

`to_json` encodes data that you provide to JavaScript Object Notation (JSON). If you provide an object, it serializes it.

`grant_read_access`

`grant_read_access` takes an Amazon Simple Storage Service (Amazon S3) URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible to workers.

Example Example of the `to_json` and `grant_read_access` filters

Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?" }}  
explicit escape: {{ "Have you read 'James & the Giant Peach'?" | escape }}  
explicit escape_once: {{ "Have you read 'James & the Giant Peach'?" | escape_once }}  
skip_autoescape: {{ "Have you read 'James & the Giant Peach'?" | skip_autoescape }}  
to_json: {{ jsObject | to_json }}  
grant_read_access: {{ "s3://examplebucket/myphoto.png" | grant_read_access }}
```

Example

Output

```
auto-escape: Have you read 'James & the Giant Peach'?  
explicit escape: Have you read 'James & the Giant Peach'?  
explicit escape_once: Have you read 'James & the Giant Peach'?  
skip_autoescape: Have you read 'James & the Giant Peach'?  
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
```

```
grant_read_access: https://s3.amazonaws.com/examplebucket/myphoto.png?<access token and other params>
```

Example Example of an automated classification template.

To automate this simple text classification sample, include the Liquid tag {{ task.input.[source](#) }}. This example uses the [crowd-classifier](#) (p. 198) element.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'cannot determine']"
    header="Which term best describes this tweet?">
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seems to match, choose "other."
    </full-instructions>

    <short-instructions>
      Pick the term that best describes the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

Preview a Worker Task Template

To preview a custom worker task template, use the Amazon SageMaker `RenderUiTemplate` operation. You can use the `RenderUiTemplate` operation with the AWS CLI or your preferred AWS SDK. For documentation on the supported language specific SDK's for this API operation use the [See Also](#) section of the [RenderUiTemplate](#).

Prerequisites

To preview your worker task template, the AWS Identity and Access Management (IAM) role Amazon Resource Name (ARN), or `RoleArn`, that you use must have permission to access to the S3 objects that are used by the template. To learn how to configure your role or user see [Enable Worker Task Template Previews](#) (p. 174).

To preview your worker task template using the `RenderUiTemplate` operation:

1. Provide a `RoleArn` of the role with required policies attached to preview your custom template.
2. In the `Input` parameter of `Task`, provide A JSON object that contains values for the variables defined in the template. These are the variables that are substituted for the `task.input.source` variable. For example, if you define a variable `task.input.text` in your template, you can supply the variable in the JSON object as "text": "sample text".
3. In the `Content` parameter of `UiTemplate`, insert your template.

Once you've configured `RenderUiTemplate`, use your prefered SDK or the AWS CLI to submit a request to render your template. If your request was successful, the response will include `RenderedContent`, a Liquid template that renders the HTML for the worker UI.

Important

To preview your template, you need an IAM role with permissions to read Amazon S3 objects that get rendered on your user interface. For a sample policy that you can attach to your IAM role to grant these permissions, see [Enable Worker Task Template Previews \(p. 174\)](#).

Creating Good Worker Instructions

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

Creating good instructions for your human review jobs improves your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console when creating a human review workflow, or you can use the console to create a custom worker template and include your instructions in this template. The instructions are shown to the worker on the UI page where they complete their labeling task.

Create Good Worker Instructions

There are three kinds of instructions in the Amazon Augmented AI console:

- **Task Description** – The description should provide a succinct explanation of the task.
- **Instructions** – These instructions are shown on the same webpage where workers complete a task. These instructions should provide an easy reference to show the worker the correct way to complete the task.
- **Additional Instructions** – These instructions are shown in a dialog box that appears when a worker chooses **View full instructions**. We recommend that you provide detailed instructions for completing the task, and include several examples showing edge cases and other difficult situations for labeling objects.

Add Example Images to Your Instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions, do the following:

1. Place the cursor where the image should go in the instructions editor.
2. Choose the image icon in the editor toolbar.
3. Enter the URL of your image.

If your instruction image is in an S3 bucket that isn't publicly accessible, do the following:

- For the image URL, enter: `{ { 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access } }`.

This renders the image URL with a short-lived, one-time access code that's appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview. See [grant_read_access \(p. 168\)](#) for more information about the `grant_read_access` element.

Monitor and Manage Your Human Loop

Once you've started a human review loop, you can check the results of and manage the loop using the [Amazon Augmented AI Runtime API](#). Additionally, Amazon A2I integrates with Amazon EventBridge (also known as Amazon CloudWatch Events) to alert you when a human review loop changes status.

Use the procedures below to learn how to use the Amazon Augmented AI Runtime API to monitor and manage your human loops. See [Use Amazon CloudWatch Events in Amazon Augmented AI \(p. 175\)](#) to learn how Amazon A2I integrates with Amazon EventBridge.

To check your output data:

1. Check the results of your human loop by calling the [DescribeHumanLoop](#) operation. The result of this API operation contains information about the reason for and outcome of the loop activation.
2. Check the output data from your human loop in Amazon Simple Storage Service (Amazon S3). The path to the data uses the following pattern.

```
s3://customer-output-bucket-specified-in-flow-definition/flow-definition-name/YYYY/MM/DD/MM/SS/human-loop-name/output.json
```

You can integrate this structure with AWS Glue or Amazon Athena to partition and analyze your output data. For more information, see [Managing Partitions for ETL Output in AWS Glue](#) and

To stop and delete your human loop:

1. Once a human loop has been started, you can stop your human loop by calling the [StopHumanLoop](#) operation using the `HumanLoopName`. If a human loop was successfully stopped, the server sends back an HTTP 200 response.
2. To delete a human loop for which the status equals `Failed`, `Completed`, or `Stopped`, use the [DeleteHumanLoop](#) operation.

To list human loops:

1. You can list all active human loops by calling the [ListHumanLoops](#) operation. You can filter human loops by the creation date of the loop using the `CreationTimeAfter` and `CreateTimeBefore` parameters.
2. If successful, `ListHumanLoops` will return `HumanLoopSummaries` and `NextToken` objects in the response element. `HumanLoopSummaries` contains information about a single human loop. For example, it will list a loop's status and if applicable, failure reason.

Use the string returned in `NextToken` as an input in a subsequent call to `ListHumanLoops` to see the next page of human loops.

Permissions and Security in Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

When using Amazon Augmented AI (Amazon A2I) to create a human review workflow for your ML/AI application, you create and configure *resources* in Amazon SageMaker such as a human workforce and worker task templates. To configure and start a human loop, you will either integrate Amazon A2I with other AWS services such as Amazon Textract or Amazon Rekognition or use the Amazon Augmented AI Runtime API. To create a human review workflow and start a human loop, you will need to attach certain policies to your AWS Identity and Access Management(IAM) role or user. Specifically:

- When you create a flow definition, you need to provide a role that grants Amazon A2I permission to access Amazon S3 both for reading objects that will be rendered in a human task UI and for writing the results of the human review.

This role will also need to have a trust policy attached to give Amazon SageMaker permission to assume the role. This allows Amazon A2I to perform actions in accordance with permissions that you attach to the role.

See [Enable Flow Definition Creation \(p. 172\)](#) for example policies that you can modify and attach to the role you use to create a flow definition.

- When using an Amazon A2I human review loop with other services such as Amazon Textract or Amazon Rekognition, or with the Amazon A2I runtime API, you need to attach the **AmazonAugmentedAIFullAccess** policy to the IAM user that you use to invoke the service. This policy grants these services permission to use Amazon A2I operations. To learn how, see [Create an IAM User That Can Invoke Amazon Augmented AI Operations \(p. 173\)](#).
- To preview your custom worker task UI template, you need an IAM role with permissions to read Amazon S3 objects that get rendered on your user interface. See a policy example in [Enable Worker Task Template Previews \(p. 174\)](#).

Topics

- [Enable Flow Definition Creation \(p. 172\)](#)
- [Create an IAM User That Can Invoke Amazon Augmented AI Operations \(p. 173\)](#)
- [Enable Worker Task Template Previews \(p. 174\)](#)
- [Additional Permissions and Security Resources \(p. 174\)](#)

Enable Flow Definition Creation

To create a flow definition, attach the policies in this section to the role that you use when creating a human review workflow in the Amazon SageMaker console, or using the `CreateFlowDefinition` API operation.

- If you are using the console to create a human review workflow, enter the role Amazon Resource Name (ARN) in the **IAM role** field when [creating a human review workflow in the console](#).
- When creating a flow definition using the API, attach these policies to the role that is passed to the `RoleArn` parameter of the `CreateFlowDefinition` operation.

When you create a human review workflow (flow definition), Amazon A2I invokes Amazon S3 to complete your task. To grant Amazon A2I permission to retrieve and store your files in your Amazon S3 bucket, create the following policy and attach it to your role. For example, if the images, documents, and other files that you are sending for human review are stored in an S3 bucket named `my_input_bucket`, and if you want the human reviews to be stored in a bucket named `my_output_bucket`, you would create the following policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::my_input_bucket/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::my_output_bucket/*"  
            ]  
        }  
    ]  
}
```

```
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::my_output_bucket/*"
    ]
}
```

In addition, the IAM role must have the following trust policy to give Amazon SageMaker permission to assume the role. To learn more about IAM trust policies, see [Resource-Based Policies](#) section of Policies and Permissions in the *AWS Identity and Access Management* documentation.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowSageMakerToAssumeRole",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "sagemaker.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

For more information about creating and managing IAM roles and policies, see the following topics in the *AWS Identity and Access Management User Guide*:

- To create IAM role, see [Creating a Role to Delegate Permissions to an IAM User](#).
 - To learn how to create IAM policies, see [Creating IAM Policies](#).
 - To learn how to attach an IAM policy to a role, see [Adding and Removing IAM Identity Permissions](#).

Create an IAM User That Can Invoke Amazon Augmented AI Operations

To use Amazon Augmented AI (Amazon A2I) with Amazon Rekognition, Amazon Textract, or the Amazon A2I runtime API, you must use IAM user that has permissions to invoke Amazon A2I operations. To do this, use the AWS Identity and Access Management (IAM) console to attach the **AmazonAugmentedAIFullAccess** policy to a new or existing IAM user.

To create the required IAM user

1. Sign in to the IAM console at <https://console.aws.amazon.com/iam/>.
 2. Choose **Users** and choose an existing user, or create a new user by choosing **Add user**. To learn how to create a new user, see [Creating an IAM User in Your AWS Account](#) in the *AWS Identity and Access Management User Guide*.
 - If you chose to attach the policy to an existing user, choose **Add permissions**.
 - While creating a new user, follow the next step on the the **Set permissions** page.
 3. Choose **Attach existing policies directly**.
 4. In the **Search** bar, enter **AmazonAugmentedAIFullAccess** and check the box next to that policy.

To enable this IAM user to create a flow definition with the public workteam, also attach the **AmazonSageMakerMechanicalTurkAccess** managed policy.

5. After attaching the policy or policies:

- a. If you are using an existing user, choose **Next: Review**, and then choose **Add permissions**.
- b. If you are creating a new user, choose **Next: Tags** and complete the process of creating your user.

For more information, see [Adding and Removing IAM Identity Permissions](#) in the *AWS Identity and Access Management User Guide*.

Enable Worker Task Template Previews

To customize the interface and instructions that your workers see when working on your tasks, you create a worker task template. You can create the template using the [CreateHumanTaskUi](#) operation or the Amazon SageMaker console.

To preview your template, you need an IAM role with the following permissions to read Amazon S3 objects that get rendered on your user interface.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Resource": [  
                "arn:aws:s3:::my_input_bucket/*"  
            ]  
        }  
    ]  
}
```

For Amazon Rekognition and Amazon Textract task types, you can preview your template using the Amazon Augmented AI section of the Amazon SageMaker console. For custom task types, you preview your template by invoking the [RenderUiTemplate](#) operation. To preview your template, follow the instructions for your task type:

- Amazon Rekognition and Amazon Textract task types – In the Amazon SageMaker console, use the role's Amazon Resource Name (ARN) in the procedure documented in [Create a Custom Worker Template \(Console\) \(p. 162\)](#).
- Custom task types – In the [RenderUiTemplate](#) operation, use the role's ARN in the `RoleArn` parameter.

Additional Permissions and Security Resources

- the section called “Control Access to Amazon SageMaker Resources by Using Tags” (p. 773).
- the section called “Amazon SageMaker Identity-Based Policies” (p. 751)
- the section called “Control Creation of Amazon SageMaker Resources with Condition Keys” (p. 764)
- the section called “Amazon SageMaker API Permissions Reference” (p. 790)
- [Security \(p. 743\)](#)

Use Amazon CloudWatch Events in Amazon Augmented AI

Amazon Augmented AI uses Amazon CloudWatch Events (CloudWatch Events) to alert you when a human review loop changes status. When a review loop changes to the `Completed`, `Failed`, or `Stopped` status, Augmented AI sends an event to CloudWatch Events similar to the following:

```
{  
    "version": "0",  
    "id": "12345678-1111-2222-3333-12345EXAMPLE",  
    "detail-type": "Amazon Augmented AI HumanLoop Status Change",  
    "source": "aws.sagemaker",  
    "account": "111111111111",  
    "time": "2019-11-14T17:49:25Z",  
    "region": "us-east-1",  
    "resources": ["arn:aws:sagemaker:us-east-1:111111111111:human-loop/humanloop-nov-14-1"],  
    "detail": {  
        "creationTime": "2019-11-14T17:37:36.740Z",  
        "failureCode": null,  
        "failureReason": null,  
        "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111111111111:flow-definition/  
flowdef-nov-12",  
        "humanLoopArn": "arn:aws:sagemaker:us-west-2:111111111111:human-loop/humanloop-  
nov-14-1",  
        "humanLoopName": "humanloop-nov-14-1",  
        "humanLoopOutput": {  
            "outputS3Uri": "s3://path-specified-in-flow-definition/flowdef-  
nov-12/2019/11/14/humanloop-nov-14-1/20191114T173736Z/output.json"  
        },  
        "humanLoopStatus": "Completed"  
    }  
}
```

The details in the JSON output include the following:

creationTime

The timestamp when Augmented AI created the human loop.

failureCode

A failure code denoting a specific type of failure.

failureReason

The reason why a human loop has failed. The failure reason is only returned when the human review loop status is `Failed`.

flowDefinitionArn

The Amazon Resource Name (ARN) of the flow definition, or *human review workflow*.

humanLoopArn

The Amazon Resource Name (ARN) of the human loop.

humanLoopName

The name of the human loop.

humanLoopOutput

An object containing information about the output of the human loop.

`outputS3Uri`

The location of the Amazon S3 object where Augmented AI stores your human loop output.

`humanLoopStatus`

The status of the human loop.

Use Human Review Output

After you receive human review results, you can analyze the results and compare them to machine learning predictions. The JSON that is stored in the Amazon S3 bucket contains both the machine learning predictions and the human review results.

More Information

[the section called “React to Amazon SageMaker Job Status Changes with CloudWatch Events” \(p. 742\)](#)

Use APIs in Amazon Augmented AI

Amazon Augmented AI is in preview release and is subject to change. We do not recommend using this product in production environments.

You can create a human review workflow or a worker task template programmatically. The APIs you use depend on whether you are creating a Amazon Rekognition, Amazon Textract, or custom task type. This topic provides links to API reference documentation for each task type and programming task.

The following APIs can be used with Augmented AI:

Amazon Augmented AI

Use the Augmented AI API to start, stop, and delete human review loops. You can also list all human review loops and return information about human review loops in your account.

Learn more about human review loop APIs in the [Amazon Augmented AI Runtime API Reference](#).

Amazon Rekognition

Use the **HumanLoopConfig** parameter of the [DetectModerationLabels](#) API to trigger a human review workflow using Amazon Rekognition.

Amazon SageMaker

Use the Amazon SageMaker API to create a `FlowDefinition`, also known as a *human review workflow*. You can also create a `HumanTaskUi`, or *worker task template*.

For more information, see the [CreateFlowDefinition](#) or the [CreateHumanTaskUi](#) API documentation.

Amazon Textract

Use the **HumanLoopConfig** parameter of the [AnalyzeDocument](#) API to trigger a human review workflow using Amazon Textract.

Programmatic Walkthroughs

The following walkthroughs and tutorials provide example code and step-by-step instructions for creating human review workflows and worker task templates programmatically.

- the section called “[Create a Flow Definition \(API\)](#)” ([p. 146](#))
- [Using Amazon Augmented AI with Amazon Rekognition](#) in the [Amazon Rekognition Developer Guide](#)
- [Using Amazon Augmented AI with Amazon Textract AnalyzeDocument](#) in the [Amazon Textract Developer Guide](#)

Create and Manage Workforces

A *workforce* is the group of workers that you have selected to label your dataset. You can choose either the Amazon Mechanical Turk workforce, a vendor-managed workforce, or you can create your own private workforce to label or review your dataset. Whichever workforce type you choose, Amazon SageMaker takes care of sending tasks to workers.

When you use a private workforce, you also create *work teams*, a group of workers from your workforce that are assigned to specific *jobs*—Amazon SageMaker Ground Truth labeling jobs or Amazon Augmented AI human review tasks. You can have multiple work teams and can assign one or more work teams to each job.

Ground Truth and Amazon Augmented AI use Amazon Cognito to manage your private workforce and work teams. For more information about the permissions required to manage your workforce this way, see [Permissions Required to Use the Amazon SageMaker Ground Truth Console](#) ([p. 756](#)).

Topics

- [Using the Amazon Mechanical Turk Workforce](#) ([p. 177](#))
- [Managing Vendor Workforces](#) ([p. 178](#))
- [Use a Private Workforce](#) ([p. 179](#))

Using the Amazon Mechanical Turk Workforce

The Amazon Mechanical Turk workforce provides the most workers for your Amazon Augmented AI task review and Amazon SageMaker Ground Truth labeling job.

You can use the console to choose the Amazon Mechanical Turk workforce for your Amazon SageMaker Ground Truth labeling job or Amazon Augmented AI human review workflow, or you can provide the Amazon Resource Name (ARN) for the Amazon Mechanical Turk workforce when you use the Amazon A2I `CreateLabelingJob` operation.

Any Amazon Mechanical Turk workforce billing is handled as part of your Ground Truth or Amazon Augmented AI billing. You do not need to create a separate Mechanical Turk account to use the Amazon Mechanical Turk workforce.

The ARN for the Amazon Mechanical Turk workforce is:

- `arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default`

The Amazon Mechanical Turk workforce is a world-wide resource. Workers are available 24 hours a day, 7 days a week. You typically get the fastest turn-around for your human review tasks and labeling jobs when you use the Amazon Mechanical Turk workforce.

Adjust the number of workers that annotate each data object based on the complexity of the job and the quality that you need. Amazon SageMaker Ground Truth uses annotation consolidation to improve the quality of the labels. More workers can make a difference in the quality of the labels for more complex labeling jobs, but might not make a difference for simpler jobs. For more information, see [Consolidate](#)

[Annotations \(p. 89\)](#). Annotation consolidation is not supported for Amazon Augmented AI human review workflows.

Important

You should not share confidential information, personal information or protected health information with this workforce. For avoidance of doubt, you should not use the Amazon Mechanical Turk workforce when you use Amazon A2I in conjunction with AWS HIPAA-eligible services, such as Amazon Textract and Amazon Rekognition for workloads containing protected health information.

To choose the Amazon Mechanical Turk workforce when you are creating a labeling job or human review workflow using the console, do the following during the **Select workers and configure tool** step:

To use the Amazon Mechanical Turk workforce

1. Choose **Public** from **Worker types**.
2. Choose **The dataset does not contain adult content** if your dataset doesn't contain potentially offensive content. This enables workers to opt out if they don't want to work with it.
3. Acknowledge that your data will be viewed by the Amazon Mechanical Turk workforce and that all personally identifiable information (PII) has been removed.
4. Choose **Additional configuration** to set optional parameters.
5. Optional. Enable automated data labeling to have Ground Truth automatically label some of your dataset. For more information, see [Automate Data Labeling \(p. 91\)](#). Automated data labeling is not available for Amazon Augmented AI.
6. Optional. Set the number of workers that should see each object in your dataset. Using more workers can increase the quality of your labels but also increases the cost.

Your labeling job or human review task will now be sent to the Amazon Mechanical Turk workforce. You can use the console to continue configuring your labeling job.

Managing Vendor Workforces

You can use a vendor-managed workforce to label your data using Amazon SageMaker Ground Truth (Ground Truth) and Amazon Augmented AI (Amazon A2I). Vendors have extensive experience in providing data labeling services for the purpose of machine learning. Vendor workforces for these two services must be created and managed separately through the Amazon SageMaker console.

Vendors make their services available via the AWS Marketplace. You can find details of the vendor's services on their detail page, such as the number of workers and the hours that they work. You can use these details to make estimates of how much the labeling job will cost and the amount of time that you can expect the job to take. Once you have chosen a vendor you subscribe to their services using the AWS Marketplace.

A subscription is an agreement between you and the vendor. The agreement spells out the details of the agreement, such as price, schedule, or refund policy. You work directly with the vendor if there are any issues with your labeling job.

You can subscribe to any number of vendors to meet your data annotation needs. When you create a labeling job or human review workflow you can specify that the job be routed to a specific vendor.

Before you send sensitive data to a vendor, check the vendor's security practices on their detail page and review the end user license agreement (EULA) that is part of your subscription agreement.

You must use the console to subscribe to a vendor workforce. Once you have a subscription, you can use the [ListSubscribedWorkteams](#) operation to list your subscribed vendors.

To subscribe to a vendor workforce

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 2. Choose the appropriate page in the Amazon SageMaker console.
 - For Ground Truth labeling jobs, choose **Labeling workforces**, choose **Vendor**, and then choose **Find data labeling services**.
 - For Amazon A2I human review workflows, choose **Human review workforces**, choose **Vendor**, and then choose **Find human review services**.
 3. The console opens the AWS Marketplace with:
 - data labeling services category selected for Ground Truth
 - human review services category selected for Amazon A2I
- Here you see a list of the vendor services available for this service.
4. Choose a vendor. The AWS Marketplace shows detailed information about the data labeling or human review service. Use this information to determine if the vendor meets your requirements for your task.
 5. If the vendor meets your requirements, choose **Continue to subscribe**.
 6. Review the details of the subscription. If you agree to the terms, choose **Subscribe** to complete your subscription to the service.

Use a Private Workforce

A **private workforce** is a group of workers that *you* choose. These can be employees of your company or a group of subject matter experts from your industry. For example, if the task is to label medical images, you could create a private workforce of people knowledgeable about the images in question.

Each AWS account has access to a single private workforce per region, and the owner has the ability to create multiple **private work teams** within that workforce. A single private work team is used to complete a labeling job or human review task, or a *job*. You can assign each work team to a separate job or use a single team for multiple jobs. A single worker can be in more than one work team.

Topics

- [Create a Private Workforce \(p. 179\)](#)
- [Manage a Private Workforce \(p. 182\)](#)
- [Create and manage Amazon SNS topics for your work teams \(p. 188\)](#)

Create a Private Workforce

There are three ways that you can create a private workforce:

- Create a new workforce while you are creating your labeling job. Do this in the Amazon SageMaker console in the Ground Truth section.
- Create a new workforce before you create your labeling job. Do this in the Amazon SageMaker console in the Ground Truth section.
- Import an existing workforce after creating a user pool in the Amazon Cognito console.

Once you create a private workforce, that workforce and all work teams and workers associated with it will be available to use for all Ground Truth labeling job tasks and Amazon Augmented AI human review workflows tasks. If you are new to Amazon SageMaker and want to test Ground Truth or Amazon A2I, we suggest that you create a private work team consisting of people from your organization using

the console. Use this work team when creating labeling or human review workflows (flow definitions) to test your worker UI and job-workflow.

Topics

- [Create a Private Workforce \(Console\) \(p. 180\)](#)
- [Create a Private Workforce \(Amazon Cognito Console\) \(p. 181\)](#)

Create a Private Workforce (Console)

You can create a private workforce in the Amazon SageMaker console in one of two ways:

- When creating a labeling job in the **Labeling jobs** page of the Amazon SageMaker Ground Truth section
- Using the **Labeling workforces** page of the Amazon SageMaker Ground Truth section. If you are creating a private workforce for an Amazon A2I human review workflow, use this method.

Both of these methods also create a default work team containing all of the members of the workforce. This private workforce will be available to use for both Ground Truth and Amazon Augmented AI jobs.

Create a Workforce When Creating a Labeling Job

If you haven't created a private workforce when you create your labeling job, you are prompted to create one.

To create a workforce while creating a labeling job (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, choose **Labeling jobs** and fill in all required fields. For instructions on how to start a labeling job, see [Getting started \(p. 65\)](#). Choose **Next**.
3. Choose **Private** for the workforce type.
4. In the **Workers** section, Enter:
 - a. The **Team name**.
 - b. Email addresses for up to 100 workforce members. Email addresses are case sensitive. Your workers must log in using the same case used when the address was initially entered. You can add additional workforce members after the job has been created.
 - c. The name of your organization. Amazon SageMaker uses this to customize the email sent to the workers.
 - d. A contact email address for workers to report issues related to the task.

When you create the labeling job, an email is sent to each worker inviting them to join the workforce. After creating the workforce, you can add, delete, and disable workers using the Amazon SageMaker console or the Amazon Cognito console.

Create a Workforce Using the Labeling Workforces Page

To create and manage your private workforce, you can use the **Labeling workforces** page. When following the instructions below, you will have the option to create a private workforce by entering worker emails or importing a pre-existing workforce from an Amazon Cognito user pool. To import a workforce, see [Create a Private Workforce \(Amazon Cognito Console\) \(p. 181\)](#).

To create a private workforce (worker emails)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.

2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**, then choose **Create private team**.
4. Choose **Invite new workers by email**.
5. Paste or type a list of up to 50 email addresses, separated by commas, into the email addresses box.
6. Enter an organization name and contact email.
7. Enter an organization name and contact email.
8. Optionally choose an SNS topic to which to subscribe the team so workers are notified by email when new labeling jobs become available.
9. Click the **Create private team** button.

After you import your private workforce, refresh the page. On the **Private workforce summary** page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce.

Note

If you delete all of your private work teams, you have to repeat this process to use a private workforce in that region.

Create a Private Workforce (Amazon Cognito Console)

Amazon Cognito is used to define and manage your private workforce and your work teams. It is a service that you can use to create identities for your workers and authenticate these identities with identity providers. A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool.

Example identity providers supported by Amazon Cognito:

- Social sign-in providers such as Facebook and Google
- OpenID Connect (OIDC) providers
- Security Assertion Markup Language (SAML) providers such as Active Directory
- The Amazon Cognito built-in identity provider

For more information, see [What Is Amazon Cognito?](#)

To create a private workforce using Amazon Cognito, you must have an existing Amazon Cognito user pool containing at least one user group. See [Tutorial: Creating a User Pool](#) to learn how to create a user pool. See [Adding Groups to a User Pool](#) to learn how to add a user group to a pool.

Once your user pool has been created, follow the steps below to create a private workforce by importing that user pool into Amazon SageMaker.

To create a private workforce by importing a Amazon Cognito user pool

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**.
4. Choose **Create private team**. This creates a private workforce and a work team.
5. Choose **Import workers from existing Amazon Cognito user groups**.
6. Choose a user pool that you have created. User pools require a domain and an existing user group. If you get an error that the domain is missing, set it in the **Domain name** options on the **App integration** page of the Amazon Cognito console for your group.
7. Choose an app client. We recommend using a client generated by Amazon SageMaker.

8. Choose a user group from your pool to import its members.
9. Optionally choose an Amazon Simple Notification Service (Amazon SNS) topic to which to subscribe the team so that workers are notified by email when new labeling jobs become available.
10. Choose **Create private team**.

Important

After you create a workforce using a Amazon Cognito user pool, it should not be deleted without first deleting all work teams associated with that pool in the Amazon SageMaker console.

After you import your private workforce, refresh the page to see the **Private workforce summary** page. On this page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce. This workforce will now be available to use in both Amazon Augmented AI and Amazon SageMaker Ground Truth for human review tasks and data labeling jobs respectively.

Manage a Private Workforce

After you have created a private workforce, you can do the following using either the Amazon SageMaker or Amazon Cognito console. Note, if you add workers to a workforce using the Amazon Cognito console, you must use the same console to remove the worker from the workforce.

- Add work teams to your workforce
- Add workers to your workforce and one or more work teams
- Disable or remove workers from your workforce and one or more workteams

You can restrict access to tasks to workers at specific IP addresses using the Amazon SageMaker API. For more information, see [Manage Private Workforce Access to Tasks Using IP Addresses \(p. 186\)](#).

Note

Your private workforce is shared between Amazon SageMaker Ground Truth and Amazon Augmented AI. To manage private work teams and workers used by Amazon Augmented AI, use the Ground Truth section of the Amazon SageMaker console or the Amazon Cognito user pool that you used to create your shared private workforce.

Topics

- [Manage a Private Workforce \(Amazon SageMaker Console\) \(p. 182\)](#)
- [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 184\)](#)
- [Manage Private Workforce Access to Tasks Using IP Addresses \(p. 186\)](#)
- [Track Worker Performance \(p. 186\)](#)

Manage a Private Workforce (Amazon SageMaker Console)

You can use the Amazon SageMaker console to create and manage the work teams and individual workers that make up a private workforce.

Use a work team to assign members of your private workforce to a *job* - a labeling or human review job. When you create your workforce using the Amazon SageMaker console, there is a work team called **Everyone-in-private-workforce** that you can use if you want to assign your entire workforce to a job. Because an imported Amazon Cognito user pool may contain members that you don't want to include in your work teams, a similar work team is not created for Amazon Cognito user pools.

You have two choices to create a new work team:

- You can create a work team in the Amazon SageMaker console and add members from your workforce to the team.
- You can create a user group by using the Amazon Cognito console and then create a work team by importing the user group. You can import more than one user group into each work team. You manage the members of the work team by updating the user group in the Amazon Cognito console. See [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 184\)](#) for more information.

Create a Work Team Using the Amazon SageMaker Console

You can create a workteam using the Amazon SageMaker console, on the **Labeling workforces** page. You can create a workteam by creating a new Amazon Cognito user group or by importing an existing user group. For more information on creating a user group in the Amazon Cognito console, see [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 184\)](#).

To create a work team using the Amazon SageMaker console:

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>
2. Choose **Labeling workforces** from the left menu.
3. Under **Private**, choose **Create private team**.
4. Under **Team details**, enter a **Team name**. The name must be unique in your account in an AWS Region.
5. Under **Add workers**, choose a method to add workers to the team using a user group.
 - If you chose **Create a team by adding workers to a new Amazon Cognito user group**, select the workers to add to the team.
 - If you chose **Create a team by importing existing Amazon Cognito user groups**, choose the user groups that are part of the new team.
6. If you select an **SNS topic**, all workers added to the team are subscribed to the Amazon SNS topic and notified when new work items are available to the team. Select from a list of your existing Ground Truth or Amazon Augmented AI related Amazon SNS topics or select **Create new topic** to open a topic-creation dialog.

Workers in a workteam subscribed to a topic receive notifications when a new job for that team becomes available and when one is about to expire.

Read [Create and manage Amazon SNS topics for your work teams \(p. 188\)](#) for more information about using Amazon SNS topic.

Subscriptions

After you have created a work team, you can see more information about the team and change or set the Amazon SNS topic to which its members are subscribed by visiting the Amazon Cognito console. Any members of the team who were added to the team prior to the team being subscribed to a topic will need to be subscribed to that topic manually. Read [Create and manage Amazon SNS topics for your work teams](#) for more information on creating and managing the Amazon SNS topic.

Add or Remove Workers

A *work team* is a group of workers within your workforce that you can assign jobs to. A worker can be added to more than one work team. Once a worker has been added to a work team, that worker can be disabled or removed.

Add Workers to the Workforce

Adding a worker to the workforce will allow you to add that worker to any work team within that work force.

To add workers using the Private workforce summary page:

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>
2. Choose **Labeling workforces** to navigate to your Private workforce summary page.
3. Choose **Private**
4. Choose **Invite new workers**
5. Paste or type a list of email addresses, separated by commas, into the email addresses box. You can have up to 50 email addresses in this list.

Add a Worker to a Work Team

A worker must be added to the workforce before being added to a work team. To add a worker to a work team, first navigate to the **Private workforce summary** page using the steps above.

To add a worker to a work team from the workforce summary page

1. In the **Private teams** section, choose the team that you want to add the workers to
2. Choose the **Workers** tab.
3. Choose **Add workers to team** and choose the boxes next to the workers that you want to add
4. Click **Add workers to team**

Disable and Remove a Worker from the Workforce

Disabling a worker stops the worker from receiving a job. This action does not remove the worker from the workforce, or any work team the worker is associated with. To disable or remove a worker to a work team, first navigate to the Private workforce summary page using the steps above.

To deactivate a worker using the Private workforce summary page

1. In the **Workers** section, choose the worker that you would like to disable.
2. Choose **Disable**.

If desired, you can subsequently **Enable** a worker after they have been disabled.

You can remove workers from your private workforce directly in the Amazon SageMaker console if that worker was added in this console. If you added the worker (user) in the Amazon Cognito console, see [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 184\)](#) to learn how to remove the worker in the Amazon Cognito console.

To remove a worker using the Private workforce summary page

1. In the **Workers** section, choose the worker that you would like to delete.
2. If the worker has not been disabled, choose **Disable**.
3. Select the worker and choose **Delete**.

Manage a Private Workforce (Amazon Cognito Console)

A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool. Workers correspond to **Amazon Cognito users** within those groups.

After your workforce has been created, you can add work teams and individual workers through the Amazon Cognito console. You can also delete workers from your private workforce and/or remove them from individual teams in the Amazon Cognito console.

Important

You can't delete work teams from the Amazon Cognito console. Deleting a Amazon Cognito user group that is associated with a Amazon SageMaker work team will result in an error. To remove work teams, use the Amazon SageMaker console.

Create Work Teams (Amazon Cognito Console)

You can create a new work team to complete a job by adding a Amazon Cognito user group to the user pool associated with your private workforce. To add a Amazon Cognito user group to an existing worker pool, see [Adding groups to a User Pool](#).

To create a work team using an existing Amazon Cognito user group

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, choose **Workforces**.
3. For **Private teams**, choose **Create private team**.
4. Under **Team details**, give the team a name. The name must be unique in your account in an AWS Region.
5. For **Add workers**, choose **Import existing Amazon Cognito user groups**, and choose one or more user groups that are part of the new team.
6. If you choose an **SNS topic**, all workers added to the team are subscribed to the Amazon Simple Notification Service (Amazon SNS) topic and notified when new work items are available to the team. Choose from a list of your existing SNS topics related to Amazon SageMaker Ground Truth or Amazon Augmented AI or choose **Create new topic** to create one.

Subscriptions

After you have created a work team, you can see more information about the team and change or set the SNS topic to which its members are subscribed using the Amazon Cognito console. Any members of the team who were added to the team prior to the team being subscribed to a topic need to be subscribed to that topic manually. For information, see [Create and manage Amazon SNS topics for your work teams \(p. 188\)](#).

Add and Remove Workers (Amazon Cognito Console)

When using the Amazon Cognito console to add workers to a work team, you must add a user to the user pool associated with the workforce before adding that user to a user group. Users can be added to a user pool in various ways. For more information, see [Signing Up and Confirming User Accounts](#).

Add a Worker to a Work Team

After a user has been added to a pool, the user can be associated with user groups inside of that pool. After a user has been added to a user group, that user becomes a worker on any work team created using that user group.

To add a user to a user group

1. Open the Amazon Cognito console: <https://us-east-2.console.aws.amazon.com/Amazon%20Cognito/home>
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups** and do one of the following:
 - Choose **Groups**, choose the group that you want to add the user to, and choose **Add users**. Choose the users that you want to add by choosing the plus-icon to the right of the users' name.

- Choose **Users**, choose the user that you want to add to the user group, and choose **Add to group**. From the drop down menu, choose the group and choose **Add to group**.

Disable and Remove a Worker From a Work Team

Disabling a worker stops the worker from receiving a jobs. This action doesn't remove the worker from the workforce, or any work team the worker is associated with. To remove a user from a work team in Amazon Cognito, you remove the user from the user group associated with that team.

To deactivate a worker (Amazon Cognito console)

1. Open the Amazon Cognito console: [https://us-east-2.console.aws.amazon.com/Amazon Cognito/home](https://us-east-2.console.aws.amazon.com/Amazon%20Cognito/home).
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. Choose the user that you want to disable.
6. Choose **Disable User**

You can enable a disabled user by choosing **Enable User**.

To remove a user from a user group (Amazon Cognito console)

1. Open the Amazon Cognito console: <https://us-east-2.console.aws.amazon.com/cognito/home>.
2. Choose **Manage User Pools**
3. Choose the user pool associated with your Amazon SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. For **User** tab, choose the x-icon to the right of the group that you want to remove the user from.

Manage Private Workforce Access to Tasks Using IP Addresses

By default, a workforce isn't restricted to specific IP addresses. You can use the [UpdateWorkforce](#) operation to restrict access to tasks to workers at allowable IP address ranges ([CIDRs](#)). If you specify one or more CIDRs, workers who attempt to access tasks using any IP address outside the specified ranges are denied access and get a `Not Found` error message on the worker portal. You can specify up to four CIDR values using [UpdateWorkforce](#).

After you have restricted your workforce to one or more CIDRs, you can view all allowable CIDRs using the [DescribeWorkforce](#) operation.

Track Worker Performance

Amazon SageMaker Ground Truth logs worker events to Amazon CloudWatch, such as when a worker starts or submits a task. Use Amazon CloudWatch metrics to measure and track throughput across a team or for individual workers.

Important

Worker event tracking is not available for Amazon Augmented AI human review workflows.

Enable Tracking

During the set-up process for a new work team, the permissions for Amazon CloudWatch logging of worker events are created. Since this feature was added in August of 2019, work teams created prior

to that may not have the correct permissions. If all of your work teams were created before August 2019, create a new work team. It does not need any members and may be deleted after creation, but by creating it, the permissions will be established and will apply to all of your work teams, regardless of when they were created.

Examine Logs

After tracking is enabled, the activity of your workers is logged. Open the Amazon CloudWatch console and choose **Logs** in the navigation pane. You should see a log group named **/aws/sagemaker/groundtruth/WorkerActivity**.

Each completed task is represented by a log entry, which contains information about the worker, their team, the job, when the task was accepted, and when it was submitted.

Example Log entry

```
{  
    "worker_id": "cd449a289e129409",  
    "cognito_user_pool_id": "us-east-2_IpicJXXXX",  
    "cognito_sub_id": "d6947aeb-0650-447a-ab5d-894db61017fd",  
    "task_accepted_time": "Wed Aug 14 16:00:59 UTC 2019",  
    "task_submitted_time": "Wed Aug 14 16:01:04 UTC 2019",  
    "task_returned_time": "",  
    "workteam_arn": "arn:aws:sagemaker:us-east-2:#####:workteam/private-crowd/Sample-labeling-team",  
    "labeling_job_arn": "arn:aws:sagemaker:us-east-2:#####:labeling-job/metrics-demo",  
    "work_requester_account_id": "#####",  
    "job_reference_code": "#####",  
    "job_type": "Private",  
    "event_type": "TasksSubmitted",  
    "event_timestamp": "1565798464"  
}
```

A useful data point in each event is the `cognito_sub_id`. You can match that to an individual worker.

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Under the **Ground Truth** section, choose **Workforces**.
3. Choose **Private**.
4. Choose the name of a team in the **Private teams** section.
5. In the **Team summary** section, choose the user group identified under **Amazon Cognito user group**. That will take you to the group in the Amazon Cognito console.
6. The **Group** page lists the users in the group. Choose any user's link in the **Username** column to see more information about the user, including a unique `sub` ID.

To get information about all of the team's members, use the `ListUsers` action ([examples](#)) in the Amazon Cognito API.

Use Log Metrics

If you don't want to write your own scripts to process and visualize the raw log information, Amazon CloudWatch metrics provide insights into worker activity for you.

To view metrics

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.

3. Choose the **AWS/SageMaker/Workteam** name space, then explore the [available metrics \(p. 731\)](#). For example, selecting the **Workflow, Workteam** metrics lets you calculate the average time per submitted task for a specific labeling job.

For more information, see [Using Amazon CloudWatch Metrics](#).

Create and manage Amazon SNS topics for your work teams

Use the procedures in this topic when you need to:

- Create a topic that you want an existing work team to subscribe to.
- Create a topic before you've created a work team.
- Create or modify the work team with an API call, and you need to specify a topic Amazon Resource Name (ARN).

If you create a work team using the console, the console provides an option to create a new topic for the team so that you don't have to perform these steps.

Create the Amazon SNS topic

The steps for creating Amazon SNS topics for work team notifications is similar to the steps in [Getting Started](#) in the *Amazon SNS Developer Guide*, with one significant addition—you must add an access policy so that Amazon SageMaker can publish messages to the topic on your behalf.

To add the policy when you create the topic

1. Open the Amazon SNS console at <https://console.aws.amazon.com/sns/>.
2. In **Create topic**, enter the name of your topic and then choose **Next steps**.
3. In **Access policy**, choose **Advanced**.
4. In the **JSON editor**, find the **Resource** property, which displays the topic's ARN.
5. Copy the **Resource** ARN value.
6. Before the final closing brace (]), add the following policy.

```
, {
  "Sid": "AwsSagemaker_SnsAccessPolicy",
  "Effect": "Allow",
  "Principal": {
    "Service": "sagemaker.amazonaws.com"
  },
  "Action": "sns:Publish",
  "Resource": "ARN of the topic you copied in the previous step"
}
```

7. Create the topic.

After you create the topic, it appears in your **Topics** summary screen. For more information about creating topics, see [Creating a Topic](#) in the *Amazon SNS Developer Guide*.

Manage worker subscriptions

If you subscribe a work team to a topic after you've already created the work team, the individual work team members who were added to the team when the work team was created are not automatically subscribed to the topic. For information about subscribing workers' email addresses to the topic, see [Subscribing an Endpoint to an Amazon SNS Topic](#) in the *Amazon SNS Developer Guide*.

The only situation where workers are automatically subscribed to your topic is when you create or import an Amazon Cognito user group at the time that you create a work team *and* you set up the topic subscription when you create that work team. For more information about creating and managing your workteams with Amazon Cognito, see [Create Work Teams \(Amazon Cognito Console\) \(p. 185\)](#).

HTML Elements Reference

Crowd HTML Elements are web components, a web standard that abstracts HTML markup, CSS, and JavaScript functionality into an HTML tag or set of tags. Amazon SageMaker provides customers with the ability to design their own custom task templates in HTML. For more information about how to implement custom templates in Amazon SageMaker Ground Truth, see [Creating Custom Labeling Workflows \(p. 114\)](#). To learn more about custom templates in Amazon Augmented AI, see [Create Custom Worker Templates \(p. 163\)](#).

Amazon SageMaker crowd-elements

Following is a list of enhanced HTML elements that make building a custom template easier and provide a familiar UI for workers. These elements are supported in Ground Truth, Augmented AI, and Mechanical Turk.

Topics

- [crowd-alert \(p. 190\)](#)
- [crowd-badge \(p. 190\)](#)
- [crowd-button \(p. 191\)](#)
- [crowd-bounding-box \(p. 192\)](#)
- [crowd-card \(p. 195\)](#)
- [crowd-checkbox \(p. 196\)](#)
- [crowd-classifier \(p. 198\)](#)
- [crowd-classifier-multi-select \(p. 199\)](#)
- [crowd-entity-annotation \(p. 201\)](#)
- [crowd-fab \(p. 203\)](#)
- [crowd-form \(p. 204\)](#)
- [crowd-icon-button \(p. 205\)](#)
- [crowd-image-classifier \(p. 205\)](#)
- [crowd-image-classifier-multi-select \(p. 208\)](#)
- [crowd-input \(p. 210\)](#)
- [crowd-instance-segmentation \(p. 212\)](#)
- [crowd-instructions \(p. 213\)](#)
- [crowd-keypoint \(p. 214\)](#)
- [crowd-modal \(p. 217\)](#)
- [crowd-polygon \(p. 217\)](#)
- [crowd-radio-button \(p. 222\)](#)
- [crowd-radio-group \(p. 224\)](#)
- [crowd-semantic-segmentation \(p. 225\)](#)
- [crowd-slider \(p. 227\)](#)
- [crowd-tab \(p. 228\)](#)
- [crowd-tabs \(p. 229\)](#)

- [crowd-text-area \(p. 229\)](#)
- [crowd-toast \(p. 231\)](#)
- [crowd-toggle-button \(p. 231\)](#)

crowd-alert

A message that alerts the worker to a current situation.

Attributes

The following attributes are supported by this element.

[dismissible](#)

A Boolean switch that, if present, allows the message to be closed by the worker.

[type](#)

A string that specifies the type of message to be displayed. The possible values are "info" (the default), "success", "error", and "warning".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-badge

An icon that floats over the top right corner of another element to which it is attached.

Attributes

The following attributes are supported by this element.

[for](#)

A string that specifies the ID of the element to which the badge is attached.

[icon](#)

A string that specifies the icon to be displayed in the badge. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

This attribute overrides the *label* attribute.

label

The text to display in the badge. Three characters or less is recommended because text that is too large will overflow the badge area. An icon can be displayed instead of text by setting the *icon* attribute.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-button

A styled button that represents some action.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

form-action

A switch that either submits its parent [crowd-form \(p. 204\)](#) element, if set to "submit", or resets its parent `<crowd-form>` element, if set to "reset".

href

The URL to an online resource. Use this property if you need a link styled as a button.

icon

A string that specifies the icon to be displayed next to the button's text. The string must be the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded. For example, to insert the `search` iron-icon, use the following:

```
<crowd-button>
  <iron-icon icon="search"></iron-icon>
</crowd-button>
```

The icon is positioned to either the left or the right of the text, as specified by the *icon-align* attribute.

To use a custom icon see [icon-url](#).

icon-align

The left or right position of the icon relative to the button's text. The default is "left".

icon-url

A URL to a custom image for the icon. A custom image can be used in place of a standard icon that is specified by the *icon* attribute.

loading

A Boolean switch that, if present, displays the button as being in a loading state. This attribute has precedence over the *disabled* attribute if both attributes are present.

target

When you use the `href` attribute to make the button act as a hyperlink to a specific URL, the `target` attribute optionally targets a frame or window where the linked URL should load.

variant

The general style of the button. Use "primary" for primary buttons, "normal" for secondary buttons, "link" for tertiary buttons, or "icon" to display only the icon without text.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-bounding-box

A widget for drawing rectangles on an image and assigning a label to the portion of the image that is enclosed in each rectangle.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

An array of JSON objects, each of which sets a bounding box when the component is loaded. Each JSON object in the array contains the following properties. Bounding boxes set via the `initial-value` property can be adjusted and whether or not a worker answer was adjusted is tracked via an `initialValueModified` boolean in the worker answer output.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the `labels` attribute of the `<crowd-bounding-box>` element.

- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

You can extract the bounding box initial value from a manifest file of a previous job in a custom template using the Liquid templating language:

```
initial-value="[
    {% for box in task.input.manifestLine.label-attribute-name-from-prior-job.annotations %}
        {% capture class_id %}{{ box.class_id }}{% endcapture %}
        {% assign label = task.input.manifestLine.label-attribute-name-from-prior-job-
metadata.class-map[class_id] %}
        {
            label: {{label | to_json}},
            left: {{box.left}},
            top: {{box.top}},
            width: {{box.width}},
            height: {{box.height}},
        },
    {% endfor %}
]"
```

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a rectangle. **Limit:** 10 labels.

name

The name of this widget. It's used as a key for the widget's input in the form output.

src

The URL of the image on which to draw bounding boxes.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 193\)](#), [short-instructions \(p. 193\)](#)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to draw bounding boxes.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

[boundingBoxes](#)

An array of JSON objects, each of which specifies a bounding box that has been created by the worker. Each JSON object in the array contains the following properties.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the `<crowd-bounding-box>` element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

[inputImageProperties](#)

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

Single Label, Single Box / Multiple Label, Single Box

```
[  
  {  
    "annotatedResult": {  
      "boundingBoxes": [  
        {  
          "height": 401,  
          "label": "Dog",  
          "left": 243,  
          "top": 117,  
          "width": 187  
        }  
      ],  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      }  
    }  
  }  
]
```

Single Label, Multiple Box

```
[  
  {  
    "annotatedResult": {  
      "boundingBoxes": [  
        {  
          "height": 401,  
          "label": "Dog",  
          "left": 243,  
          "top": 117,  
          "width": 187  
        },  
        {  
          "height": 401,  
          "label": "Dog",  
          "left": 243,  
          "top": 117,  
          "width": 187  
        }  
      ]  
    }  
  }  
]
```

```
[  
  {  
    "height": 283,  
    "label": "Dog",  
    "left": 684,  
    "top": 120,  
    "width": 116  
  },  
  ],  
  "inputImageProperties": {  
    "height": 533,  
    "width": 800  
  }  
}  
]  
]
```

Multiple Label, Multiple Box

```
[  
  {  
    "annotatedResult": {  
      "boundingBoxes": [  
        {  
          "height": 395,  
          "label": "Dog",  
          "left": 241,  
          "top": 125,  
          "width": 158  
        },  
        {  
          "height": 298,  
          "label": "Cat",  
          "left": 699,  
          "top": 116,  
          "width": 101  
        }  
      ],  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      }  
    }  
  }  
]
```

You could have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-card

A box with an elevated appearance for displaying information.

Attributes

The following attributes are supported by this element.

heading

The text displayed at the top of the box.

image

A URL to an image to be displayed within the box.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-checkbox

A UI component that can be checked or unchecked allowing a user to select multiple options from a set.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the check box as checked.

disabled

A Boolean switch that, if present, displays the check box as disabled and prevents it from being checked.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A string used as the name for the check box state in the output. Defaults to "on" if not specified.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)

- **Child elements:** none

Output

Provides a JSON object. The name string is the object name and the valuemstring is the property name for a Boolean value based on the check box state; true if checked, false if not checked.

Example : Sample Element Outputs

Using the same name value for multiple boxes.

```
<!-- INPUT -->
<div><crowd-checkbox name="myformbit" value="Red"> Red </div>
<div><crowd-checkbox name="myformbit" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="myformbit" value="Green"> Green </div>
```

```
//Output with "Red" checked
[
  {
    "myformbit": {
      "Green": false,
      "Red": true,
      "Yellow": false
    }
  }
]
```

Note that all three color values are properties of a single object.

Using different name values for each box.

```
<!-- INPUT -->
<div><crowd-checkbox name="Stop" value="Red"> Red </div>
<div><crowd-checkbox name="Slow" value="Yellow"> Yellow </div>
<div><crowd-checkbox name="Go" value="Green"> Green </div>
```

```
//Output with "Red" checked
[
  {
    "Go": {
      "Green": false
    },
    "Slow": {
      "Yellow": false
    },
    "Stop": {
      "Red": true
    }
  }
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)

- [HTML Elements Reference \(p. 189\)](#)

crowd-classifier

A widget for classifying non-image content, such as audio, video, or text.

The following is an example of an HTML worker task template built using `crowd-classifier`. This example uses the [Liquid template language](#) to automate:

- Label categories in the `categories` parameter
- The objects that are being classified in the `classification-target` parameter.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="category"
    categories="{{ task.input.labels | to_json | escape }}"
    header="What type of a document is this?"
  >
    <classification-target>
      <iframe style="width: 100%; height: 600px;" src="{{ task.input.taskObject | grant_read_access }}" type="application/pdf"></iframe>
    </classification-target>

    <full-instructions header="Document Classification Instructions">
      <p>Read the task carefully and inspect the document.</p>
      <p>Choose the appropriate label that best suits the document.</p>
    </full-instructions>

    <short-instructions>
      Please choose the correct category for the document
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

Attributes

The following attributes are supported by this element.

categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the to the text. You should include "other" as a category, otherwise the worker my not be able to provide an answer.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

name

The name of this widget. It is used as a key for the widget's input in the form output.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [classification-target \(p. 199\)](#), [full-instructions \(p. 199\)](#), [short-instructions \(p. 199\)](#)

Regions

The following regions are supported by this element.

classification-target

The content to be classified by the worker. This can be plain text or HTML. Examples of how the HTML can be used include *but are not limited to* embedding a video or audio player, embedding a PDF, or performing a comparison of two or more images.

full-instructions

General instructions about how to do text classification.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The output of this element is an object using the specified name value as a property name, and a string from the categories as the property's value.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      "label": "<value>"  
    }  
  }  
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-classifier-multi-select

A widget for classifying various forms of content—such as audio, video, or text—into one or more categories. The content to classify is referred to as an *object*. The following is an example of an HTML worker task template built using this crowd element.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>  
  
<crowd-form>  
  <crowd-classifier-multi-select  
    name="category"  
    categories="['Positive', 'Negative', 'Neutral']"  
    header="Select the relevant categories"  
    exclusion-category="{ text: 'None of the above' }"  
  >  
    <classification-target>
```

```
    {{ task.input.taskObject }}
```

```
</classification-target>
```

```
<full-instructions header="Text Categorization Instructions">
```

```
    <p><strong>Positive</strong> sentiment include: joy, excitement, delight</p>
```

```
    <p><strong>Negative</strong> sentiment include: anger, sarcasm, anxiety</p>
```

```
    <p><strong>Neutral</strong>: neither positive or negative, such as stating a fact</p>
```

```
    <p><strong>N/A</strong>: when the text cannot be understood</p>
```

```
    <p>When the sentiment is mixed, such as both joy and sadness, choose both labels.</p>
```

```
</full-instructions>
```

```
<short-instructions>
```

```
    Choose all categories that are expressed by the text.
```

```
</short-instructions>
```

```
</crowd-classifier-multi-select>
```

```
</crowd-form>
```

Attributes

The following attributes are supported by the `crowd-classifier-multi-select` element. Each attribute accepts a string value or string values.

categories

Required. A JSON-formatted array of strings, each of which is a category that a worker can assign to the object.

header

Required. The text to display above the image. This is typically a question or simple instruction for workers.

name

Required. The name of this widget. In the form output, the name is used as a key for the widget's input.

exclusion-category

Optional. A JSON-formatted string with the following format: "`{ text: 'default-value' }`". This attribute sets a default value that workers can choose if none of the labels applies to the object shown in the worker UI.

Element Hierarchy

This element has the following parent and child elements:

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [classification-target \(p. 199\)](#), [full-instructions \(p. 199\)](#), [short-instructions \(p. 199\)](#)

Regions

This element uses the following regions.

classification-target

The content to be classified by the worker. Content can be plain text or an object that you specify in the template using HTML. For example, you can use HTML elements to include a video or audio player, embedding a PDF file, or include a comparison of two or more images.

full-instructions

General instructions about how to classify text.

short-instructions

Important task-specific instructions. These instructions are displayed prominently.

Output

The output of this element is an object that uses the specified `name` value as a property name, and a string from `categories` as the property's value.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      labels: ["label_a", "label_b"]  
    }  
  }  
]
```

See Also

For more information, see the following:

- [Text Classification \(Multi-label\) \(p. 80\)](#)
- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-entity-annotation

A widget for labeling words, phrases, or character strings within a longer text.

Important: Self-contained Widget

Do not use `<crowd-entity-annotation>` element with the `<crowd-form>` element. It contains its own form submission logic and **Submit** button.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

A JSON formatted array of objects, each of which defines an annotation to apply to the text at initialization. Objects contain a `label` value that matches one in the `labels` attribute, an integer `startOffset` value for labeled span's starting unicode offset, and an integer `endOffset` value for the ending unicode offset.

Example

```
[
```

```
[  
  {  
    label: 'person',  
    startOffset: 0,  
    endOffset: 16  
  },  
  ...  
]
```

labels

A JSON formatted array of objects, each of which contains:

- **label** (required): The name used to identify entities.
- **fullDisplayName** (optional): Used for the label list in the task widget. Defaults to the label value if not specified.
- **shortDisplayName** (optional): An abbreviation of 3-4 letters to display above selected entities. Defaults to the label value if not specified.

shortDisplayName is highly recommended

Values displayed above the selections can overlap and create difficulty managing labeled entities in the workspace. Providing a 3-4 character shortDisplayName for each label is highly recommended to prevent overlap and keep the workspace manageable for your workers.

Example

```
[  
  {  
    label: 'person',  
    shortDisplayName: 'per',  
    fullDisplayName: 'person'  
  }  
]
```

name

Serves as the widget's name in the DOM. It is also used as the label attribute name in form output and the output manifest.

text

The text to be annotated. The templating system escapes quotes and HTML strings by default. If your code is already escaped or partially escaped, see [Variable filters \(p. 119\)](#) for more ways to control escaping.

Element Hierarchy

This element has the following parent and child elements.

- **Child elements:** [full-instructions \(p. 202\)](#), [short-instructions \(p. 203\)](#)

Regions

The following regions are supported by this element.

[full-instructions](#)

General instructions about how to work with the widget.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

entities

A JSON object that specifies the start, end, and label of an annotation. This object contains the following properties.

- **label** – The assigned label.
- **startOffset** – The Unicode offset of the beginning of the selected text.
- **endOffset** – The Unicode offset of the first character after the selection.

Example : Sample Element Outputs

The following is a sample of the output from this element.

```
{  
  "myAnnotatedResult": {  
    "entities": [  
      {  
        "endOffset": 54,  
        "label": "person",  
        "startOffset": 47  
      },  
      {  
        "endOffset": 97,  
        "label": "event",  
        "startOffset": 93  
      },  
      {  
        "endOffset": 219,  
        "label": "date",  
        "startOffset": 212  
      },  
      {  
        "endOffset": 271,  
        "label": "location",  
        "startOffset": 260  
      }  
    ]  
  }  
}
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-fab

A floating button with an image in its center.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the floating button as disabled and prevents clicks.

icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

label

A string consisting of a single character that can be used instead of an icon. Emojis or multiple characters may result in the button displaying an ellipsis instead.

title

A string that will display as a tool tip when the mouse hovers over the button.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-form

The form wrapper for all custom tasks. Sets and implements important actions for the proper submission of your form data.

If a [crowd-button \(p. 191\)](#) of type "submit" is not included inside the <crowd-form> element, it will automatically be appended within the <crowd-form> element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** none
- **Child elements:** Any of the [UI Template \(p. 189\)](#) elements

Element Events

The `crowd-form` element extends the [standard HTML form element](#) and inherits its events, such as `onclick` and `onsubmit`.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-icon-button

A button with an image placed in the center. When the user touches the button, a ripple effect emanates from the center of the button.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-image-classifier

A widget for classifying an image, which can be a JPG, PNG, or GIF, with no size limit.

Attributes

The following attributes are required by this element.

categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the image. You should include "other" as a category, so that the worker can provide an answer. You can specify up to 10 categories.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

name

The name of this widget. It is used as a key for the widget's input in the form output.

overlay

Information to be overlaid on the source image. This is for verification workflows of bounding-box and semantic-segmentation tasks.

It is a JSON object containing an object with the name of the task-type in camelCase as the key. That key's value is an object that contains the labels and other necessary information from the previous task.

An example of a `crowd-image-classifier` element with attributes for verifying a semantic segmentation task follows:

```
<crowd-image-classifier
    name='crowd-image-classifier'
    categories='["good", "bad"]'
    src='URL of image to be classified'
    header='Please classify'
    overlay='{
        "semanticSegmentation": {
            "labels": ["Cat", "Dog", "Bird", "Cow"],
            "labelMappings": {
                "Bird": {
                    "color": "#ff7f0e"
                },
                "Cat": {
                    "color": "#2ca02c"
                },
                "Cow": {
                    "color": "#d62728"
                },
                "Dog": {
                    "color": "#2acf59"
                }
            },
            "src": "URL of overlay image",
        }
    }'
>
```

A bounding-box verification task would use the `overlay` value like follows:

```
<crowd-image-classifier
    name="boundingBoxClassification"
    header="Rate the quality of the annotations based on the background section
           in the instructions on the left hand side."
    src="https://i.imgur.com/CIPKVJ0.jpg"
    categories="['good', 'bad', 'okay']"
    overlay='{
        "boundingBox": {
            labels: ["bird", "cat"],
            value: [
                {
                    height: 284,
                    label: "bird",
                    left: 230,
                    top: 974,
                    width: 223
                },
                {
                    height: 69,
```

```
        label: "bird",
        left: 79,
        top: 889,
        width: 247
    }
],
},
>
```

src

The URL of the image to be classified.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 207\)](#), [short-instructions \(p. 207\)](#), [worker-comment \(p. 207\)](#)

Regions

The following regions are used by this element.

full-instructions

General instructions for the worker on how to classify an image.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

worker-comment

Use this in verification workflows when you need workers to explain why they made the choice they did. Use the text between the opening and closing tags to provide instructions for workers on what information should be included in the comment.

It uses the following attributes:

header

A phrase with a call to action for leaving a comment. Used as the title text for a modal window where the comment is added.

Optional. Defaults to "Add a comment."

link-text

This text appears below the categories in the widget. When clicked, it opens a modal window where the worker may add a comment.

Optional. Defaults to "Add a comment."

placeholder

An example text in the comment text area that is overwritten when worker begins to type. This does not appear in output if the worker leaves the field blank.

Optional. Defaults to blank.

Output

The output of this element is a string that specifies one of the values defined in the *categories* attribute of the <crowd-image-classifier> element.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      "label": "<value>"  
      "workerComment": "Comment - if no comment is provided, this field will not be  
present"  
    }  
  }  
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-image-classifier-multi-select

A widget for classifying an image into one or more categories. The image can be a JPG, PNG, or GIF file, and has no size limit. The following is an example of an HTML worker task template built using this crowd element.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>  
  
<crowd-form>  
  <crowd-image-classifier-multi-select  
    name="animals"  
    categories="['Cat', 'Dog', 'Horse', 'Pig', 'Bird']"  
    src="https://images.unsplash.com/photo-1509205477838-a534e43a849f?  
ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDD9&auto=format&fit=crop&w=1998&q=80"  
    header="Please identify the animals in this image"  
    exclusion-category="{ text: 'None of the above' }"  
  >  
    <full-instructions header="Classification Instructions">  
      <p>If more than one label applies to the image, select multiple labels.</p>  
      <p>If no labels apply, select <b>None of the above</b></p>  
    </full-instructions>  
  
    <short-instructions>  
      <p>Read the task carefully and inspect the image.</p>  
      <p>Choose the appropriate label(s) that best suit the image.</p>  
    </short-instructions>  
  </crowd-image-classifier-multi-select>  
</crowd-form>
```

Attributes

The following attributes are supported by the `crowd-image-classifier-multi-select` element. Each attribute accepts a string value or string values.

categories

Required. A JSON-formatted array of strings, each of which is a category that a worker can assign to the image. A worker must choose at least one category and can choose all categories.

header

Required. The text to display above the image. This is typically a question or simple instruction for workers.

name

Required. The name of this widget. In the form output, the name is used as a key for the widget's input.

src

Required. The URL of the image to be classified.

exclusion-category

Optional. A JSON-formatted string with the following format: "{ text: '[default-value](#)' }". This attribute sets a default value that workers can choose if none of the labels applies to the image shown in the worker UI.

Element Hierarchy

This element has the following parent and child elements:

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 207\)](#), [short-instructions \(p. 207\)](#), [worker-comment \(p. 207\)](#)

Regions

This element uses the following regions

full-instructions

General instructions for the worker on how to classify an image.

short-instructions

Important task-specific instructions. These instructions are displayed prominently.

Output

The output of this element is a string that specifies one or more of the values defined in the categories attribute of the <crowd-image-classifier-multi-select> element.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      labels: ["label_a", "label_b"]  
    }  
  }  
]
```

See Also

For more information, see the following:

- [Image Classification \(Multi-label\) \(p. 72\)](#)
- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-input

A box that accepts input data.

Cannot be self-closing

Unlike the `input` element in the HTML standard, this element cannot be self-closed by putting a slash before the ending bracket, e.g. `<crowd-input ... />`. It must be followed with a `</crowd-input>` to close the element.

Attributes

The following attributes are supported by this element.

allowed-pattern

A regular expression that is used with the `auto-validate` attribute to ignore non-matching characters as the worker types.

auto-focus

When the value is set to true, the browser places focus inside the input area after loading. This way, the worker can start typing without having to select it first.

auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the `error-message` and `allowed-pattern` attributes.

disabled

A Boolean switch that, if present, displays the input area as disabled.

error-message

The text to be displayed below the input field, on the left side, if validation fails.

label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the `value` attribute is set.

max-length

A maximum number of characters the input will accept. Input beyond this limit is ignored.

min-length

A minimum length for the input in the field

[name](#)

Sets the name of the input to be used in the DOM and the output of the form.

[placeholder](#)

A string value that is used as placeholder text, displayed until the worker starts entering data into the input. It is not used as a default value.

[required](#)

A Boolean switch that, if present, requires the worker to provide input.

[type](#)

Takes a string to set the HTML5 `input-type` behavior for the input. Examples include `file` and `date`.

[value](#)

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

Output

Provides a `name` string as the property name, and the text that was entered in the field as its value.

Example : Sample JSON Output

The values for multiple elements are output in the same object, with their `name` attribute value as their property name. Elements with no input do not appear in the output. For example, let's use three inputs:

```
<crowd-input name="tag1" label="Word/phrase 1"></crowd-input>
<crowd-input name="tag2" label="Word/phrase 2"></crowd-input>
<crowd-input name="tag3" label="Word/phrase 3"></crowd-input>
```

This is the output if only two have input:

```
[  
  {  
    "tag1": "blue",  
    "tag2": "red"  
  }  
]
```

This means any code built to parse these results should be able to handle the presence or absence of each input in the answers.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-instance-segmentation

A widget for identifying individual instances of specific objects within an image and creating a colored overlay for each labeled instance.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to an instance of an object in the image. Workers can generate different overlay colors for each relevant instance by selecting "add instance" under the label in the tool.

name

The name of this widget. It is used as a key for the labeling data in the form output.

src

The URL of the image that is to be labeled.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 212\)](#), [short-instructions \(p. 212\)](#)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

instances

A JSON Array containing objects with the instance labels and colors.

- **color** – The hexadecimal value of the label's RGB color in the `labeledImage` PNG.

- **label** – The label given to overlay(s) using that color. This value may repeat, because the different instances of the label are identified by their unique color.

[inputImageProperties](#)

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "annotatedResult": {  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      },  
      "instances": [  
        {  
          "color": "#1f77b4",  
          "label": "<Label 1>":  
        },  
        {  
          "color": "#2ca02c",  
          "label": "<Label 1>":  
        },  
        {  
          "color": "#ff7f0e",  
          "label": "<Label 3>":  
        },  
      ],  
      "labeledImage": {  
        "pngImageData": "<Base-64 Encoded Data>"  
      }  
    }  
  ]
```

[See Also](#)

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

[crowd-instructions](#)

An element that displays instructions on three tabbed pages, **Summary**, **Detailed Instructions**, and **Examples**, when the worker clicks on a link or button.

[Attributes](#)

The following attributes are supported by this element.

link-text

The text to display for opening the instructions. The default is **Click for instructions**.

link-type

A string that specifies the type of trigger for the instructions. The possible values are "link" (default) and "button".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

Regions

The following regions are supported by this element.

detailed-instructions

Content that provides specific instructions for a task. This appears on the page of the "Detailed Instructions" tab.

negative-example

Content that provides examples of inadequate task completion. This appears on the page of the "Examples" tab. More than one example may be provided within this element.

positive-example

Content that provides examples of proper task completion. This appears on the page of the "Examples" tab.

short-summary

A brief statement that summarizes the task to be completed. This appears on the page of the "Summary" tab. More than one example may be provided within this element.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-keypoint

Generates a tool to select and annotate key points on an image.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

An array, in JSON format, of keypoints to be applied to the image on start. For example:

```
initial-value="[
  {
    'label': 'Left Eye',
    'x': 1022,
    'y': 429
  },
  {
    'label': 'Beak',
    'x': 941,
    'y': 403
  }
]
```

Note

Please note that label values used in this attribute must have a matching value in the `labels` attribute or the point will not be rendered.

labels

An array, in JSON format, of strings to be used as keypoint annotation labels.

name

A string used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

src

The source URI of the image to be annotated.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 215\)](#), [short-instructions \(p. 215\)](#)

Regions

The following regions are required by this element.

full-instructions

General instructions about how to annotate the image.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

keypoints

An array of JSON objects containing the coordinates and label of a keypoint. Each object contains the following properties.

- **label** – The assigned label for the keypoint.
- **x** – The X coordinate, in pixels, of the keypoint on the image.
- **y** – The Y coordinate, in pixels, of the keypoint on the image.

Note

X and Y coordinates are based on 0,0 being the top left corner of the image.

Example : Sample Element Outputs

The following is a sample output from using this element.

```
[  
  {  
    "crowdKeypoint": {  
      "inputImageProperties": {  
        "height": 1314,  
        "width": 962  
      },  
      "keypoints": [  
        {  
          "label": "dog",  
          "x": 155,  
          "y": 275  
        },  
        {  
          "label": "cat",  
          "x": 341,  
          "y": 447  
        },  
        {  
          "label": "cat",  
          "x": 491,  
          "y": 513  
        },  
        {  
          "label": "dog",  
          "x": 714,  
          "y": 578  
        },  
        {  
          "label": "cat",  
          "x": 712,  
          "y": 763  
        },  
        {  
          "label": "cat",  
          "x": 397,  
          "y": 531  
        }  
      ]  
    }  
  }]
```

```
        "y": 814
    }
}
]
```

You may have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-modal

A small window that pops up on the display when it is opened.

Attributes

The following attributes are supported by this element.

link-text

The text to display for opening the modal. The default is "Click to open modal".

link-type

A string that specifies the type of trigger for the modal. The possible values are "link" (default) and "button".

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-polygon

A widget for drawing polygons on an image and assigning a label to the portion of the image that is enclosed in each polygon.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a polygon.

name

The name of this widget. It's used as a key for the widget's input in the form output.

src

The URL of the image on which to draw polygons.

initial-value

An array of JSON objects, each of which defines a polygon to be drawn when the component is loaded. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the <crowd-polygon> element.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon.

Example

An `initial-value` attribute might look something like this.

```
initial-value =
  [
    {
      "label": "dog",
      "vertices":
        [
          {
            ...
            {
              "x": 570,
              "y": 239
            },
            ...
            {
              "x": 759,
              "y": 281
            }
          ]
    }
  ]'
```

Because this will be within an HTML element, the JSON array must be enclosed in single or double quotes. The example above uses single quotes to encapsulate the JSON and double quotes within the JSON itself. If you must mix single and double quotes inside your JSON, replace them with their HTML entity codes (" for double quote, ' for single) to safely escape them.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)

- **Child elements:** [full-instructions \(p. 219\)](#), [short-instructions \(p. 219\)](#)

Regions

The following regions are required.

[full-instructions](#)

General instructions about how to draw polygons.

[short-instructions](#)

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

[polygons](#)

An array of JSON objects, each of which describes a polygon that has been created by the worker. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon. The top left corner of the image is 0,0.

[inputImageProperties](#)

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

Single Label, Single Polygon

```
{  
    "annotatedResult":  
    {  
        "inputImageProperties": {  
            "height": 853,  
            "width": 1280  
        },  
        "polygons":  
        [  
            {  
                "label": "dog",  
                "vertices":  
                [  
                    {  
                        "x": 570,  
                        "y": 239  
                    }  
                ]  
            }  
        ]  
    }  
}
```

```
        },
        {
          "x": 603,
          "y": 513
        },
        {
          "x": 823,
          "y": 645
        },
        {
          "x": 901,
          "y": 417
        },
        {
          "x": 759,
          "y": 281
        }
      ]
    }
  }
]
```

Single Label, Multiple Polygons

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons": [
        {
          "label": "dog",
          "vertices": [
            {
              "x": 570,
              "y": 239
            },
            {
              "x": 603,
              "y": 513
            },
            {
              "x": 823,
              "y": 645
            },
            {
              "x": 901,
              "y": 417
            },
            {
              "x": 759,
              "y": 281
            }
          ]
        },
        {
          "label": "dog",
          "vertices": [
            {
              "x": 870,
              "y": 278
            }
          ]
        }
      ]
    }
  }
]
```

```
        },
        {
          "x": 908,
          "y": 446
        },
        {
          "x": 1009,
          "y": 602
        },
        {
          "x": 1116,
          "y": 519
        },
        {
          "x": 1174,
          "y": 498
        },
        {
          "x": 1227,
          "y": 479
        },
        {
          "x": 1179,
          "y": 405
        },
        {
          "x": 1179,
          "y": 337
        }
      ]
    }
  ]
}
```

Multiple Labels, Multiple Polygons

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons": [
        {
          "label": "dog",
          "vertices": [
            {
              "x": 570,
              "y": 239
            },
            {
              "x": 603,
              "y": 513
            },
            {
              "x": 823,
              "y": 645
            },
            {
              "x": 901,
              "y": 417
            },
            {
              "x": 570,
              "y": 239
            }
          ]
        }
      ]
    }
  }
]
```

```
{  
    "x": 759,  
    "y": 281  
}  
]  
},  
{  
    "label": "cat",  
    "vertices": [  
        {  
            "x": 870,  
            "y": 278  
        },  
        {  
            "x": 908,  
            "y": 446  
        },  
        {  
            "x": 1009,  
            "y": 602  
        },  
        {  
            "x": 1116,  
            "y": 519  
        },  
        {  
            "x": 1174,  
            "y": 498  
        },  
        {  
            "x": 1227,  
            "y": 479  
        },  
        {  
            "x": 1179,  
            "y": 405  
        },  
        {  
            "x": 1179,  
            "y": 337  
        }  
    ]  
}  
]  
]
```

You could have many labels available, but only the ones that are used appear in the output.

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-radio-button

A button that can be either checked or unchecked. When radio buttons are inside a radio group, exactly one radio button in the group can be checked at any time. The following is an example of how to configure a `crowd-radio-button` element inside of a `crowd-radio-group` element.

```
<crowd-radio-group>
    <crowd-radio-button name="tech" value="tech">Technology</crowd-radio-button>
    <crowd-radio-button name="politics" value="politics">Politics</crowd-radio-button>
</crowd-radio-group>
```

The previous example can be seen in a custom worker task template in this GitHub example: [entity recognition labeling job custom template](#).

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the radio button as checked.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents it from being checked.

name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

Note

If you use the buttons outside of a [crowd-radio-group \(p. 224\)](#) element, but with the same name string and different value strings, the name object in the output will contain a Boolean value for each value string. To ensure that only one button in a group is selected, make them children of a [crowd-radio-group \(p. 224\)](#) element and use different name values.

value

A property name for the element's boolean value. If not specified, it uses "on" as the default, e.g. { "<name>": { "<value>": <true or false> } }.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-radio-group \(p. 224\)](#)
- **Child elements:** none

Output

Outputs an object with the following pattern: { "<name>": { "<value>": <true or false> } }. If you use the buttons outside of a [crowd-radio-group \(p. 224\)](#) element, but with the same name string and different value strings, the name object will contain a Boolean value for each value string. To ensure that only one in a group of buttons is selected, make them children of a [crowd-radio-group \(p. 224\)](#) element and use different name values.

Example Sample output of this element

```
[  
  {  
    "btn1": {
```

```
        "yes": true
    },
    "btn2": {
        "no": false
    }
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-radio-group

A group of radio buttons. Only one radio button within the group can be selected. Choosing one radio button clears any previously chosen radio button within the same group. For an example of a custom UI template that uses the `crowd-radio-group` element, see this [entity recognition labeling job custom template](#).

Attributes

No special attributes are supported by this element.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [crowd-radio-button \(p. 222\)](#)

Output

Outputs an array of objects representing the [crowd-radio-button \(p. 222\)](#) elements within it.

Example Sample of Element Output

```
[{
    {
        "btn1": {
            "yes": true
        },
        "btn2": {
            "no": false
        }
    }
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)

- [HTML Elements Reference \(p. 189\)](#)

crowd-semantic-segmentation

A widget for segmenting an image and assigning a label to each image segment.

Attributes

The following attributes are supported by this element.

header

The text to display above the image. This is typically a question or simple instruction for the worker.

initial-value

A JSON object containing the color mappings of a prior semantic segmentation job and a link to the overlay image output by the prior job. Include this when you want a human worker to verify the results of a prior labeling job and adjust it if necessary.

The attribute would appear as follows:

```
initial-value='{
  "labelMappings": {
    "Bird": {
      "color": "#ff7f0e"
    },
    "Cat": {
      "color": "#2ca02c"
    },
    "Cow": {
      "color": "#d62728"
    },
    "Dog": {
      "color": "#1f77b4"
    }
  },
  "src": {{ "S3 file URL for image" | grant_read_access }}
}'
```

While label mappings are included in individual worker output records, the overall result is represented as the `internal-color-map` in the consolidated results.

You can convert the `internal-color-map` to `label-mappings` in a custom template using the Liquid templating language:

```
initial-value="{
  'src' : '{{ task.input.manifestLine.'label-attribute-name-from-prior-job'| grant_read_access }}',
  'labelMappings': {
    '% for box in task.input.manifestLine.'label-attribute-name-from-prior-job-
metadata.internal-color-map %'
      '{% if box[1]['class-name'] != 'BACKGROUND' %}
        {{ box[1]['class-name'] | to_json }}: {
          'color': {{ box[1]['hex-color'] | to_json }}
        },
      {% endif %}
      {% endfor %}
  }
}"
```

labels

A JSON formatted array of strings, each of which is a label that a worker can assign to a segment of the image.

name

The name of this widget. It is used as a key for the widget's input in the form output.

src

The URL of the image that is to be segmented.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [full-instructions \(p. 226\)](#), [short-instructions \(p. 226\)](#)

Regions

The following regions are supported by this element.

full-instructions

General instructions about how to do image segmentation.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Output

The following output is supported by this element.

labeledImage

A JSON Object containing a Base64 encoded PNG of the labels.

labelMappings

A JSON Object containing objects with named with the segmentation labels.

- **color** – The hexadecimal value of the label's RGB color in the labeledImage PNG.

initialValueModified

A boolean representing whether the initial values have been modified. This is only included when the output is from an adjustment task.

inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "annotatedResult": {  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      },  
      "labelMappings": {  
        "<Label 2>": {  
          "color": "#ff7f0e"  
        },  
        "<label 3>": {  
          "color": "#2ca02c"  
        },  
        "<label 1>": {  
          "color": "#1f77b4"  
        }  
      },  
      "labeledImage": {  
        "pngImageData": "<Base-64 Encoded Data>"  
      }  
    }  
  }  
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-slider

A bar with a sliding knob that allows a worker to select a value from a range of values by moving the knob. The slider makes it a great choice for settings that reflect intensity levels, such as volume, brightness, or color saturation.

Attributes

The following attributes are supported by this element.

disabled

A Boolean switch that, if present, displays the slider as disabled.

editable

A Boolean switch that, if present, displays an up/down button that can be chosen to select the value.

Selecting the value via the up/down button is an alternative to selecting the value by moving the knob on the slider. The knob on the slider will move synchronously with the up/down button choices.

max

A number that specifies the maximum value on the slider.

[min](#)

A number that specifies the minimum value on the slider.

[name](#)

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

[pin](#)

A Boolean switch that, if present, displays the current value above the knob as the knob is moved.

[required](#)

A Boolean switch that, if present, requires the worker to provide input.

[secondary-progress](#)

When used with a `crowd-slider-secondary-color` CSS attribute, the progress bar is colored to the point represented by the `secondary-progress`. For example, if this was representing the progress on a streaming video, the value would represent where the viewer was in the video timeline. The `secondary-progress` value would represent the point on the timeline to which the video had buffered.

[step](#)

A number that specifies the difference between selectable values on the slider.

[value](#)

A preset that becomes the default if the worker does not provide input.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

[crowd-tab](#)

A component styled to look like a tab with information below.

Attributes

The following attributes are supported by this element.

[header](#)

The text appearing on the tab. This is usually some short descriptive name indicative of the information contained below the tab.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-tabs \(p. 229\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

[crowd-tabs](#)

A container for tabbed information.

Attributes

This element has no attributes.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** [crowd-tab \(p. 228\)](#)

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

[crowd-text-area](#)

A field for text input.

Attributes

The following attributes are supported by this element.

[auto-focus](#)

A Boolean switch that, if present, puts the cursor in this element on-load so that users can immediately begin typing without having to click inside the element.

[auto-validate](#)

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the *error-message* and *allowed-pattern* attributes.

[char-counter](#)

A Boolean switch that, if present, puts a small text field beneath the lower-right corner of the element, displaying the number of characters inside the element.

[disabled](#)

A Boolean switch that, if present, displays the input area as disabled.

[error-message](#)

The text to be displayed below the input field, on the left side, if validation fails.

[label](#)

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the *value* attribute is set.

[max-length](#)

An integer that specifies the maximum number of characters allowed by the element. Characters typed or pasted beyond the maximum are ignored.

[max-rows](#)

An integer that specifies the maximum number of rows of text that are allowed within a crowd-text-area. Normally the element expands to accommodate new rows. If this is set, after the number of rows exceeds it, content scrolls upward out of view and a scrollbar control appears.

[name](#)

A string used to represent the element's data in the output.

[placeholder](#)

A string presented to the user as placeholder text. It disappears after the user puts something in the input area.

[rows](#)

An integer that specifies the height of the element in rows of text.

[value](#)

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

Output

This element outputs the `name` as a property name and the element's text contents as the value. Carriage returns in the text are represented as `\n`.

Example Sample output for this element

```
[  
  {  
    "textInput1": "This is the text; the text that\nmakes the crowd go wild."  
  }  
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-toast

A subtle notification that temporarily appears on the display. Only one crowd-toast is visible.

Attributes

The following attributes are supported by this element.

duration

A number that specifies the duration, in seconds, that the notification appears on the screen.

text

The text to display in the notification.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

crowd-toggle-button

A button that acts as an ON/OFF switch, toggling a state.

Attributes

The following attributes are supported by this element.

checked

A Boolean switch that, if present, displays the button switched to the ON position.

disabled

A Boolean switch that, if present, displays the button as disabled and prevents toggling.

invalid

When in an off position, a button using this attribute, will display in an alert color. The standard is red, but may be changed in CSS. When toggled on, the button will display in the same color as other buttons in the on position.

name

A string that is used to identify the answer submitted by the worker. This value matches a key in the JSON object that specifies the answer.

required

A Boolean switch that, if present, requires the worker to provide input.

value

A value used in the output as the property name for the element's Boolean state. Defaults to "on" if not provided.

Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 204\)](#)
- **Child elements:** none

Output

This element outputs the name as the name of an object, containing the value as a property name and the element's state as Boolean value for the property. If no value for the element is specified, the property name defaults to "on."

Example Sample output for this element

```
[  
  {  
    "theToggler": {  
      "on": true  
    }  
  }  
]
```

See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth for Labeling \(p. 64\)](#)
- [HTML Elements Reference \(p. 189\)](#)

Augmented AI crowd-elements

The following HTML elements are only available for Amazon Augmented AI human workflow tasks.

Topics

- [crowd-textract-document-analysis \(p. 233\)](#)
- [crowd-rekognition-detect-moderation-labels \(p. 236\)](#)

crowd-textract-document-analysis

A widget to enable human review of a Amazon Textract document analysis result.

Attributes

The following attributes are supported by this element.

header

This is the text that is displayed as the header.

src

This is a link to the image to be analyzed by the worker.

initialValue

This sets initial values for attributes found in the worker UI.

The following is an example of an initialValue input:

```
[  
  {  
    "BlockType": "KEY_VALUE_SET",  
    "Confidence": 38.43309020996094,  
    "Geometry": {  
      "BoundingBox": {  
        "Width": 0.32613086700439453,  
        "Height": 0.0942094624042511,  
        "Left": 0.4833833575248718,  
        "Top": 0.5227988958358765  
      },  
      "Polygon": [  
        {"X": 0.123, "Y": 0.345}, ...  
      ]  
    }  
    "Id": "8c97b240-0969-4678-834a-646c95da9cf4",  
    "Relationships": [  
      {  
        "Type": "CHILD",  
        "Ids": [  
          "7ee7b7da-ee1b-428d-a567-55a3e3affa56",  
          "4d6da730-ba43-467c-a9a5-c6137ba0c472"  
        ]  
      },  
      {  
        "Type": "VALUE",  
        "Ids": [  
          "6ee7b7da-ee1b-428d-a567-55a3e3affa54"  
        ]  
      }  
    ],  
    "EntityTypes": [  
      "KEY"  
    ],  
    "Text": "Foo bar"
```

```
        },
    ]
```

blockTypes

This determines the kind of analysis the workers can do. Only `KEY_VALUE_SET` is currently supported.

keys

This specifies new keys and the associated text value the worker can add. The input values for `keys` can include the following elements:

- `importantFormKey` accepts strings, and is used to specify a single key.
- `importantFormKeyAliases` can be used to specify aliases that are acceptable alternatives to the keys supplied. Use this element to identify alternative spellings or presentations of your keys. This parameter accepts a list of one or more strings.

The following is an example of an input for `keys`.

```
[
  {
    importantFormKey: 'Address',
    importantFormKeyAliases: [
      'address',
      'Addr.',
      'Add.'
    ],
    {
      importantFormKey: 'Last name',
      importantFormKeyAliases: ['Surname']
    }
]
```

no-key-edit

This prevents the workers from editing the keys of annotations passed through `initialValue`. If you want to prevent workers from editing the keys that have been detected on your documents, you should include this attribute.

no-geometry-edit

This prevents workers from editing the polygons of annotations passed through `initialValue`. For example, this would prevent the worker from editing the bounding box around a given key. In most scenarios, you should include this attribute.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements – `crowd-form`
- Child elements – [full-instructions \(p. 235\)](#), [short-instructions \(p. 235\)](#)

AWS Regions

The following AWS Regions are supported by this element. You can use custom HTML and CSS code within these Regions to format your instructions to workers. For example, use the `short-instructions` section to provide good and bad examples of how to complete a task.

full-instructions

General instructions about how to work with the widget.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Example of a Worker Template Using the `crowd` Element

An example of a worker template using this `crowd` element would look like the following.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<% capture s3_arn %>http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.document.s3Object.bucket }}/
{{ task.input.aiServiceRequest.document.s3Object.name }}{%
endcapture %}

<crowd-form>
  <crowd-extract-analyze-document
    src="{{ s3_arn | grant_read_access }}"
    initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
    header="Review the key-value pairs listed on the right and correct them if they don't
match the following document."
    no-key-edit
    no-geometry-edit
    keys="{{ task.input.humanLoopContext.importantFormKeys }}"
    block-types="['KEY_VALUE_SET']"
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
        .instructionsImage {
          display: inline-block;
          max-width: 100%;
        }
      </style>
      <p class='instructions'>Click on a key-value block to highlight the corresponding
key-value pair in the document.

If it is a valid key-value pair, review the content for the value. If the content is
incorrect, correct it.

The text of the value is incorrect, correct it.


A wrong value is identified, correct it.


If it is not a valid key-value relationship, choose No.


If you can't find the key in the document, choose Key not found.


If the content of a field is empty, choose Value is blank.

```

```

<b>Examples</b>
Key and value are often displayed next or below to each other.

Key and value displayed in one line.


Key and value displayed in two lines.


If the content of the value has multiple lines, enter all the text without line break.
Include all value text even if it extends beyond the highlight box.
</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-extract-analyze-document>
</crowd-form>

```

Output

The following is a sample of the output from this element. You can find a detailed explanation of this output in the Amazon Textract [AnalyzeDocument](#) API documentation.

```
{
    "AWS/Textract/AnalyzeDocument/Forms/V1": {
        "blocks": [
            {
                "blockType": "KEY_VALUE_SET",
                "id": "8c97b240-0969-4678-834a-646c95da9cf4",
                "relationships": [
                    {
                        "type": "CHILD",
                        "ids": ["7ee7b7da-ee1b-428d-a567-55a3e3affa56", "4d6da730-ba43-467c-a9a5-c6137ba0c472"]
                    },
                    {
                        "type": "VALUE",
                        "ids": ["6ee7b7da-ee1b-428d-a567-55a3e3affa54"]
                    }
                ],
                "entityTypes": ["KEY"],
                "text": "Foo bar baz"
            }
        ]
    }
}
```

crowd-rekognition-detect-moderation-labels

A widget to enable human review of an Amazon Rekognition image moderation result.

Attributes

The following attributes are supported by this element.

header

This is the text that is displayed as the header.

src

This is a link to the image to be analyzed by the worker.

categories

This supports `categories` as an array of strings **or** an array of objects where each object has a `name` field.

If the categories come in as objects, the following applies:

- The displayed categories are the value of the `name` field.
- The returned answer contains the **full** objects of any selected categories.

If the categories come in as strings, the following applies:

- The returned answer is an array of all the strings that were selected.

exclusion-category

By setting this attribute you create a button underneath the categories in the UI.

- When a user chooses the button, all categories are deselected and disabled.
- Choosing the button again re-enables the categories so that users can choose them.
- If you submit after choosing the button, it returns an empty array.

Element Hierarchy

This element has the following parent and child elements.

- Parent elements – `crowd-form`
- Child elements – [full-instructions \(p. 237\)](#), [short-instructions \(p. 237\)](#)

AWS Regions

The following AWS Regions are supported by this element. You can use custom HTML and CSS code within these Regions to format your instructions to workers. For example, use the `short-instructions` section to provide good and bad examples of how to complete a task.

full-instructions

General instructions about how to work with the widget.

short-instructions

Important task-specific instructions that are displayed in a prominent place.

Example Worker Template with the crowd Element

An example of a worker template using the `crowd` element would look like the following.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_arn %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/
{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}
```

```

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
    categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
        {
          name: "{{ label.name }}",
          parentName: "{{ label.parentName }}",
        },
      {% endfor %}
    ]'
    src="{{ s3_arn | grant_read_access }}"
    header="Review the image and choose all applicable categories."
  >
  <short-instructions header="Instructions">
    <style>
      .instructions {
        white-space: pre-wrap;
      }
    </style>
    <p class='instructions'>Review the image and choose all applicable categories.  
If no categories apply, choose None.</p>
  </short-instructions>
  <b>Nudity</b>
  Visuals depicting nude male or female person or persons

  <b>Graphic Male Nudity</b>
  Visuals depicting full frontal male nudity, often close ups

  <b>Graphic Female Nudity</b>
  Visuals depicting full frontal female nudity, often close ups

  <b>Sexual Activity</b>
  Visuals depicting various types of explicit sexual activities and pornography

  <b>Illustrated Nudity or Sexual Activity</b>
  Visuals depicting animated or drawn sexual activity, nudity or pornography

  <b>Adult Toys</b>
  Visuals depicting adult toys, often in a marketing context

  <b>Female Swimwear or Underwear</b>
  Visuals depicting female person wearing only swimwear or underwear

  <b>Male Swimwear Or Underwear</b>
  Visuals depicting male person wearing only swimwear or underwear

  <b>Partial Nudity</b>
  Visuals depicting covered up nudity, for example using hands or pose

  <b>Revealing Clothes</b>
  Visuals depicting revealing clothes and poses, such as deep cut dresses

  <b>Graphic Violence or Gore</b>
  Visuals depicting prominent blood or bloody injuries

  <b>Physical Violence</b>
  Visuals depicting violent physical assault, such as kicking or punching

  <b>Weapon Violence</b>
  Visuals depicting violence using weapons like firearms or blades, such as shooting

  <b>Weapons</b>
  Visuals depicting weapons like firearms and blades

  <b>Self Injury</b>

```

```
Visuals depicting self-inflicted cutting on the body, typically in distinctive patterns
using sharp objects

<b>Emaciated Bodies</b>
Visuals depicting extremely malnourished human bodies

<b>Corpses</b>
Visuals depicting human dead bodies

<b>Hanging</b>
Visuals depicting death by hanging</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>
```

Output

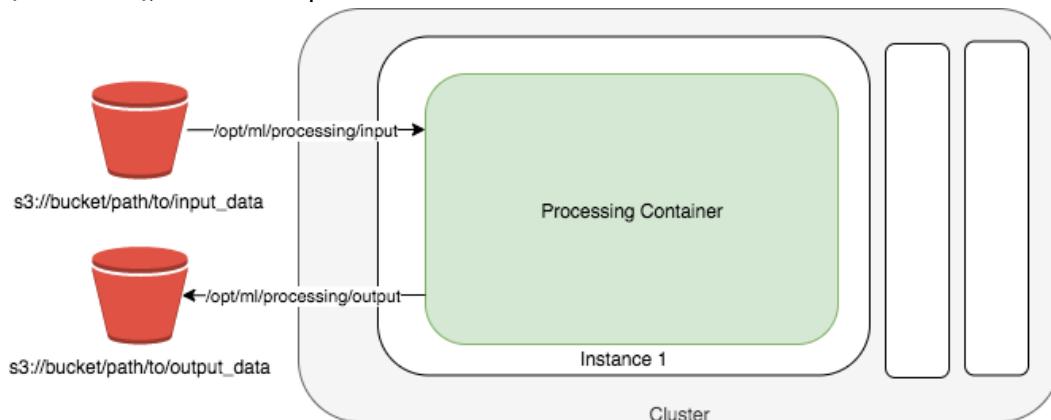
The following is a sample of the output from this element. For details about this output, see Amazon Rekognition [DetectModerationLabels](#) API documentation.

```
{
    "AWS/Rekognition/DetectModerationLabels/Image/V3": {
        "ModerationLabels": [
            { name: 'Gore', parentName: 'Violence' },
            { name: 'Corpses', parentName: 'Violence' },
        ]
    }
}
```

Process Data and Evaluate Models

Use Amazon SageMaker Processing to analyze data and evaluate models on the Amazon SageMaker machine learning platform. Amazon SageMaker is a fully managed service that covers the entire machine learning workflow, from preparing your data, to training and deploying the model to make predictions, and monitoring model performance when in production. Data processing tasks such as feature engineering, data validation, model evaluation, and model interpretation are essential steps performed by engineers and data scientists in this machine learning workflow. With Processing you can leverage a simplified, managed experience to run your data processing workloads on the Amazon SageMaker platform or by using the Amazon SageMaker APIs, in the experimentation phase and after code is deployed in production.

You can run an Amazon SageMaker Processing Job to process data from Amazon Simple Storage Service (Amazon S3), and save the processed data back to Amazon S3.



Topics

- [Amazon SageMaker Processing Sample Notebooks \(p. 240\)](#)
- [Monitor Amazon SageMaker Processing with CloudWatch Logs and Metrics \(p. 241\)](#)
- [Data Processing and Model Evaluation with Scikit-Learn \(p. 241\)](#)
- [Use Your Own Processing Code \(p. 241\)](#)

Amazon SageMaker Processing Sample Notebooks

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see [scikit-learn Processing](#). This notebook also shows how to bring your own container to run processing workloads with your own dependencies.

For a sample notebook that shows how to use Processing to do distributed data processing with Spark, see [Distributed Processing \(Spark\)](#).

For instructions on how to create and access Jupyter notebook instances that you can use to run the samples in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After you

have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, choose its **Use** tab and choose **Create copy**.

Monitor Amazon SageMaker Processing with CloudWatch Logs and Metrics

Amazon SageMaker Processing provides Amazon CloudWatch logs and metrics to monitor processing jobs. CloudWatch provides CPU, GPU, memory, GPU memory, and disk metrics and event logging. For more information, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#) and [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

Data Processing and Model Evaluation with Scikit-Learn

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see [scikit-learn Processing](#). This notebook runs a processing job using `SKLearnProcessor` to execute a scikit-learn script that you provide that preprocesses data, trains a model using an Amazon SageMaker training job, and then runs a processing job to evaluate the trained model to estimate how the model is expected to perform in production.

The following example shows how to use a `SKLearnProcessor` to run your own scikit-learn script using a Docker image provided and maintained by Amazon SageMaker, rather than your own Docker image.

```
from sagemaker.sklearn.processing import SKLearnProcessor
from sagemaker.processing import ProcessingInput, ProcessingOutput

sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                      role=role,
                                      instance_type='ml.m5.xlarge',
                                      instance_count=1)

sklearn_processor.run(code='preprocessing.py',
                      inputs=[ProcessingInput(
                          source='s3://path/to/my/input-data.csv',
                          destination='/opt/ml/processing/input')],
                      outputs=[ProcessingOutput(source='/opt/ml/processing/output/train'),
                               ProcessingOutput(source='/opt/ml/processing/output/
validation'),
                               ProcessingOutput(source='/opt/ml/processing/output/test')])
```

Use Your Own Processing Code

You can use your own container to process and install libraries to run your scripts or to build your own processing container. For a formal specification for building a Processing container contract is provided in the [Build Your Own Processing Container \(p. 242\)](#) topic for those who need the details.

Topics

- [Run Scripts with Your Own Processing Container \(p. 242\)](#)
- [Build Your Own Processing Container \(p. 242\)](#)

Run Scripts with Your Own Processing Container

For a sample notebook that shows how to run scikit-learn scripts to do data preprocessing and model evaluation with the Amazon SageMaker Python SDK for Processing, see the [scikit-learn Processing](#) sample.

The following example shows how to use a `ScriptProcessor` class from the Amazon SageMaker Python SDK to run a Python script with your own image to run a processing job that processes input data, and saves the processed data in Amazon S3.

The notebook shows the general workflow as follows. It includes the code from the steps that use the Amazon SageMaker Python SDK.

1. Write the Dockerfile with the dependencies, build it, and push it to an Amazon Elastic Container Registry (Amazon ECR) repository.
2. Write the `preprocessing.py` that contains the preprocessing code.
3. Set up the `ScriptProcessor` to run the script.

```
from sagemaker.processing import ScriptProcessor, ProcessingInput, ProcessingOutput

script_processor = ScriptProcessor(command=['python3'],
                                    image_uri='<image_uri>',
                                    role='<role_arn>',
                                    instance_count=1,
                                    instance_type='ml.m5.xlarge')
```

4. Run the script.

```
script_processor.run(code='preprocessing.py',
                      inputs=[ProcessingInput(
                          source='s3://path/to/my/input-data.csv',
                          destination='/opt/ml/processing/input')],
                      outputs=[ProcessingOutput(source='/opt/ml/processing/output/
train'),
                               ProcessingOutput(source='/opt/ml/processing/output/
validation'),
                               ProcessingOutput(source='/opt/ml/processing/output/test')])
```

The same procedure could be used with any other library or system dependencies. More generally, you can use existing Docker images on Amazon SageMaker Processing, including images that you run on other platforms, such as Kubernetes.

Build Your Own Processing Container

You can provide SageMaker Processing a Docker image to run your data processing, feature engineering, and model evaluation workloads. You can add your own dependencies to this Docker image, and SageMaker Processing will run this image with your own code and dependencies.

The following example of a Dockerfile builds a container with the Python libraries “scikit-learn” and “pandas” that you can run as a processing job.

```
FROM python:3.7-slim-buster

# Install scikit-learn and pandas
RUN pip3 install pandas==0.25.3 scikit-learn==0.21.3

# Add a python script and configure Docker to run it
```

```
ADD processing_script.py /  
ENTRYPOINT ["python3", "/processing_script.py"]
```

After building and pushing this Docker image to an Amazon Elastic Container Registry (Amazon ECR) repository and ensuring that your Amazon SageMaker IAM role can pull the image from Amazon ECR, you can run this image on Amazon SageMaker Processing.

How Amazon SageMaker Runs Your Processing Image

Amazon SageMaker Processing runs the container in a similar way as the following command, where `AppSpecification.ImageUri` is the Amazon ECR image URI that you specify in your `CreateProcessingJob` API.

```
docker run [AppSpecification.ImageUri]
```

This runs the entrypoint command configured in your Docker image.

You can also override the entrypoint command in the image or give command-line arguments to your entrypoint command using the `AppSpecification.ContainerEntrypoint` and `AppSpecification.ContainerArguments` parameters in your `CreateProcessingJob` request. Specifying these parameters configures Amazon SageMaker Processing to run the container in a similar way as the following command.

```
docker run --entry-point [AppSpecification.ContainerEntrypoint]  
[AppSpecification.ImageUri] [AppSpecification.ContainerArguments]
```

For example, if you specify the `ContainerEntrypoint` to be `["python3", "-v", "/processing_script.py"]` in your `CreateProcessingJob` request, and `ContainerArguments` to be `["--data-format", "csv"]`, Processing runs your container with the command `"python3 -v /processing_script.py --data-format csv"`.

Be aware of the following details when building your processing container:

- Processing decides whether the job completes or fails depending on the exit code of the command run. A processing job completes if all of the processing containers exit successfully with an exit code of 0, and fails if any of the containers exits with a non-zero exit code.
- Processing lets you override the processing container's entrypoint and set command-line arguments just like you can with the Docker API. Docker images can also configure the entrypoint and command-line arguments using the `ENTRYPOINT` and `CMD` instructions. The way `CreateProcessingJob`'s `ContainerEntrypoint` and `ContainerArgument` parameters configure a Docker image's entrypoint and arguments mirrors how Docker overrides the entrypoint and arguments through the Docker API:
 - If neither `ContainerEntrypoint` nor `ContainerArguments` are provided, the default `ENTRYPOINT` or `CMD` in the image is used.
 - If `ContainerEntrypoint` is provided, but not `ContainerArguments`, the image is run with the given entrypoint, and the `ENTRYPOINT` and `CMD` in the image are ignored.
 - If `ContainerArguments` is provided, but not `ContainerEntrypoint`, the image is run with the default `ENTRYPOINT` in the image with the arguments provided.
 - If both `ContainerEntrypoint` and `ContainerArguments` are provided, the image is run with the given entrypoint and arguments, and the `ENTRYPOINT` and `CMD` in the image are ignored.
 - Use the "exec" form of the `ENTRYPOINT` instruction in your Dockerfile (`ENTRYPOINT ["executable", "param1", "param2"]`) instead of the "shell" form (`ENTRYPOINT command param1 param2`). This lets your processing container receive `SIGINT` and `SIGKILL` signals, which Processing uses to stop processing jobs using the `StopProcessingJob` API.

- `/opt/ml` and all subdirectories are reserved by Amazon SageMaker. When building your processing Docker image, don't place any data required by your processing container in these directories.
- If you plan to use GPU devices, 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](#).

How Amazon SageMaker Processing Configures Input and Output For Your Processing Container

When you create a processing job using the `CreateProcessingJob` API, you can specify multiple `ProcessingInputs` and `ProcessingOutputs`. values

A `ProcessingInput` configures an Amazon S3 URI to download data from, and a path in your processing container to download the data to. A `ProcessingOutput` configures a path in your processing container to upload data from, and where in Amazon S3 to upload that data to. For both `ProcessingInput` and `ProcessingOutput`, the path in the processing container must begin with `/opt/ml/processing/`

For example, you might create a processing job with one `ProcessingInput` that downloads data from `s3://your-data-bucket/path/to/input/csv/data` into a path `/opt/ml/processing/csv` in your processing container, and a `ProcessingOutput` that uploads data from `/opt/ml/processing/processed_csv` to `s3://your-data-bucket/path/to/output/csv/data`. The code running in your processing code might read this data, and write output data to `/opt/ml/processing/processed_csv`. Processing uploads the data written into this path to the Amazon S3 output location.

How Amazon SageMaker Processing Provides Logs and Metrics for your Processing Container

When your processing container writes to `stdout` or `stderr`, Processing saves the output from each processing container and puts the output in Amazon CloudWatch logs. For information about logging, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

Processing also provides CloudWatch metrics for each instance running your processing container. For information about metrics, see the [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#).

How Amazon SageMaker Processing Provides Configuration to Your Processing Container

Amazon SageMaker Processing provides configuration to your processing container through environment variables and two JSON files at predefined locations in the processing container, `/opt/ml/config/processingjobconfig.json` and `/opt/ml/config/resourceconfig.json`.

Your processing job is started with the environment variables configured using the `Environment` map in the `CreateProcessingJob` API request. The file `/opt/ml/config/processingjobconfig.json` contains information from the `CreateProcessingJob` request.

The following example shows the format of the file.

```
{  
    "ProcessingJobArn": "<processing_job_arn>",  
    "ProcessingJobName": "<processing_job_name>",  
    "AppSpecification": {  
        "ImageUri": "<image_uri>",  
        "ContainerEntrypoint": null,  
        "ContainerArguments": null
```

```

},
"Environment": {
    "KEY": "VALUE"
},
"ProcessingInputs": [
    {
        "InputName": "input-1",
        "S3Input": {
            "LocalPath": "/opt/ml/processing/input/dataset",
            "S3Uri": "<s3_uri>",
            "S3DataDistributionType": "FullyReplicated",
            "S3DataType": "S3Prefix",
            "S3InputMode": "File",
            "S3CompressionType": "None",
            "S3DownloadMode": "StartOfJob"
        }
    }
],
"ProcessingOutputConfig": {
    "Outputs": [
        {
            "OutputName": "output-1",
            "S3Output": {
                "LocalPath": "/opt/ml/processing/output/dataset",
                "S3Uri": "<s3_uri>",
                "S3UploadMode": "EndOfJob"
            }
        }
    ],
    "KmsKeyId": null
},
"ProcessingResources": {
    "ClusterConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m5.xlarge",
        "VolumeSizeInGB": 30,
        "VolumeKmsKeyId": null
    }
},
"RoleArn": "<IAM role>",
"StoppingCondition": {
    "MaxRuntimeInSeconds": 86400
}
}
}

```

The `/opt/ml/config/resourceconfig.json` file contains information about the hostnames of your processing containers. Use these hostnames when creating or running distributed processing code.

```
{
    "current_host": "algo-1",
    "hosts": ["algo-1", "algo-2", "algo-3"]
}
```

Don't use the information in `/etc/hostname` or `/etc/hosts` because it might be inaccurate. Hostname information might not be immediately available to the processing container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

How You Can Save Metadata Information About Your Processing Job

Your processing containers can write to a UTF-8 to the file `/opt/ml/output/message` to communicate metadata from the processing container after the processing

job enters any terminal status ("Completed", "Stopped", or "Failed"), the "ExitMessage" field in [DescribeProcessingJob](#) will contain the first 1 KB of this file. The [DescribeProcessingJob](#) call returns up to 1 KB of data from your processing containers through the `ExitMessage` parameter. For example, for failed processing jobs, you can use this field to communicate why the processing container failed.

Do not write sensitive data to this file. If the data in this file is not UTF-8 encoded, the job will fail with a [ClientError](#). If multiple containers exit with "ExitMessage", the "ExitMessage" content from each processing container is concatenated, then truncated to 1 KB.

How You Can Run Your Processing Container Using the SageMaker Python SDK

You can use the Amazon SageMaker Python SDK to run your own processing image by using the `Processor` class. The following example shows how to run your own processing container with one input from Amazon Simple Storage Service (Amazon S3), and one output to Amazon S3.

```
from sagemaker.processing import Processor, ProcessingInput, ProcessingOutput

processor = Processor(image_uri='<your_ecr_image_uri>',
                      role=role,
                      instance_count=1,
                      instance_type="ml.m5.xlarge")

processor.run(inputs=[ProcessingInput(
    source='<s3_uri or local path>',
    destination='/opt/ml/processing/input_data')],
              outputs=[ProcessingOutput(
    source='/opt/ml/processing/processed_data',
    destination='<s3_uri>'),
              ])
```

Instead of building your processing code into your processing image, you can provide a `ScriptProcessor` with your own image and the command that you want to run, along with the code that you want to run inside that container, as in the [Run Scripts with Your Own Processing Container \(p. 242\)](#) example.

You can also use the scikit-learn image that Processing provides through `SKLearnProcessor` to run scikit-learn scripts, as in [Data Processing and Model Evaluation with Scikit-Learn \(p. 241\)](#). To learn more about using the Amazon SageMaker Python SDK with Processing containers, see [SageMaker Python SDK ReadTheDocs](#).

Build Models

To build machine learning models in Amazon SageMaker, you have the following options:

- Use one of the built-in algorithms. Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types. For more information, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).
- Write a custom training script in a machine learning framework that Amazon SageMaker supports, and use one of the pre-built framework containers to run it in Amazon SageMaker. For information, see [Use Machine Learning Frameworks with Amazon SageMaker \(p. 466\)](#).
- Bring your own algorithm or model to train or host in Amazon SageMaker. For information, see [Use Your Own Algorithms or Models with Amazon SageMaker \(p. 478\)](#).
- Use an algorithm that you subscribe to from AWS Marketplace. For information, see [Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace \(p. 519\)](#).

Topics

- [Use Amazon SageMaker Notebooks \(p. 247\)](#)
- [Use Amazon SageMaker Notebook Instances \(p. 252\)](#)
- [Choose an Algorithm \(p. 272\)](#)
- [Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace \(p. 519\)](#)

Use Amazon SageMaker Notebooks

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Amazon SageMaker Studio notebooks are collaborative notebooks that are built into Amazon SageMaker Studio that you can launch quickly. You can access your notebooks without setting up compute instances and file storage so you can get started fast. You pay only for the resources consumed when you run the notebooks.

After your organization is set up with Amazon SageMaker Studio, your organization's data scientists, developers, and other SageMaker users can start working on notebooks within seconds without having to provision any compute resources. Each of your users is automatically assigned their own instance to execute the code of the notebooks, but they can also have multiple instances for different notebooks. Amazon SageMaker Studio notebooks provides persistent storage for your users' notebooks, which enables them to view and share notebooks even if the instances are shut down.

You can share your notebooks with others in your organization, so that they can easily reproduce your results and collaborate while building models and exploring your data. Dependencies for your notebook are included in the notebooks' environment settings, so sharing is seamless. Sharing publishes a reproducible snapshot of your notebook environments through secure sharable URLs.

To get started, you or your organization's administrator need to complete the Amazon SageMaker Studio setup for the team. There are two modes for authentication: AWS Single Sign-On (AWS SSO) and AWS Identity and Access Management (IAM). When choosing the appropriate method for your team, you must carefully consider the advantages of each method. Switching to another authentication method after you set up the domain requires a manual migration of the team's notebooks so you want to make sure you choose the appropriate method for your organization at the start. For more information, see the [setup documentation](#).

Note

Because Amazon SageMaker Studio Notebooks is in preview, visual elements of Amazon SageMaker Studio might be impacted.

How Are These Notebooks Different from Instance-based Amazon SageMaker Notebooks?

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

When starting a new notebook, we recommend that you use the notebook feature in Amazon SageMaker Studio instead of launching a notebook instance from the console. There are many benefits to using an Amazon SageMaker Studio notebook:

- Starting an Amazon SageMaker Studio notebook is faster than launching an instance-based notebook. Typically, it is 5-10 times faster than instance-based notebooks.
- Notebook sharing is an integrated feature in Amazon SageMaker Studio. Users can generate a shareable link that reproduces not only the notebook code, but also the environment required to execute it, in just a few clicks. You can also share the environment that hosts.
- Amazon SageMaker Studio notebooks come pre-installed with the latest Amazon SageMaker SDK and can be accessed within the studio's IDE, allowing you to build, train, debug, track, and monitor your models.
- As a member of an Amazon SageMaker Studio team, you get a home directory independent of a particular instance. This directory is automatically mounted into all notebook servers and kernels as they're started, so you always have your notebooks and other files. The home directories are in your account in EFS so that you can access them from other services.
- When using AWS SSO, you use your AWS domain credentials and have a unique URL for your team. You never have to interact with the AWS Management Console to run your notebooks.
- Amazon SageMaker Studio Notebooks are equipped with a set of predefined environments to get your organization started on data science projects faster.

When you open a notebook in Amazon SageMaker Studio, the default view is a JupyterLab interface. The primary features are the same, however, so what you can expect to find the typical features of a Jupyter notebook and JupyterLab, you will find here as well.

If you're unfamiliar with JupyterLab features, it is recommended that you take a tour of the [JupyterLab user interface features in the JupyterLab documentation](#).

Get Started

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

To get started with Amazon SageMaker Studio notebooks you need to set up AWS SSO if you haven't already. You don't have to be in an organization to use AWS SSO. You can sign up as an individual or as an enterprise. The process is similar except enterprises have the option of batch loading their users from their existing AWS SSO organization.

If your organization has already set up a team and you were invited to join the organization or view a specific notebook, you can log in and use the Amazon SageMaker Studio dashboard. From there you can try a tutorial that walks you through all of the features of Amazon SageMaker Studio, including how to use Amazon SageMaker Studio notebooks.

Login

Logging in to an Amazon SageMaker notebook is the same as logging in to Amazon SageMaker Studio. When you receive a shared notebook or if you want to create a new notebook, or run experiments, it is all the same process. Use the console or the link provided to you in an invite email to log in to Amazon SageMaker Studio.

To log in from the console

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Amazon SageMaker Studio**.
3. You should see an Amazon SageMaker Studio landing page.
4. If your admin has completed the setup, you will see two sections:
 - **Summary** section that shows your team's domain URL and the status of the domain.
 - In the **Profile** section under the **Summary**, you can access notebooks and manage profiles.

Choose your profile name, and then choose **Open Amazon SageMaker Studio**. In the JupyterLab UI, you can open an existing notebook or create notebooks.

If you can't log in or you didn't receive an invitation in email, you must contact your organization's administrator. For password reset and account recovery instructions, see the [SSO guide](#).

Create a Notebook

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

After you're logged in to Amazon SageMaker Studio, you can create a notebook in the following ways:

- Using the JupyterLab launcher.
- On the **File** menu, choose **New**, and then choose **Notebooks**.

Environment Options

Depending on your goals and processing requirements, SageMaker has a collection of pre-built environments.

- Data Science (includes most common data science packages such as NumPy, SciKit Learn, and more)
- Base Python (a plain, vanilla Python environment)
- MXNet CPU optimized
- TensorFlow CPU optimized
- PyTorch CPU optimized

Use the File Menu to Create a New Notebook

When you create a new notebook from the File menu, you will be asked to choose a kernel from a dropdown first, before your notebook is created.

Use the Launcher to Create a New Notebook

Selecting an Environment

When you create a notebook, first you must select the environment. Environments are the combination of software, such as frameworks, and other dependencies that your notebook needs to run. For example, an environment could be TensorFlow with Python 3, Keras with an MXNet backend, or PyTorch and Python 3 customized with additional Python packages.

When you're logged in as an Amazon SageMaker Studio user, you can have one instance of each instance type. Each instance can have only one environment running on it at a time. However, an environment can run multiple kernels or terminal instances.

Selecting a Kernel

After selecting an environment, you need to select an kernel. The options are a Python 3 kernel or console, or you can launch a Terminal.

After you choose the kernel, your new notebook launches and opens in a JupyterLab interface. Your notebook will be hosted in a t3.medium instance by default.

Run and Manage Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

Topics

- [Change a Notebook's Environment \(p. 250\)](#)

Change a Notebook's Environment

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

You can change the notebook's kernel in the same way that you change the kernel for any standard Jupyter notebook.

To change a notebook's environment

- In the top-right corner of the JupyterLab UI, choose the kernel name.
- Choose from the drop-down list the environment that fits your need. As described previously, you are choosing a kernel that is part of an environment. For example, if you switch from Python 3 (Data Science) to Python 3 (TensorFlow CPU Optimized), you load your notebook in the TensorFlow CPU Optimized environment with a Python 3 kernel.

Share Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

You can share any of your notebooks with others in your organization. It is a copy, so any changes you make to your notebook aren't reflected in a previous version that you shared. If you want to share your latest version, you must create a new snapshot and then share it.

To share a notebook

On the menu directly above your notebook, choose the camera icon to create a shareable screenshot.

Choose any of the following options:

- **Include github information** - If available, the GitHub repository associated with the notebook. This enables you and your recipient to collaborate and contribute to the same git repository.
- **Include outputs** - You can share a clean notebook, or include the output of your notebook's last run. Check this if you want to share the last run result, especially if the notebook takes long to run.

Note

If you don't see some or any of these options, your administrator probably disabled the feature. Contact your administrator.

After selecting your sharing options, you are provided with a URL. You can share this link with users that have access to Studio in your organization. When the user opens the URL, they're prompted to log in using AWS SSO or IAM. This shared notebook becomes a copy, so changes made by the recipient will not occur in your original notebook.

Use a Shared Notebook

You use a shared notebook in the same way you would any notebook that you created yourself. When you click a link to a shared notebook for the first time you will open a read-only version of the notebook. To edit, run and save the shared notebook, choose **Create a Copy** to copy the shared notebook to your personal storage. The notebook launches with the instance and built-in environment that the sender was using when they shared it. Customization to the environment cannot be shared at this moment. You can also inspect the notebook snapshot by choosing **Snapshot Details**.

Important considerations about sharing and authentication:

If you have an active session, you see a read-only view of the notebook until you choose **Create a Copy**.

If you don't have an active session, you need to log in.

- If you use IAM to login, after you login, select your user profile then choose **Open SageMaker Studio**. Then you need to choose the link you were sent.
- If you use SSO to login, after you login the shared notebook is opened automatically in JupyterLab.

Notebooks are opened based on the sender's instance type and environments (those are passed as metadata to the notebooks). A new instance of the same type is launched for you when you create a copy of the notebook.

Use Sample Notebooks

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

For sample notebooks using Amazon SageMaker, see the [amazon-sagemaker-examples](#) repository. The provided Amazon SageMaker Studio examples come with the environment settings needed to reproduce the execution.

View Personal Storage

Amazon SageMaker Studio Notebooks is in preview release and is subject to change.

You can browse the contents of your personal storage by looking at the File Browser in JupyterLab.

Manage Your Storage Volume

When you set up Amazon SageMaker Studio, an Amazon Elastic File System (Amazon EFS) file system is created in your AWS account. This volume contains all of the home directories for all of the users in your AWS SSO domain. This is where notebook files and data files are stored. Users can't directly access each other's home directories. Don't delete the Amazon EFS file system. If you do, your AWS SSO domain is longer functional, and all of your users will lose their work.

Use Amazon SageMaker Notebook Instances

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

Amazon SageMaker also provides sample notebooks that contain complete code walkthroughs. These walkthroughs show how to use Amazon SageMaker to perform common machine learning tasks. For more information, see [Use Example Notebooks \(p. 260\)](#).

Topics

- [Create a Notebook Instance \(p. 252\)](#)
- [Access Notebook Instances \(p. 255\)](#)
- [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#)
- [Control an Amazon EMR Spark Instance Using a Notebook \(p. 258\)](#)
- [Use Example Notebooks \(p. 260\)](#)
- [Notebook Instance Software Updates \(p. 261\)](#)
- [Set the Notebook Kernel \(p. 262\)](#)
- [Install External Libraries and Kernels in Notebook Instances \(p. 262\)](#)
- [Associate Git Repositories with Amazon SageMaker Notebook Instances \(p. 264\)](#)
- [Get Notebook Instance Metadata \(p. 271\)](#)
- [Monitor Jupyter Logs in Amazon CloudWatch Logs \(p. 271\)](#)

Create 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. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to Amazon SageMaker hosting, and test or validate your models.

To create a notebook instance, use either the Amazon SageMaker console or the [CreateNotebookInstance API](#).

The notebook instance type you choose depends on how you use your notebook instance. You want to ensure that your notebook instance is not bound by memory, CPU, or IO. If you plan to load a dataset into memory on the notebook instance for exploration or preprocessing, we recommend that you choose an instance type with enough RAM memory for your dataset. This would require an instance with at least 16 Gb of memory (.xlarge or larger). If you plan to use the notebook for compute intensive preprocessing, we recommend you choose a compute-optimized instance such as a c4 or c5.

A best practice when using an Amazon SageMaker notebook is to use the notebook instance to orchestrate other AWS services. For example, you can use the notebook instance to manage large dataset

processing by making calls to AWS Glue for ETL (extract, transform, and load) services or Amazon EMR for mapping and data reduction using Hadoop. You can use AWS services as temporary forms of computation or storage for your data.

You can store and retrieve your training and test data using an Amazon S3 bucket. You can then use Amazon SageMaker to train and build your model, so the instance type of your notebook would have no bearing on the speed of your model training and testing.

After receiving the request, Amazon SageMaker does the following:

- **Creates a network interface**—If you choose the optional VPC configuration, Amazon SageMaker 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 [Connect a Notebook Instance to Resources in a VPC \(p. 798\)](#).
- **Launches an ML compute instance**—Amazon SageMaker launches an ML compute instance in an Amazon SageMaker VPC. Amazon SageMaker performs the configuration tasks that allow it to manage your notebook instance, and if you specified your VPC, it enables traffic between your VPC and the notebook instance.
- **Installs Anaconda packages and libraries for common deep learning platforms**—Amazon SageMaker installs all of the Anaconda packages that are included in the installer. For more information, see [Anaconda package list](#). In addition, Amazon SageMaker installs the TensorFlow and Apache MXNet deep learning libraries.
- **Attaches an ML storage volume**—Amazon SageMaker attaches an ML storage volume to the ML compute instance. You can use the volume as a working area to clean up the training dataset or to temporarily store validation, test, or other data. Choose any size between 5 GB and 16384 GB, in 1 GB increments, for the volume. The default is 5 GB. ML storage volumes are encrypted, so Amazon SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

Only files and data saved within the `/home/ec2-user/SageMaker` folder persist between notebook instance sessions. Files and data that are saved outside this directory are overwritten when the notebook instance stops and restarts. Each notebook instance's `/tmp` directory provides a minimum of 10 GB of storage in an instant store. An instance store is temporary, block-level storage that isn't persistent. When the instance is stopped or restarted, Amazon SageMaker deletes the directory's contents. This temporary storage is part of the root volume of the notebook instance.

- **Copies example Jupyter notebooks**— These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

To create a Amazon SageMaker notebook instance:

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information:
 - a. For **Notebook instance name**, type a name for your notebook instance.
 - b. For **Notebook instance type**, choose an instance type for your notebook instance. For a list of supported instance types, see [Amazon SageMaker Limits](#).
 - c. For **Elastic Inference**, choose an inference accelerator type to associate with the notebook instance if you plan to conduct inferences from the notebook instance, or choose **none**. For information about elastic inference, see [Use Amazon SageMaker Elastic Inference \(EI\) \(p. 702\)](#).
 - d. (Optional) **Additional configuration** lets advanced users create a shell script that can run when you create or start the instance. This script, called a lifecycle configuration script, can be used

to set the environment for the notebook or to perform other functions. For information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).

- e. (Optional) **Additional configuration** also lets you specify the size, in GB, of the ML storage volume that is attached to the notebook instance. You can choose a size between 5 GB and 16,384 GB, in 1 GB increments. You can use the volume to clean up the training dataset or to temporarily store validation or other data.
- f. For **IAM role**, choose either an existing IAM role in your account that has the necessary permissions to access Amazon SageMaker resources or choose **Create a new role**. If you choose **Create a new role**, Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmSS`. The AWS managed policy `AmazonSageMakerFullAccess` is attached to the role. The role provides permissions that allow the notebook instance to call Amazon SageMaker and Amazon S3.
- g. For **Root access**, to enable root access for all notebook instance users, choose **Enable**. To disable root access for users, choose **Disable**. If you enable root access, all notebook instance users have administrator privileges and can access and edit all files on it.
- h. (Optional) **Encryption key** lets you encrypt data on the ML storage volume attached to the notebook instance using an AWS Key Management Service (AWS KMS) key. If you plan to store sensitive information on the ML storage volume, consider encrypting the information.
- i. (Optional) **Network** lets you put your notebook instance inside a Virtual Private Cloud (VPC). A VPC provides additional security and restricts access to resources in the VPC from sources outside the VPC. For more information on VPCs, see [Amazon VPC User Guide](#).

To add your notebook instance to a VPC:

- i. Choose the **VPC** and a **SubnetId**.
- ii. For **Security Group**, choose your VPC's default security group.
- iii. If you need your notebook instance to have internet access, enable direct internet access. For **Direct internet access**, choose **Enable**. Internet access can make your notebook instance less secure. For more information, see [Connect a Notebook Instance to Resources in a VPC \(p. 798\)](#).
- j. (Optional) To associate Git repositories with the notebook instance, choose a default repository and up to three additional repositories. For more information, see [Associate Git Repositories with Amazon SageMaker Notebook Instances \(p. 264\)](#).
- k. Choose **Create notebook instance**.

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

4. When the status of the notebook instance is `InService`, in the console, the notebook instance is ready to use. Choose **Open Jupyter** next to the notebook name to open the classic Jupyter dashboard.

You can choose **Open JupyterLab** to open the JupyterLab dashboard. The dashboard provides access to your notebook instance and sample Amazon SageMaker notebooks that contain complete code walkthroughs. These walkthroughs show how to use Amazon SageMaker to perform common machine learning tasks. For more information, see [Use Example Notebooks \(p. 260\)](#). For more information, see [Access Notebook Instances \(p. 255\)](#).

For more information about Jupyter notebooks, see [The Jupyter notebook](#).

Access Notebook Instances

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

- Use the console.

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

The screenshot shows the 'Notebook instances' section of the Amazon SageMaker console. At the top, there's a breadcrumb navigation: 'Amazon SageMaker > Notebook instances'. Below that is a search bar labeled 'Search notebook instances'. A table follows, with columns: 'Name', 'Instance', 'Creation time', 'Status', and 'Actions'. There is one entry named 'test' with details: 'ml.t2.medium', 'Nov 16, 2018 22:35 UTC', 'InService' status, and 'Open Jupyter' action link.

Name	Instance	Creation time	Status	Actions
test	ml.t2.medium	Nov 16, 2018 22:35 UTC	InService	Open Jupyter

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

Note

The URL that you get from a call to

[CreatePresignedNotebookInstanceUrl](#) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the AWS Management Console sign-in page.

- Use the API.

To get the URL for the notebook instance, call the

[CreatePresignedNotebookInstanceUrl](#) API and use the URL that the API returns to open the notebook instance.

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

Control Root Access to a Notebook Instance

By default, when you create a notebook instance, users that log into that notebook instance have root access. Data science is an iterative process that might require the data scientist to test and use different software tools and packages, so many notebook instance users need to have root access to be able to install these tools and packages. Because users with root access have administrator privileges, users can access and edit all files on a notebook instance with root access enabled.

If you don't want users to have root access to a notebook instance, when you call

[CreateNotebookInstance](#) or

[UpdateNotebookInstance](#) operations, set the `RootAccess` field to `Disabled`. You can also disable root access for users when you create or update a notebook instance in the Amazon SageMaker console. For information, see [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#).

Note

Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

Customize a Notebook Instance Using a Lifecycle Configuration Script

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

You can also use a lifecycle configuration script to access AWS services from your notebook. For example, you can create a script that lets you use your notebook to control other AWS resources, such as an Amazon EMR instance.

We maintain a public repository of notebook lifecycle configuration scripts that address common use cases for customizing notebook instances at <https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-configuration-samples>.

Note

Each script has a limit of 16384 characters.

The value of the \$PATH environment variable that is available to both scripts is /usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin. The working directory, which is the value of the \$PWD environment variable, is /.

View CloudWatch Logs for notebook instance lifecycle configurations in log group /aws/sagemaker/NotebookInstances in log stream [notebook-instance-name]/[LifecycleConfigHook].

Scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started. To help decrease the run time of scripts, try the following:

- Cut down on necessary steps. For example, limit which conda environments in which to install large packages.
- Run tasks in parallel processes.
- Use the nohup command in your script.

To create a lifecycle configuration

1. For **Lifecycle configuration - Optional**, choose **Create a new lifecycle configuration**.
2. For **Name**, type a name using alphanumeric characters and "-", but no spaces. The name can have a maximum of 63 characters.
3. (Optional) To create a script that runs when you create the notebook and every time you start it, choose **Start notebook**.
4. In the **Start notebook** editor, type the script.
5. (Optional) To create a script that runs only once, when you create the notebook, choose **Create notebook**.
6. In the **Create notebook** editor, type the script configure networking.
7. Choose **Create configuration**.

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

Lifecycle Configuration Best Practices

The following are best practices for using lifecycle configurations:

- Lifecycle configurations run as the `root` user. If your script makes any changes within the `/home/ec2-user/SageMaker` directory, (for example, installing a package with `pip`), use the command `sudo -u ec2-user` to run as the `ec2-user` user. This is the same user that Amazon SageMaker runs as.
- Amazon SageMaker notebook instances use `conda` environments to implement different kernels for Jupyter notebooks. If you want to install packages that are available to one or more notebook kernels, enclose the commands to install the packages with `conda` environment commands that activate the `conda` environment that contains the kernel where you want to install the packages.

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

```
#!/bin/bash
sudo -u ec2-user -i <<'EOF'

# This will affect only the Jupyter kernel called "conda_python3".
source activate python3

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

source deactivate

EOF
```

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

```
#!/bin/bash
sudo -u ec2-user -i <<'EOF'

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

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

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

    source /home/ec2-user/anaconda3/bin/deactivate
done

EOF
```

- You must store all `conda` environments in the default environments folder (`/home/user/anaconda3/envs`).

Important

When you create or change a script, we recommend that you use a text editor that provides Unix-style line breaks, such as the text editor available in the console when you create a notebook. Copying text from a non-Linux operating system might introduce incompatible line breaks and result in an unexpected error.

Control an Amazon EMR Spark Instance Using a Notebook

You can use a notebook instance created with a custom lifecycle configuration script to access AWS services from your notebook. For example, you can create a script that lets you use your notebook with Sparkmagic to control other AWS resources, such as an Amazon EMR instance. You can then use the Amazon EMR instance to process your data instead of running the data analysis on your notebook. This allows you to create a smaller notebook instance because you won't use the instance to process data. This is helpful when you have large datasets that would require a large notebook instance to process the data.

The process requires three procedures using the Amazon SageMaker console:

- Create the Amazon EMR Spark instance
- Create the Jupyter Notebook
- Test the notebook-to-Amazon EMR connection

To create an Amazon EMR Spark instance that can be controlled from a notebook using Sparkmagic

1. Open the Amazon EMR console at <https://console.aws.amazon.com/elasticmapreduce/>.
2. In the navigation pane, choose **Create cluster**.
3. On the **Create Cluster - Quick Options** page, under **Software configuration**, choose **Spark: Spark 2.4.4 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.2**.
4. Set additional parameters on the page and then choose **Create cluster**.
5. On the **Cluster** page, choose the cluster name that you created. Note the **Master Public DNS**, the **EMR master's security group**, and the VPC name and subnet ID where the EMR cluster was created. You will use these values when you create a notebook.

To create a notebook that uses Sparkmagic to control an Amazon EMR Spark instance

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, under **Notebook instances**, choose **Create notebook**.
3. Enter the notebook instance name and choose the instance type.
4. Choose **Additional configuration**, then, under **Lifecycle configuration**, choose **Create a new lifecycle configuration**.
5. Add the following code to the lifecycle configuration script:

```
# OVERVIEW
# This script connects an Amazon EMR cluster to an Amazon SageMaker notebook instance
# that uses Sparkmagic.
#
# Note that this script will fail if the Amazon EMR cluster's master node IP address is
# not reachable.
#   1. Ensure that the EMR master node IP is resolvable from the notebook instance.
```

```

#      One way to accomplish this is to have the notebook instance and the Amazon EMR
#      cluster in the same subnet.
#  2. Ensure the EMR master node security group provides inbound access from the
#      notebook instance security group.
#      Type          - Protocol - Port - Source
#      Custom TCP   - TCP       - 8998   - $NOTEBOOK_SECURITY_GROUP
#  3. Ensure the notebook instance has internet connectivity to fetch the SparkMagic
#      example config.
#
# https://aws.amazon.com/blogs/machine-learning/build-amazon-sagemaker-notebooks-
# backed-by-spark-in-amazon-emr/
#
# PARAMETERS
EMR_MASTER_IP=your.emr.master.ip

cd /home/ec2-user/.sparkmagic

echo "Fetching Sparkmagic example config from GitHub..."
wget https://raw.githubusercontent.com/jupyter-incubator/sparkmagic/master/sparkmagic/
example_config.json

echo "Replacing EMR master node IP in Sparkmagic config..."
sed -i -- "s/localhost/$EMR_MASTER_IP/g" example_config.json
mv example_config.json config.json

echo "Sending a sample request to Livy.."
curl "$EMR_MASTER_IP:8998/sessions"

```

6. In the **PARAMETERS** section of the script, replace `your.emr.master.ip` with the Master Public DNS name for the Amazon EMR instance.
7. Choose **Create configuration**.
8. On the **Create notebook** page, choose **Network - optional**.
9. Choose the VPC and subnet where the Amazon EMR instance is located.
10. Choose the security group used by the Amazon EMR master node.
11. Choose **Create notebook instance**.

While the notebook instance is being created, the status is **Pending**. After the instance has been created and the lifecycle configuration script has successfully run, the status is **InService**.

Note

If the notebook instance can't connect to the Amazon EMR instance, Amazon SageMaker can't create the notebook instance. The connection can fail if the Amazon EMR instance and notebook are not in the same VPC and subnet, if the Amazon EMR master security group is not used by the notebook, or if the Master Public DNS name in the script is incorrect.

To test the connection between the Amazon EMR instance and the notebook

1. When the status of the notebook is **InService**, choose **Open Jupyter** to open the notebook.
2. Choose **New**, then choose **Sparkmagic (PySpark)**.
3. In the code cell, enter `%%info` and then run the cell.

The output should be similar to the following

```

Current session configs: {'driverMemory': '1000M', 'executorCores': 2, 'kind':
'pyspark'}
No active sessions.

```

Use Example Notebooks

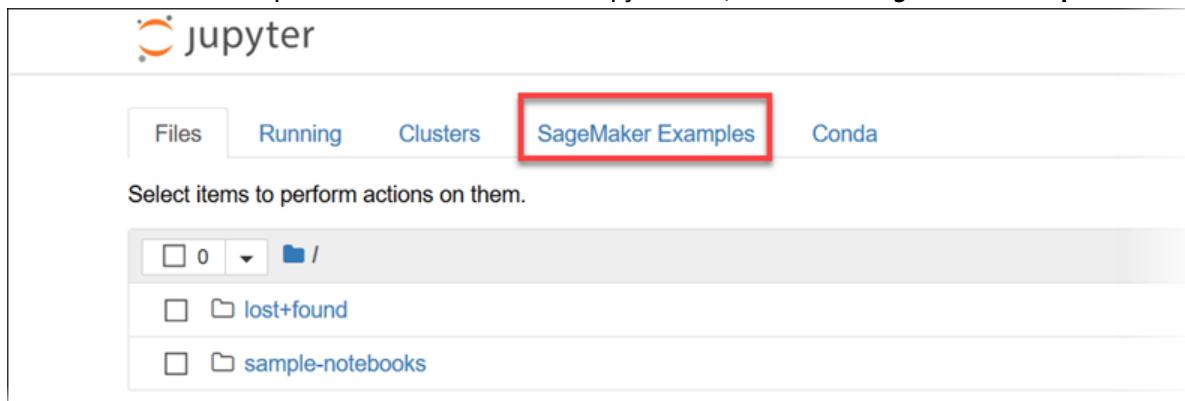
Your notebook instance contains example notebooks provided by Amazon SageMaker. The example notebooks contain code that shows how to apply machine learning solutions by using Amazon SageMaker. Notebook instances use the `nbexamples` Jupyter extension, which enables you to view a read-only version of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

Note

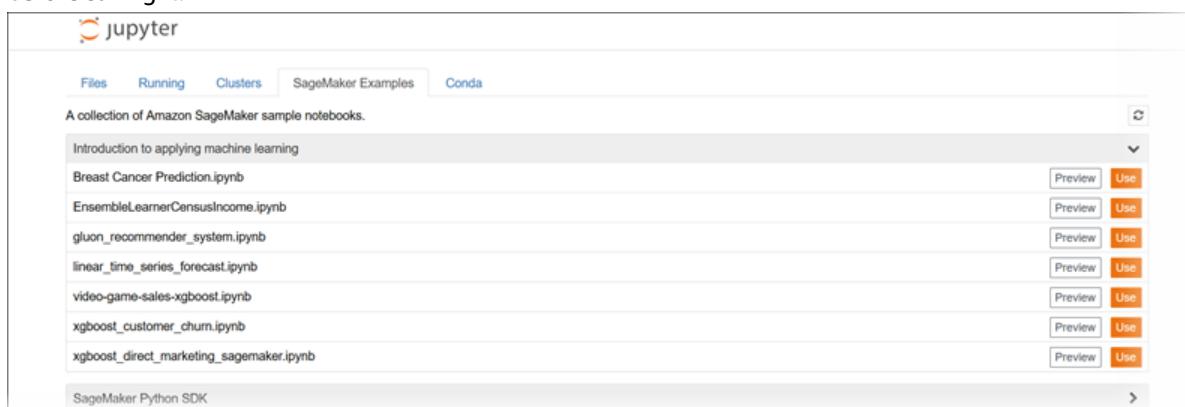
Example notebooks typically download datasets from the internet. If you disable Amazon SageMaker-provided internet access when you create your notebook instance, example notebooks might not work. For more information, see [Connect a Notebook Instance to Resources in a VPC \(p. 798\)](#).

Use or View Example Notebooks in Jupyter Classic

To view or use the example notebooks in the classic Jupyter view, choose the **SageMaker Examples** tab.

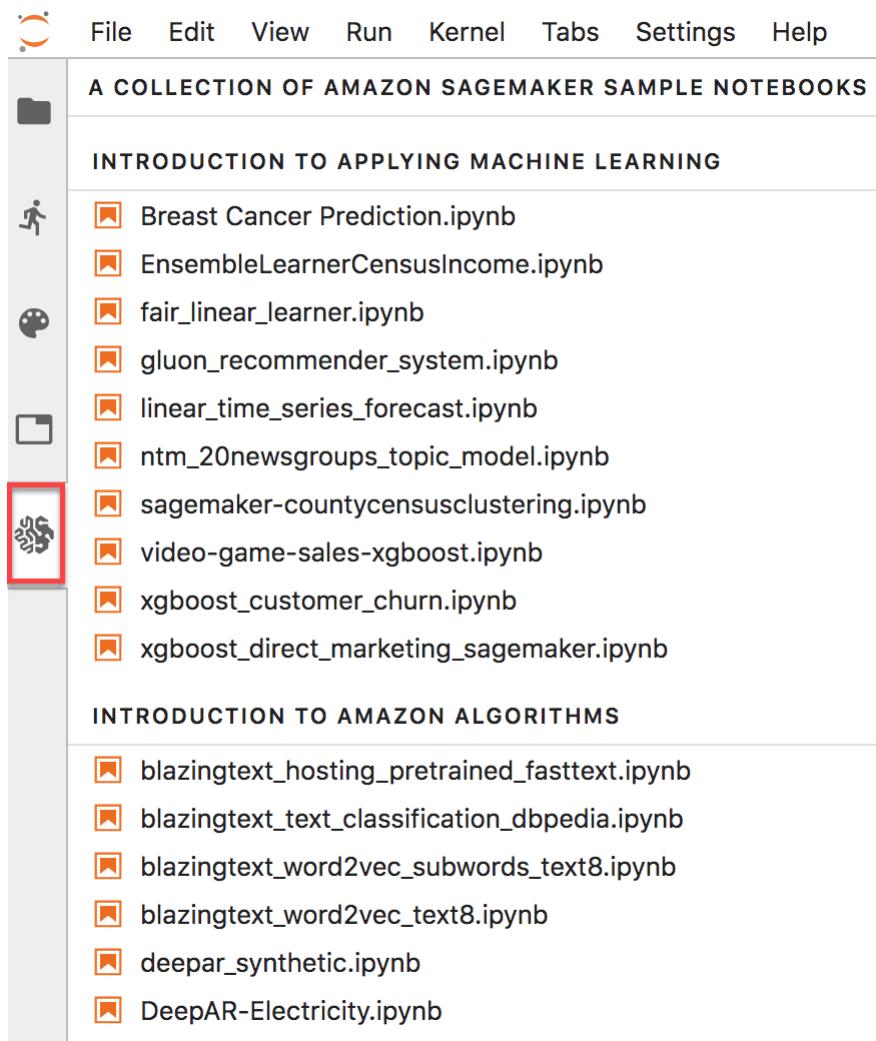


To view a read-only version of an example notebook in the Jupyter classic view, on the **SageMaker Examples** tab, choose **Preview** for that notebook. To create a copy of an example notebook in the home directory of your notebook instance, choose **Use**. In the dialog box, you can change the notebook's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



Use or View Example Notebooks in Jupyterlab

To view or use the example notebooks in the Jupyterlab view, choose the examples icon in the left navigation panel.



To view a read-only version of an example notebook, choose the name of the notebook. This opens the notebook as a tab in the main area. To create a copy of an example notebook in the home directory of your notebook instance, choose **Create a Copy** in the top banner. In the dialog box, type a name for the notebook and then choose **CREATE COPY**.

For more information about the example notebooks, see the [Amazon SageMaker examples GitHub repository](#).

Notebook Instance Software Updates

Amazon SageMaker periodically tests and releases software that is installed on notebook instances. This includes:

- Kernel updates
- Security patches
- AWS SDK updates
- Amazon SageMaker Python SDK updates
- Open source software updates

Amazon SageMaker does not automatically update software on a notebook instance when it is in service. To ensure that you have the most recent software updates, stop and restart your notebook instance, either in the Amazon SageMaker console or by calling [StopNotebookInstance](#).

You can also manually update software installed on your notebook instance while it is running by using update commands in a terminal or in a notebook.

Note

Updating kernels and some packages might depend on whether root access is enabled for the notebook instance. For more information, see [Control Root Access to a Notebook Instance \(p. 255\)](#).

Notebook instances do not notify you if you are running outdated software. You can check the [Personal Health Dashboard](#) or the security bulletin at <https://aws.amazon.com/security/security-bulletins/> for updates.

Set the Notebook Kernel

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



Install External Libraries and Kernels in Notebook Instances

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

The different Jupyter kernels in Amazon SageMaker notebook instances are separate conda environments. For information about conda environments, see [Managing environments](#) in the *Conda* documentation. If you want to use an external library in a specific kernel, install the library in the environment for that kernel. You can do this either in the terminal or in a notebook cell. The following procedures show how to install Theano so that you can use it in a notebook with a `conda_mxnet_p36` kernel.

To install Theano from a terminal

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

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

To install Theano from a Jupyter notebook cell

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

```
!pip install theano
```

Maintain a Sandboxed Python Environment

Amazon SageMaker periodically updates the Python and dependency versions in the environments installed on a notebook instance when it is stopped and restarted. For more information, see [Notebook Instance Software Updates \(p. 261\)](#). To maintain an isolated Python environment that does not change versions, create a lifecycle configuration that runs each time you start your notebook instance. For information about creating lifecycle configurations, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).

The following example lifecycle configuration script installs Miniconda on your notebook instance. This allows you to create environments in your notebook instance with specific versions of Python and dependencies that Amazon SageMaker does not update:

```
#!/bin/bash

set -e

WORKING_DIR=/home/ec2-user/.myproject
mkdir -p "$WORKING_DIR"

# Install Miniconda to get a separate python and pip
wget https://repo.anaconda.com/miniconda/Miniconda3-4.5.12-Linux-x86_64.sh -O
"$WORKING_DIR/miniconda.sh"

# Install Miniconda into the working directory
bash "$WORKING_DIR/miniconda.sh" -b -u -p "$WORKING_DIR/miniconda"

# Install pinned versions of any dependencies
source "$WORKING_DIR/miniconda/bin/activate"
pip install boto3==1.9.86
pip install requests==2.21.0

# Bootstrapping code

# Cleanup
source "$WORKING_DIR/miniconda/bin/deactivate"
rm -rf "$WORKING_DIR/miniconda.sh"
```

You can also add a sandboxed Python installation as a kernel that you can use in a Jupyter notebook by including the following code to the above lifecycle configuration:

```
source "$WORKING_DIR/miniconda/bin/activate"
```

```
# If required, add this as a kernel
pip install ipykernel
python -m ipykernel install --user --name MyProjectEnv --display-name "Python
(myprojectenv)"

source "$WORKING_DIR/miniconda/bin/deactivate"
```

Associate Git Repositories with Amazon SageMaker Notebook Instances

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

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

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

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

Topics

- [Add a Git Repository to Your Amazon SageMaker Account \(p. 264\)](#)
- [Create a Notebook Instance with an Associated Git Repository \(p. 267\)](#)
- [Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance \(p. 268\)](#)
- [Use Git Repositories in a Notebook Instance \(p. 269\)](#)

Add a Git Repository to Your Amazon SageMaker Account

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

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

Note

You can use the Amazon SageMaker API

[CreateCodeRepository](#) to add Git repositories to your Amazon SageMaker account, but step-by-step instructions are not provided here.

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

To add a Git repository as a resource in your Amazon SageMaker account

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Git repositories**, then choose **Add repository**.
3. To add an CodeCommit repository, choose **AWS CodeCommit**. To add a GitHub or other Git-based repository, choose **GitHub/Other Git-based repo**.

To add an existing CodeCommit repository

1. Choose **Use existing repository**.
2. For **Repository**, choose a repository from the list.
3. Enter a name to use for the repository in Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
4. Choose **Add repository**.

To create a new CodeCommit repository

1. Choose **Create new repository**.
2. Enter a name for the repository that you can use in both CodeCommit and Amazon SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
3. Choose **Create repository**.

To add a Git repository hosted somewhere other than CodeCommit

1. Choose **GitHub/Other Git-based repo**.
2. Enter a name of up to 63 characters. Valid characters include alpha-numeric characters, a hyphen (-), and 0-9.
3. Enter the URL for the repository. Do not provide a user name in the URL. Add the username and password in AWS Secrets Manager as described in the next step.
4. For **Git credentials**, choose the credentials to use to authenticate to the repository. This is necessary only if the Git repository is private.

Note

If you have two-factor authentication enabled for your Git repository, use a personal access token generated by your Git service provider instead of a password.

- a. To use an existing AWS Secrets Manager secret, choose **Use existing secret**, and then choose a secret from the list. For information about creating and storing a secret, see [Creating a Basic Secret](#) in the *AWS Secrets Manager User Guide*. The name of the secret you use must contain the string `sagemaker`.

Note

The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```
{"username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password. For information, see <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>.

-
- b. To create a new AWS Secrets Manager secret, choose **Create secret**, enter a name for the secret, and then enter the username and password to use to authenticate to the repository. The name for the secret must contain the string `sagemaker`.

Note

The IAM role you use to create the secret must have the `secretsmanager:GetSecretValue` permission in its IAM policy.

The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```
{ "username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password.

-
-
- c. To not use any credentials, choose **No secret**.

5. Choose **Create secret**.

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

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

- The default branch
- The URL of the Git repository

Note

Do not provide a user name in the URL. Add the username and password in AWS Secrets Manager as described in the next step.

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

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

For Linux, OS X, or Unix:

```
aws sagemaker create-code-repository \
    --code-repository-name "MyRepository" \
    --git-config '{\"Branch\":\"master\", \"RepositoryUrl\" :
    \"https://github.com/myprofile/my-repo\", \"SecretArn\" :
    \"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABc0DE\"}'
```

For Windows:

```
aws sagemaker create-code-repository ^
    --code-repository-name "MyRepository" ^
    --git-config "{\"Branch\":\"master\", \"RepositoryUrl\" :
    \"https://github.com/myprofile/my-repo\", \"SecretArn\" :
    \"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABc0DE\"}"
```

Note

The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```
{ "username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password.

Create a Notebook Instance with an Associated Git Repository

You can associate Git repositories with a notebook instance when you create the notebook instance by using the AWS Management Console, or the AWS CLI. If you want to use a CodeCommit repository that is in a different AWS than the notebook instance, set up cross-account access for the repository. For information, see [Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance \(p. 268\)](#).

Topics

- [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 267\)](#)
- [Create a Notebook Instance with an Associated Git Repository \(CLI\) \(p. 267\)](#)

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

To create a notebook instance and associate Git repositories in the Amazon SageMaker console

1. Follow the instructions at [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#).
2. For **Git repositories**, choose Git repositories to associate with the notebook instance.
 - a. For **Default repository**, choose a repository that you want to use as your default repository. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to Amazon SageMaker** ([opens the Add repository flow in a new window](#)) and then follow the instructions at [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 267\)](#). To clone a public repository that is not stored in your account, choose **Clone a public Git repository to this notebook instance only**, and then specify the URL for that repository.
 - b. For **Additional repository 1**, choose a repository that you want to add as an additional directory. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to Amazon SageMaker** ([opens the Add repository flow in a new window](#)) and then follow the instructions at [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 267\)](#). To clone a repository that is not stored in your account, choose **Clone a public Git repository to this notebook instance only**, and then specify the URL for that repository.

Repeat this step up to three times to add up to three additional repositories to your notebook instance.

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

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

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

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

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

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

Note

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

Associate a CodeCommit Repository in a Different AWS Account with a Notebook Instance

To associate a CodeCommit repository in a different AWS account with your notebook instance, set up cross-account access for the CodeCommit repository.

To set up cross-account access for a CodeCommit repository and associate it with a notebook instance:

- In the AWS account that contains the CodeCommit repository, create an IAM policy that allows access to the repository from users in the account that contains your notebook instance. For information, see [Step 1: Create a Policy for Repository Access in AccountA](#) in the *CodeCommit User Guide*.
- In the AWS account that contains the CodeCommit repository, create an IAM role, and attach the policy that you created in the previous step to that role. For information, see [Step 2: Create a Role for Repository Access in AccountA](#) in the *CodeCommit User Guide*.
- Create a profile in the notebook instance that uses the role that you created in the previous step:
 - Open the notebook instance.
 - Open a terminal in the notebook instance.
 - Edit a new profile by typing the following in the terminal:

```
vi /home/ec2-user/.aws/config
```

- Edit the file with the following profile information:

```
[profile CrossAccountAccessProfile]
region = us-west-2
role_arn =
  arn:aws:iam::CodeCommitAccount:role/CrossAccountRepositoryContributorRole
credential_source=Ec2InstanceMetadata
output = json
```

Where `CodeCommitAccount` is the account that contains the CodeCommit repository, `CrossAccountAccessProfile` is the name of the new profile, and `CrossAccountRepositoryContributorRole` is the name of the role you created in the previous step.

4. On the notebook instance, configure git to use the profile you created in the previous step:
 - a. Open the notebook instance.
 - b. Open a terminal in the notebook instance.
 - c. Edit the Git configuration file typing the following in the terminal:

```
vi /home/ec2-user/.gitconfig
```

- d. Edit the file with the following profile information:

```
[credential]
  helper = !aws codecommit credential-helper --
profile CrossAccountAccessProfile $@
  UseHttpPath = true
```

Where `CrossAccountAccessProfile` is the name of the profile that you created in the previous step.

Use Git Repositories in a Notebook Instance

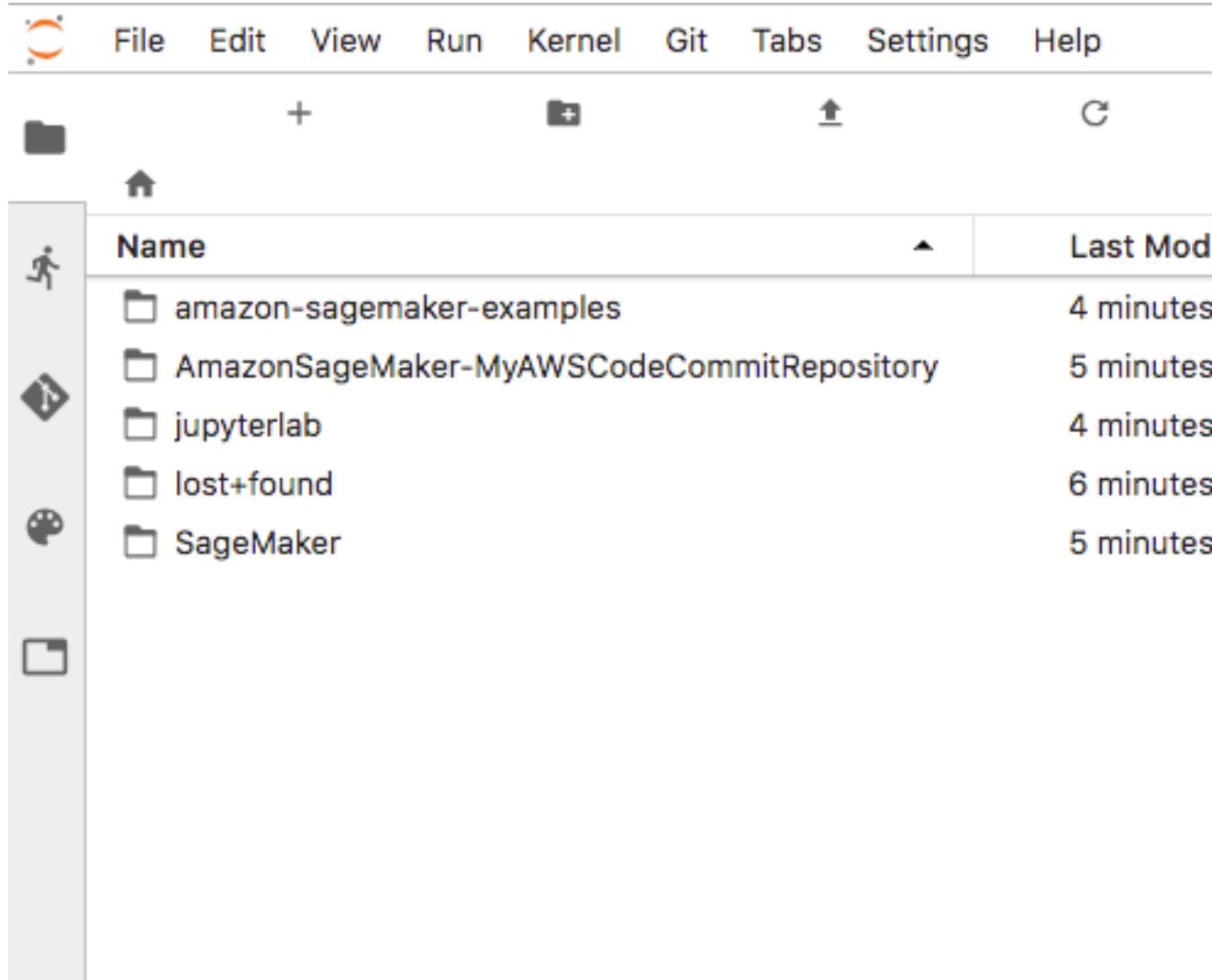
When you open a notebook instance that has Git repositories associated with it, it opens in the default repository, which is installed in your notebook instance directly under `/home/ec2-user/SageMaker`. You can open and create notebooks, and you can manually run Git commands in a notebook cell. For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

```
!git pull origin master
```

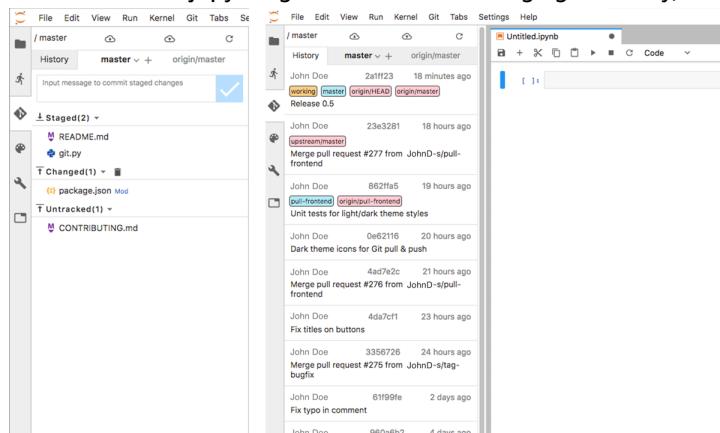
To open any of the additional repositories, navigate up one folder. The additional repositories are also installed as directories under `/home/ec2-user/SageMaker`.

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

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



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



Get Notebook Instance Metadata

When you create a notebook instance, Amazon SageMaker creates a JSON file on the instance at the location `/opt/ml/metadata/resource-metadata.json` that contains the `ResourceName` and `ResourceArn` of the notebook instance. You can access this metadata from anywhere within the notebook instance, including in lifecycle configurations. For information about notebook instance lifecycle configurations, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).

The `resource-metadata.json` file has the following structure:

```
{  
    "ResourceArn": "NotebookInstanceArn",  
    "ResourceName": "NotebookInstanceName"  
}
```

You can use this metadata from within the notebook instance to get other information about the notebook instance. For example, the following commands get the tags associated with the notebook instance:

```
NOTEBOOK_ARN=$(jq '.ResourceArn'  
    /opt/ml/metadata/resource-metadata.json --raw-output)  
aws sagemaker list-tags --resource-arn $NOTEBOOK_ARN
```

The output looks like the following:

```
{  
    "Tags": [  
        {  
            "Key": "test",  
            "Value": "true"  
        }  
    ]  
}
```

Monitor Jupyter Logs in Amazon CloudWatch Logs

Jupyter logs include important information such as events, metrics, and health information that provide actionable insights when running Amazon SageMaker notebooks. By importing Jupyter logs into CloudWatch Logs, customers can use CloudWatch Logs to detect anomalous behaviors, set alarms, and discover insights to keep the Amazon SageMaker notebooks running more smoothly. You can access the logs even when the Amazon EC2 instance that hosts the notebook is unresponsive, and use the logs to troubleshoot the unresponsive notebook. Sensitive information such as AWS account IDs, secret keys, and authentication tokens in presigned URLs are removed so that customers can share logs without leaking private information.

To view Jupyter logs for a notebook instance:

1. Sign in to the AWS Management Console and open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Notebook instances**.
3. In the list of notebook instances, choose the notebook instance for which you want to view Jupyter logs.
4. Under **Monitor** on the notebook instance details page, choose **View logs**.
5. In the CloudWatch console, choose the log stream for your notebook instance. Its name is in the form `NotebookInstanceName/jupyter.log`.

For more information about monitoring CloudWatch logs for Amazon SageMaker, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

Choose an Algorithm

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. You have several choices when choosing an algorithm to use with Amazon SageMaker. These choices include:

- Amazon SageMaker provides several built-in machine learning algorithms that you can use for a variety of problem types. For more information on Amazon SageMaker built-in algorithms, see [.](#)
- You can use your code to access the most popular machine learning and deep learning framework algorithms to build models in Amazon SageMaker. For more information on the machine learning and deep learning frameworks supported by Amazon SageMaker, see [.](#)
- You can use your own algorithm or model with Amazon SageMaker. For more information on using your own algorithms with Amazon SageMaker, see [Use Your Own Algorithms or Models with Amazon SageMaker \(p. 478\)](#).

Use Amazon SageMaker Built-in Algorithms

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

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

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

- Answers that fit into discrete categories—for example, answers to these questions:
 - "Based on past customer responses, should I mail this particular customer?" Answers to this question fall into two categories, "yes" or "no." In this case, you use the answer to narrow the recipients of the mail campaign.
 - "Based on past customer segmentation, which segment does this customer fall into?" Answers might fall into categories such as "empty nester," "suburban family," or "urban professional." You could use these segments to decide who should receive the mailing.

For this type of discrete classification problem, Amazon SageMaker provides two algorithms: [Linear Learner Algorithm \(p. 365\)](#) and the [XGBoost Algorithm \(p. 445\)](#). You set the following hyperparameters to direct these algorithms to produce discrete results:

- For the Linear Learner algorithm, set the `predictor_type` hyperparameter to `binary_classifier`.
- For the XGBoost algorithm, set the `objective` hyperparameter to `reg:logistic`.
- Answers that are quantitative—Consider this question: "Based on the return on investment (ROI) from past mailings, what is the ROI for mailing this customer?" In this case, you use the ROI to target customers for the mail campaign. For these quantitative analysis problems, you can also use the [Linear Learner Algorithm \(p. 365\)](#) or the [XGBoost Algorithm \(p. 445\)](#) algorithms. You set the following hyperparameters to direct these algorithms to produce quantitative results:
 - For the Linear Learner algorithm, set the `predictor_type` hyperparameter to `regressor`.
 - For the XGBoost algorithm, set the `objective` hyperparameter to `reg:linear`.
- Answers in the form of discrete recommendations—Consider this question: "Based on past responses to mailings, what is the recommended content for each customer?" In this case, you are looking for a recommendation on what to mail, not whether to mail, the customer. For this problem, Amazon SageMaker provides the [Factorization Machines Algorithm \(p. 314\)](#) 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 \(p. 344\)](#).
- "What are the attributes that differentiate these customers, and what are the values for each customer along those dimensions." You use these answers to simplify the view of current and prospective customers, and, maybe, to better understand these customer attributes. For this type of question, Amazon SageMaker provides the [Principal Component Analysis \(PCA\) Algorithm \(p. 412\)](#) algorithm.

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

- [Image Classification Algorithm \(p. 324\)](#)—Use this algorithm to classify images. It uses example data with answers (referred to as *supervised algorithm*).
- [Sequence-to-Sequence Algorithm \(p. 432\)](#)—This supervised algorithm is commonly used for neural machine translation.

- [Latent Dirichlet Allocation \(LDA\) Algorithm \(p. 360\)](#)—This algorithm is suitable for determining topics in a set of documents. It is an *unsupervised algorithm*, which means that it doesn't use example data with answers during training.
- [Neural Topic Model \(NTM\) Algorithm \(p. 380\)](#)—Another unsupervised technique for determining topics in a set of documents, using a neural network approach.

Topics

- [Common Elements of Built-in Algorithms \(p. 274\)](#)
- [BlazingText Algorithm \(p. 290\)](#)
- [DeepAR Forecasting Algorithm \(p. 299\)](#)
- [Factorization Machines Algorithm \(p. 314\)](#)
- [Image Classification Algorithm \(p. 324\)](#)
- [IP Insights Algorithm \(p. 334\)](#)
- [K-Means Algorithm \(p. 344\)](#)
- [K-Nearest Neighbors \(k-NN\) Algorithm \(p. 351\)](#)
- [Latent Dirichlet Allocation \(LDA\) Algorithm \(p. 360\)](#)
- [Linear Learner Algorithm \(p. 365\)](#)
- [Neural Topic Model \(NTM\) Algorithm \(p. 380\)](#)
- [Object2Vec Algorithm \(p. 386\)](#)
- [Object Detection Algorithm \(p. 402\)](#)
- [Principal Component Analysis \(PCA\) Algorithm \(p. 412\)](#)
- [Random Cut Forest \(RCF\) Algorithm \(p. 417\)](#)
- [Semantic Segmentation Algorithm \(p. 424\)](#)
- [Sequence-to-Sequence Algorithm \(p. 432\)](#)
- [XGBoost Algorithm \(p. 445\)](#)

Common Elements of Built-in Algorithms

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

Topics

- [Common Parameters for Built-In Algorithms \(p. 274\)](#)
- [Common Data Formats for Built-in Algorithms \(p. 280\)](#)
- [Instance Types for Built-in Algorithms \(p. 288\)](#)
- [Logs for Built-In Algorithms \(p. 289\)](#)

Common Parameters for Built-In Algorithms

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

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
BlazingText	train	<ecr_path>/blazingtext:<tag>	File or Pipe	Text file (one)	GPU (single)	No

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
				sentence per line with space-separated tokens)	instance only) or CPU	
DeepAR Forecasting	train and (optionally) test	<ecr_path>/forecasting-deepar:<tag>	File	JSON Lines or Parquet	GPU or CPU	Yes
Factorization Machines	train and (optionally) test	<ecr_path>/factorization-machines:<tag>	File or Pipe	recordIO-protobuf	CPU (GPU for dense data)	Yes
Image Classification	train and validation, (optionally) train_lst, validation_lst, and model	<ecr_path>/image-classification:<tag>	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
IP Insights	train and (optionally) validation	<ecr_path>/ipinsights:<tag>	File	CSV	CPU or GPU	Yes
k-means	train and (optionally) test	<ecr_path>/kmeans:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPUCommon (single GPU device on one or more instances)	No
k-nearest-neighbor (k-NN)	train and (optionally) test	<ecr_path>/knn:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPU (single GPU device on one or more instances)	Yes
LDA	train and (optionally) test	<ecr_path>/lda:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU (single instance only)	No
Linear Learner	train and (optionally) validation, test, or both	<ecr_path>/linear-learner:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU or GPU	Yes

Algorithm Name	Channel Name	Training Image and Inference Image Registry Path	Training Input Mode	File Type	Instance Class	Parallelizable
Neural Topic Model	train and (optionally) validation, test, or both	<ecr_path>/ntm:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Object2Vec	train and (optionally) validation, test, or both	<ecr_path>/object2vec:<tag>	File	JSON Lines	GPU or CPU (single instance only)	No
Object Detection	train and validation, (optionally) train_annotation, validation_annotation, and model	<ecr_path>/object-detection:<tag>	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes
PCA	train and (optionally) test	<ecr_path>/pca:<tag>	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes
Random Cut Forest	train and (optionally) test	<ecr_path>/randomcutforest:<tag>	File or Pipe	recordIO-protobuf or CSV	CPU	Yes
Semantic Segmentation	train and validation, (optionally) label_map and model	<ecr_path>/semantic-segmentation:<tag>	File or Pipe	image files	GPU (single instance only)	No
Seq2Seq Modeling	train, validation, and vocab	<ecr_path>/seq2seq:<tag>	File	recordIO-protobuf	GPU (single instance only)	No
XGBoost	train and (optionally) validation	<ecr_path>/xgboost:<tag>	File	CSV or LibSVM	CPU	Yes

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

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

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
Factorization Machines, IP Insights, k-means, k-nearest-neighbor, Linear Learner, Object2Vec, Neural Topic Model, PCA, and Random Cut Forest	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	174872318107.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	382416733822.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	404615174143.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com
	ap-northeast-1	351501993468.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	835164637446.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	712309505854.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	664544806723.dkr.ecr.eu-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	438346466558.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	me-south-1	249704162688.dkr.ecr.me-south-1.amazonaws.com
	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com
LDA	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	266724342769.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	766337827248.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	999911452149.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-northeast-1	258307448986.dkr.ecr.ap-northeast-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
BlazingText, Image Classification, Object Detection, Semantic Segmentation, Seq2Seq, and XGBoost (0.72)	ap-northeast-2	293181348795.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	297031611018.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	353608530281.dkr.ecr.eu-central-1.amazonaws.com
	eu-west-1	999678624901.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	433757028032.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	811284229777.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	825641698319.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com
	ap-northeast-1	501404015308.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	306986355934.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	544295431143.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	813361260812.dkr.ecr.eu-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	685385470294.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	me-south-1	249704162688.dkr.ecr.me-south-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
XGBoost (0.90)	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com
	us-west-1	746614075791.dkr.ecr.us-west-1.amazonaws.com
	us-west-2	246618743249.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	683313688378.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	257758044811.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	414596584902.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-northeast-1	354813040037.dkr.ecr.ap-northeast-1.amazonaws.com
	ap-northeast-2	366743142698.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-southeast-1	121021644041.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	783357654285.dkr.ecr.ap-southeast-2.amazonaws.com
	ap-south-1	720646828776.dkr.ecr.ap-south-1.amazonaws.com
	ap-east-1	651117190479.dkr.ecr.ap-east-1.amazonaws.com
	ca-central-1	341280168497.dkr.ecr.ca-central-1.amazonaws.com
	eu-central-1	492215442770.dkr.ecr.eu-central-1.amazonaws.com
DeepAR Forecasting	eu-north-1	662702820516.dkr.ecr.eu-north-1.amazonaws.com
	eu-west-1	141502667606.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	764974769150.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	659782779980.dkr.ecr.eu-west-3.amazonaws.com
	me-south-1	801668240914.dkr.ecr.me-south-1.amazonaws.com
	sa-east-1	737474898029.dkr.ecr.sa-east-1.amazonaws.com
	us-west-1	632365934929.dkr.ecr.us-west-1.amazonaws.com
DeepAR Forecasting	us-west-2	156387875391.dkr.ecr.us-west-2.amazonaws.com
	us-east-1	522234722520.dkr.ecr.us-east-1.amazonaws.com
	us-east-2	566113047672.dkr.ecr.us-east-2.amazonaws.com
	us-gov-west-1	226302683700.dkr.ecr.us-gov-west-1.amazonaws.com
	ap-east-1	286214385809.dkr.ecr.ap-east-1.amazonaws.com
	ap-northeast-1	633353088612.dkr.ecr.ap-northeast-1.amazonaws.com

Algorithm Name	AWS Region	Training Image and Inference Image Registry Path
	ap-northeast-2	204372634319.dkr.ecr.ap-northeast-2.amazonaws.com
	ap-south-1	991648021394.dkr.ecr.ap-south-1.amazonaws.com
	ap-southeast-1	475088953585.dkr.ecr.ap-southeast-1.amazonaws.com
	ap-southeast-2	514117268639.dkr.ecr.ap-southeast-2.amazonaws.com
	ca-central-1	469771592824.dkr.ecr.ca-central-1.amazonaws.com
	eu-north-1	669576153137.dkr.ecr.eu-north-1.amazonaws.com
	eu-central-1	495149712605.dkr.ecr.eu-central-1.amazonaws.com
	eu-west-1	224300973850.dkr.ecr.eu-west-1.amazonaws.com
	eu-west-2	644912444149.dkr.ecr.eu-west-2.amazonaws.com
	eu-west-3	749696950732.dkr.ecr.eu-west-3.amazonaws.com
	me-south-1	249704162688.dkr.ecr.me-south-1.amazonaws.com
	sa-east-1	855470959533.dkr.ecr.sa-east-1.amazonaws.com

Use the paths and training input mode as follows:

- To create a training job (with a request to the [CreateTrainingJob API](#)), specify the Docker Registry path and the training input mode for the training image. You create a training job to train a model using a specific dataset.
- To create a model (with a [CreateModel](#) request), specify the Docker Registry path for the inference image. Amazon SageMaker launches machine learning compute instances that are based on the endpoint configuration and deploys the model, which includes the artifacts (the result of model training).

Common Data Formats for Built-in Algorithms

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

Topics

- [Common Data Formats for Training \(p. 280\)](#)
- [Common Data Formats for Inference \(p. 284\)](#)

Common Data Formats for Training

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

- Training data serialization (handled by you)

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

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

The following table lists supported [Content Type](#) values:

Content Type	Definition
text/csv; label_size=n	Comma-separated values, where n specifies the number of starting columns in a row that are labels. The default value for n is 1.
application/x-recordio-protobuf	A protobuf message wrapped in a RecordIO record.

Training Data Formats

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

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

Note

For an example that shows how to convert the commonly used numpy array into the protobuf recordIO format, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/factorization_machines_mnist/factorization_machines_mnist.ipynb.

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

```
syntax = "proto2";

package aialgs.data;

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

```

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

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

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

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

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

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

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

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

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

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

    // If the content type of the data is known, stores it.
    // This allows for the possibility of using decoders for common formats
    // in the future.
    optional string content_type = 2;
}

message Value {

```

```

oneof value {
    // The numbering assumes the possible use of:
    // - float16, float128
    // - int8, int16, int32
    Float32Tensor float32_tensor = 2;
    Float64Tensor float64_tensor = 3;
    Int32Tensor int32_tensor = 7;
    Bytes bytes = 9;
}
}

message Record {
    // Map from the name of the feature to the value.
    //
    // For vectors and libsvm-like datasets,
    // a single feature with the name `values`
    // should be specified.
    map<string, Value> features = 1;

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

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

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

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

```

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

Note

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

Trained Model Deserialization

Amazon SageMaker models are stored as `model.tar.gz` in the S3 bucket specified in `OutputDataConfig.S3OutputPath` parameter of the `create_training_job` call. You can specify most of these model

artifacts when creating a hosting model. You can also open and review them in your notebook instance. When `model.tar.gz` is untarred, it contains `model_algo-1`, which is a serialized Apache MXNet object. For example, you use the following to load the k-means model into memory and view it:

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

Common Data Formats for Inference

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

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

Convert Data for Inference Request Serialization

Content type options for Amazon SageMaker algorithm inference requests include: `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Algorithms that don't support all of these types can support other types. XGBoost, for example, only supports `text/csv` from this list, but also supports `text/libsvm`.

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

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

```
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:

```
// Has the same format as the protocol buffers implementation described for training.
let dataElement = {
    "keys": [],
    "values": [],
    "shape": []
```

```
}
```

Simple numeric vector:

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

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

And, for multiple records:

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

Convert Data for Inference Response Deserialization

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

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

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

Single-record inference:

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

Multi-record inference:

```
let response = {
  "predictions": [
    // First instance prediction.
    {
      "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:

```

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

```

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

```

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

```

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

Common Request Formats for All Algorithms

Most algorithms use several of the following inference request formats.

JSON Request Format

Content-type: application/json

Dense Format

```

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

```

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

JSONLINES Request Format

Content-type: application/jsonlines

Dense Format

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

```

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

or:

```

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

Sparse Format

A single record in sparse format is represented as:

```

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

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

```

{"data": {"features": { "keys": [0, 1, 3], "shape": [4], "values": [1, 4, 1] } } }
{ "data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } }
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

CSV Request Format

Content-type: text/csv;label_size=0

Note

CSV support is not available for factorization machines.

RECORDIO Request Format

Content-type: application/x-recordio-protobuf

Use Batch Transform with Build-in Algorithms

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

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

ContentType	Recommended SplitType
application/x-recordio-protobuf	RecordIO
text/csv	Line
application/jsonlines	Line
application/json	None
application/x-image	None
image/*	None
Accept	Recommended AssembleWith
application/x-recordio-protobuf	None
application/json	None
application/jsonlines	Line

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

- [PCA Response Formats \(p. 416\)](#)
- [Linear Learner Response Formats \(p. 378\)](#)
- [NTM Response Formats \(p. 385\)](#)
- [K-Means Response Formats \(p. 351\)](#)
- [Factorization Machine Response Formats \(p. 323\)](#)

Instance Types for Built-in Algorithms

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

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

Most Amazon SageMaker algorithms have been engineered to take advantage of GPU computing for training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective.

Exceptions, such as XGBoost, are noted in this guide. (XGBoost implements an open-source algorithm that has been optimized for CPU computation.)

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

For more information on Amazon SageMaker hardware specifications, see [Amazon SageMaker ML Instance Types](#).

Logs for Built-In Algorithms

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

Note

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

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

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

Common Errors

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

```
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_size' was
unexpected)
```

2. Specifying an invalid value for a hyperparameter.

FailureReason:

```
AlgorithmError: u'abc' is not valid under any of the given
schemas\n\nFailed validating u'oneOf' in
schema[u'properties'][u'feature_dim']: \n      {u'oneOf':
[ {u'pattern': u'^([1-9][0-9]*)$', u'type': u'string'}, \n
```

```
{u'minimum': 1, u'type': u'integer'}]]}\
```

FailureReason:

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

3. Inaccurate protobuf file format.

From the CloudWatch log:

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

BlazingText Algorithm

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms. The Word2vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc. Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.

The Word2vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is called a *word embedding*. Words that are semantically similar correspond to vectors that are close together. That way, word embeddings capture the semantic relationships between words.

Many natural language processing (NLP) applications learn word embeddings by training on large collections of documents. These pretrained vector representations provide information about semantics and word distributions that typically improves the generalizability of other models that are later trained on a more limited amount of data. Most implementations of the Word2vec algorithm are not optimized for multi-core CPU architectures. This makes it difficult to scale to large datasets.

With the BlazingText algorithm, you can scale to large datasets easily. Similar to Word2vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures. BlazingText's implementation of the supervised multi-class, multi-label text classification algorithm extends the fastText text classifier to use GPU acceleration with custom CUDA kernels. You can train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. And, you achieve performance on par with the state-of-the-art deep learning text classification algorithms.

The Amazon SageMaker BlazingText algorithms provides the following features:

- Accelerated training of the fastText text classifier on multi-core CPUs or a GPU and Word2Vec on GPUs using highly optimized CUDA kernels. For more information, see [BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs](#).
- [Enriched Word Vectors with Subword Information](#) by learning vector representations for character n-grams. This approach enables BlazingText to generate meaningful vectors for out-of-vocabulary (OOV) words by representing their vectors as the sum of the character n-gram (subword) vectors.
- A `batch_skipgram` mode for the Word2Vec algorithm that allows faster training and distributed computation across multiple CPU nodes. The `batch_skipgram` mode does mini-batching using the Negative Sample Sharing strategy to convert level-1 BLAS operations into level-3 BLAS operations. This efficiently leverages the multiply-add instructions of modern architectures. For more information, see [Parallelizing Word2Vec in Shared and Distributed Memory](#).

To summarize, the following modes are supported by BlazingText on different types instances:

Modes	Word2Vec (Unsupervised Learning)	Text Classification (Supervised Learning)
Single CPU instance	cbow Skip-gram Batch Skip-gram	supervised
Single GPU instance (with 1 or more GPUs)	cbow Skip-gram	supervised with one GPU
Multiple CPU instances	Batch Skip-gram	None

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

Topics

- [Input/Output Interface for the BlazingText Algorithm \(p. 291\)](#)
- [EC2 Instance Recommendation for the BlazingText Algorithm \(p. 294\)](#)
- [BlazingText Sample Notebooks \(p. 294\)](#)
- [BlazingText Hyperparameters \(p. 294\)](#)
- [Tune a BlazingText Model \(p. 298\)](#)

Input/Output Interface for the BlazingText Algorithm

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

Training and Validation Data Format

Training and Validation Data Format for the Word2Vec Algorithm

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

Training and Validation Data Format for the Text Classification Algorithm

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

Train with File Mode

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

```
__label__4 linux ready for prime time , intel says , despite all the linux hype , the
open-source movement has yet to make a huge splash in the desktop market . that may be
about to change , thanks to chipmaking giant intel corp .

__label__2 bowled by the slower one again , kolkata , november 14 the past caught up with
sourav ganguly as the indian skippers return to international cricket was short lived .
```

Note

The order of labels within the sentence doesn't matter.

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

Train with Augmented Manifest Text Format

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

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

Multi-label training is also supported by specifying a JSON array of labels.

```
{"source":"linux ready for prime time , intel says , despite all the linux hype", "label": [1, 3]}  
{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly", "label": [2, 4, 5]}
```

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Model Artifacts and Inference

Model Artifacts for the Word2Vec Algorithm

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

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

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

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

Sample JSON Request

Mime-type: `application/json`

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

Model Artifacts for the Text Classification Algorithm

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

Sample JSON Request

Mime-type: `application/json`

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

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

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

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

```
content-type: application/jsonlines  
  
{"source": "source_0"}  
{"source": "source_1"}  
  
if you need to pass the value of k for top-k, then you can do it in the following way:  
  
{"source": "source_0", "k": 2}  
{"source": "source_1", "k": 3}
```

The format for output follows:

```
accept: application/jsonlines  
  
{"prob": [prob_1], "label": ["__label__1"]}  
{"prob": [prob_1], "label": ["__label__1"]}  
  
If you have passed the value of k to be more than 1, then response will be in this format:  
  
{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}  
{"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
```

For both supervised (text classification) and unsupervised (Word2Vec) modes, the binaries (`*.bin`) produced by BlazingText can be cross-consumed by fastText and vice versa. You can use binaries produced by BlazingText by fastText. Likewise, you can host the model binaries created with fastText using BlazingText. However, the binaries are only supported when training on CPU and single GPU; training on multi-GPU will not produce binaries.

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

EC2 Instance Recommendation for the BlazingText Algorithm

For `cbow` and `skipgram` modes, BlazingText supports single CPU and single GPU instances. Both of these modes support learning of subwords embeddings. To achieve the highest speed without compromising accuracy, we recommend that you use an `ml.p3.2xlarge` instance.

For `batch_skipgram` mode, BlazingText supports single or multiple CPU instances. When training on multiple instances, set the value of the `S3DataDistributionType` field of the `S3DataSource` object that you pass to `CreateTrainingJob` to `FullyReplicated`. BlazingText takes care of distributing data across machines.

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

BlazingText Sample Notebooks

For a sample notebook that uses the Amazon SageMaker BlazingText algorithm to train and deploy supervised binary and multiclass classification models, see [Blazing Text classification on the DBpedia dataset](#). For instructions for creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the Blazing Text are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

BlazingText Hyperparameters

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

Word2Vec Hyperparameters

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

Parameter Name	Description
<code>mode</code>	<p>The Word2vec architecture used for training.</p> <p>Required</p> <p>Valid values: <code>batch_skipgram</code>, <code>skipgram</code>, or <code>cbow</code></p>
<code>batch_size</code>	<p>The size of each batch when <code>mode</code> is set to <code>batch_skipgram</code>. Set to a number between 10 and 20.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 11</p>
<code>buckets</code>	<p>The number of hash buckets to use for subwords.</p> <p>Optional</p> <p>Valid values: positive integer</p>

Parameter Name	Description
	Default value: 2000000
epochs	<p>The number of complete passes through the training data.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
evaluation	<p>Whether the trained model is evaluated using the WordSimilarity-353 Test.</p> <p>Optional</p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p> <p>Default value: <code>True</code></p>
learning_rate	<p>The step size used for parameter updates.</p> <p>Optional</p> <p>Valid values: Positive float</p> <p>Default value: 0.05</p>
min_char	<p>The minimum number of characters to use for subwords/character n-grams.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
min_count	<p>Words that appear less than <code>min_count</code> times are discarded.</p> <p>Optional</p> <p>Valid values: Non-negative integer</p> <p>Default value: 5</p>
max_char	<p>The maximum number of characters to use for subwords/character n-grams</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 6</p>

Parameter Name	Description
<code>negative_samples</code>	The number of negative samples for the negative sample sharing strategy. Optional Valid values: Positive integer Default value: 5
<code>sampling_threshold</code>	The threshold for the occurrence of words. Words that appear with higher frequency in the training data are randomly down-sampled. Optional Valid values: Positive fraction. The recommended range is (0, 1e-3] Default value: 0.0001
<code>subwords</code>	Whether to learn subword embeddings or not. Optional Valid values: (Boolean) <code>True</code> or <code>False</code> Default value: <code>False</code>
<code>vector_dim</code>	The dimension of the word vectors that the algorithm learns. Optional Valid values: Positive integer Default value: 100
<code>window_size</code>	The size of the context window. The context window is the number of words surrounding the target word used for training. Optional Valid values: Positive integer Default value: 5

Text Classification Hyperparameters

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

Note

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

Parameter Name	Description
<code>mode</code>	The training mode. Required

Parameter Name	Description
	Valid values: <code>supervised</code>
<code>buckets</code>	<p>The number of hash buckets to use for word n-grams.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 2000000</p>
<code>early_stopping</code>	<p>Whether to stop training if validation accuracy doesn't improve after a patience number of epochs.</p> <p>Optional</p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>
<code>epochs</code>	<p>The maximum number of complete passes through the training data.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
<code>learning_rate</code>	<p>The step size used for parameter updates.</p> <p>Optional</p> <p>Valid values: Positive float</p> <p>Default value: 0.05</p>
<code>min_count</code>	<p>Words that appear less than <code>min_count</code> times are discarded.</p> <p>Optional</p> <p>Valid values: Non-negative integer</p> <p>Default value: 5</p>
<code>min_epochs</code>	<p>The minimum number of epochs to train before early stopping logic is invoked.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>

Parameter Name	Description
<code>patience</code>	<p>The number of epochs to wait before applying early stopping when no progress is made on the validation set. Used only when <code>early_stopping</code> is <code>True</code>.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 4</p>
<code>vector_dim</code>	<p>The dimension of the embedding layer.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 100</p>
<code>word_ngrams</code>	<p>The number of word n-gram features to use.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 2</p>

Tune a BlazingText Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the BlazingText Algorithm

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

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

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

Tunable BlazingText Hyperparameters

Tunable Hyperparameters for the Word2Vec Algorithm

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

Parameter Name	Parameter Type	Recommended Ranges or Values
batch_size	IntegerParameterRange	[8-32]
epochs	IntegerParameterRange	[5-15]
learning_rate	ContinuousParameterRange	MinValue: 0.005, MaxValue: 0.01
min_count	IntegerParameterRange	[0-100]
mode	CategoricalParameterRange	['batch_skipgram', 'skipgram', 'cbow']
negative_samples	IntegerParameterRange	[5-25]
sampling_threshold	ContinuousParameterRange	MinValue: 0.0001, MaxValue: 0.001
vector_dim	IntegerParameterRange	[32-300]
window_size	IntegerParameterRange	[1-10]

Tunable Hyperparameters for the Text Classification Algorithm

Tune an Amazon SageMaker BlazingText text classification model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges or Values
buckets	IntegerParameterRange	[1000000-10000000]
epochs	IntegerParameterRange	[5-15]
learning_rate	ContinuousParameterRange	MinValue: 0.005, MaxValue: 0.01
min_count	IntegerParameterRange	[0-100]
mode	CategoricalParameterRange	['supervised']
vector_dim	IntegerParameterRange	[32-300]
word_ngrams	IntegerParameterRange	[1-3]

DeepAR Forecasting Algorithm

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting

methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

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

The training input for the DeepAR algorithm is one or, preferably, more target time series that have been generated by the same process or similar processes. Based on this input dataset, the algorithm trains a model that learns an approximation of this process/processes and uses it to predict how the target time series evolves. Each target time series can be optionally associated with a vector of static (time-independent) categorical features provided by the `cat` field and a vector of dynamic (time-dependent) time series provided by the `dynamic_feat` field. Amazon SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the `context_length` hyperparameter. To control how far in the future predictions can be made, use the `prediction_length` hyperparameter. For more information, see [How the DeepAR Algorithm Works \(p. 304\)](#).

Topics

- [Input/Output Interface for the DeepAR Algorithm \(p. 300\)](#)
- [Best Practices for Using the DeepAR Algorithm \(p. 303\)](#)
- [EC2 Instance Recommendations for the DeepAR Algorithm \(p. 303\)](#)
- [DeepAR Sample Notebooks \(p. 303\)](#)
- [How the DeepAR Algorithm Works \(p. 304\)](#)
- [DeepAR Hyperparameters \(p. 306\)](#)
- [Tune a DeepAR Model \(p. 310\)](#)
- [DeepAR Inference Formats \(p. 311\)](#)

[Input/Output Interface for the DeepAR Algorithm](#)

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

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

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

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

- `start`—A string with the format `YYYY-MM-DD HH:MM:SS`. The start timestamp can't contain time zone information.

- **target**—An array of floating-point values or integers that represent the time series. You can encode missing values as `null` literals, or as "NaN" strings in JSON, or as `nan` floating-point values in Parquet.
- **dynamic_feat** (optional)—An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must have the same length as the associated `target` value. Missing values are not supported in the features. For example, if `target` time series represents the demand of different products, an associated `dynamic_feat` might be a boolean time-series which indicates whether a promotion was applied (1) to the particular product or not (0):

```
{ "start": ..., "target": [1, 5, 10, 2], "dynamic_feat": [[0, 1, 1, 0]]}
```

- **cat** (optional)—An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive integers. For example, the categorical domain {R, G, B} can be encoded as {0, 1, 2}. All values from each categorical domain must be represented in the training dataset. That's because the DeepAR algorithm can forecast only for categories that have been observed during training. And, each categorical feature is embedded in a low-dimensional space whose dimensionality is controlled by the `embedding_dimension` hyperparameter. For more information, see [DeepAR Hyperparameters \(p. 306\)](#).

If you use a JSON file, it must be in [JSON Lines](#) format. For example:

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

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

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

If the algorithm is trained without `cat` and `dynamic_feat` fields, it learns a "global" model, that is a model that is agnostic to the specific identity of the target time series at inference time and is conditioned only on its shape.

If the model is conditioned on the `cat` and `dynamic_feat` feature data provided for each time series, the prediction will probably be influenced by the character of time series with the corresponding `cat` features. For example, if the `target` time series represents the demand of clothing items, you can associate a two-dimensional `cat` vector that encodes the type of item (e.g. 0 = shoes, 1 = dress) in the first component and the color of an item (e.g. 0 = red, 1 = blue) in the second component. A sample input would look as follows:

```
{ "start": ..., "target": ..., "cat": [0, 0], ... } # red shoes  
{ "start": ..., "target": ..., "cat": [1, 1], ... } # blue dress
```

At inference time, you can request predictions for targets with `cat` values that are combinations of the `cat` values observed in the training data, for example:

```
{ "start": ..., "target": ..., "cat": [0, 1], ... } # blue shoes  
{ "start": ..., "target": ..., "cat": [1, 0], ... } # red dress
```

The following guidelines apply to training data:

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

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

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

$y_{i,t}$ is the true value of time series i at the time t . $\hat{y}_{i,t}$ 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 length of the forecast horizon by setting the `prediction_length` hyperparameter. For more information, see [DeepAR Hyperparameters \(p. 306\)](#).

In addition, the algorithm evaluates the accuracy of the forecast distribution using weighted quantile loss. For a quantile in the range [0, 1], the weighted quantile loss is defined as follows:

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

$q_{i,t}^{(\tau)}$ is the τ -quantile of the distribution that the model predicts. To specify which quantiles to calculate loss for, set the `test_quantiles` hyperparameter. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see [DeepAR Hyperparameters \(p. 306\)](#).

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

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

For more information, see [DeepAR Inference Formats \(p. 311\)](#).

Best Practices for Using the DeepAR Algorithm

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

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference. Regardless of how you set `context_length`, don't break up the time series or provide only a part of it. The model uses data points further back than the value set in `context_length` for the lagged values feature.
- When tuning a DeepAR model, you can split the dataset to create a training dataset and a test dataset. In a typical evaluation, you would test the model on the same time series used for training, but on the future `prediction_length` time points that follow immediately after the last time point visible during training. You can create training and test datasets that satisfy this criteria by using the entire dataset (the full length of all time series that are available) as a test set and removing the last `prediction_length` points from each time series for training. During training, the model doesn't see the target values for time points on which it is evaluated during testing. During testing, the algorithm withholds the last `prediction_length` points of each time series in the test set and generates a prediction. Then it compares the forecast with the withheld values. You can create more complex evaluations by repeating time series multiple times in the test set, but cutting them at different endpoints. With this approach, accuracy metrics are averaged over multiple forecasts from different time points. For more information, see [Tune a DeepAR Model \(p. 310\)](#).
- Avoid using very large values (>400) for the `prediction_length` because it makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating your data at a higher frequency. For example, use `5min` instead of `1min`.
- Because lags are used, a model can look further back in the time series than the value specified for `context_length`. Therefore, you don't need to set this parameter to a large value. We recommend starting with the value that you used for `prediction_length`.
- We recommend training a DeepAR model on as many time series as are available. Although a DeepAR model trained on a single time series might work well, standard forecasting algorithms, such as ARIMA or ETS, might provide more accurate results. The DeepAR algorithm starts to outperform the standard methods when your dataset contains hundreds of related time series. Currently, DeepAR requires that the total number of observations available across all training time series is at least 300.

EC2 Instance Recommendations for the DeepAR Algorithm

You can train DeepAR on both GPU and CPU instances and in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, `ml.c4.2xlarge` or `ml.c4.4xlarge`), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves throughput only for larger models (with many cells per layer and many layers) and for large mini-batch sizes (for example, greater than 512).

For inference, DeepAR supports only CPU instances.

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

DeepAR Sample Notebooks

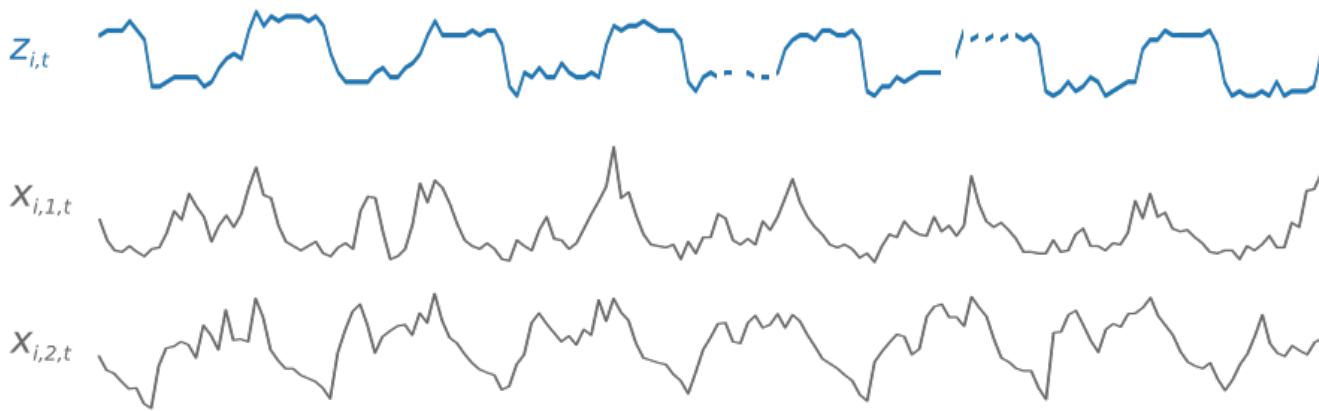
For a sample notebook that shows how to prepare a time series dataset for training the Amazon SageMaker DeepAR algorithm and how to deploy the trained model for performing inferences, see [Time series forecasting with DeepAR - Synthetic data](#) as well as [DeepAR demo on electricity dataset](#), which illustrates the advanced features of DeepAR on a real world dataset. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see

[Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker examples. To open a notebook, choose its **Use** tab, and choose **Create copy**.

How the DeepAR Algorithm Works

During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series. You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector of feature time series and a vector of categorical features. For more information, see [Input/Output Interface for the DeepAR Algorithm \(p. 300\)](#).

For example, the following is an element of a training set indexed by i which consists of a target time series, $Z_{i,t}$, and two associated feature time series, $X_{i,1,t}$ and $X_{i,2,t}$:

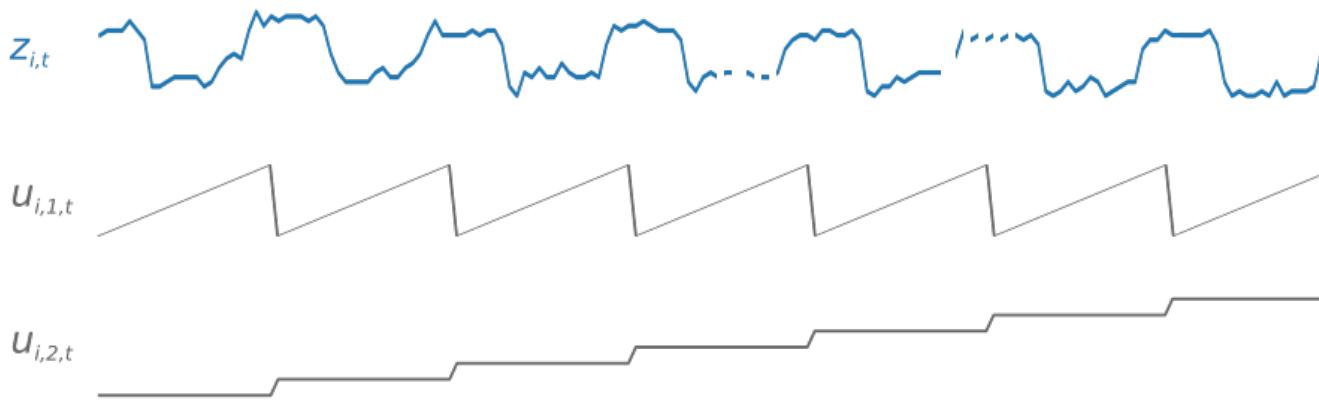


The target time series might contain missing values, which are represented by line breaks in the time series. DeepAR supports only feature time series that are known in the future. This allows you to run "what if?" scenarios. What happens, for example, if I change the price of a product in some way?

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

How Feature Time Series Work in the DeepAR Algorithm

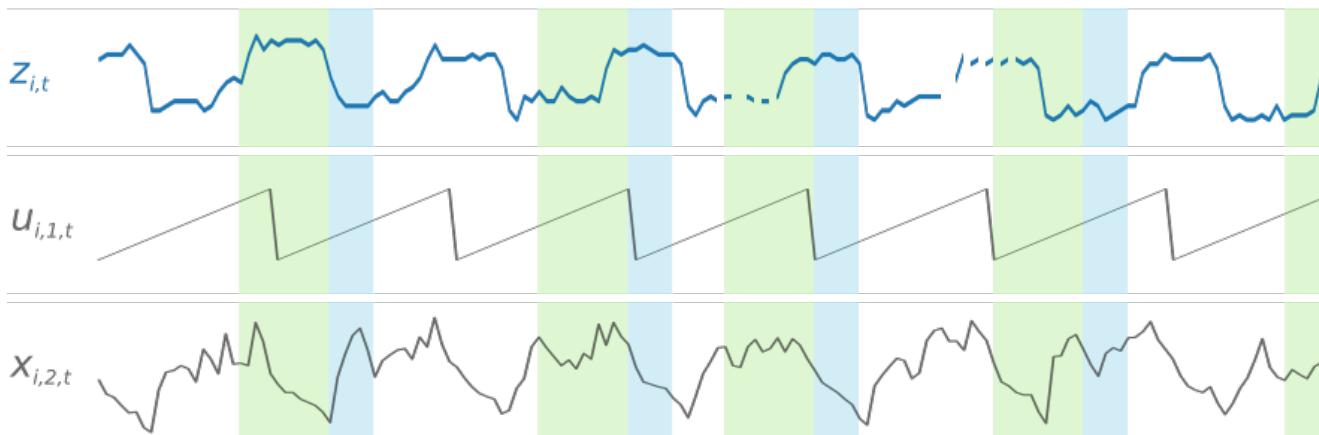
To facilitate learning time-dependent patterns, such as spikes during weekends, DeepAR automatically creates feature time series based on the frequency of the target time series. For example, DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency.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



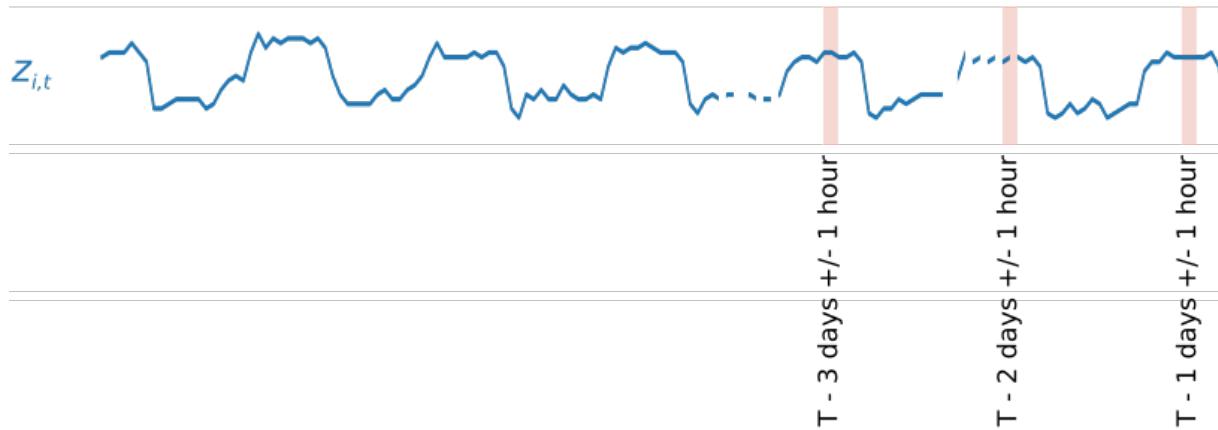
The DeepAR algorithm automatically generates these feature time series. The following table lists the derived features for the supported basic time frequencies.

Frequency of the Time Series	Derived Features
Minute	minute-of-hour, hour-of-day, day-of-week, day-of-month, day-of-year
Hour	hour-of-day, day-of-week, day-of-month, day-of-year
Day	day-of-week, day-of-month, day-of-year
Week	day-of-month, week-of-year
Month	month-of-year

DeepAR trains a model by randomly sampling several training examples from each of the time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. The `context_length` hyperparameter controls how far in the past the network can see, and the `prediction_length` hyperparameter controls how far in the future predictions can be made. During training, the algorithm ignores training set elements containing time series that are shorter than a specified prediction length. The following figure represents five samples with context lengths of 12 hours and prediction lengths of 6 hours drawn from element i . For brevity, we've omitted the feature time series $x_{i,1,t}$ and $u_{i,2,t}$.



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



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

For information on the mathematics behind DeepAR, see [DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks](#).

DeepAR Hyperparameters

Parameter Name	Description
<code>context_length</code>	<p>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 <code>prediction_length</code>. The model also receives lagged inputs from the target, so <code>context_length</code> 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.</p> <p>Required</p> <p>Valid values: Positive integer</p>
<code>epochs</code>	<p>The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also <code>early_stopping_patience</code>. Typical values range from 10 to 1000.</p> <p>Required</p> <p>Valid values: Positive integer</p>
<code>prediction_length</code>	<p>The number of time-steps that the model is trained to predict, also called the forecast horizon. The trained model always generates forecasts with this length. It can't generate longer forecasts. The</p>

Parameter Name	Description
	<p><code>prediction_length</code> is fixed when a model is trained and it cannot be changed later.</p> <p>Required</p> <p>Valid values: Positive integer</p>
<code>time_freq</code>	<p>The granularity of the time series in the dataset. Use <code>time_freq</code> to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, <code>5min</code> specifies a frequency of 5 minutes.</p> <ul style="list-style-type: none"> • <code>M</code>: monthly • <code>W</code>: weekly • <code>D</code>: daily • <code>H</code>: hourly • <code>min</code>: every minute <p>Required</p> <p>Valid values: An integer followed by <code>M</code>, <code>W</code>, <code>D</code>, <code>H</code>, or <code>min</code>. For example, <code>5min</code>.</p>
<code>cardinality</code>	<p>When using the categorical features (<code>cat</code>), <code>cardinality</code> is an array specifying the number of categories (groups) per categorical feature. Set this to <code>auto</code> to infer the cardinality from the data. The <code>auto</code> mode also works when no categorical features are used in the dataset. This is the recommended setting for the parameter.</p> <p>Set <code>cardinality</code> to <code>ignore</code> to force DeepAR to not use categorical features, even if they are present in the data.</p> <p>To perform additional data validation, it is possible to explicitly set this parameter to the actual value. For example, if two categorical features are provided where the first has 2 and the other has 3 possible values, set this to <code>[2, 3]</code>.</p> <p>For more information on how to use categorical feature, see the data-section on the main documentation page of DeepAR.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>ignore</code>, array of positive integers, empty string, or</p> <p>Default value: <code>auto</code></p>
<code>dropout_rate</code>	<p>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.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.1</p>

Parameter Name	Description
early_stopping_patience	<p>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.</p> <p>Optional</p> <p>Valid values: integer</p>
embedding_dimension	<p>Size of embedding vector learned per categorical feature (same value is used for all categorical features).</p> <p>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.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
learning_rate	<p>The learning rate used in training. Typical values range from 1e-4 to 1e-1.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1e-3</p>

Parameter Name	Description
<code>likelihood</code>	<p>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:</p> <ul style="list-style-type: none"> • <i>gaussian</i>: Use for real-valued data. • <i>beta</i>: Use for real-valued targets between 0 and 1 inclusive. • <i>negative-binomial</i>: Use for count data (non-negative integers). • <i>student-T</i>: An alternative for real-valued data that works well for bursty data. • <i>deterministic-L1</i>: A loss function that does not estimate uncertainty and only learns a point forecast. <p>Optional</p> <p>Valid values: One of <i>gaussian</i>, <i>beta</i>, <i>negative-binomial</i>, <i>student-T</i>, or <i>deterministic-L1</i>.</p> <p>Default value: <code>student-T</code></p>
<code>mini_batch_size</code>	<p>The size of mini-batches used during training. Typical values range from 32 to 512.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 128</p>
<code>num_cells</code>	<p>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 40</p>
<code>num_dynamic_feat</code>	<p>The number of <code>dynamic_feat</code> provided in the data. Set this to <code>auto</code> to infer the number of dynamic features from the data. The <code>auto</code> mode also works when no dynamic features are used in the dataset. This is the recommended setting for the parameter.</p> <p>To force DeepAR to not use dynamic features, even if they are present in the data, set <code>num_dynamic_feat</code> to <code>ignore</code>.</p> <p>To perform additional data validation, it is possible to explicitly set this parameter to the actual integer value. For example, if two dynamic features are provided, set this to 2.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>ignore</code>, positive integer, or empty string</p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
<code>num_eval_samples</code>	<p>The number of samples that are used per time-series when calculating test accuracy metrics. This parameter does not have any influence on the training or the final model. In particular, the model can be queried with a different number of samples. This parameter only affects the reported accuracy scores on the test channel after training. Smaller values result in faster evaluation, but then the evaluation scores are typically worse and more uncertain. When evaluating with higher quantiles, for example 0.95, it may be important to increase the number of evaluation samples.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 100</p>
<code>num_layers</code>	<p>The number of hidden layers in the RNN. Typical values range from 1 to 4.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 2</p>
<code>test_quantiles</code>	<p>Quantiles for which to calculate quantile loss on the test channel.</p> <p>Optional</p> <p>Valid values: array of floats</p> <p>Default value: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]</p>

Tune a DeepAR Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the DeepAR Algorithm

The DeepAR algorithm reports three metrics, which are computed during training. When tuning a model, choose one of these as the objective. For the objective, use either the forecast accuracy on a provided test channel (recommended) or the training loss. For recommendations for the training/test split for the DeepAR algorithm, see [Best Practices for Using the DeepAR Algorithm \(p. 303\)](#).

Metric Name	Description	Optimization Direction
<code>test:RMSE</code>	The root mean square error between the forecast and the actual target computed on the test set.	Minimize

Metric Name	Description	Optimization Direction
test:mean_wQuantileLoss	The average overall quantile losses computed on the test set. To control which quantiles are used, set the <code>test_quantiles</code> hyperparameter.	Minimize
train:final_loss	The training negative log-likelihood loss averaged over the last training epoch for the model.	Minimize

Tunable Hyperparameters for the DeepAR Algorithm

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact, listed in order from the most to least impactful, on DeepAR objective metrics are: `epochs`, `context_length`, `mini_batch_size`, `learning_rate`, and `num_cells`.

Parameter Name	Parameter Type	Recommended Ranges
<code>mini_batch_size</code>	<code>IntegerParameterRanges</code>	MinValue: 32, MaxValue: 1028
<code>epochs</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 1000
<code>context_length</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 200
<code>num_cells</code>	<code>IntegerParameterRanges</code>	MinValue: 30, MaxValue: 200
<code>num_layers</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 8
<code>dropout_rate</code>	<code>ContinuousParameterRange</code>	MinValue: 0.00, MaxValue: 0.2
<code>embedding_dimension</code>	<code>IntegerParameterRanges</code>	MinValue: 1, MaxValue: 50
<code>learning_rate</code>	<code>ContinuousParameterRange</code>	MinValue: 1e-5, MaxValue: 1e-1

DeepAR Inference Formats

DeepAR JSON Request Formats

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

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

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

If the model was trained with a custom feature time series (`dynamic_feat`), you have to provide the same number of `dynamic_feat` values for each instance. Each of them should have a length given by `length(target) + prediction_length`, where the last `prediction_length` values correspond to

the time points in the future that will be predicted. If the model was trained without custom feature time series, the field should not be included in the request.

```
{
    "instances": [
        {
            "start": "2009-11-01 00:00:00",
            "target": [4.0, 10.0, "NaN", 100.0, 113.0],
            "cat": [0, 1],
            "dynamic_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]
        },
        {
            "start": "2012-01-30",
            "target": [1.0],
            "cat": [2, 1],
            "dynamic_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]
        },
        {
            "start": "1999-01-30",
            "target": [2.0, 1.0],
            "cat": [1, 3],
            "dynamic_feat": [[1.0, 0.1, -2.5, 0.3, 2.0, -1.2, -0.1, -3.0, ...]]
        }
    ],
    "configuration": {
        "num_samples": 50,
        "output_types": ["mean", "quantiles", "samples"],
        "quantiles": ["0.5", "0.9"]
    }
}
```

The `configuration` field is optional. `configuration.num_samples` sets the number of sample paths that the model generates to estimate the mean and quantiles. `configuration.output_types` describes the information that will be returned in the request. Valid values are `"mean"`, `"quantiles"` and `"samples"`. If you specify `"quantiles"`, each of the quantile values in `configuration.quantiles` is returned as a time series. If you specify `"samples"`, the model also returns the raw samples used to calculate the other outputs.

DeepAR JSON Response Formats

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

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

```

        "0.5": [...]
    },
    "samples": [...],
    "mean": [...]
}
]
}

```

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

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

Batch Transform with the DeepAR Algorithm

DeepAR forecasting supports getting inferences by using batch transform from data using the JSON Lines format. In this format, each record is represented on a single line as a JSON object, and lines are separated by newline characters. The format is identical to the JSON Lines format used for model training. For information, see [Input/Output Interface for the DeepAR Algorithm \(p. 300\)](#). For example:

```

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

```

Note

When creating the transformation job with [CreateTransformJob](#), set the `BatchStrategy` value to `SingleRecord` and set the `SplitType` value in the `TransformInput` configuration to `Line`, as the default values currently cause runtime failures.

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

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

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

```

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

```

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

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

Note that in the `TransformInput` configuration of the Amazon SageMaker `CreateTransformJob` request clients must explicitly set the `AssembleWith` value to `Line`, as the default value `None` concatenates all JSON objects on the same line.

For example, here is an Amazon SageMaker `CreateTransformJob` request for a DeepAR job with a custom `DEEPAR_INFERENCE_CONFIG`:

```
{
    "BatchStrategy": "SingleRecord",
    "Environment": {
        "DEEPAR_INFERENCE_CONFIG" : "{ \"num_samples\": 200, \"output_types\": [\"mean\"] }",
        ...
    },
    "TransformInput": {
        "SplitType": "Line",
        ...
    },
    "TransformOutput": {
        "AssembleWith": "Line",
        ...
    },
    ...
}
```

Factorization Machines Algorithm

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

Note

The Amazon SageMaker implementation of factorization machines considers only pair-wise (2nd order) interactions between features.

Topics

- [Input/Output Interface for the Factorization Machines Algorithm \(p. 314\)](#)
- [EC2 Instance Recommendation for the Factorization Machines Algorithm \(p. 315\)](#)
- [Factorization Machines Sample Notebooks \(p. 315\)](#)
- [How Factorization Machines Work \(p. 315\)](#)
- [Factorization Machines Hyperparameters \(p. 316\)](#)
- [Tune a Factorization Machines Model \(p. 321\)](#)
- [Factorization Machine Response Formats \(p. 323\)](#)

Input/Output Interface for the Factorization Machines Algorithm

The factorization machine algorithm can be run in either in binary classification mode or regression mode. In each mode, a dataset can be provided to the `test` channel along with the train channel dataset.

The scoring depends on the mode used. In regression mode, the testing dataset is scored using Root Mean Square Error (RMSE). In binary classification mode, the test dataset is scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold =0.5).

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

For **inference**, factorization machines support the `application/json` and `x-recordio-protobuf` formats.

- For the **binary classification** problem, the algorithm predicts a score and a label. The label is a number and can be either 0 or 1. The score is a number that indicates how strongly the algorithm believes that the label should be 1. The algorithm computes score first and then derives the label from the score value. If the score is greater than or equal to 0.5, the label is 1.
- For the **regression** problem, just a score is returned and it is the predicted value. For example, if Factorization Machines is used to predict a movie rating, score is the predicted rating value.

Please see [Factorization Machines Sample Notebooks \(p. 315\)](#) for more details on training and inference file formats.

EC2 Instance Recommendation for the Factorization Machines Algorithm

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

Factorization Machines Sample Notebooks

For a sample notebook that uses the Amazon SageMaker factorization machine learning algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see [An Introduction to Factorization Machines with MNIST](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Factorization Machines Work

The prediction task for a factorization machine model is to estimate a function \hat{y} from a feature set x_i to a target domain. This domain is real-valued for regression and binary for classification. The factorization machine model is supervised and so has a training dataset (x_i, y_i) available. The advantages this model presents lie in the way it uses a factorized parametrization to capture the pairwise feature interactions. It can be represented mathematically as follows:

$$\hat{y} = w_0 + \sum_i w_i x_i + \sum_i \sum_{j>i} \langle v_i, v_j \rangle x_i x_j$$

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

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

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

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

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

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

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

where:

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

For more information about loss functions for classification, see [Loss functions for classification](#).

Factorization Machines Hyperparameters

The following table contains the hyperparameters for the factorization machines algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

Parameter Name	Description
<code>feature_dim</code>	<p>The dimension of the input feature space. This could be very high with sparse input.</p> <p>Required</p> <p>Valid values: Positive integer. Suggested value range: [10000,10000000]</p>
<code>num_factors</code>	<p>The dimensionality of factorization.</p> <p>Required</p> <p>Valid values: Positive integer. Suggested value range: [2,1000], 64 is usually optimal.</p>
<code>predictor_type</code>	<p>The type of predictor.</p> <ul style="list-style-type: none"> • <code>binary_classifier</code>: For binary classification tasks. • <code>regressor</code>: For regression tasks. <p>Required</p>

Parameter Name	Description
	Valid values: String: <code>binary_classifier</code> or <code>regressor</code>
<code>bias_init_method</code>	<p>The initialization method for the bias term:</p> <ul style="list-style-type: none"> <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>. <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>-bias_init_scale</code>, <code>+bias_init_scale</code>. <code>constant</code>: Initializes the weights to a scalar value specified by <code>bias_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code></p> <p>Default value: <code>normal</code></p>
<code>bias_init_scale</code>	<p>Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>bias_init_sigma</code>	<p>The standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>bias_init_value</code>	<p>The initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>bias_lr</code>	<p>The learning rate for the bias term.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.1</p>

Parameter Name	Description
bias_wd	<p>The weight decay for the bias term.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
clip_gradient	<p>Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval [-clip_gradient, +clip_gradient].</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: None</p>
epochs	<p>The number of training epochs to run.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1</p>
eps	<p>Epsilon parameter to avoid division by 0.</p> <p>Optional</p> <p>Valid values: Float. Suggested value: small.</p> <p>Default value: None</p>
factors_init_method	<p>The initialization method for factorization terms:</p> <ul style="list-style-type: none"> • <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>factors_init_sigma</code>. • <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>[-factors_init_scale, +factors_init_scale]</code>. • <code>constant</code>: Initializes the weights to a scalar value specified by <code>factors_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>

Parameter Name	Description
<code>factors_init_scale</code>	<p>The range for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>factors_init_sigma</code>	<p>The standard deviation for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>
<code>factors_init_value</code>	<p>The initial value of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>factors_lr</code>	<p>The learning rate for factorization terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.0001</p>
<code>factors_wd</code>	<p>The weight decay for factorization terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.00001</p>
<code>linear_lr</code>	<p>The learning rate for linear terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>

Parameter Name	Description
<code>linear_init_method</code>	<p>The initialization method for linear terms:</p> <ul style="list-style-type: none"> • <code>normal</code> Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>. • <code>uniform</code> Initializes weights with random values uniformly sampled from a range specified by <code>[-linear_init_scale, +linear_init_scale]</code>. • <code>constant</code> Initializes the weights to a scalar value specified by <code>linear_init_value</code>. <p>Optional</p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>
<code>linear_init_scale</code>	<p>Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_init_sigma</code>	<p>The standard deviation for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>linear_init_value</code>	<p>The initial value of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>constant</code>.</p> <p>Optional</p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_wd</code>	<p>The weight decay for linear terms.</p> <p>Optional</p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>

Parameter Name	Description
<code>mini_batch_size</code>	<p>The size of mini-batch used for training.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>rescale_grad</code>	<p>Gradient rescaling optimizer parameter. If set, multiplies the gradient with <code>rescale_grad</code> before updating. Often choose to be $1.0/\text{batch_size}$.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: None</p>

Tune a Factorization Machines Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Factorization Machines Algorithm

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

Metric Name	Description	Optimization Direction
<code>test:rmse</code>	Root Mean Square Error	Minimize

The factorization machines algorithm reports three binary classification metrics, which are computed during training. When tuning the model for binary classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
<code>test:binary_accuracy</code>	Accuracy	Maximize
<code>test:binary_entropy</code>	Cross Entropy	Minimize
<code>test:binary_f_beta</code>	Beta	Maximize

Tunable Factorization Machines Hyperparameters

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

Parameter Name	Parameter Type	Recommended Ranges	Dependency
<code>bias_init_scale</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==uniform</code>
<code>bias_init_sigma</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==normal</code>
<code>bias_init_value</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==constant</code>
<code>bias_lr</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
<code>bias_wd</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
<code>epoch</code>	IntegerParameterRange	MinValue: 1, MaxValue: 1000	None
<code>factors_init_scale</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==uniform</code>
<code>factors_init_sigma</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==normal</code>
<code>factors_init_value</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==constant</code>
<code>factors_lr</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
<code>factors_wd</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512]	None
<code>linear_init_scale</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==uniform</code>
<code>linear_init_sigma</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==normal</code>
<code>linear_init_value</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==constant</code>
<code>linear_lr</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None

Parameter Name	Parameter Type	Recommended Ranges	Dependency
linear_wd	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
mini_batch_size	IntegerParameterRange	MinValue: 100, MaxValue: 10000	None

Factorization Machine Response Formats

JSON Response Format

Binary classification

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

Regression

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

JSONLINES Response Format

Binary classification

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

Regression

```
{"score": 0.4}
```

RECORDIO Response Format

Binary classification

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

```
        values: [0.0] # float32
    }
}
]
```

Regression

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

Image Classification Algorithm

The Amazon SageMaker image classification algorithm is a supervised learning algorithm that supports multi-label classification. It takes an image as input and outputs one or more labels assigned to that image. 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. Refer to [this discussion](#) for a broad overview of efficient data preparation and loading for machine learning systems.

Note

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

For more information on convolutional networks, see:

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

Topics

- [Input/Output Interface for the Image Classification Algorithm \(p. 324\)](#)
- [EC2 Instance Recommendation for the Image Classification Algorithm \(p. 326\)](#)
- [Image Classification Sample Notebooks \(p. 326\)](#)
- [How Image Classification Works \(p. 327\)](#)
- [Image Classification Hyperparameters \(p. 327\)](#)
- [Tune an Image Classification Model \(p. 333\)](#)

Input/Output Interface for the Image Classification Algorithm

The Amazon SageMaker Image Classification algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode and supports RecordIO (`application/x-recordio`) content type for training in pipe mode. However you can also train in pipe mode using the image files (`image/png`, `image/jpeg`,

and application/x-image), without creating RecordIO files, by using the augmented manifest format. Distributed training is currently not supported in pipe mode and can only be used in file mode. The algorithm supports image/png, image/jpeg, and application/x-image for inference.

Train with RecordIO Format

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

Train with Image Format

If you use the Image format for training, specify `train`, `validation`, `train_1st`, and `validation_1st` channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` request. Specify the individual image data (.jpg or .png files) for the `train` and `validation` channels. Specify one .1st file in each of the `train_1st` and `validation_1st` channels. Set the content type for all four channels to `application/x-image`.

Note

Amazon SageMaker reads the training and validation data separately from different channels, so you must store the training and validation data in different folders.

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

The following is an example of a .1st file:

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

For example, if your training images are stored in `s3://<your_bucket>/train/class_dog`, `s3://<your_bucket>/train/class_cat`, and so on, specify the path for your `train` channel as `s3://<your_bucket>/train`, which is the top-level directory for your data. In the .1st file, specify the relative path for an individual file named `train_image_dog1.jpg` in the `class_dog` class directory as `class_dog/train_image_dog1.jpg`. You can also store all your image files under one subdirectory inside the `train` directory. In that case, use that subdirectory for the relative path. For example, `s3://<your_bucket>/train/your_image_directory`.

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. You need to specify both `train` and `validation` channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in `JSON Lines` format in which each line represents one sample. The images are specified using the `'source-ref'` tag that points to the S3 location of the image. The annotations are provided under the `"AttributeNames"` parameter value as specified in the `CreateTrainingJob` request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the following example, the `"AttributeNames"` are contained in the list of image and annotation references `["source-ref", "class"]`. The corresponding label value is `"0"` for the first image and `"1"` for the second image:

```
{"source-ref": "s3://image/filename1.jpg", "class": "0"}
{"source-ref": "s3://image/filename2.jpg", "class": "1", "class-metadata": {"class-name": "cat", "type": "groundtruth/image-classification"}}
```

The order of "AttributeNames" in the input files matters when training the ImageClassification algorithm. It accepts piped data in a specific order, with `image` first, followed by `label`. So the "AttributeNames" in this example are provided with "`source-ref`" first, followed by "`class`". When using the ImageClassification algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be "`RecordIO`".

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Incremental Training

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

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

For a sample notebook that shows how to use incremental training with the Amazon SageMaker image classification algorithm, see the [End-to-End Incremental Training Image Classification Example](#). For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 613\)](#).

Inference with the Image Format Algorithm

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

```
accept: application/jsonlines
{"prediction": [prob_0, prob_1, prob_2, prob_3, ...]}
```

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

EC2 Instance Recommendation for the Image Classification Algorithm

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

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

Image Classification Sample Notebooks

For a sample notebook that uses the Amazon SageMaker image classification algorithm to train a model on the [caltech-256 dataset](#) and then to deploy it to perform inferences, see the [End-to-End Multiclass Image Classification Example](#). For instructions how to create and access Jupyter notebook instances

that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The example image classification notebooks are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Image Classification Works

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

Image classification in Amazon SageMaker can be run in two modes: full training and transfer learning. In full training mode, the network is initialized with random weights and trained on user data from scratch. In transfer learning mode, the network is initialized with pre-trained weights and just the top fully connected layer is initialized with random weights. Then, the whole network is fine-tuned with new data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data.

Image Classification Hyperparameters

Hyperparameters are parameters that are set before a machine learning model begins learning. The following hyperparameters are supported by the Amazon SageMaker built-in Image Classification algorithm. See [Tune an Image Classification Model \(p. 333\)](#) for information on image classification hyperparameter tuning.

Parameter Name	Description
<code>num_classes</code>	<p>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.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>num_training_samples</code>	<p>Number of training examples in the input dataset.</p> <p>If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter is undefined and distributed training accuracy might be affected.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>augmentation_type</code>	<p>Data augmentation type. The input images can be augmented in multiple ways as specified below.</p> <ul style="list-style-type: none">• <code>crop</code>: Randomly crop the image and flip the image horizontally

Parameter Name	Description
	<ul style="list-style-type: none"> • <code>crop_color</code>: 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 • <code>crop_color_transform</code>: In addition to <code>crop_color</code>, 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. <p>Optional</p> <p>Valid values: <code>crop</code>, <code>crop_color</code>, or <code>crop_color_transform</code>.</p> <p>Default value: no default value</p>
<code>beta_1</code>	<p>The beta1 for adam, that is the exponential decay rate for the first moment estimates.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
<code>beta_2</code>	<p>The beta2 for adam, that is the exponential decay rate for the second moment estimates.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.999</p>
<code>checkpoint_frequency</code>	<p>Period to store model parameters (in number of epochs).</p> <p>Note that all checkpoint files are saved as part of the final model file "model.tar.gz" and uploaded to S3 to the specified model location. This increases the size of the model file proportionally to the number of checkpoints saved during training.</p> <p>Optional</p> <p>Valid values: positive integer no greater than epochs.</p> <p>Default value: no default value (Save checkpoint at the epoch that has the best validation accuracy)</p>
<code>early_stopping</code>	<p>True to use early stopping logic during training. False not to use it.</p> <p>Optional</p> <p>Valid values: <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>

Parameter Name	Description
<code>early_stopping_min_epochs</code>	<p>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
<code>early_stopping_patience</code>	<p>The number of epochs to wait before ending training if no improvement is made in the relevant metric. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
<code>early_stopping_tolerance</code>	<p>Relative tolerance to measure an improvement in accuracy validation metric. If the ratio of the improvement in accuracy divided by the previous best accuracy is smaller than the <code>early_stopping_tolerance</code> value set, early stopping considers there is no improvement. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 1$</p> <p>Default value: 0.0</p>
<code>epochs</code>	<p>Number of training epochs.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 30</p>
<code>eps</code>	<p>The epsilon for <code>adam</code> and <code>rmsprop</code>. It is usually set to a small value to avoid division by 0.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 1e-8</p>
<code>gamma</code>	<p>The gamma for <code>rmsprop</code>, the decay factor for the moving average of the squared gradient.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>

Parameter Name	Description
<code>image_shape</code>	<p>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.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: '3, 224, 224'</p>
<code>kv_store</code>	<p>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.</p> <p>This parameter is not applicable to single machine training.</p> <ul style="list-style-type: none"> • <code>dist_sync</code>: The gradients are synchronized after every batch with all the workers. With <code>dist_sync</code>, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then <code>dist_sync</code> behaves like local with batch size $n \times b$ • <code>dist_async</code>: 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. <p>Optional</p> <p>Valid values: <code>dist_sync</code> or <code>dist_async</code></p> <p>Default value: no default value</p>
<code>learning_rate</code>	<p>Initial learning rate.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate used in conjunction with the <code>lr_scheduler_step</code> parameter, defined as $lr_{new} = lr_{old} * lr_scheduler_factor$.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>

Parameter Name	Description
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. As explained in the <code>lr_scheduler_factor</code> parameter, the learning rate is reduced by <code>lr_scheduler_factor</code> at these epochs. For example, if the value is set to "10, 20", then the learning rate is reduced by <code>lr_scheduler_factor</code> after 10th epoch and again by <code>lr_scheduler_factor</code> after 20th epoch. The epochs are delimited by ",".</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: no default value</p>
<code>mini_batch_size</code>	<p>The batch size for training. In a single-machine multi-GPU setting, each GPU handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. See MXNet docs for more details.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 32</p>
<code>momentum</code>	<p>The momentum for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
<code>multi_label</code>	<p>Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>

Parameter Name	Description
<code>num_layers</code>	<p>Number of layers for the network. For data with large image size (for example, 224x224 - like ImageNet), we suggest selecting the number of layers from the set [18, 34, 50, 101, 152, 200]. For data with small image size (for example, 28x28 - like CIFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200].</p> <p>Optional</p> <p>Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110]</p> <p>Default value: 152</p>
<code>optimizer</code>	<p>The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API.</p> <p>Optional</p> <p>Valid values: One of <code>sgd</code>, <code>adam</code>, <code>rmsprop</code>, or <code>nag</code>.</p> <ul style="list-style-type: none"> • <code>sgd</code>: Stochastic gradient descent • <code>adam</code>: Adaptive momentum estimation • <code>rmsprop</code>: Root mean square propagation • <code>nag</code>: Nesterov accelerated gradient <p>Default value: <code>sgd</code></p>
<code>precision_dtype</code>	<p>The precision of the weights used for training. The algorithm can use either single precision (<code>float32</code>) or half precision (<code>float16</code>) for the weights. Using half-precision for weights results in reduced memory consumption.</p> <p>Optional</p> <p>Valid values: <code>float32</code> or <code>float16</code></p> <p>Default value: <code>float32</code></p>
<code>resize</code>	<p>Resizes the image before using it for training. The images are resized so that the shortest side has the number of pixels specified by this parameter. If the parameter is not set, then the training data is used without resizing.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: no default value</p>

Parameter Name	Description
<code>top_k</code>	<p>Reports the top-k accuracy during training. This parameter has to be greater than 1, since the top-1 training accuracy is the same as the regular training accuracy that has already been reported.</p> <p>Optional</p> <p>Valid values: positive integer larger than 1.</p> <p>Default value: no default value</p>
<code>use_pretrained_model</code>	<p>Flag to use pre-trained model for training. If set to 1, then the pretrained model with the corresponding number of layers is loaded and used for training. Only the top FC layer are reinitialized with random weights. Otherwise, the network is trained from scratch.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>use_weighted_loss</code>	<p>Flag to use weighted cross-entropy loss for multi-label classification (used only when <code>multi_label = 1</code>), where the weights are calculated based on the distribution of classes.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>weight_decay</code>	<p>The coefficient weight decay for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.0001</p>

Tune an Image Classification Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Image Classification Algorithm

The image classification algorithm is a supervised algorithm. It reports an accuracy metric that is computed during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
validation:accuracy	The ratio of the number of correct predictions to the total number of predictions made.	Maximize

Tunable Image Classification Hyperparameters

Tune an image classification model with the following hyperparameters. The hyperparameters that have the greatest impact on image classification objective metrics are: `mini_batch_size`, `learning_rate`, and `optimizer`. Tune the optimizer-related hyperparameters, such as `momentum`, `weight_decay`, `beta_1`, `beta_2`, `eps`, and `gamma`, based on the selected optimizer. For example, use `beta_1` and `beta_2` only when `adam` is the optimizer.

For more information about which hyperparameters are used in each optimizer, see [Image Classification Hyperparameters \(p. 327\)](#).

Parameter Name	Parameter Type	Recommended Ranges
<code>beta_1</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>beta_2</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>eps</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 1.0
<code>gamma</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 0.999
<code>learning_rate</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.5
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: 8, MaxValue: 512
<code>momentum</code>	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999
<code>optimizer</code>	CategoricalParameterRanges	[<code>'sgd'</code> , <code>'adam'</code> , <code>'rmsprop'</code> , <code>'nag'</code>]
<code>weight_decay</code>	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999

IP Insights Algorithm

Amazon SageMaker IP Insights is an unsupervised learning algorithm that learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers. You can use it to identify a user attempting to log into a web service from an anomalous IP address, for example. Or you can use it to identify an account that is attempting to create computing resources from an unusual IP address. Trained IP Insight models can be hosted at an endpoint for making real-time predictions or used for processing batch transforms.

Amazon SageMaker IP insights ingests historical data as (entity, IPv4 Address) pairs and learns the IP usage patterns of each entity. When queried with an (entity, IPv4 Address) event, an Amazon SageMaker

IP Insights model returns a score that infers how anomalous the pattern of the event is. For example, when a user attempts to log in from an IP address, if the IP Insights score is high enough, a web login server might decide to trigger a multi-factor authentication system. In more advanced solutions, you can feed the IP Insights score into another machine learning model. For example, you can combine the IP Insight score with other features to rank the findings of another security system, such as those from [Amazon GuardDuty](#).

The Amazon SageMaker IP Insights algorithm can also learn vector representations of IP addresses, known as *embeddings*. You can use vector-encoded embeddings as features in downstream machine learning tasks that use the information observed in the IP addresses. For example, you can use them in tasks such as measuring similarities between IP addresses in clustering and visualization tasks.

Topics

- [Input/Output Interface for the IP Insights Algorithm \(p. 335\)](#)
- [EC2 Instance Recommendation for the IP Insights Algorithm \(p. 335\)](#)
- [IP Insights Sample Notebooks \(p. 336\)](#)
- [How IP Insights Works \(p. 336\)](#)
- [IP Insights Hyperparameters \(p. 337\)](#)
- [Tune an IP Insights Model \(p. 340\)](#)
- [IP Insights Data Formats \(p. 342\)](#)

Input/Output Interface for the IP Insights Algorithm

Training and Validation

The Amazon SageMaker IP Insights algorithm supports training and validation data channels. It uses the optional validation channel to compute an area-under-the-curve (AUC) score on a predefined negative sampling strategy. The AUC metric validates how well the model discriminates between positive and negative samples. Training and validation data content types need to be in `text/csv` format. The first column of the CSV data is an opaque string that provides a unique identifier for the entity. The second column is an IPv4 address in decimal-dot notation. IP Insights currently supports only File mode. For more information and some examples, see [IP Insights Training Data Formats \(p. 342\)](#).

Inference

For inference, IP Insights supports `text/csv`, `application/json`, and `application/jsonlines` data content types. For more information about the common data formats for inference provided by Amazon SageMaker, see [Common Data Formats for Inference \(p. 284\)](#). IP Insights inference returns output formatted as either `application/json` or `application/jsonlines`. Each record in the output data contains the corresponding `dot_product` (or compatibility score) for each input data point. For more information and some examples, see [IP Insights Inference Data Formats \(p. 342\)](#).

EC2 Instance Recommendation for the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm can run on both GPU and CPU instances. For training jobs, we recommend using GPU instances. However, for certain workloads with large training datasets, distributed CPU instances might reduce training costs. For inference, we recommend using CPU instances.

GPU Instances for the IP Insights Algorithm

IP Insights supports all available GPUs. If you need to speed up training, we recommend starting with a single GPU instance, such as `ml.p3.2xlarge`, and then moving to a multi-GPU environment, such as `ml.p3.8xlarge` and `ml.p3.16xlarge`. Multi-GPUs automatically divide the mini batches of training data across themselves. If you switch from a single GPU to multiple GPUs, the `mini_batch_size` is divided equally into the number of GPUs used. You may want to increase the value of the `mini_batch_size` to compensate for this.

CPU Instances for the IP Insights Algorithm

The type of CPU instance that we recommend depends largely on the instance's available memory and the model size. The model size is determined by two hyperparameters: `vector_dim` and `num_entity_vectors`. The maximum supported model size is 8 GB. The following table lists typical EC2 instance types that you would deploy based on these input parameters for various model sizes. In Table 1, the value for `vector_dim` in the first column range from 32 to 2048 and the values for `num_entity_vectors` in the first row range from 10,000 to 50,000,000.

<code>vector_dim</code> \ <code>num_entity_vectors</code>	10,000	50,000	100,000	500,000	1,000,000	5,000,000	10,000,000	50,000,000
32	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.2xlarge	ml.m5.4xlarge
64	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge	
128	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge	
256	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge		
512	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge			
1024	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge			
2048	ml.m5.large	ml.m5.large	ml.m5.xlarge					

The values for the `mini_batch_size`, `num_ip_encoder_layers`, `random_negative_sampling_rate`, and `shuffled_negative_sampling_rate` hyperparameters also affect the amount of memory required. If these values are large, you might need to use a larger instance type than normal.

IP Insights Sample Notebooks

For a sample notebook that shows how to train the Amazon SageMaker IP Insights algorithm and perform inferences with it, see [An Introduction to the Amazon SageMaker IP Insights Algorithm](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After creating a notebook instance, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker examples. To open a notebook, choose its **Use** tab and choose **Create copy**.

How IP Insights Works

Amazon SageMaker IP Insights is an unsupervised algorithm that consumes observed data in the form of (entity, IPv4 address) pairs that associates entities with IP addresses. IP Insights determines how likely it is that an entity would use a particular IP address by learning latent vector representations for both entities and IP addresses. The distance between these two representations can then serve as the proxy for how likely this association is.

The IP Insights algorithm uses a neural network to learn the latent vector representations for entities and IP addresses. Entities are first hashed to a large but fixed hash space and then encoded by a simple embedding layer. Character strings such as user names or account IDs can be fed directly into IP Insights as they appear in log files. You don't need to preprocess the data for entity identifiers. You can provide entities as an arbitrary string value during both training and inference. The hash size should be configured with a value that is high enough to insure that the number of *collisions*, which occur when distinct entities are mapped to the same latent vector, remain insignificant. For more information about how to select appropriate hash sizes, see [Feature Hashing for Large Scale Multitask Learning](#). For

representing IP addresses, on the other hand, IP Insights uses a specially designed encoder network to uniquely represent each possible IPv4 address by exploiting the prefix structure of IP addresses.

During training, IP Insights automatically generates negative samples by randomly pairing entities and IP addresses. These negative samples represent data that is less likely to occur in reality. The model is trained to discriminate between positive samples that are observed in the training data and these generated negative samples. More specifically, the model is trained to minimize the *cross entropy*, also known as the *log loss*, defined as follows:

$$L = \frac{1}{N} \sum_n [y_n \log p_n + (1 - y_n) \log (1 - p_n)]$$

y_n is the label that indicates whether the sample is from the real distribution governing observed data ($y_n=1$) or from the distribution generating negative samples ($y_n=0$). p_n is the probability that the sample is from the real distribution, as predicted by the model.

Generating negative samples is an important process that is used to achieve an accurate model of the observed data. If negative samples are extremely unlikely, for example, if all of the IP addresses in negative samples are 10.0.0.0, then the model trivially learns to distinguish negative samples and fails to accurately characterize the actual observed dataset. To keep negative samples more realistic, IP Insights generates negative samples both by randomly generating IP addresses and randomly picking IP addresses from training data. You can configure the type of negative sampling and the rates at which negative samples are generated with the `random_negative_sampling_rate` and `shuffled_negative_sampling_rate` hyperparameters.

Given an nth (entity, IP address pair), the IP Insights model outputs a *score*, S_n , that indicates how compatible the entity is with the IP address. This score corresponds to the log odds ratio for a given (entity, IP address) of the pair coming from a real distribution as compared to coming from a negative distribution. It is defined as follows:

$$S_n = \log \left(\frac{P_{real}(n)}{P_{neg}(n)} \right)$$

The score is essentially a measure of the similarity between the vector representations of the nth entity and IP address. It can be interpreted as how much more likely it would be to observe this event in reality than in a randomly generated dataset. During training, the algorithm uses this score to calculate an estimate of the probability of a sample coming from the real distribution, p_n , to use in the cross entropy minimization, where:

$$p_n = \frac{1}{1 + e^{-S_n}}$$

IP Insights Hyperparameters

In the `CreateTransformJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker IP Insights algorithm.

Parameter Name	Description
<code>num_entity_vectors</code>	The number of entity vector representations (entity embedding vectors) to train. Each entity in the training set is randomly assigned to one of these vectors using a hash function. Because of hash collisions, it might be possible to have multiple entities assigned to the same vector. This would cause the same vector to represent multiple entities. This generally has a negligible effect on model performance, as long as the collision rate is not too severe. To keep the collision rate low, set this value as

Parameter Name	Description
	<p>high as possible. However, the model size, and, therefore, the memory requirement, for both training and inference, scales linearly with this hyperparameter. We recommend that you set this value to twice the number of unique entity identifiers.</p> <p>Required</p> <p>Valid values: $1 \leq \text{positive integer} \leq 250,000,000$</p>
<code>vector_dim</code>	<p>The size of embedding vectors to represent entities and IP addresses. The larger the value, the more information that can be encoded using these representations. In practice, model size scales linearly with this parameter and limits how large the dimension can be. In addition, using vector representations that are too large can cause the model to overfit, especially for small training datasets. Overfitting occurs when a model doesn't learn any pattern in the data but effectively memorizes the training data and, therefore, cannot generalize well and performs poorly during inference. The recommended value is 128.</p> <p>Required</p> <p>Valid values: $4 \leq \text{positive integer} \leq 4096$</p>
<code>batch_metrics_publish_interval</code>	<p>The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second).</p> <p>Optional</p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 1,000</p>
<code>epochs</code>	<p>The number of passes over the training data. The optimal value depends on your data size and learning rate. Typical values range from 5 to 100.</p> <p>Optional</p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 10</p>

Parameter Name	Description
<code>learning_rate</code>	<p>The learning rate for the optimizer. IP Insights use a gradient-descent-based Adam optimizer. The learning rate effectively controls the step size to update model parameters at each iteration. Too large a learning rate can cause the model to diverge because the training is likely to overshoot a minima. On the other hand, too small a learning rate slows down convergence. Typical values range from 1e-4 to 1e-1.</p> <p>Optional</p> <p>Valid values: $1e-6 \leq \text{float} \leq 10.0$</p> <p>Default value: 0.001</p>
<code>mini_batch_size</code>	<p>The number of examples in each mini batch. The training procedure processes data in mini batches. The optimal value depends on the number of unique account identifiers in the dataset. In general, the larger the <code>mini_batch_size</code>, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large <code>mini_batch_size</code>, the training is more likely to converge to a poor local minimum and perform relatively worse for inference.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{positive integer} \leq 500000$</p> <p>Default value: 10,000</p>
<code>num_ip_encoder_layers</code>	<p>The number of fully connected layers used to encode the IP address embedding. The larger the number of layers, the greater the model's capacity to capture patterns among IP addresses. However, using a large number of layers increases the chance of overfitting.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 100$</p> <p>Default value: 1</p>

Parameter Name	Description
<code>random_negative_sampling_rate</code>	<p>The number of random negative samples, R, to generate per input example. The training procedure relies on negative samples to prevent the vector representations of the model collapsing to a single point. Random negative sampling generates R random IP addresses for each input account in the mini batch. The sum of the <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: $1 \leq R + S \leq 500$.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 500$</p> <p>Default value: 1</p>
<code>shuffled_negative_sampling_rate</code>	<p>The number of shuffled negative samples, S, to generate per input example. In some cases, it helps to use more realistic negative samples that are randomly picked from the training data itself. This kind of negative sampling is achieved by shuffling the data within a mini batch. Shuffled negative sampling generates S negative IP addresses by shuffling the IP address and account pairings within a mini batch. The sum of the <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: $1 \leq R + S \leq 500$.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{positive integer} \leq 500$</p> <p>Default value: 1</p>
<code>weight_decay</code>	<p>The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data.</p> <p>Optional</p> <p>Valid values: $0.0 \leq \text{float} \leq 10.0$</p> <p>Default value: 0.00001</p>

Tune an IP Insights Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm is an unsupervised learning algorithm that learns associations between IP addresses and entities. The algorithm trains a discriminator model , which learns to separate observed data points (*positive samples*) from randomly generated data points (*negative samples*). Automatic model tuning on IP Insights helps you find the model that can most accurately distinguish between unlabeled validation data and automatically generated negative samples. The model accuracy on the validation dataset is measured by the area under the receiver operating characteristic (ROC) curve. This validation:discriminator_auc metric can take values between 0.0 and 1.0, where 1.0 indicates perfect accuracy.

The IP Insights algorithm computes a validation:discriminator_auc metric during validation, the value of which is used as the objective function to optimize for hyperparameter tuning.

Metric Name	Description	Optimization Direction
validation:discriminator_auc	Area under the ROC curve on the validation dataset. The validation dataset is not labeled. AUC is a metric that describes the model's ability to discriminate validation data points from randomly generated data points.	Maximize

Tunable IP Insights Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker IP Insights algorithm.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRange	MinValue: 1, MaxValue: 100
learning_rate	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1
mini_batch_size	IntegerParameterRanges	MinValue: 100, MaxValue: 50000
num_entity_vectors	IntegerParameterRanges	MinValue: 10000, MaxValue: 1000000
num_ip_encoder_layer	IntegerParameterRanges	MinValue: 1, MaxValue: 10
random_negative_samples	IntegerParameterRanges	MinValue: 0, MaxValue: 10
shuffled_negative_samples	IntegerParameterRanges	MinValue: 0, MaxValue: 10
vector_dim	IntegerParameterRanges	MinValue: 8, MaxValue: 256
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0

IP Insights Data Formats

This section provides examples of the available input and output data formats used by the IP Insights algorithm during training and inference.

Topics

- [IP Insights Training Data Formats \(p. 342\)](#)
- [IP Insights Inference Data Formats \(p. 342\)](#)

IP Insights Training Data Formats

The following are the available data input formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input training format described in [Common Data Formats for Training \(p. 280\)](#). However, the Amazon SageMaker IP Insights algorithm currently supports only the CSV data input format.

IP Insights Training Data Input Formats

INPUT: CSV

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

IP Insights Inference Data Formats

The following are the available input and output formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats for Inference \(p. 284\)](#). However, the Amazon SageMaker IP Insights algorithm does not currently support RecordIO format.

IP Insights Input Request Formats

INPUT: CSV Format

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

INPUT: JSON Format

JSON data can be provided in different formats. IP Insights follows the common Amazon SageMaker formats. For more information about inference formats, see [Common Data Formats for Inference \(p. 284\)](#).

content-type: application/json

```
{
```

```

    "instances": [
        {"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},
        {"features": ["entity_id_2", "10.10.1.2"]}
    ]
}

```

INPUT: JSONLINES Format

The JSON Lines content type is useful for running batch transform jobs. For more information on Amazon SageMaker inference formats, see [Common Data Formats for Inference \(p. 284\)](#). For more information on running batch transform jobs, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#).

content-type: application/jsonlines

```
{"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},
{"features": ["entity_id_2", "10.10.1.2"]}]
```

IP Insights Output Response Formats

OUTPUT: JSON Response Format

The default output of the Amazon SageMaker IP Insights algorithm is the dot_product between the input entity and IP address. The dot_product signifies how compatible the model considers the entity and IP address. The dot_product is unbounded. To make predictions about whether an event is anomalous, you need to set a threshold based on your defined distribution. For information about how to use the dot_product for anomaly detection, see the [An Introduction to the Amazon SageMaker IP Insights Algorithm](#).

accept: application/json

```
{
    "predictions": [
        {"dot_product": 0.0},
        {"dot_product": 2.0}
    ]
}
```

Advanced users can access the model's learned entity and IP embeddings by providing the additional content-type parameter `verbose=True` to the Accept heading. You can use the `entity_embedding` and `ip_embedding` for debugging, visualizing, and understanding the model. Additionally, you can use these embeddings in other machine learning techniques, such as classification or clustering.

accept: application/json;verbose=True

```
{
    "predictions": [
        {
            "dot_product": 0.0,
            "entity_embedding": [1.0, 0.0, 0.0],
            "ip_embedding": [0.0, 1.0, 0.0]
        },
        {
            "dot_product": 2.0,
            "entity_embedding": [1.0, 0.0, 1.0],
            "ip_embedding": [1.0, 0.0, 1.0]
        }
    ]
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"dot_product": 0.0}  
{"dot_product": 2.0}
```

accept: application/jsonlines; verbose=True

```
{"dot_product": 0.0, "entity_embedding": [1.0, 0.0, 0.0], "ip_embedding": [0.0, 1.0, 0.0]}  
{"dot_product": 2.0, "entity_embedding": [1.0, 0.0, 1.0], "ip_embedding": [1.0, 0.0, 1.0]}
```

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 \(p. 345\)](#).

Topics

- [Input/Output Interface for the K-Means Algorithm \(p. 344\)](#)
- [EC2 Instance Recommendation for the K-Means Algorithm \(p. 345\)](#)
- [K-Means Sample Notebooks \(p. 345\)](#)
- [How K-Means Clustering Works \(p. 345\)](#)
- [K-Means Hyperparameters \(p. 348\)](#)
- [Tune a K-Means Model \(p. 350\)](#)
- [K-Means Response Formats \(p. 351\)](#)

Input/Output Interface for the K-Means Algorithm

For training, the k-means algorithm expects data to be provided in the `train` channel (recommended `S3DataDistributionType=ShardedByS3Key`), with an optional `test` channel (recommended `S3DataDistributionType=FullyReplicated`) to score the data on. Both `recordIO-wrapped-protobuf` and `csv` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, `text/csv`, `application/json`, and `application/x-recordio-protobuf` are supported. k-means returns a `closest_cluster` label and the `distance_to_cluster` for each observation.

For more information on input and output file formats, see [K-Means Response Formats \(p. 351\)](#) for inference and the [K-Means Sample Notebooks \(p. 345\)](#). The k-means algorithm does not support multiple instance learning, in which the training set consists of labeled “bags”, each of which is a collection of unlabeled instances.

EC2 Instance Recommendation for the K-Means Algorithm

We recommend training k-means on CPU instances. You can train on GPU instances, but should limit GPU training to `p*.xlarge` instances because only one GPU per instance is used.

K-Means Sample Notebooks

For a sample notebook that uses the Amazon SageMaker K-means algorithm to segment the population of counties in the United States by attributes identified using principle component analysis, see [Analyze US census data for population segmentation using Amazon SageMaker](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

How K-Means Clustering Works

K-means is an algorithm that trains a model that groups similar objects together. The k-means algorithm accomplishes this by mapping each observation in the input dataset to a point in the n -dimensional space (where n is the number of attributes of the observation). For example, your dataset might contain observations of temperature and humidity in a particular location, which are mapped to points (t, h) in 2-dimensional space.

Note

Clustering algorithms are unsupervised. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

In k-means clustering, each cluster has a center. During model training, the k-means algorithm uses the distance of the point that corresponds to each observation in the dataset to the cluster centers as the basis for clustering. You choose the number of clusters (k) to create.

For example, suppose that you want to create a model to recognize handwritten digits and you choose the MNIST dataset for training. The dataset provides thousands of images of handwritten digits (0 through 9). In this example, you might choose to create 10 clusters, one for each digit (0, 1, ..., 9). As part of model training, the k-means algorithm groups the input images into 10 clusters.

Each image in the MNIST dataset is a 28x28-pixel image, with a total of 784 pixels. Each image corresponds to a point in a 784-dimensional space, similar to a point in a 2-dimensional space (x,y) . To find a cluster to which a point belongs, the k-means algorithm finds the distance of that point from all of the cluster centers. It then chooses the cluster with the closest center as the cluster to which the image belongs.

Note

Amazon SageMaker uses a customized version of the algorithm where, instead of specifying that the algorithm create k clusters, you might choose to improve model accuracy by specifying extra cluster centers ($K = k*x$). However, the algorithm ultimately reduces these to k clusters.

In Amazon SageMaker, you specify the number of clusters when creating a training job. For more information, see [CreateTrainingJob](#). In the request body, you add the `HyperParameters` string map to specify the `k` and `extra_center_factor` strings.

The following is a summary of how k-means works for model training in Amazon SageMaker:

1. It determines the initial K cluster centers.

Note

In the following topics, K clusters refer to $k * x$, where you specify k and x when creating a model training job.

2. It iterates over input training data and recalculates cluster centers.

3. It reduces resulting clusters to k (if the data scientist specified the creation of $k \times$ 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 \(p. 346\)](#)
- [Step 2: Iterate over the Training Dataset and Calculate Cluster Centers \(p. 347\)](#)
- [Step 3: Reduce the Clusters from K to k \(p. 347\)](#)

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](#). For an example, see [Create and Run a Training Job \(AWS SDK for Python \(Boto 3\)\) \(p. 47\)](#).

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:

```
p1 = proportion of points assigned to c1 = 25/(100+25)
p2 = proportion of points assigned to c2 = 40/(150+40)
p3 = proportion of points assigned to c3 = 5/(450+5)
```

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

```
d1 = center of the new points added to cluster 1
d2 = center of the new points added to cluster 2
d3 = center of the new points added to cluster 3
```

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

```
Center of cluster 1 = ((1 - p1) * center of cluster 1) + (p1 * d1)
Center of cluster 2 = ((1 - p2) * center of cluster 2) + (p2 * d2)
Center of cluster 3 = ((1 - p3) * center of cluster 3) + (p3 * d3)
```

2. Read the next mini-batch, and repeat Step 1 to recalculate the cluster centers.

3. For more information about mini-batch k -means, see [Web-Scale \$k\$ -means Clustering](#)).

Step 3: Reduce the Clusters from K to k

If the algorithm created K clusters—($K = k*x$) 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 \(p. 345\)](#).

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. Required Valid values: Positive integer
<code>k</code>	The number of required clusters. Required Valid values: Positive integer
<code>epochs</code>	The number of passes done over the training data. Optional Valid values: Positive integer Default value: 1
<code>eval_metrics</code>	A JSON list of metric types used to report a score for the model. Allowed values are <code>msd</code> for Means Square Error and <code>ssd</code> for Sum of Square Distance. If test data is provided, the score is reported for each of the metrics requested. Optional Valid values: Either <code>[\"msd\"]</code> or <code>[\"ssd\"]</code> or <code>[\"msd\", \"ssd\"]</code> . Default value: <code>[\"msd\"]</code>
<code>extra_center_factor</code>	The algorithm creates K centers = <code>num_clusters * extra_center_factor</code> as it runs and reduces the number of centers from K to <code>k</code> when finalizing the model. Optional Valid values: Either a positive integer or <code>auto</code> . Default value: <code>auto</code>
<code>half_life_time_size</code>	Used to determine the weight given to an observation when computing a cluster mean. This weight decays exponentially as more points are observed. When a point is first observed, it is assigned a weight of 1 when computing the cluster mean. The decay constant for the exponential decay function is chosen so that after observing <code>half_life_time_size</code> points, its weight is 1/2. If set to 0, there is no decay. Optional

Parameter Name	Description
	<p>Valid values: Non-negative integer</p> <p>Default value: 0</p>
<code>init_method</code>	<p>Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An alternative k-means++ method chooses the first cluster center at random. Then it spreads out the position of the remaining initial clusters by weighting the selection of centers with a probability distribution that is proportional to the square of the distance of the remaining data points from existing centers.</p> <p>Optional</p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>random</code></p>
<code>local_lloyd_init_method</code>	<p>The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>kmeans++</code></p>
<code>local_lloyd_max_iter</code>	<p>The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 300</p>
<code>local_lloyd_num_trials</code>	<p>The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Either a positive integer or <code>auto</code>.</p> <p>Default value: <code>auto</code></p>
<code>local_lloyd_tol</code>	<p>The tolerance for change in loss for early stopping of Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p>Optional</p> <p>Valid values: Float. Range in [0, 1].</p> <p>Default value: 0.0001</p>

Parameter Name	Description
mini_batch_size	<p>The number of observations per mini-batch for the data iterator.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 5000</p>

Tune a K-Means Model

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

The Amazon SageMaker k-means algorithm is an unsupervised algorithm that groups data into clusters whose members are as similar as possible. Because it is unsupervised, it doesn't use a validation dataset that hyperparameters can optimize against. But it does take a test dataset and emits metrics that depend on the squared distance between the data points and the final cluster centroids at the end of each training run. To find the model that reports the tightest clusters on the test dataset, you can use a hyperparameter tuning job. The clusters optimize the similarity of their members.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the K-Means Algorithm

The k-means algorithm computes the following metrics during training. When tuning a model, choose one of these metrics as the objective metric.

Metric Name	Description	Optimization Direction
test:msd	Mean squared distances between each record in the test set and the closest center of the model.	Minimize
test:ssd	Sum of the squared distances between each record in the test set and the closest center of the model.	Minimize

Tunable K-Means Hyperparameters

Tune the Amazon SageMaker k-means model with the following hyperparameters. The hyperparameters that have the greatest impact on k-means objective metrics are: `mini_batch_size`, `extra_center_factor`, and `init_method`. Tuning the hyperparameter `epochs` generally results in minor improvements.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRanges	MinValue: 1, MaxValue:10
extra_center_factor	IntegerParameterRanges	MinValue: 4, MaxValue:10

Parameter Name	Parameter Type	Recommended Ranges
init_method	CategoricalParameterRanges	['kmeans++', 'random']
mini_batch_size	IntegerParameterRanges	MinValue: 3000, MaxValue:15000

K-Means Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker k-means algorithm.

JSON Response Format

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

JSONLINES Response Format

```
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```

RECORDIO Response Format

```
[
  Record = {
    features = {},
    label = {
      'closest_cluster': {
        keys: [],
        values: [1.0, 2.0] # float32
      },
      'distance_to_cluster': {
        keys: [],
        values: [3.0, 5.0] # float32
      },
    }
  }
]
```

K-Nearest Neighbors (k-NN) Algorithm

Amazon SageMaker k-nearest neighbors (k-NN) algorithm is an index-based algorithm. It uses a non-parametric method for classification or regression. For classification problems, the algorithm queries the k points that are closest to the sample point and returns the most frequently used label of their class as

the predicted label. For regression problems, the algorithm queries the k closest points to the sample point and returns the average of their feature values as the predicted value.

Training with the k-NN algorithm has three steps: sampling, dimension reduction, and index building. Sampling reduces the size of the initial dataset so that it fits into memory. For dimension reduction, the algorithm decreases the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency. We provide two methods of dimension reduction methods: random projection and the fast Johnson-Lindenstrauss transform. Typically, you use dimension reduction for high-dimensional ($d > 1000$) datasets to avoid the “curse of dimensionality” that troubles the statistical analysis of data that becomes sparse as dimensionality increases. The main objective of k-NN’s training is to construct the index. The index enables efficient lookups of distances between points whose values or class labels have not yet been determined and the k nearest points to use for inference.

Topics

- [Input/Output Interface for the k-NN Algorithm \(p. 352\)](#)
- [k-NN Sample Notebooks \(p. 353\)](#)
- [How the k-NN Algorithm Works \(p. 353\)](#)
- [EC2 Instance Recommendation for the k-NN Algorithm \(p. 354\)](#)
- [k-NN Hyperparameters \(p. 354\)](#)
- [Tune a k-NN Model \(p. 356\)](#)
- [Data Formats for k-NN Training Input \(p. 357\)](#)
- [k-NN Request and Response Formats \(p. 357\)](#)

Input/Output Interface for the k-NN Algorithm

Amazon SageMaker k-NN supports train and test data channels.

- Use a *train channel* for data that you want to sample and construct into the k-NN index.
- Use a *test channel* to emit scores in log files. Scores are listed as one line per mini-batch: accuracy for classifier, mean-squared error (mse) for regressor for score.

For training inputs, k-NN supports `text/csv` and `application/x-recordio-protobuf` data formats. For input type `text/csv`, the first `label_size` columns are interpreted as the label vector for that row. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as CSV.

For inference inputs, k-NN supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` data formats. The `text/csv` format accepts a `label_size` and `encoding` parameter. It assumes a `label_size` of 0 and a UTF-8 encoding.

For inference outputs, k-NN supports the `application/json` and `application/x-recordio-protobuf` data formats. These two data formats also support a verbose output mode. In verbose output mode, the API provides the search results with the distances vector sorted from smallest to largest, and corresponding elements in the labels vector.

For batch transform, k-NN supports the `application/jsonlines` data format for both input and output. An example input is as follows:

```
content-type: application/jsonlines

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

An example output is as follows:

```
accept: application/jsonlines

{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

For more information on input and output file formats, see [Data Formats for k-NN Training Input \(p. 357\)](#) for training, [k-NN Request and Response Formats \(p. 357\)](#) for inference, and the [k-NN Sample Notebooks \(p. 353\)](#).

k-NN Sample Notebooks

For a sample notebook that uses the Amazon SageMaker k-nearest neighbor algorithm to predict wilderness cover types from geological and forest service data, see the [K-Nearest Neighbor Covertype](#).

Use a Jupyter notebook instance to run the example in Amazon SageMaker. To learn how to create and open a Jupyter notebook instance in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker example notebooks. Find K-Nearest Neighbor notebooks in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How the k-NN Algorithm Works

Step 1: Sample

To specify the total number of data points to be sampled from the training dataset, use the `sample_size` parameter. For example, if the initial dataset has 1,000 data points and the `sample_size` is set to 100, where the total number of instances is 2, each worker would sample 50 points. A total set of 100 data points would be collected. Sampling runs in linear time with respect to the number of data points.

Step 2: Perform Dimension Reduction

The current implementation of k-NN has two methods of dimension reduction. You specify the method in the `dimension_reduction_type` hyperparameter. The `sign` method specifies a random projection, which uses a linear projection using a matrix of random signs, and the `fjlt` method specifies a fast Johnson-Lindenstrauss transform, a method based on the Fourier transform. Both methods preserve the L2 and inner product distances. The `fjlt` method should be used when the target dimension is large and has better performance with CPU inference. The methods differ in their computational complexity. The `sign` method requires $O(ndk)$ time to reduce the dimension of a batch of n points of dimension d into a target dimension k . The `fjlt` method requires $O(nd \log(d))$ time, but the constants involved are larger. Using dimension reduction introduces noise into the data and this noise can reduce prediction accuracy.

Step 3: Build an Index

During inference, the algorithm queries the index for the k-nearest-neighbors of a sample point. Based on the references to the points, the algorithm makes the classification or regression prediction. It makes the prediction based on the class labels or values provided. k-NN provides three different types of indexes: a flat index, an inverted index, and an inverted index with product quantization. You specify the type with the `index_type` parameter.

Serialize the Model

When the k-NN algorithm finishes training, it serializes three files to prepare for inference.

- `model_algo-1`: Contains the serialized index for computing the nearest neighbors.

- `model_algo-1.labels`: Contains serialized labels (np.float32 binary format) for computing the predicted label based on the query result from the index.
- `model_algo-1.json`: Contains the JSON-formatted model metadata which stores the `k` and `predictor_type` hyper-parameters from training for inference along with other relevant state.

With the current implementation of k-NN, you can modify the metadata file to change the way predictions are computed. For example, you can change `k` to 10 or change `predictor_type` to `regressor`.

```
{
  "k": 5,
  "predictor_type": "classifier",
  "dimension_reduction": {"type": "sign", "seed": 3, "target_dim": 10, "input_dim": 20},
  "normalize": False,
  "version": "1.0"
}
```

EC2 Instance Recommendation for the k-NN Algorithm

Instance Recommendation for Training with the k-NN Algorithm

To start, try running training on a CPU, using, for example, an `ml.m5.2xlarge` instance, or on a GPU using, for example, an `ml.p2.xlarge` instance.

Instance Recommendation for Inference with the k-NN Algorithm

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.

k-NN Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. Required Valid values: positive integer.
<code>k</code>	The number of nearest neighbors. Required Valid values: positive integer
<code>predictor_type</code>	The type of inference to use on the data labels. Required Valid values: <code>classifier</code> for classification or <code>regressor</code> for regression.
<code>sample_size</code>	The number of data points to be sampled from the training data set. Required Valid values: positive integer
<code>dimension_reduction_table</code>	The target dimension to reduce to.

Parameter Name	Description
	<p>Required when you specify the <code>dimension_reduction_type</code> parameter.</p> <p>Valid values: positive integer greater than 0 and less than <code>feature_dim</code>.</p>
<code>dimension_reduction_type</code>	<p>The type of dimension reduction method.</p> <p>Optional</p> <p>Valid values: <code>sign</code> for random projection or <code>fjlt</code> for the fast Johnson-Lindenstrauss transform.</p> <p>Default value: No dimension reduction</p>
<code>faiss_index_ivf_nlists</code>	<p>The number of centroids to construct in the index when <code>index_type</code> is <code>faiss.IVFFlat</code> or <code>faiss.IVFPQ</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: <code>auto</code>, which resolves to <code>sqrt(sample_size)</code>.</p>
<code>faiss_index_pq_m</code>	<p>The number of vector sub-components to construct in the index when <code>index_type</code> is set to <code>faiss.IVFPQ</code>.</p> <p>The FaceBook AI Similarity Search (FAISS) library requires that the value of <code>faiss_index_pq_m</code> is a divisor of the data dimension. If <code>faiss_index_pq_m</code> is not a divisor of the data dimension, we increase the data dimension to smallest integer divisible by <code>faiss_index_pq_m</code>. If no dimension reduction is applied, the algorithm adds a padding of zeros. If dimension reduction is applied, the algorithm increase the value of the <code>dimension_reduction_target</code> hyper-parameter.</p> <p>Optional</p> <p>Valid values: One of the following positive integers: 1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 96</p>
<code>index_metric</code>	<p>The metric to measure the distance between points when finding nearest neighbors. When training with <code>index_type</code> set to <code>faiss.IVFPQ</code>, the <code>INNER_PRODUCT</code> distance and <code>COSINE</code> similarity are not supported.</p> <p>Optional</p> <p>Valid values: <code>L2</code> for Euclidean-distance, <code>INNER_PRODUCT</code> for inner-product distance, <code>COSINE</code> for cosine similarity.</p> <p>Default value: <code>L2</code></p>
<code>index_type</code>	<p>The type of index.</p> <p>Optional</p> <p>Valid values: <code>faiss.Flat</code>, <code>faiss.IVFFlat</code>, <code>faiss.IVFPQ</code>.</p> <p>Default values: <code>faiss.Flat</code></p>

Parameter Name	Description
mini_batch_size	<p>The number of observations per mini-batch for the data iterator.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5000</p>

Tune a k-NN Model

The Amazon SageMaker k-nearest neighbors algorithm is a supervised algorithm. The algorithm consumes a test data set and emits a metric about the accuracy for a classification task or about the mean squared error for a regression task. These accuracy metrics compare the model predictions for their respective task to the ground truth provided by the empirical test data. To find the best model that reports the highest accuracy or lowest error on the test dataset, run a hyperparameter tuning job for k-NN.

Automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric appropriate for the prediction task of the algorithm. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric. The hyperparameters are used only to help estimate model parameters and are not used by the trained model to make predictions.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the k-NN Algorithm

The k-nearest neighbors algorithm computes one of two metrics in the following table during training depending on the type of task specified by the `predictor_type` hyper-parameter.

- *classifier* specifies a classification task and computes `test:accuracy`
- *regressor* specifies a regression task and computes `test:mse`.

Choose the `predictor_type` value appropriate for the type of task undertaken to calculate the relevant objective metric when tuning a model.

Metric Name	Description	Optimization Direction
<code>test:accuracy</code>	When <code>predictor_type</code> is set to <i>classifier</i> , k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The accuracy reported ranges from 0.0 (0%) to 1.0 (100%).	Maximize
<code>test:mse</code>	When <code>predictor_type</code> is set to <i>regressor</i> , k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The mean squared error is computed by comparing the two labels.	Minimize

Tunable k-NN Hyperparameters

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
k	IntegerParameterRanges	MinValue: 1, MaxValue: 1024
sample_size	IntegerParameterRanges	MinValue: 256, MaxValue: 20000000

Data Formats for k-NN Training Input

All Amazon SageMaker built-in algorithms adhere to the common input training formats described in [Common Data Formats - Training](#). This topic contains a list of the available input formats for the Amazon SageMaker k-nearest-neighbor algorithm.

CSV Data Format

content-type: text/csv; label_size=1

```
4,1.2,1.3,9.6,20.3
```

The first `label_size` columns are interpreted as the label vector for that row.

RECORDIO Data Format

content-type: application/x-recordio-protobuf

```
[  
    Record = {  
        features = {  
            'values': {  
                values: [1.2, 1.3, 9.6, 20.3] # float32  
            }  
        },  
        label = {  
            'values': {  
                values: [4] # float32  
            }  
        }  
    }  
]
```

k-NN Request and Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker k-nearest-neighbor algorithm.

INPUT: CSV Request Format

content-type: text/csv

```
1.2,1.3,9.6,20.3
```

This accepts a `label_size` or `encoding` parameter. It assumes a `label_size` of 0 and a utf-8 encoding.

INPUT: JSON Request Format

content-type: application/json

```
{  
    "instances": [  
        {"data": {"features": {"values": [-3, -1, -4, 2]}},  
        {"features": [3.0, 0.1, 0.04, 0.002]}]  
    ]  
}
```

INPUT: JSONLINES Request Format

content-type: application/jsonlines

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

INPUT: RECORDIO Request Format

content-type: application/x-recordio-protobuf

```
[  
    Record = {  
        features = {  
            'values': {  
                values: [-3, -1, -4, 2] # float32  
            }  
        },  
        label = {}  
    },  
    Record = {  
        features = {  
            'values': {  
                values: [3.0, 0.1, 0.04, 0.002] # float32  
            }  
        },  
        label = {}  
    },  
]
```

OUTPUT: JSON Response Format

accept: application/json

```
{  
    "predictions": [  
        {"predicted_label": 0.0},  
        {"predicted_label": 2.0}  
    ]  
}
```

OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"predicted_label": 0.0}  
{"predicted_label": 2.0}
```

OUTPUT: VERBOSE JSON Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/json; verbose=true

```
{
  "predictions": [
    {
      "predicted_label": 0.0,
      "distances": [3.11792408, 3.89746071, 6.32548437],
      "labels": [0.0, 1.0, 0.0]
    },
    {
      "predicted_label": 2.0,
      "distances": [1.08470316, 3.04917915, 5.25393973],
      "labels": [2.0, 2.0, 0.0]
    }
  ]
}
```

OUTPUT: RECORDIO-PROTOBUF Response Format

content-type: application/x-recordio-protobuf

```
[
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [0.0] # float32
      }
    }
  },
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [2.0] # float32
      }
    }
  }
]
```

OUTPUT: VERBOSE RECORDIO-PROTOBUF Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/x-recordio-protobuf; verbose=true

```
[
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [0.0] # float32
      },
      'distances': {
        values: [3.11792408, 3.89746071, 6.32548437] # float32
      },
      'labels': {
        values: [0.0, 1.0, 0.0] # float32
      }
    }
  }
]
```

```

        values: [0.0, 1.0, 0.0] # float32
    }
}
Record = {
    features = {},
    label = {
        'predicted_label': {
            values: [0.0] # float32
        },
        'distances': {
            values: [1.08470316, 3.04917915, 5.25393973] # float32
        },
        'labels': {
            values: [2.0, 2.0, 0.0] # float32
        }
    }
}
]
```

SAMPLE OUTPUT for the k-NN Algorithm

For regressor tasks:

```
[06/08/2018 20:15:33 INFO 140026520049408] #test_score (algo-1) : ('mse',
0.01333333333333334)
```

For classifier tasks:

```
[06/08/2018 20:15:46 INFO 140285487171328] #test_score (algo-1) : ('accuracy',
0.9866666666666669)
```

Latent Dirichlet Allocation (LDA) Algorithm

The Amazon SageMaker Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

The exact content of two documents with similar topic mixtures will not be the same. But overall, you would expect these documents to more frequently use a shared subset of words, than when compared with a document from a different topic mixture. This allows LDA to discover these word groups and use them to form topics. As an extremely simple example, given a set of documents where the only words that occur within them are: *eat*, *sleep*, *play*, *meow*, and *bark*, LDA might produce topics like the following:

Topic	<i>eat</i>	<i>sleep</i>	<i>play</i>	<i>meow</i>	<i>bark</i>
Topic 1	0.1	0.3	0.2	0.4	0.0
Topic 2	0.2	0.1	0.4	0.0	0.3

You can infer that documents that are more likely to fall into Topic 1 are about cats (who are more likely to *meow* and *sleep*), and documents that fall into Topic 2 are about dogs (who prefer to *play* and *bark*). These topics can be found even though the words dog and cat never appear in any of the texts.

Topics

- [Input/Output Interface for the LDA Algorithm \(p. 361\)](#)
- [EC2 Instance Recommendation for the LDA Algorithm \(p. 361\)](#)
- [LDA Sample LDA Notebooks \(p. 361\)](#)
- [How LDA Works \(p. 361\)](#)
- [LDA Hyperparameters \(p. 363\)](#)
- [Tune an LDA Model \(p. 364\)](#)

[Input/Output Interface for the LDA Algorithm](#)

LDA expects data to be provided on the train channel, and optionally supports a test channel, which is scored by the final model. LDA supports both recordIO-wrapped-protobuf (dense and sparse) and csv file formats. For csv, the data must be dense and have dimension equal to *number of records * vocabulary size*. LDA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the csv format.

For inference, text/csv, application/json, and application/x-recordio-protobuf content types are supported. Sparse data can also be passed for application/json and application/x-recordio-protobuf. LDA inference returns application/json or application/x-recordio-protobuf *predictions*, which include the topic_mixture vector for each observation.

Please see the [LDA Sample LDA Notebooks \(p. 361\)](#) for more detail on training and inference formats.

[EC2 Instance Recommendation for the LDA Algorithm](#)

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

[LDA Sample LDA Notebooks](#)

For a sample notebook that shows how to train the Amazon SageMaker Latent Dirichlet Allocation algorithm on a dataset and then how to deploy the trained model to perform inferences about the topic mixtures in input documents, see the [An Introduction to SageMaker LDA](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

[How LDA Works](#)

Amazon SageMaker LDA is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of different categories. These categories are themselves a probability distribution over the features. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. This is opposed to discriminative models, which attempt to learn how inputs map to outputs.

You can use LDA for a variety of tasks, from clustering customers based on product purchases to automatic harmonic analysis in music. However, it is most commonly associated with topic modeling in text corpuses. Observations are referred to as documents. The feature set is referred to as vocabulary. A feature is referred to as a word. And the resulting categories are referred to as topics.

Note

Lemmatization significantly increases algorithm performance and accuracy. Consider pre-processing any input text data.

An LDA model is defined by two parameters:

- α —A prior estimate on topic probability (in other words, the average frequency that each topic within a given document occurs).
- β —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 β_z .
- Draw a word $w \sim \text{Multinomial}(\beta_z)$.

When training the model, the goal is to find parameters α and β , 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 α and β , 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 α and β 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:

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

For more information about algorithms for LDA and the Amazon SageMaker implementation, see the following:

- Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. *Tensor Decompositions for Learning Latent Variable Models*. Journal of Machine Learning Research, 15:2773–2832, 2014.
- David M Blei, Andrew Y Ng, and Michael I Jordan. *Latent Dirichlet Allocation*. Journal of Machine Learning Research, 3(Jan):993–1022, 2003.
- Thomas L Griffiths and Mark Steyvers. *Finding Scientific Topics*. Proceedings of the National Academy of Sciences, 101(suppl 1):5228–5235, 2004.
- Tamara G Kolda and Brett W Bader. *Tensor Decompositions and Applications*. SIAM Review, 51(3):455–500, 2009.

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 \(p. 361\)](#).

Parameter Name	Description
<code>num_topics</code>	The number of topics for LDA to find within the data. Required Valid values: positive integer
<code>feature_dim</code>	The size of the vocabulary of the input document corpus. Required Valid values: positive integer
<code>mini_batch_size</code>	The total number of documents in the input document corpus. Required Valid values: positive integer
<code>alpha0</code>	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. Optional Valid values: Positive float

Parameter Name	Description
	Default value: 0.1
<code>max_restarts</code>	<p>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> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 10</p>
<code>max_iterations</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>tol</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Positive float</p> <p>Default value: 1e-8</p>

Tune an LDA Model

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

LDA is an unsupervised topic modeling algorithm that attempts to describe a set of observations (documents) as a mixture of different categories (topics). The “per-word log-likelihood” (PWLL) metric measures the likelihood that a learned set of topics (an LDA model) accurately describes a test document dataset. Larger values of PWLL indicate that the test data is more likely to be described by the LDA model.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the LDA Algorithm

The LDA algorithm reports on a single metric during training: `test:pwll`. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
test:pwll	Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.	Maximize

Tunable LDA Hyperparameters

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, `alpha0` and `num_topics`, can affect the LDA objective metric (`test:pwll`). If you don't already know the optimal values for these hyperparameters, which maximize per-word log-likelihood and produce an accurate LDA model, automatic model tuning can help find them.

Parameter Name	Parameter Type	Recommended Ranges
<code>alpha0</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 10
<code>num_topics</code>	IntegerParameterRanges	MinValue: 1, MaxValue: 150

Linear Learner Algorithm

Linear models are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples (x, y) . x is a high-dimensional vector and y is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to `num_classes` - 1. For regression problems, y is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector x to an approximation of the label y .

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the Amazon SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from a validation set. You can also explore a large number of models and choose the best. The best model optimizes either of the following:

- Continuous objectives, such as mean square error, cross entropy loss, absolute error.
- Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.

Compared with methods that provide a solution for only continuous objectives, the Amazon SageMaker linear learner algorithm provides a significant increase in speed over naive hyperparameter optimization techniques. It is also more convenient.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification or regression) as arguments. The feature dimension is also required. For more information, see [CreateTrainingJob](#). You can specify additional parameters in the `HyperParameters` string map of the request body. These parameters control the optimization procedure, or specifics of the objective function that you train on. For example, the number of epochs, regularization, and loss type.

Topics

- [Input/Output Interface for the Linear Learner Algorithm \(p. 366\)](#)

- [EC2 Instance Recommendation for the Linear Learner Algorithm \(p. 366\)](#)
- [Linear Learner Sample Notebooks \(p. 366\)](#)
- [How Linear Learner Works \(p. 367\)](#)
- [Linear Learner Hyperparameters \(p. 367\)](#)
- [Tune a Linear Learner Model \(p. 376\)](#)
- [Linear Learner Response Formats \(p. 378\)](#)

Input/Output Interface for the Linear Learner Algorithm

The Amazon SageMaker linear learner algorithm supports three data channels: train, validation (optional), and test (optional). If you provide validation data, it should be `FullyReplicated`. The algorithm logs validation loss at every epoch, and uses a sample of the validation data to calibrate and select the best model. If you don't provide validation data, the algorithm uses a sample of the training data to calibrate and select the model. If you provide test data, the algorithm logs include the test score for the final model.

For training, the linear learner algorithm supports both `recordIO-wrapped protobuf` and `csv` formats. For the `application/x-recordio-protobuf` input type, only `Float32` tensors are supported. For the `text/csv` input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as `recordIO-wrapped-protobuf` or as `csv`.

For inference, the linear learner algorithm supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` formats. When you make predictions on new data, the format of the response depends on the type of model. **For regression** (`predictor_type='regressor'`), the `score` is the prediction produced by the model. **For classification** (`predictor_type='binary_classifier'` or `predictor_type='multiclass_classifier'`), the model returns a `score` and also a `predicted_label`. The `predicted_label` is the class predicted by the model and the `score` measures the strength of that prediction.

- **For binary classification**, `predicted_label` is 0 or 1, and `score` is a single floating point number correspond class.
- **For multiclass classification**, the `predicted_class` will be an integer from 0 to `num_classes-1`, and the `score` will be a list of one floating point number per class.

To interpret the `score` in classification problems, you have to consider the loss function used. If the `loss` hyperparameter value is `logistic` for binary classification or `softmax_loss` for multiclass classification, then the `score` can be interpreted as the probability of the corresponding class. These are the loss values used by the linear learner when the `loss` value is `auto` default value. But if the `loss` is set to `hinge_loss`, then the `score` cannot be interpreted as a probability. This is because hinge loss corresponds to a Support Vector Classifier, which does not produce probability estimates.

For more information on input and output file formats, see [Linear Learner Response Formats \(p. 378\)](#). For more information on inference formats, and the [Linear Learner Sample Notebooks \(p. 366\)](#).

EC2 Instance Recommendation for the Linear Learner Algorithm

You can train the linear learner algorithm on single- or multi-machine CPU and GPU instances. During testing, we have not found substantial evidence that multi-GPU computers are faster than single-GPU computers. Results can vary, depending on your specific use case.

Linear Learner Sample Notebooks

For a sample notebook that uses the Amazon SageMaker linear learner algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see [An Introduction to Linear Learner with](#)

MNIST. For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. The topic modeling example notebooks using the linear learning algorithm are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

How Linear Learner Works

There are three steps involved in the implementation of the linear learner algorithm: preprocess, train, and validate.

Step 1: Preprocess

Normalization, or feature scaling, is an important preprocessing step for certain loss functions that ensures the model being trained on a dataset does not become dominated by the weight of a single feature. The Amazon SageMaker Linear Learner algorithm has a normalization option to assist with this preprocessing step. If normalization is turned on, the algorithm first goes over a small sample of the data to learn the mean value and standard deviation for each feature and for the label. Each of the features in the full dataset is then shifted to have a mean of zero and scaled to have a unit standard deviation.

Note

For best results, ensure your data is shuffled before training. Training with unshuffled data may cause training to fail.

You can configure whether the linear learner algorithm normalizes the feature data and the labels using the `normalize_data` and `normalize_label` hyperparameters respectively. Normalization is enabled by default for both features and labels for regression. Only the features can be normalized for binary classification and this is the default behavior.

Step 2: Train

With the linear learner algorithm, you train with a distributed implementation of stochastic gradient descent (SGD). You can control the optimization process by choosing the optimization algorithm. For example, you can choose to use Adam, AdaGrad, stochastic gradient descent, or other optimization algorithms. You also specify their hyperparameters, such as momentum, learning rate, and the learning rate schedule. If you aren't sure which algorithm or hyperparameter value to use, choose a default that works for the majority of datasets.

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

Step 3: Validate and Set the Threshold

When training multiple models in parallel, the models are evaluated against a validation set to select the most optimal model once training is complete. For regression, the most optimal model is the one that achieves the best loss on the validation set. For classification, a sample of the validation set is used to calibrate the classification threshold. The most optimal model selected is the one that achieves the best binary classification selection criteria on the validation set. Examples of such criteria include the F1 measure, accuracy, and cross-entropy loss.

Note

If the algorithm is not provided a validation set, then evaluating and selecting the most optimal model is not possible. To take advantage of parallel training and model selection ensure you provide a validation set to the algorithm.

Linear Learner Hyperparameters

The following table contains the hyperparameters for the linear learner algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required

hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

Parameter Name	Description
<code>feature_dim</code>	<p>The number of features in the input data.</p> <p>Required</p> <p>Valid values: Positive integer</p>
<code>num_classes</code>	<p>The number of classes for the response variable. The algorithm assumes that classes are labeled 0, ..., <code>num_classes</code> – 1.</p> <p>Required when <code>predictor_type</code> is <code>multiclass_classifier</code>. Otherwise, the algorithm ignores it.</p> <p>Valid values: Integers from 3 to 1,000,000</p>
<code>predictor_type</code>	<p>Specifies the type of target variable as a binary classification, multiclass classification, or regression.</p> <p>Required</p> <p>Valid values: <code>binary_classifier</code>, <code>multiclass_classifier</code>, or <code>regressor</code></p>
<code>accuracy_top_k</code>	<p>When computing the top-k accuracy metric for multiclass classification, the value of <i>k</i>. If the model assigns one of the top-k scores to the true label, an example is scored as correct.</p> <p>Optional</p> <p>Valid values: Positive integers</p> <p>Default value: 3</p>
<code>balance_multiclass_weights</code>	<p>Specifies whether to use class weights, which give each class equal importance in the loss function. Used only when the <code>predictor_type</code> is <code>multiclass_classifier</code>.</p> <p>Optional</p> <p>Valid values: <code>true</code>, <code>false</code></p> <p>Default value: <code>false</code></p>
<code>beta_1</code>	<p>The exponential decay rate for first-moment estimates. Applies only when the <code>optimizer</code> value is <code>adam</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or floating-point value between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>beta_2</code>	<p>The exponential decay rate for second-moment estimates. Applies only when the <code>optimizer</code> value is <code>adam</code>.</p> <p>Optional</p>

Parameter Name	Description
	<p>Valid values: <code>auto</code> or floating-point integer between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>bias_lr_mult</code>	<p>Allows a different learning rate for the bias term. The actual learning rate for the bias is <code>learning_rate * bias_lr_mult</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive floating-point integer</p> <p>Default value: <code>auto</code></p>
<code>bias_wd_mult</code>	<p>Allows different regularization for the bias term. The actual L2 regularization weight for the bias is <code>wd * bias_wd_mult</code>. By default, there is no regularization on the bias term.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or non-negative floating-point integer</p> <p>Default value: <code>auto</code></p>
<code>binary_classifier_model</code>	<p>When <code>prediction_type</code> is set to <code>binary_classifier</code>, the model evaluation criteria for the validation dataset (or for the training dataset if you don't provide a validation dataset). Criteria include:</p> <ul style="list-style-type: none"> <code>accuracy</code>—The model with the highest accuracy. <code>f_beta</code>—The model with the highest F1 score. The default is F1. <code>precision_at_target_recall</code>—The model with the highest precision at a given recall target. <code>recall_at_target_precision</code>—The model with the highest recall at a given precision target. <code>loss_function</code>—The model with the lowest value of the loss function used in training. <p>Optional</p> <p>Valid values: <code>accuracy</code>, <code>f_beta</code>, <code>precision_at_target_recall</code>, <code>recall_at_target_precision</code>, or <code>loss_function</code></p> <p>Default value: <code>accuracy</code></p>

Parameter Name	Description
early_stopping_patience	<p>If no improvement is made in the relevant metric, the number of epochs to wait before ending training. If you have provided a value for <code>binary_classifier_model_selection_criteria</code>, the metric is that value. Otherwise, the metric is the same as the value specified for the <code>loss</code> hyperparameter.</p> <p>The metric is evaluated on the validation data. If you haven't provided validation data, the metric is always the same as the value specified for the <code>loss</code> hyperparameter and is evaluated on the training data. To disable early stopping, set <code>early_stopping_patience</code> to a value greater than the value specified for <code>epochs</code>.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 3</p>
early_stopping_tolerance	<p>The relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping considers the improvement to be zero.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.001</p>
epochs	<p>The maximum number of passes over the training data.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 15</p>
f_beta	<p>The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if the value specified for <code>binary_classifier_model_selection_criteria</code> is <code>f_beta</code>.</p> <p>Optional</p> <p>Valid values: Positive floating-point integers</p> <p>Default value: 1.0</p>
huber_delta	<p>The parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than delta and L1 loss for errors larger than delta.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 1.0</p>

Parameter Name	Description
<code>init_bias</code>	<p>Initial weight for the bias term.</p> <p>Optional</p> <p>Valid values: Floating-point integer</p> <p>Default value: 0</p>
<code>init_method</code>	<p>Sets the initial distribution function used for model weights. Functions include:</p> <ul style="list-style-type: none"> • <code>uniform</code>—Uniformly distributed between (-scale, +scale) • <code>normal</code>—Normal distribution, with mean 0 and sigma <p>Optional</p> <p>Valid values: <code>uniform</code> or <code>normal</code></p> <p>Default value: <code>uniform</code></p>
<code>init_scale</code>	<p>Scales an initial uniform distribution for model weights. Applies only when the <code>init_method</code> hyperparameter is set to <code>uniform</code>.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.07</p>
<code>init_sigma</code>	<p>The initial standard deviation for the normal distribution. Applies only when the <code>init_method</code> hyperparameter is set to <code>normal</code>.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.01</p>
<code>l1</code>	<p>The L1 regularization parameter. If you don't want to use L1 regularization, set the value to 0.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or non-negative float</p> <p>Default value: <code>auto</code></p>
<code>learning_rate</code>	<p>The step size used by the optimizer for parameter updates.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive floating-point integer</p> <p>Default value: <code>auto</code>, whose value depends on the optimizer chosen.</p>

Parameter Name	Description
loss	<p>Specifies the loss function.</p> <p>The available loss functions and their default values depend on the value of <code>predictor_type</code>:</p> <ul style="list-style-type: none"> If the <code>predictor_type</code> is set to <code>regressor</code>, the available options are <code>auto</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, and <code>huber_loss</code>. The default value for <code>auto</code> is <code>squared_loss</code>. If the <code>predictor_type</code> is set to <code>binary_classifier</code>, the available options are <code>auto</code>, <code>logistic</code>, and <code>hinge_loss</code>. The default value for <code>auto</code> is <code>logistic</code>. If the <code>predictor_type</code> is set to <code>multiclass_classifier</code>, the available options are <code>auto</code> and <code>softmax_loss</code>. The default value for <code>auto</code> is <code>softmax_loss</code>. <p>Valid values: <code>auto</code>, <code>logistic</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>hinge_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, or <code>huber_loss</code></p> <p>Optional</p> <p>Default value: <code>auto</code></p>
loss_insensitivity	<p>The parameter for the epsilon-insensitive loss type. During training and metric evaluation, any error smaller than this value is considered to be zero.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.01</p>
lr_scheduler_factor	<p>For every <code>lr_scheduler_step</code> hyperparameter, the learning rate decreases by this quantity. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive floating-point integer between 0 and 1</p> <p>Default value: <code>auto</code></p>
lr_scheduler_minimum_lr	<p>The learning rate never decreases to a value lower than the value set for <code>lr_scheduler_minimum_lr</code>. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive floating-point integer</p> <p>Default values: <code>auto</code></p>

Parameter Name	Description
<code>lr_scheduler_step</code>	<p>The number of steps between decreases of the learning rate. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
<code>margin</code>	<p>The margin for the <code>hinge_loss</code> function.</p> <p>Optional</p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 1.0</p>
<code>mini_batch_size</code>	<p>The number of observations per mini-batch for the data iterator.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>momentum</code>	<p>The momentum of the <code>sgd</code> optimizer.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or a floating-point integer between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
<code>normalize_data</code>	<p>Normalizes the feature data before training. Data normalization shifts the data for each feature to have a mean of zero and scales it to have unit standard deviation.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>true</code></p>
<code>normalize_label</code>	<p>Normalizes the label. Label normalization shifts the label to have a mean of zero and scales it to have unit standard deviation.</p> <p>The <code>auto</code> default value normalizes the label for regression problems but does not for classification problems. If you set the <code>normalize_label</code> hyperparameter to <code>true</code> for classification problems, the algorithm ignores it.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
<code>num_calibration_samples</code>	<p>The number of observations from the validation dataset to use for model calibration (when finding the best threshold).</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
<code>num_models</code>	<p>The number of models to train in parallel. For the default, <code>auto</code>, 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.</p> <p>Optional</p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default values: <code>auto</code></p>
<code>num_point_for_scaler</code>	<p>The number of data points to use for calculating normalization or unbiassing of terms.</p> <p>Optional</p> <p>Valid values: Positive integer</p> <p>Default value: 10,000</p>
<code>optimizer</code>	<p>The optimization algorithm to use.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>auto</code>—The default value. • <code>sgd</code>—Stochastic gradient descent. • <code>adam</code>—Adaptive momentum estimation. • <code>rmsprop</code>—A gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient. <p>Default value: <code>auto</code>. The default setting for <code>auto</code> is <code>adam</code>.</p>
<code>positive_example_weight</code>	<p>The weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If you want the algorithm to choose a weight so that errors in classifying negative vs. positive examples have equal impact on training loss, specify <code>balanced</code>. If you want the algorithm to choose the weight that optimizes performance, specify <code>auto</code>.</p> <p>Optional</p> <p>Valid values: <code>balanced</code>, <code>auto</code>, or a positive floating-point integer</p> <p>Default value: 1.0</p>

Parameter Name	Description
quantile	<p>The quantile for quantile loss. For quantile q, the model attempts to produce predictions so that the value of <code>true_label</code> is greater than the prediction with probability q.</p> <p>Optional</p> <p>Valid values: Floating-point integer between 0 and 1</p> <p>Default value: 0.5</p>
target_precision	<p>The target precision. If <code>binary_classifier_model_selection_criteria</code> is <code>recall_at_target_precision</code>, then precision is held at this value while recall is maximized.</p> <p>Optional</p> <p>Valid values: Floating-point integer between 0 and 1.0</p> <p>Default value: 0.8</p>
target_recall	<p>The target recall. If <code>binary_classifier_model_selection_criteria</code> is <code>precision_at_target_recall</code>, then recall is held at this value while precision is maximized.</p> <p>Optional</p> <p>Valid values: Floating-point integer between 0 and 1.0</p> <p>Default value: 0.8</p>
unbias_data	<p>Unbiases the features before training so that the mean is 0. By default, data is unbiased if the <code>use_bias</code> hyperparameter is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
unbias_label	<p>Unbiases labels before training so that the mean is 0. Applies to regression only if the <code>use_bias</code> hyperparameter is set to <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
use_bias	<p>Specifies whether the model should include a bias term, which is the intercept term in the linear equation.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>

Parameter Name	Description
use_lr_scheduler	<p>Whether to use a scheduler for the learning rate. If you want to use a scheduler, specify <code>true</code>.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
wd	<p>The weight decay parameter, also known as the L2 regularization parameter. If you don't want to use L2 regularization, set the value to 0.</p> <p>Optional</p> <p>Valid values:<code>auto</code> or non-negative floating-point integer</p> <p>Default value: <code>auto</code></p>

Tune a Linear Learner Model

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

The linear learner algorithm also has an internal mechanism for tuning hyperparameters separate from the automatic model tuning feature described here. By default, the linear learner algorithm tunes hyperparameters by training multiple models in parallel. When you use automatic model tuning, the linear learner internal tuning mechanism is turned off automatically. This sets the number of parallel models, `num_models`, to 1. The algorithm ignores any value that you set for `num_models`.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Linear Learner Algorithm

The linear learner algorithm reports the metrics in the following table, which are computed during training. Choose one of them as the objective metric. To avoid overfitting, we recommend tuning the model against a validation metric instead of a training metric.

Metric Name	Description	Optimization Direction
<code>test:objective_loss</code>	The mean value of the objective loss function on the test dataset after the model is trained. By default, the loss is logistic loss for binary classification and squared loss for regression. To set the loss to other types, use the <code>loss</code> hyperparameter.	Minimize
<code>test:binary_classification_accuracy</code>	The accuracy of the final model on the test dataset.	Maximize
<code>test:binary_f_beta</code>	The F_beta score of the final model on the test dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall.	Maximize

Metric Name	Description	Optimization Direction
<code>test:precision</code>	The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>test:recall</code>	The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize
<code>validation:objective</code>	The mean value of the objective loss function on the validation dataset every epoch. By default, the loss is logistic loss for binary classification and squared loss for regression. To set loss to other types, use the <code>loss</code> hyperparameter.	Minimize
<code>validation:binary_classification_accuracy</code>	The accuracy of the final model on the validation dataset.	Maximize
<code>validation:binary_f_beta</code>	The F_beta score of the final model on the validation dataset. By default, the F_beta score is the F1 score, which is the harmonic mean of the <code>validation:precision</code> and <code>validation:recall</code> metrics.	Maximize
<code>validation:precision</code>	The precision of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>validation:recall</code>	The recall of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize

Tuning Linear Learner Hyperparameters

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

Parameter Name	Parameter Type	Recommended Ranges
wd	ContinuousParameterRanges	MinValue: 1e-7, MaxValue: 1
l1	ContinuousParameterRanges	MinValue: 1e-7, MaxValue: 1
learning_rate	ContinuousParameterRanges	MinValue: 1e-5, MaxValue: 1
mini_batch_size	IntegerParameterRanges	MinValue: 100, MaxValue: 5000
use_bias	CategoricalParameterRanges	[True, False]
positive_example_weight	ContinuousParameterRanges	MinValue: 1e-5, MaxValue: 1e5

Linear Learner Response Formats

JSON Response Format

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). The following are the available output formats for the Amazon SageMaker linear learner algorithm.

Binary Classification

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

Multiclass Classification

```
let response = {
  "predictions": [
    {
      "score": [0.1, 0.2, 0.4, 0.3],
      "predicted_label": 2
    }
  ]
}
```

Regression

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

JSONLINES Response Format

Binary Classification

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

Multiclass Classification

```
{"score": [0.1, 0.2, 0.4, 0.3], "predicted_label": 2}
```

Regression

```
{"score": 0.4}
```

RECORDIO Response Format

Binary Classification

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

Multiclass Classification

```
[  
    Record = {  
        "features": [],  
        "label": {  
            "score": {  
                "values": [0.1, 0.2, 0.3, 0.4]  
            },  
            "predicted_label": {  
                "values": [3]  
            }  
        },  
        "uid": "abc123",  
        "metadata": "{created_at: '2017-06-03'}"  
    }  
]
```

Regression

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

```
        keys: [],
        values: [0.4] # float32
    }
}
]
```

Neural Topic Model (NTM) Algorithm

Amazon SageMaker NTM is an unsupervised learning algorithm that is used to organize a corpus of documents into *topics* that contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as "bike", "car", "train", "mileage", and "speed" are likely to share a topic on "transportation" for example. Topic modeling can be used to classify or summarize documents based on the topics detected or to retrieve information or recommend content based on topic similarities. The topics from documents that NTM learns are characterized as a *latent representation* because the topics are inferred from the observed word distributions in the corpus. The semantics of topics are usually inferred by examining the top ranking words they contain. Because the method is unsupervised, only the number of topics, not the topics themselves, are prespecified. In addition, the topics are not guaranteed to align with how a human might naturally categorize documents.

Topic modeling provides a way to visualize the contents of a large document corpus in terms of the learned topics. Documents relevant to each topic might be indexed or searched for based on their soft topic labels. The latent representations of documents might also be used to find similar documents in the topic space. You can also use the latent representations of documents that the topic model learns for input to another supervised algorithm such as a document classifier. Because the latent representations of documents are expected to capture the semantics of the underlying documents, algorithms based in part on these representations are expected to perform better than those based on lexical features alone.

Although you can use both the Amazon SageMaker NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results on the same input data.

For more information on the mathematics behind NTM, see [Neural Variational Inference for Text Processing](#).

Topics

- [Input/Output Interface for the NTM Algorithm \(p. 380\)](#)
- [EC2 Instance Recommendation for the NTM Algorithm \(p. 381\)](#)
- [NTM Sample Notebooks \(p. 381\)](#)
- [NTM Hyperparameters \(p. 381\)](#)
- [Tune an NTM Model \(p. 384\)](#)
- [NTM Response Formats \(p. 385\)](#)

Input/Output Interface for the NTM Algorithm

Amazon SageMaker Neural Topic Model supports four data channels: train, validation, test, and auxiliary. The validation, test, and auxiliary data channels are optional. If you specify any of these optional channels, set the value of the `S3DataDistributionType` parameter for them to `FullyReplicated`. If you provide validation data, the loss on this data is logged at every epoch, and the model stops training as soon as it detects that the validation loss is not improving. If you don't provide validation data, the algorithm stops early based on the training data, but this can be less efficient. If you provide test data, the algorithm reports the test loss from the final model.

The train, validation, and test data channels for NTM support both `recordIO-wrapped-protobuf` (dense and sparse) and CSV file formats. For CSV format, each row must be represented densely with zero counts for words not present in the corresponding document, and have dimension equal to: (number of records) * (vocabulary size). You can use either File mode or Pipe mode to train models on

data that is formatted as recordIO-wrapped-protobuf or as csv. The auxiliary channel is used to supply a text file that contains vocabulary. By supplying the vocabulary file, users are able to see the top words for each of the topics printed in the log instead of their integer IDs. Having the vocabulary file also allows NTM to compute the Word Embedding Topic Coherence (WETC) scores, a new metric displayed in the log that captures similarity among the top words in each topic effectively. The ContentType for the auxiliary channel is text/plain, with each line containing a single word, in the order corresponding to the integer IDs provided in the data. The vocabulary file must be named vocab.txt and currently only UTF-8 encoding is supported.

For inference, text/csv, application/json, application/jsonlines, and application/x-recordio-protobuf content types are supported. Sparse data can also be passed for application/json and application/x-recordio-protobuf. NTM inference returns application/json or application/x-recordio-protobuf *predictions*, which include the topic_weights vector for each observation.

See the [blog post](#) and the companion [notebook](#) for more details on using the auxiliary channel and the WETC scores. For more information on how to compute the WETC score, see [Coherence-Aware Neural Topic Modeling](#). We used the pairwise WETC described in this paper for the Amazon SageMaker Neural Topic Model.

For more information on input and output file formats, see [NTM Response Formats \(p. 385\)](#) for inference and the [NTM Sample Notebooks \(p. 381\)](#).

EC2 Instance Recommendation for the NTM Algorithm

NTM training supports both GPU and CPU instance types. We recommend GPU instances, but for certain workloads, CPU instances may result in lower training costs. CPU instances should be sufficient for inference.

NTM Sample Notebooks

For a sample notebook that uses the Amazon SageMaker NTM algorithm to uncover topics in documents from a synthetic data source where the topic distributions are known, see the [Introduction to Basic Functionality of NTM](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

NTM Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	The vocabulary size of the dataset. Required Valid values: Positive integer (min: 1, max: 1,000,000)
<code>num_topics</code>	The number of required topics. Required Valid values: Positive integer (min: 2, max: 1000)
<code>batch_norm</code>	Whether to use batch normalization during training. Optional

Parameter Name	Description
	<p>Valid values: <i>true</i> or <i>false</i></p> <p>Default value: <i>false</i></p>
<code>clip_gradient</code>	<p>The maximum magnitude for each gradient component.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-3)</p> <p>Default value: Infinity</p>
<code>encoder_layers</code>	<p>The number of layers in the encoder and the output size of each layer. When set to <i>auto</i>, the algorithm uses two layers of sizes 3 x <code>num_topics</code> and 2 x <code>num_topics</code> respectively.</p> <p>Optional</p> <p>Valid values: Comma-separated list of positive integers or <i>auto</i></p> <p>Default value: <i>auto</i></p>
<code>encoder_layers_activation</code>	<p>The activation function to use in the encoder layers.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>sigmoid</code>: Sigmoid function • <code>tanh</code>: Hyperbolic tangent • <code>relu</code>: Rectified linear unit <p>Default value: <code>sigmoid</code></p>
<code>epochs</code>	<p>The maximum number of passes over the training data.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 50</p>
<code>learning_rate</code>	<p>The learning rate for the optimizer.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-6, max: 1.0)</p> <p>Default value: 0.001</p>
<code>mini_batch_size</code>	<p>The number of examples in each mini batch.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1, max: 10000)</p> <p>Default value: 256</p>

Parameter Name	Description
<code>num_patience_epochs</code>	<p>The number of successive epochs over which early stopping criterion is evaluated. Early stopping is triggered when the change in the loss function drops below the specified tolerance within the last <code>num_patience_epochs</code> number of epochs. To disable early stopping, set <code>num_patience_epochs</code> to a value larger than <code>epochs</code>.</p> <p>Optional</p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 3</p>
<code>optimizer</code>	<p>The optimizer to use for training.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>sgd</code>: Stochastic gradient descent • <code>adam</code>: Adaptive momentum estimation • <code>adagrad</code>: Adaptive gradient algorithm • <code>adadelta</code>: An adaptive learning rate algorithm • <code>rmsprop</code>: Root mean square propagation <p>Default value: <code>adadelta</code></p>
<code>rescale_gradient</code>	<p>The rescale factor for gradient.</p> <p>Optional</p> <p>Valid values: float (min: 1e-3, max: 1.0)</p> <p>Default value: 1.0</p>
<code>sub_sample</code>	<p>The fraction of the training data to sample for training per epoch.</p> <p>Optional</p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 1.0</p>
<code>tolerance</code>	<p>The maximum relative change in the loss function. Early stopping is triggered when change in the loss function drops below this value within the last <code>num_patience_epochs</code> number of epochs.</p> <p>Optional</p> <p>Valid values: Float (min: 1e-6, max: 0.1)</p> <p>Default value: 0.001</p>

Parameter Name	Description
weight_decay	<p>The weight decay coefficient. Adds L2 regularization.</p> <p>Optional</p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 0.0</p>

Tune an NTM Model

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

Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. Latent representations use inferred variables that are not directly measured to model the observations in a dataset. Automatic model tuning on NTM helps you find the model that minimizes loss over the training or validation data. *Training loss* measures how well the model fits the training data. *Validation loss* measures how well the model can generalize to data that it is not trained on. Low training loss indicates that a model is a good fit to the training data. Low validation loss indicates that a model has not overfit the training data and so should be able to model documents on which it has not been trained successfully. Usually, it's preferable to have both losses be small. However, minimizing training loss too much might result in overfitting and increase validation loss, which would reduce the generality of the model.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the NTM Algorithm

The NTM algorithm reports a single metric that is computed during training: `validation:total_loss`. The total loss is the sum of the reconstruction loss and Kullback-Leibler divergence. When tuning hyperparameter values, choose this metric as the objective.

Metric Name	Description	Optimization Direction
<code>validation:total_loss</code>	Total Loss on validation set	Minimize

Tunable NTM Hyperparameters

You can tune the following hyperparameters for the NTM algorithm. Usually setting low `mini_batch_size` and small `learning_rate` values results in lower validation losses, although it might take longer to train. Low validation losses don't necessarily produce more coherent topics as interpreted by humans. The effect of other hyperparameters on training and validation loss can vary from dataset to dataset. To see which values are compatible, see [NTM Hyperparameters \(p. 381\)](#).

Parameter Name	Parameter Type	Recommended Ranges
<code>encoder_layers_activation</code>	CategoricalParameterRanges	['sigmoid', 'tanh', 'relu']
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1

Parameter Name	Parameter Type	Recommended Ranges
mini_batch_size	IntegerParameterRanges	MinValue: 16, MaxValue:2048
optimizer	CategoricalParameterRanges	['sgd', 'adam', 'adadelta']
rescale_gradient	ContinuousParameterRange	MinValue: 0.1, MaxValue: 1.0
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0

NTM Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker NTM algorithm.

JSON Response Format

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

JSONLINES Response Format

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

RECORDIO Response Format

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

Object2Vec Algorithm

The Amazon SageMaker Object2Vec algorithm is a general-purpose neural embedding algorithm that is highly customizable. It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in a way that preserves the semantics of the relationship between pairs of objects in the original space in the embedding space. You can use the learned embeddings to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space, for example. You can also use the embeddings as features of the corresponding objects in downstream supervised tasks, such as classification or regression.

Object2Vec generalizes the well-known Word2Vec embedding technique for words that is optimized in the Amazon SageMaker [BlazingText Algorithm \(p. 290\)](#). For a blog post that discusses how to apply Object2Vec to some practical use cases, see [Introduction to Amazon SageMaker Object2Vec](#).

Topics

- [I/O Interface for the Object2Vec Algorithm \(p. 386\)](#)
- [EC2 Instance Recommendation for the Object2Vec Algorithm \(p. 387\)](#)
- [Object2Vec Sample Notebooks \(p. 387\)](#)
- [How Object2Vec Works \(p. 388\)](#)
- [Object2Vec Hyperparameters \(p. 389\)](#)
- [Tune an Object2Vec Model \(p. 397\)](#)
- [Data Formats for Object2Vec Training \(p. 399\)](#)
- [Data Formats for Object2Vec Inference \(p. 399\)](#)
- [Encoder Embeddings for Object2Vec \(p. 401\)](#)

I/O Interface for the Object2Vec Algorithm

You can use Object2Vec on many input data types, including the following examples.

Input Data Type	Example
Sentence-sentence pairs	"A soccer game with multiple males playing." and "Some men are playing a sport."
Labels-sequence pairs	The genre tags of the movie "Titanic", such as "Romance" and "Drama", and its short description: "James Cameron's Titanic is an epic, action-packed romance set against the ill-fated maiden voyage of the R.M.S. Titanic. She was the most luxurious liner of her era, a ship of dreams, which ultimately carried over 1,500 people to their death in the ice cold waters of the North Atlantic in the early hours of April 15, 1912."
Customer-customer pairs	The customer ID of Jane and customer ID of Jackie.
Product-product pairs	The product ID of football and product ID of basketball.
Item review user-item pairs	A user's ID and the items she has bought, such as apple, pear, and orange.

To transform the input data into the supported formats, you must preprocess it. Currently, Object2Vec natively supports two types of input:

- A discrete token, which is represented as a list of a single `integer-id`. For example, [10].

- A sequences of discrete tokens, which is represented as a list of `integer-ids`. For example, `[0, 12, 10, 13]`.

The object in each pair can be asymmetric. For example, the pairs can be (token, sequence) or (token, token) or (sequence, sequence). For token inputs, the algorithm supports simple embeddings as compatible encoders. For sequences of token vectors, the algorithm supports the following as encoders:

- Average-pooled embeddings
- Hierarchical convolutional neural networks (CNNs),
- Multi-layered bidirectional long short-term memory (BiLSTMs)

The input label for each pair can be one of the following:

- A categorical label that expresses the relationship between the objects in the pair
- A score that expresses the strength of the similarity between the two objects

For categorical labels used in classification, the algorithm supports the cross-entropy loss function. For ratings/score-based labels used in regression, the algorithm supports the mean squared error (MSE) loss function. Specify these loss functions with the `output_layer` hyperparameter when you create the model training job.

EC2 Instance Recommendation for the Object2Vec Algorithm

The type of Amazon Elastic Compute Cloud (Amazon EC2) instance that you use depends on whether you are training or running inferences.

Instance Recommendation for Training

When training a model using the Object2Vec algorithm on a CPU, start with an `ml.m5.2xlarge` instance. For training on a GPU, start with an `ml.p2.xlarge` instance. If the training takes too long on this instance, you can use a larger instance, such as an `ml.m5.4xlarge` or an `ml.m5.12xlarge` instance. Currently, the Object2Vec algorithm can train only on a single machine. However, it does offer support for multiple GPUs.

Instance Recommendation for Inference

For inference with a trained Object2Vec model that has a deep neural network, we recommend using `ml.p3.2xlarge` GPU instance. Due to GPU memory scarcity, the `INFERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the [the section called "GPU optimization: Classification or Regression" \(p. 399\)](#) or [the section called "GPU optimization: Encoder Embeddings" \(p. 401\)](#) inference network is loaded into GPU.

Object2Vec Sample Notebooks

For a sample notebook that uses the Amazon SageMaker Object2Vec algorithm to encode sequences into fixed-length embeddings, see [Using Object2Vec to Encode Sentences into Fixed Length Embeddings](#). For a sample notebook that uses the Object2Vec algorithm in a multi-label prediction setting to predict the genre of a movie from its plot description, see [Movie genre prediction with Object2Vec Algorithm](#). For a sample notebook that uses the Object2Vec algorithm to make movie recommendations, see [An Introduction to SageMaker ObjectToVec model for MovieLens recommendation](#). For a sample notebook that uses the Object2Vec algorithm to learn document embeddings, see [Using Object2Vec to learn document embeddings](#). For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After you have created a notebook instance

and opened it, choose **SageMaker Examples** to see a list of Amazon SageMaker samples. To open a notebook, choose its **Use** tab and choose **Create copy**.

How Object2Vec Works

When using the Amazon SageMaker Object2Vec algorithm, you follow the standard workflow: process the data, train the model, and produce inferences.

Topics

- [Step 1: Process Data \(p. 388\)](#)
- [Step 2: Train a Model \(p. 388\)](#)
- [Step 3: Produce Inferences \(p. 389\)](#)

Step 1: Process Data

During preprocessing, convert the data to the [JSON Lines](#) text file format specified in [Data Formats for Object2Vec Training \(p. 399\)](#). To get the highest accuracy during training, also randomly shuffle the data before feeding it into the model. How you generate random permutations depends on the language. For python, you could use `np.random.shuffle`; for Unix, `shuf`.

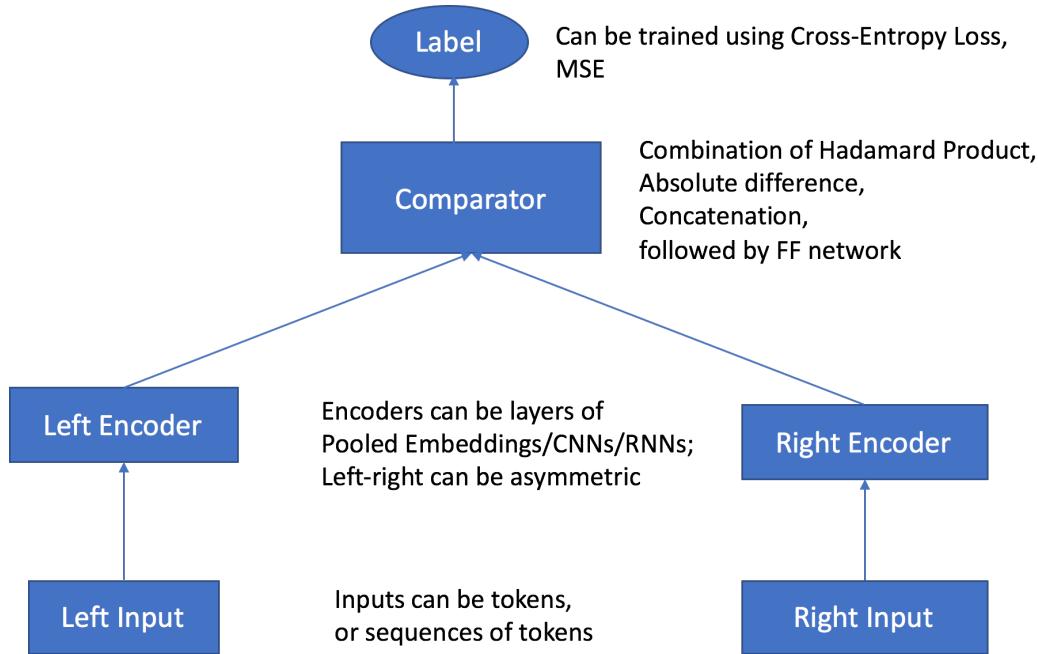
Step 2: Train a Model

The Amazon SageMaker Object2Vec algorithm has the following main components:

- **Two input channels** – The input channels take a pair of objects of the same or different types as inputs, and pass them to independent and customizable encoders.
- **Two encoders** – The two encoders, enc0 and enc1, convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair are then passed into a comparator.
- **A comparator** – The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects. In the output score for a sentence pair. For example, 1 indicates a strong relationship between a sentence pair, and 0 represents a weak relationship.

During training, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types, as described earlier. If the inputs to both encoders are composed of the same token-level units, you can use a shared token embedding layer by setting the `tied_token_embedding_weight` hyperparameter to `True` when you create the training job. This is possible, for example, when comparing sentences that both have word token-level units. To generate negative samples at a specified rate, set the `negative_sampling_rate` hyperparameter to the desired ratio of negative to positive samples. This hyperparameter expedites learning how to discriminate between the positive samples observed in the training data and the negative samples that are not likely to be observed.

Pairs of objects are passed through independent, customizable encoders that are compatible with the input types of corresponding objects. The encoders convert each object in a pair into a fixed-length embedding vector of equal length. The pair of vectors are passed to a comparator operator, which assembles the vectors into a single vector using the value specified in the `he comparator_list` hyperparameter. The assembled vector then passes through a multilayer perceptron (MLP) layer, which produces an output that the loss function compares with the labels that you provided. This comparison evaluates the strength of the relationship between the objects in the pair as predicted by the model. The following figure shows this workflow.



Architecture of the Object2Vec Algorithm from Data Inputs to Scores

Step 3: Produce Inferences

After the model is trained, you can use the trained encoder to preprocess input objects or to perform two types of inference:

- To convert singleton input objects into fixed-length embeddings using the corresponding encoder
- To predict the relationship label or score between a pair of input objects

The inference server automatically figures out which of the types is requested based on the input data. To get the embeddings as output, provide only one input. To predict the relationship label or score, provide both inputs in the pair.

Object2Vec 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 Object2Vec training algorithm.

Parameter Name	Description
<code>enc0_max_seq_len</code>	<p>The maximum sequence length for the enc0 encoder.</p> <p>Required</p> <p>Valid values: $1 \leq \text{integer} \leq 5000$</p>
<code>enc0_vocab_size</code>	<p>The vocabulary size of enc0 tokens.</p> <p>Required</p> <p>Valid values: $2 \leq \text{integer} \leq 3000000$</p>

Parameter Name	Description
bucket_width	<p>The allowed difference between data sequence length when bucketing is enabled. To enable bucketing, specify a non-zero value for this parameter.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{integer} \leq 100$</p> <p>Default value: 0 (no bucketing)</p>
comparator_list	<p>A list used to customize the way in which two embeddings are compared. The Object2Vec comparator operator layer takes the encodings from both encoders as inputs and outputs a single vector. This vector is a concatenation of subvectors. The string values passed to the <code>comparator_list</code> and the order in which they are passed determine how these subvectors are assembled. For example, if <code>comparator_list="hadamard, concat"</code>, then the comparator operator constructs the vector by concatenating the Hadamard product of two encodings and the concatenation of two encodings. If, on the other hand, <code>comparator_list="hadamard"</code>, then the comparator operator constructs the vector as the hadamard product of only two encodings.</p> <p>Optional</p> <p>Valid values: A string that contains any combination of the names of the three binary operators: <code>hadamard</code>, <code>concat</code>, or <code>abs_diff</code>. The Object2Vec algorithm currently requires that the two vector encodings have the same dimension. These operators produce the subvectors as follows:</p> <ul style="list-style-type: none"> • <code>hadamard</code>: Constructs a vector as the Hadamard (element-wise) product of two encodings. • <code>concat</code>: Constructs a vector as the concatenation of two encodings. • <code>abs_diff</code>: Constructs a vector as the absolute difference between two encodings. <p>Default value: "hadamard, concat, abs_diff"</p>
dropout	<p>The dropout probability for network layers. <i>Dropout</i> is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons.</p> <p>Optional</p> <p>Valid values: $0.0 \leq \text{float} \leq 1.0$</p> <p>Default value: 0.0</p>

Parameter Name	Description
<code>early_stopping_patience</code>	<p>The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by with the <code>early_stopping_tolerance</code> hyperparameter.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 5$</p> <p>Default value: 3</p>
<code>early_stopping_tolerance</code>	<p>The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after the number of consecutive epochs specified in the <code>early_stopping_patience</code> hyperparameter concludes.</p> <p>Optional</p> <p>Valid values: $0.000001 \leq \text{float} \leq 0.1$</p> <p>Default value: 0.01</p>
<code>enc_dim</code>	<p>The dimension of the output of the embedding layer.</p> <p>Optional</p> <p>Valid values: $4 \leq \text{integer} \leq 10000$</p> <p>Default value: 4096</p>
<code>enc0_network</code>	<p>The network model for the enc0 encoder.</p> <p>Optional</p> <p>Valid values: <code>hcnn</code>, <code>bilstm</code>, or <code>pooled_embedding</code></p> <ul style="list-style-type: none"> • <code>hcnn</code>: A hierarchical convolutional neural network. • <code>bilstm</code>: A bidirectional long short-term memory network (biLSTM), in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. • <code>pooled_embedding</code>: Averages the embeddings of all of the tokens in the input. <p>Default value: <code>hcnn</code></p>
<code>enc0_cnn_filter_width</code>	<p>The filter width of the convolutional neural network (CNN) enc0 encoder.</p> <p>Conditional</p> <p>Valid values: $1 \leq \text{integer} \leq 9$</p> <p>Default value: 3</p>

Parameter Name	Description
<code>enc0_freeze_pretrained_embedding</code>	<p>Whether to freeze enc0 pretrained embedding weights.</p> <p>Conditional</p> <p>Valid values: <code>True</code> or <code>False</code></p> <p>Default value: <code>True</code></p>
<code>enc0_layers</code>	<p>The number of layers in the enc0 encoder.</p> <p>Conditional</p> <p>Valid values: <code>auto</code> or $1 \leq \text{integer} \leq 4$</p> <ul style="list-style-type: none"> For <code>hcnn</code>, <code>auto</code> means 4. For <code>bilstm</code>, <code>auto</code> means 1. For <code>pooled_embedding</code>, <code>auto</code> ignores the number of layers. <p>Default value: <code>auto</code></p>
<code>enc0_pretrained_embedding_file</code>	<p>filename of the pretrained enc0 token embedding file in the auxiliary data channel.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. <code>[A-Za-z0-9\.__]</code></p> <p>Default value: <code>""</code> (empty string)</p>
<code>enc0_token_embedding_dim</code>	<p>The output dimension of the enc0 token embedding layer.</p> <p>Conditional</p> <p>Valid values: $2 \leq \text{integer} \leq 1000$</p> <p>Default value: 300</p>
<code>enc0_vocab_file</code>	<p>The vocabulary file for mapping pretrained enc0 token embedding vectors to numerical vocabulary IDs.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. <code>[A-Za-z0-9\.__]</code></p> <p>Default value: <code>""</code> (empty string)</p>

Parameter Name	Description
enc1_network	<p>The network model for the enc1 encoder. If you want the enc1 encoder to use the same network model as enc0, including the hyperparameter values, set the value to enc0.</p> <p>Note Even when the enc0 and enc1 encoder networks have symmetric architectures, you can't shared parameter values for these networks.</p> <p>Optional</p> <p>Valid values: enc0, hcnn, bilstm, or pooled_embedding</p> <ul style="list-style-type: none"> enc0: The network model for the enc0 encoder. hcnn: A hierarchical convolutional neural network. bilstm: A bidirectional LSTM, in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks. pooled_embedding: The averages of the embeddings of all of the tokens in the input. <p>Default value: enc0</p>
enc1_cnn_filter_width	<p>The filter width of the CNN enc1 encoder.</p> <p>Conditional</p> <p>Valid values: 1 ≤ integer ≤ 9</p> <p>Default value: 3</p>
enc1_freeze_pretrained_embedding	<p>Whether to freeze enc1 pretrained embedding weights.</p> <p>Conditional</p> <p>Valid values: True or False</p> <p>Default value: True</p>
enc1_layers	<p>The number of layers in the enc1 encoder.</p> <p>Conditional</p> <p>Valid values: auto or 1 ≤ integer ≤ 4</p> <ul style="list-style-type: none"> For hcnn, auto means 4. For bilstm, auto means 1. For pooled_embedding, auto ignores the number of layers. <p>Default value: auto</p>
enc1_max_seq_len	<p>The maximum sequence length for the enc1 encoder.</p> <p>Conditional</p> <p>Valid values: 1 ≤ integer ≤ 5000</p>

Parameter Name	Description
enc1_pretrained_embedding_file	<p>The filename of the enc1 pretrained token embedding file in the auxiliary data channel.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._]</p> <p>Default value: "" (empty string)</p>
enc1_token_embedding_dim	<p>The output dimension of the enc1 token embedding layer.</p> <p>Conditional</p> <p>Valid values: $2 \leq \text{integer} \leq 1000$</p> <p>Default value: 300</p>
enc1_vocab_file	<p>The vocabulary file for mapping pretrained enc1 token embeddings to vocabulary IDs.</p> <p>Conditional</p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\._]</p> <p>Default value: "" (empty string)</p>
enc1_vocab_size	<p>The vocabulary size of enc0 tokens.</p> <p>Conditional</p> <p>Valid values: $2 \leq \text{integer} \leq 3000000$</p>
epochs	<p>The number of epochs to run for training.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 100$</p> <p>Default value: 30</p>
learning_rate	<p>The learning rate for training.</p> <p>Optional</p> <p>Valid values: $1.0E-6 \leq \text{float} \leq 1.0$</p> <p>Default value: 0.0004</p>
mini_batch_size	<p>The batch size that the dataset is split into for an optimizer during training.</p> <p>Optional</p> <p>Valid values: $1 \leq \text{integer} \leq 10000$</p> <p>Default value: 32</p>

Parameter Name	Description
mlp_activation	<p>The type of activation function for the multilayer perceptron (MLP) layer.</p> <p>Optional</p> <p>Valid values: <code>tanh</code>, <code>relu</code>, or <code>linear</code></p> <ul style="list-style-type: none"> • <code>tanh</code>: Hyperbolic tangent • <code>relu</code>: Rectified linear unit (ReLU) • <code>linear</code>: Linear function <p>Default value: <code>linear</code></p>
mlp_dim	<p>The dimension of the output from MLP layers.</p> <p>Optional</p> <p>Valid values: $2 \leq \text{integer} \leq 10000$</p> <p>Default value: 512</p>
mlp_layers	<p>The number of MLP layers in the network.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{integer} \leq 10$</p> <p>Default value: 2</p>
negative_sampling_rate	<p>The ratio of negative samples, generated to assist in training the algorithm, to positive samples that are provided by users. Negative samples represent data that is unlikely to occur in reality and are labelled negatively for training. They facilitate training a model to discriminate between the positive samples observed and the negative samples that are not. To specify the ratio of negative samples to positive samples used for training, set the value to a positive integer. For example, if you train the algorithm on input data in which all of the samples are positive and set <code>negative_sampling_rate</code> to 2, the Object2Vec algorithm internally generates two negative samples per positive sample. If you don't want to generate or use negative samples during training, set the value to 0.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{integer}$</p> <p>Default value: 0 (off)</p>
num_classes	<p>The number of classes for classification training. Amazon SageMaker ignores this hyperparameter for regression problems.</p> <p>Optional</p> <p>Valid values: $2 \leq \text{integer} \leq 30$</p> <p>Default value: 2</p>

Parameter Name	Description
optimizer	<p>The optimizer type.</p> <p>Optional</p> <p>Valid values: <code>adadelta</code>, <code>adagrad</code>, <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</p> <ul style="list-style-type: none"> • <code>adadelta</code>: A per-dimension learning rate method for gradient descent • <code>adagrad</code>: The adaptive gradient algorithm • <code>adam</code>: The adaptive moment estimation algorithm • <code>sgd</code>: Stochastic gradient descent • <code>rmsprop</code>: Root mean square propagation <p>Default value: <code>adam</code></p>
output_layer	<p>The type of output layer where you specify that the task is regression or classification.</p> <p>Optional</p> <p>Valid values: <code>softmax</code> or <code>mean_squared_error</code></p> <ul style="list-style-type: none"> • <code>softmax</code>: The Softmax function used for classification. • <code>mean_squared_error</code>: The MSE used for regression. <p>Default value: <code>softmax</code></p>
tied_token_embedding_weight	<p>Whether to use a shared embedding layer for both encoders. If the inputs to both encoders use the same token-level units, use a shared token embedding layer. For example, for a collection of documents, if one encoder encodes sentences and another encodes whole documents, you can use a shared token embedding layer. That's because both sentences and documents are composed of word tokens from the same vocabulary.</p> <p>Optional</p> <p>Valid values: <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>

Parameter Name	Description
token_embedding_storage_type	<p>The mode of gradient update used during training: when the <code>dense</code> mode is used, the optimizer calculates the full gradient matrix for the token embedding layer even if most rows of the gradient are zero-valued. When <code>sparse</code> mode is used, the optimizer only stores rows of the gradient that are actually being used in the mini-batch. If you want the algorithm to perform lazy gradient updates, which calculate the gradients only in the non-zero rows and which speed up training, specify <code>row_sparse</code>. Setting the value to <code>row_sparse</code> constrains the values available for other hyperparameters, as follows:</p> <ul style="list-style-type: none"> The optimizer hyperparameter must be set to <code>adam</code>, <code>adagrad</code>, or <code>sgd</code>. Otherwise, the algorithm throws a <code>CustomerValueError</code>. The algorithm automatically disables bucketing, setting the <code>bucket_width</code> hyperparameter to 0. <p>Optional</p> <p>Valid values: <code>dense</code> or <code>row_sparse</code></p> <p>Default value: <code>dense</code></p>
weight_decay	<p>The weight decay parameter used for optimization.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 10000$</p> <p>Default value: 0 (no decay)</p>

Tune an Object2Vec Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm has both classification and regression metrics. The `output_layer` type determines which metric you can use for automatic model tuning.

Regressor Metrics Computed by the Object2Vec Algorithm

The algorithm reports a mean squared error regressor metric, which is computed during testing and validation. When tuning the model for regression tasks, choose this metric as the objective.

Metric Name	Description	Optimization Direction
<code>test:mean_squared_error</code>	Root Mean Square Error	Minimize

Metric Name	Description	Optimization Direction
validation:mean_squared_error	Mean Square Error	Minimize

Classification Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm reports accuracy and cross-entropy classification metrics, which are computed during test and validation. When tuning the model for classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
test:accuracy	Accuracy	Maximize
test:cross_entropy	Cross-entropy	Minimize
validation:accuracy	Accuracy	Maximize
validation:cross_entropy	Cross-entropy	Minimize

Tunable Object2Vec Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

Hyperparameter Name	Hyperparameter Type	Recommended Ranges and Values
dropout	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0
early_stopping_patience	IntegerParameterRange	MinValue: 1, MaxValue: 5
early_stopping_tolerance	ContinuousParameterRange	MinValue: 0.001, MaxValue: 0.1
enc_dim	IntegerParameterRange	MinValue: 4, MaxValue: 4096
enc0_cnn_filter_width	IntegerParameterRange	MinValue: 1, MaxValue: 5
enc0_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4
enc0_token_embedding_dim	IntegerParameterRange	MinValue: 5, MaxValue: 300
enc1_cnn_filter_width	IntegerParameterRange	MinValue: 1, MaxValue: 5
enc1_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4
enc1_token_embedding_dim	IntegerParameterRange	MinValue: 5, MaxValue: 300

Hyperparameter Name	Hyperparameter Type	Recommended Ranges and Values	
epochs	IntegerParameterRange	MinValue: 4, MaxValue: 20	
learning_rate	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 1.0	
mini_batch_size	IntegerParameterRange	MinValue: 1, MaxValue: 8192	
mlp_activation	CategoricalParameterRanges	[tanh, relu, linear]	
mlp_dim	IntegerParameterRange	MinValue: 16, MaxValue: 1024	
mlp_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
optimizer	CategoricalParameterRanges	[adagrad, adam, rmsprop, sgd, adadelta]	
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0	

Data Formats for Object2Vec Training

Input: JSON Lines Request Format

Content-type: application/jsonlines

```
{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}  
{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}  
{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

The “in0” and “in1” are the inputs for encoder0 and encoder1, respectively. The same format is valid for both classification and regression problems. For regression, the field “label” can accept real valued inputs.

Data Formats for Object2Vec Inference

GPU optimization: Classification or Regression

Due to GPU memory scarcity, the `INFERENCER_PREFERRED_MODE` environment variable can be specified to optimize on whether the classification/regression or the [the section called “Output: Encoder Embeddings” \(p. 401\)](#) inference network is loaded into GPU. If the majority of your inference is for classification or regression, specify `INFERENCER_PREFERRED_MODE=classification`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for classification/regression inference:

```
transformer = o2v.transformer(instance_count=4,
                             instance_type="ml.p2.xlarge",
```

```

    max_concurrent_transforms=2,
    max_payload=1, # 1MB
    strategy='MultiRecord',
    env={'INFERENCE_PREFERRED_MODE': 'classification'}, # only
useful with GPU
    output_path=output_s3_path)

```

Input: Classification or Regression Request Format

Content-type: application/json

```
{
  "instances" : [
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}, {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}, {"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}]
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}
{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

For classification problems, the length of the scores vector corresponds to `num_classes`. For regression problems, the length is 1.

Ouput: Classification or Regression Response Format

Accept: application/json

```
{
  "predictions": [
    {
      "scores": [
        0.6533935070037842,
        0.07582679390907288,
        0.2707797586917877
      ]
    },
    {
      "scores": [
        0.026291321963071823,
        0.6577019095420837,
        0.31600672006607056
      ]
    }
  ]
}
```

Accept: application/jsonlines

```
{"scores": [0.195667684078216, 0.395351558923721, 0.408980727195739]}
{"scores": [0.251988261938095, 0.258233487606048, 0.489778339862823]}
```

```
{ "scores": [0.280087798833847, 0.368331134319305, 0.351581096649169]}
```

In both the classification and regression formats, the scores apply to individual labels.

Encoder Embeddings for Object2Vec

GPU optimization: Encoder Embeddings

Due to GPU memory scarcity, the `INFEERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the [the section called “Inference Formats: Scoring” \(p. 399\)](#) or the encoder embedding inference network is loaded into GPU. If the majority of your inference is for encoder embeddings, specify `INFEERENCE_PREFERRED_MODE=embedding`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for encoder embedding inference:

```
transformer = o2v.transformer(instance_count=4,
                             instance_type="ml.p2.xlarge",
                             max_concurrent_transforms=2,
                             max_payload=1, # 1MB
                             strategy='MultiRecord',
                             env={'INFEERENCE_PREFERRED_MODE': 'embedding'}, # only useful
                             with GPU
                             output_path=output_s3_path)
```

Input: Encoder Embeddings

Content-type: application/json

```
{
  "instances" : [
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]},
    {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]},
    {"in0": [774, 14, 21, 206]}
  ]
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]}
{"in0": [774, 14, 21, 206]}
```

In both of these formats, you specify only one input type: “`in0`” or “`in1`.” The inference service then invokes the corresponding encoder and outputs the embeddings for each of the instances.

Output: Encoder Embeddings

Content-type: application/json

```
{
  "predictions": [
    {"embeddings": [
      [0.057368703186511, 0.030703511089086, 0.099890425801277, 0.063688032329082, 0.026327300816774, 0.0036375711
      {"embeddings": [
        [0.150190666317939, 0.05145975202322, 0.098204270005226, 0.064249359071254, 0.056249320507049, 0.01513972133
      ]
    }
  ]]
```

Content-type: application/jsonlines

```
{"embeddings":  
[0.057368703186511, 0.030703511089086, 0.099890425801277, 0.063688032329082, 0.026327300816774, 0.0036375711  
{"embeddings":  
[0.150190666317939, 0.05145975202322, 0.098204270005226, 0.064249359071254, 0.056249320507049, 0.01513972133}
```

The vector length of the embeddings output by the inference service is equal to the value of one of the following hyperparameters that you specify at training time: `enc0_token_embedding_dim`, `enc1_token_embedding_dim`, or `enc_dim`.

Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene. The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box. It uses the [Single Shot multibox Detector \(SSD\)](#) framework and supports two base networks: [VGG](#) and [ResNet](#). The network can be trained from scratch, or trained with models that have been pre-trained on the [ImageNet](#) dataset.

Topics

- [Input/Output Interface for the Object Detection Algorithm \(p. 402\)](#)
- [EC2 Instance Recommendation for the Object Detection Algorithm \(p. 405\)](#)
- [Object Detection Sample Notebooks \(p. 405\)](#)
- [How Object Detection Works \(p. 405\)](#)
- [Object Detection Hyperparameters \(p. 405\)](#)
- [Tune an Object Detection Model \(p. 410\)](#)
- [Object Detection Request and Response Formats \(p. 411\)](#)

Input/Output Interface for the Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode and supports RecordIO (`application/x-recordio`) for training in pipe mode. However you can also train in pipe mode using the image files (`image/png`, `image/jpeg`, and `application/x-image`), without creating RecordIO files, by using the augmented manifest format. The recommended input format for the Amazon SageMaker object detection algorithms is [Apache MXNet RecordIO](#). However, you can also use raw images in `.jpg` or `.png` format. The algorithm supports only `application/x-image` for inference.

Note

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

See the [Object Detection Sample Notebooks \(p. 405\)](#) for more details on data formats.

Train with the RecordIO Format

If you use the RecordIO format for training, specify both train and validation channels as values for the `InputDataConfig` parameter of the [CreateTrainingJob](#) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to `application/x-recordio`. An example of how to generate RecordIO file can be found in the object

detection sample notebook. You can also use tools from the [MXNet](#) example to generate RecordIO files for popular datasets like the [PASCAL Visual Object Classes](#) and [Common Objects in Context \(COCO\)](#).

Train with the Image Format

If you use the image format for training, specify `train`, `validation`, `train_annotation`, and `validation_annotation` channels as values for the `InputDataConfig` parameter of `CreateTrainingJob` request. Specify the individual image data (.jpg or .png) files for the train and validation channels. For annotation data, you can use the JSON format. Specify the corresponding .json files in the `train_annotation` and `validation_annotation` channels. Set the content type for all four channels to `image/png` or `image/jpeg` based on the image type. You can also use the content type `application/x-image` when your dataset contains both .jpg and .png images. The following is an example of a .json file.

```
{  
    "file": "your_image_directory/sample_image1.jpg",  
    "image_size": [  
        {  
            "width": 500,  
            "height": 400,  
            "depth": 3  
        }  
    ],  
    "annotations": [  
        {  
            "class_id": 0,  
            "left": 111,  
            "top": 134,  
            "width": 61,  
            "height": 128  
        },  
        {  
            "class_id": 0,  
            "left": 161,  
            "top": 250,  
            "width": 79,  
            "height": 143  
        },  
        {  
            "class_id": 1,  
            "left": 101,  
            "top": 185,  
            "width": 42,  
            "height": 130  
        }  
    ],  
    "categories": [  
        {  
            "class_id": 0,  
            "name": "dog"  
        },  
        {  
            "class_id": 1,  
            "name": "cat"  
        }  
    ]  
}
```

Each image needs a .json file for annotation, and the .json file should have the same name as the corresponding image. The name of above .json file should be "sample_image1.json". There are four properties in the annotation .json file. The property "file" specifies the relative path of the image file. For example, if your training images and corresponding .json files are stored in `s3://your_bucket/`

train/sample_image and s3://*your_bucket*/train_annotation, specify the path for your train and train_annotation channels as s3://*your_bucket*/train and s3://*your_bucket*/train_annotation, respectively.

In the .json file, the relative path for an image named sample_image1.jpg should be sample_image/sample_image1.jpg. The "image_size" property specifies the overall image dimensions. The SageMaker object detection algorithm currently only supports 3-channel images. The "annotations" property specifies the categories and bounding boxes for objects within the image. Each object is annotated by a "class_id" index and by four bounding box coordinates ("left", "top", "width", "height"). The "left" (x-coordinate) and "top" (y-coordinate) values represent the upper-left corner of the bounding box. The "width" (x-coordinate) and "height" (y-coordinate) values represent the dimensions of the bounding box. The origin (0, 0) is the upper-left corner of the entire image. If you have multiple objects within one image, all the annotations should be included in a single .json file. The "categories" property stores the mapping between the class index and class name. The class indices should be numbered successively and the numbering should start with 0. The "categories" property is optional for the annotation .json file

Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in `JSON Lines` format in which each line represents one sample. The images are specified using the 'source-ref' tag that points to the S3 location of the image. The annotations are provided under the "AttributeNames" parameter value as specified in the `CreateTrainingJob` request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the following example, the "AttributeNames" are contained in the list `["source-ref", "bounding-box"]`:

```
{"source-ref": "s3://your_bucket/image1.jpg", "bounding-box": {"image_size": [{"width": 500, "height": 400, "depth": 3}], "annotations": [{"class_id": 0, "left": 111, "top": 134, "width": 61, "height": 128}, {"class_id": 5, "left": 161, "top": 250, "width": 80, "height": 50}], "bounding-box-metadata": {"class-map": {"0": "dog", "5": "horse"}, "type": "groundtruth/object_detection"}}, {"source-ref": "s3://your_bucket/image2.jpg", "bounding-box": {"image_size": [{"width": 400, "height": 300, "depth": 3}], "annotations": [{"class_id": 1, "left": 100, "top": 120, "width": 43, "height": 78}], "bounding-box-metadata": {"class-map": {"1": "cat"}, "type": "groundtruth/object_detection"}}}
```

The order of "AttributeNames" in the input files matters when training the Object Detection algorithm. It accepts piped data in a specific order, with `image` first, followed by `annotations`. So the "AttributeNames" in this example are provided with "source-ref" first, followed by "bounding-box". When using Object Detection with Augmented Manifest, the value of parameter `RecordWrapperType` must be set as "RecordIO".

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Incremental Training

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

To use a pretrained model, in the `CreateTrainingJob` request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/`

`x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `base_network` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by Amazon SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the Amazon SageMaker object detection algorithm, see [Amazon SageMaker Object Detection Incremental Training](#) sample notebook. For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 613\)](#).

EC2 Instance Recommendation for the Object Detection Algorithm

For object detection, we support the following GPU instances for training: `ml.p2.xlarge`, `ml.p2.8xlarge`, `ml.p2.16xlarge`, `ml.p3.2xlarge`, `ml.p3.8xlarge` and `ml.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training. However, both CPU (such as C5 and M5) and GPU (such as P2 and P3) instances can be used for the inference. All the supported instance types for inference are itemized on [Amazon SageMaker ML Instance Types](#).

Object Detection Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Object Detection algorithm to train and host a model on the COCO dataset using the Single Shot multibox Detector algorithm, see [Object Detection using the Image and JSON format](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Object Detection Works

The object detection algorithm identifies and locates all instances of objects in an image from a known collection of object categories. The algorithm takes an image as input and outputs the category that the object belongs to, along with a confidence score that it belongs to the category. The algorithm also predicts the object's location and scale with a rectangular bounding box. Amazon SageMaker Object Detection uses the [Single Shot multibox Detector \(SSD\)](#) algorithm that takes a convolutional neural network (CNN) pretrained for classification task as the base network. SSD uses the output of intermediate layers as features for detection.

Various CNNs such as [VGG](#) and [ResNet](#) have achieved great performance on the image classification task. Object detection in Amazon SageMaker supports both VGG-16 and ResNet-50 as a base network for SSD. The algorithm can be trained in full training mode or in transfer learning mode. In full training mode, the base network is initialized with random weights and then trained on user data. In transfer learning mode, the base network weights are loaded from pretrained models.

The object detection algorithm uses standard data augmentation operations, such as flip, rescale, and jitter, on the fly internally to help avoid overfitting.

Object Detection Hyperparameters

In the [CreateTrainingJob](#) request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters that are used to help estimate the parameters of the model from a training dataset. The following table lists the hyperparameters provided by Amazon SageMaker for training the object detection algorithm. For more information about how object training works, see [How Object Detection Works \(p. 405\)](#).

Parameter Name	Description
<code>num_classes</code>	<p>The number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>num_training_samples</code>	<p>The number of training examples in the input dataset.</p> <p>Note If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter will be undefined and distributed training accuracy may be affected.</p> <p>Required</p> <p>Valid values: positive integer</p>
<code>base_network</code>	<p>The base network architecture to use.</p> <p>Optional</p> <p>Valid values: 'vgg-16' or 'resnet-50' Default value: 'vgg-16'</p>
<code>early_stopping</code>	<p><code>True</code> to use early stopping logic during training. <code>False</code> not to use it.</p> <p>Optional</p> <p>Valid values: <code>True</code> or <code>False</code> Default value: <code>False</code></p>
<code>early_stopping_min_epochs</code>	<p>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: positive integer Default value: 10</p>
<code>early_stopping_patience</code>	<p>The number of epochs to wait before ending training if no improvement, as defined by the <code>early_stopping_tolerance</code> hyperparameter, is made in the relevant metric. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: positive integer Default value: 5</p>

Parameter Name	Description
<code>early_stopping_tolerance</code>	<p>The tolerance value that the relative improvement in <code>validation:mAP</code>, the mean average precision (mAP), is required to exceed to avoid early stopping. If the ratio of the change in the mAP divided by the previous best mAP is smaller than the <code>early_stopping_tolerance</code> value set, early stopping considers that there is no improvement. It is used only when <code>early_stopping = True</code>.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 1$</p> <p>Default value: 0.0</p>
<code>image_shape</code>	<p>The image size for input images. We rescale the input image to a square image with this size. We recommend using 300 and 512 for better performance.</p> <p>Optional</p> <p>Valid values: positive integer ≥ 300</p> <p>Default: 300</p>
<code>epochs</code>	<p>The number of training epochs.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default: 30</p>
<code>freeze_layer_pattern</code>	<p>The regular expression (regex) for freezing layers in the base network. For example, if we set <code>freeze_layer_pattern = "^(conv1_ conv2_).*</code>", then any layers with a name that contains "conv1_" or "conv2_" are frozen, which means that the weights for these layers are not updated during training. The layer names can be found in the network symbol files vgg16-symbol.json and resnet-50-symbol.json. Freezing a layer means that its weights can not be modified further. This can reduce training time significantly in exchange for modest losses in accuracy. This technique is commonly used in transfer learning where the lower layers in the base network do not need to be retrained.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default: No layers frozen.</p>

Parameter Name	Description
<code>kv_store</code>	<p>The weight update synchronization mode used for distributed training. The weights can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See the Distributed Training MXNet tutorial for details.</p> <p>Note This parameter is not applicable to single machine training.</p> <p>Optional</p> <p>Valid values: 'dist_sync' or 'dist_async'</p> <ul style="list-style-type: none"> 'dist_sync': The gradients are synchronized after every batch with all the workers. With 'dist_sync', batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like a single machine with batch size n*b. 'dist_async': Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed. <p>Default: -</p>
<code>label_width</code>	<p>The force padding label width used to sync across training and validation data. For example, if one image in the data contains at most 10 objects, and each object's annotation is specified with 5 numbers, [class_id, left, top, width, height], then the <code>label_width</code> should be no smaller than (10*5 + header information length). The header information length is usually 2. We recommend using a slightly larger <code>label_width</code> for the training, such as 60 for this example.</p> <p>Optional</p> <p>Valid values: Positive integer large enough to accommodate the largest annotation information length in the data.</p> <p>Default: 350</p>
<code>learning_rate</code>	<p>The initial learning rate.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.001</p>

Parameter Name	Description
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate. Used in conjunction with the <code>lr_scheduler_step</code> parameter defined as $lr_{new} = lr_{old} * lr_scheduler_factor$.</p> <p>Optional</p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.1</p>
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. The learning rate is reduced by <code>lr_scheduler_factor</code> at epochs listed in a comma-delimited string: "epoch1, epoch2, ...". For example, if the value is set to "10, 20" and the <code>lr_scheduler_factor</code> is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default: empty string</p>
<code>mini_batch_size</code>	<p>The batch size for training. In a single-machine multi-gpu setting, each GPU handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. A large <code>mini_batch_size</code> usually leads to faster training, but it may cause out of memory problem. The memory usage is related to <code>mini_batch_size</code>, <code>image_shape</code>, and <code>base_network</code> architecture. For example, on a single p3.2xlarge instance, the largest <code>mini_batch_size</code> without an out of memory error is 32 with the <code>base_network</code> set to "resnet-50" and an <code>image_shape</code> of 300. With the same instance, you can use 64 as the <code>mini_batch_size</code> with the base network vgg-16 and an <code>image_shape</code> of 300.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default: 32</p>
<code>momentum</code>	<p>The momentum for sgd. Ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.9</p>
<code>nms_threshold</code>	<p>The non-maximum suppression threshold.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.45</p>

Parameter Name	Description
optimizer	<p>The optimizer types. For details on optimizer values, see MXNet's API.</p> <p>Optional</p> <p>Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta']</p> <p>Default: 'sgd'</p>
overlap_threshold	<p>The evaluation overlap threshold.</p> <p>Optional</p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.5</p>
use_pretrained_model	<p>Indicates whether to use a pre-trained model for training. If set to 1, then the pre-trained model with corresponding architecture is loaded and used for training. Otherwise, the network is trained from scratch.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default: 1</p>
weight_decay	<p>The weight decay coefficient for sgd and rmsprop. Ignored for other optimizers.</p> <p>Optional</p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.0005</p>

Tune an Object Detection Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Object Detection Algorithm

The object detection algorithm reports on a single metric during training: validation:mAP. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
validation:mAP	Mean Average Precision (mAP) computed on the validation set.	Maximize

Tunable Object Detection Hyperparameters

Tune the Amazon SageMaker object detection model with the following hyperparameters. The hyperparameters that have the greatest impact on the object detection objective metric are: `mini_batch_size`, `learning_rate`, and `optimizer`.

Parameter Name	Parameter Type	Recommended Ranges
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 0.5
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: 8, MaxValue: 64
<code>momentum</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999
<code>optimizer</code>	CategoricalParameterRanges	['sgd', 'adam', 'rmsprop', 'adadelta']
<code>weight_decay</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999

Object Detection Request and Response Formats

Request Format

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with `image/jpeg` and `image/png` content-types.

Response Formats

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

```
{"prediction": [
    [4.0, 0.86419455409049988, 0.3088374733924866, 0.07030484080314636, 0.7110607028007507,
     0.9345266819000244],
    [0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571,
     0.9712159633636475],
    [4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057,
     0.5967587828636169],
    [8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051,
     0.8985623121261597],
    [3.0, 0.42260299175977707, 0.019305512309074402, 0.08386176824569702,
     0.39093565940856934, 0.9574796557426453]
]}
```

Each row in this .json file contains an array that represents a detected object. Each of these object arrays consists of a list of six numbers. The first number is the predicted class label. The second number is the associated confidence score for the detection. The last four numbers represent the bounding box coordinates [xmin, ymin, xmax, ymax]. These output bounding box corner indices are normalized by the overall image size. Note that this encoding is different than that use by the input .json format. For example, in the first entry of the detection result, 0.3088374733924866 is the left coordinate (x-coordinate of upper-left corner) of the bounding box as a ratio of the overall image width, 0.07030484080314636 is the top coordinate (y-coordinate of upper-left corner) of the bounding box as a ratio of the overall image height, 0.7110607028007507 is the right coordinate (x-coordinate of

lower-right corner) of the bounding box as a ratio of the overall image width, and 0.9345266819000244 is the bottom coordinate (y-coordinate of lower-right corner) of the bounding box as a ratio of the overall image height.

To avoid unreliable detection results, you might want to filter out the detection results with low confidence scores. In the [object detection sample notebook](#), we provide scripts to remove the low confidence detections. Scripts are also provided to plot the bounding boxes on the original image.

For batch transform, the response is in JSON format, where the format is identical to the JSON format described above. The detection results of each image is represented as a JSON file. For example:

```
{"prediction": [[label_id, confidence_score, xmin, ymin, xmax, ymax], [label_id, confidence_score, xmin, ymin, xmax, ymax]]}
```

For more details on training and inference, see the [Object Detection Sample Notebooks \(p. 405\)](#).

OUTPUT: JSON Response Format

accept: application/json;annotation=1

```
{
  "image_size": [
    {
      "width": 500,
      "height": 400,
      "depth": 3
    }
  ],
  "annotations": [
    {
      "class_id": 0,
      "score": 0.943,
      "left": 111,
      "top": 134,
      "width": 61,
      "height": 128
    },
    {
      "class_id": 0,
      "score": 0.0013,
      "left": 161,
      "top": 250,
      "width": 79,
      "height": 143
    },
    {
      "class_id": 1,
      "score": 0.0133,
      "left": 101,
      "top": 185,
      "width": 42,
      "height": 130
    }
  ]
}
```

Principal Component Analysis (PCA) Algorithm

PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called *components*, which are composites of the original features that are

uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

In Amazon SageMaker, PCA operates in two modes, depending on the scenario:

- **regular:** For datasets with sparse data and a moderate number of observations and features.
- **randomized:** For datasets with both a large number of observations and features. This mode uses an approximation algorithm.

PCA uses tabular data.

The rows represent observations you want to embed in a lower dimensional space. The columns represent features that you want to find a reduced approximation for. The algorithm calculates the covariance matrix (or an approximation thereof in a distributed manner), and then performs the singular value decomposition on this summary to produce the principal components.

Topics

- [Input/Output Interface for the PCA Algorithm \(p. 413\)](#)
- [EC2 Instance Recommendation for the PCA Algorithm \(p. 413\)](#)
- [PCA Sample Notebooks \(p. 413\)](#)
- [How PCA Works \(p. 414\)](#)
- [PCA Hyperparameters \(p. 415\)](#)
- [PCA Response Formats \(p. 416\)](#)

[Input/Output Interface for the PCA Algorithm](#)

For training, PCA expects data provided in the train channel, and optionally supports a dataset passed to the test dataset, which is scored by the final algorithm. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, PCA supports `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Results are returned in either `application/json` or `application/x-recordio-protobuf` format with a vector of "projections."

For more details on training and inference file formats, see the [PCA Sample Notebooks \(p. 413\)](#) and the [PCA Response Formats \(p. 416\)](#).

For more information on input and output file formats, see [PCA Response Formats \(p. 416\)](#) for inference and the [PCA Sample Notebooks \(p. 413\)](#).

[EC2 Instance Recommendation for the PCA Algorithm](#)

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

[PCA Sample Notebooks](#)

For a sample notebook that shows how to use the Amazon SageMaker Principal Component Analysis algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see [An Introduction to PCA with MNIST](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks

using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How 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, \dots, x_n each of dimension $1 * d$, 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 $1 * d$ 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 `num_components` singular vectors form the model.

Note

If `subtract_mean` is `False`, we avoid computing and subtracting $\sum x_i$.

Use this algorithm when the dimension d of the vectors is small enough so that d^2 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_t of dimension $b * d$, we randomly initialize a $(\text{num_components} + \text{extra_components}) * b$ matrix that we multiply by each mini-batch, to create a $(\text{num_components} + \text{extra_components}) * d$ matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final $(\text{num_components} + \text{extra_components}) * d$ matrix. The top right `num_components` singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let $\ell = \text{num_components} + \text{extra_components}$. Given a mini-batch X_t of dimension $b * d$, the worker draws a random matrix H_t of dimension $\ell * b$. 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 $+/-1$ or a *FJLT* (fast Johnson Lindenstrauss transform; for information, see [FJLT Transforms](#) and the follow-up papers). The worker then computes $H_t X_t$ and maintains $B = \sum H_t X_t$. The worker also maintains h^T , the sum of columns of H_1, \dots, 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 the sums of the respective inputs. It then computes $C = B - (1/n)h^T s$, and finds its singular value decomposition. The top-right singular vectors and singular values of C are used as the approximate solution to the problem.

PCA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see [How PCA Works \(p. 414\)](#).

Parameter Name	Description
<code>feature_dim</code>	<p>Input dimension. Required Valid values: positive integer</p>
<code>mini_batch_size</code>	<p>Number of rows in a mini-batch. Required Valid values: positive integer</p>
<code>num_components</code>	<p>The number of principal components to compute. Required Valid values: positive integer</p>
<code>algorithm_mode</code>	<p>Mode for computing the principal components. Optional Valid values: <i>regular</i> or <i>randomized</i> Default value: <i>regular</i></p>
<code>extra_components</code>	<p>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 <code>num_components</code>. Valid for <i>randomized</i> mode only. Optional Valid values: Non-negative integer or -1 Default value: -1</p>

Parameter Name	Description
subtract_mean	<p>Indicates whether the data should be unbiased both during training and at inference.</p> <p>Optional</p> <p>Valid values: One of <i>true</i> or <i>false</i></p> <p>Default value: <i>true</i></p>

PCA Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the Amazon SageMaker PCA algorithm.

JSON Response Format

Accept—application/json

```
{
  "projections": [
    {
      "projection": [1.0, 2.0, 3.0, 4.0, 5.0]
    },
    {
      "projection": [6.0, 7.0, 8.0, 9.0, 0.0]
    },
    ...
  ]
}
```

JSONLINES Response Format

Accept—application/jsonlines

```
{ "projection": [1.0, 2.0, 3.0, 4.0, 5.0] }
{ "projection": [6.0, 7.0, 8.0, 9.0, 0.0] }
```

RECORDIO Response Format

Accept—application/x-recordio-protobuf

```
[
  Record = {
    features = {},
    label = {
      'projection': {
        keys: [],
        values: [1.0, 2.0, 3.0, 4.0, 5.0]
      }
    }
  },
  Record = {
    features = {},
    label = {
      'projection': {
        keys: [],
        values: [1.0, 2.0, 3.0, 4.0, 5.0]
      }
    }
  }
]
```

```
        }
    }
]
```

Random Cut Forest (RCF) Algorithm

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a data set can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

With each data point, RCF associates an anomaly score. Low score values indicate that the data point is considered "normal." High values indicate the presence of an anomaly in the data. The definitions of "low" and "high" depend on the application but common practice suggests that scores beyond three standard deviations from the mean score are considered anomalous.

While there are many applications of anomaly detection algorithms to one-dimensional time series data such as traffic volume analysis or sound volume spike detection, RCF is designed to work with arbitrary-dimensional input. Amazon SageMaker RCF scales well with respect to number of features, data set size, and number of instances.

Topics

- [Input/Output Interface for the RCF Algorithm \(p. 417\)](#)
- [Instance Recommendations for the RCF Algorithm \(p. 418\)](#)
- [RCF Sample Notebooks \(p. 418\)](#)
- [How RCF Works \(p. 418\)](#)
- [RCF Hyperparameters \(p. 421\)](#)
- [Tune an RCF Model \(p. 422\)](#)
- [RCF Response Formats \(p. 422\)](#)

Input/Output Interface for the RCF Algorithm

Amazon SageMaker Random Cut Forest supports the `train` and `test` data channels. The optional `test` channel is used to compute accuracy, precision, recall, and F1-score metrics on labeled data. Train and test data content types can be either `application/x-recordio-protobuf` or `text/csv` formats. For the `test` data, when using `text/csv` format, the content must be specified as `text/csv;label_size=1` where the first column of each row represents the anomaly label: "1" for an anomalous data point and "0" for a normal data point. You can use either File mode or Pipe mode to train RCF models on data that is formatted as `recordIO-wrapped-protobuf` or as CSV

Also note that the `train` channel only supports `S3DataDistributionType=ShardedByS3Key` and the `test` channel only supports `S3DataDistributionType=FullyReplicated`. The S3 distribution type can be specified using the Python SDK as follows:

```
import sagemaker

# specify Random Cut Forest training job information and hyperparameters
rcf = sagemaker.estimator.Estimator(...)

# explicitly specify "ShardedByS3Key" distribution type
train_data = sagemaker.s3_input(
    s3_data=s3_training_data_location,
    content_type='text/csv;label_size=0',
```

```
distribution='ShardedByS3Key')  
  
# run the training job on input data stored in S3  
rcf.fit({'train': train_data})
```

See the [S3DataSource](#) for more information on customizing the S3 data source attributes. Finally, in order to take advantage of multi-instance training the training data must be partitioned into at least as many files as instances.

For inference, RCF supports application/x-recordio-protobuf, text/csv and application/json input data content types. See the [Common Data Formats for Built-in Algorithms \(p. 280\)](#) documentation for more information. RCF inference returns application/x-recordio-protobuf or application/json formatted output. Each record in these output data contains the corresponding anomaly scores for each input data point. See [Common Data Formats--Inference](#) for more information.

For more information on input and output file formats, see [RCF Response Formats \(p. 422\)](#) for inference and the [RCF Sample Notebooks \(p. 418\)](#).

Instance Recommendations for the RCF Algorithm

For training, we recommend the m1.m4, m1.c4, and m1.c5 instance families. For inference we recommend using a m1.c5.xl instance type in particular, for maximum performance as well as minimized cost per hour of usage. Although the algorithm could technically run on GPU instance types it does not take advantage of GPU hardware.

RCF Sample Notebooks

For an example of how to train an RCF model and perform inferences with it, see the [Introduction to SageMaker Random Cut Forests](#) notebook. For a sample notebook that uses the Amazon SageMaker Random Cut Forest algorithm for anomaly detection, see [An Introduction to SageMaker Random Cut Forests](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

How RCF Works

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree. The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree. The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size. The RCF algorithm is based on the one described in reference [1].

Sample Data Randomly

The first step in the RCF algorithm is to obtain a random sample of the training data. In particular, suppose we want a sample of size K from N total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw K elements from this set. However,

frequently the training data is too large to fit all at once, and this approach isn't feasible. Instead, we use a technique called reservoir sampling.

Reservoir sampling is an algorithm for efficiently drawing random samples from a dataset $S = \{S_1, \dots, S_N\}$ where the elements in the dataset can only be observed one at a time or in batches. In fact, reservoir sampling works even when N is not known *a priori*. If only one sample is requested, such as when $K = 1$, the algorithm is like this:

Algorithm: Reservoir Sampling

- Input: dataset or data stream $S = \{S_1, \dots, S_N\}$
- Initialize the random sample $X = S_1$
- For each observed sample $S_n, n = 2, \dots, N$:
 - Pick a uniform random number $\xi \in [0, 1]$
 - If $\xi < 1/n$
 - Set $X = S_n$
- Return X

This algorithm selects a random sample such that $P(X = S_n) = 1/N$ for all $n = 1, \dots, N$. When $K > 1$ the algorithm is more complicated. Additionally, a distinction must be made between random sampling that is with and without replacement. RCF performs an augmented reservoir sampling without replacement on the training data based on the algorithms described in [2].

Train a RCF Model and Produce Inferences

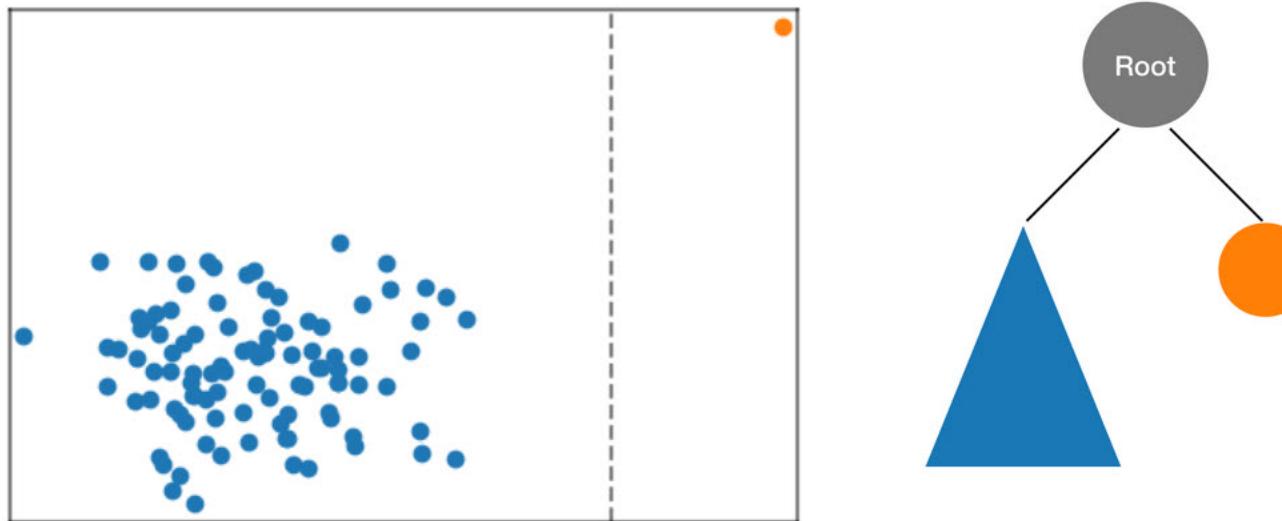
The next step in RCF is to construct a random cut forest using the random sample of data. First, the sample is partitioned into a number of equal-sized partitions equal to the number of trees in the forest. Then, each partition is sent to an individual tree. The tree recursively organizes its partition into a binary tree by partitioning the data domain into bounding boxes.

This procedure is best illustrated with an example. Suppose a tree is given the following two-dimensional dataset. The corresponding tree is initialized to the root node:



A two-dimensional dataset where the majority of data lies in a cluster (blue) except for one anomalous data point (orange). The tree is initialized with a root node.

The RCF algorithm organizes these data in a tree by first computing a bounding box of the data, selecting a random dimension (giving more weight to dimensions with higher "variance"), and then randomly determining the position of a hyperplane "cut" through that dimension. The two resulting subspaces define their own subtree. In this example, the cut happens to separate a lone point from the remainder of the sample. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.



A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

Bounding boxes are then computed for the left and right halves of the data and the process is repeated until every leaf of the tree represents a single data point from the sample. Note that if the lone point is sufficiently far away then it is more likely that a random cut would result in point isolation. This observation provides the intuition that tree depth is, loosely speaking, inversely proportional to the anomaly score.

When performing inference using a trained RCF model the final anomaly score is reported as the average across scores reported by each tree. Note that it is often the case that the new data point does not already reside in the tree. To determine the score associated with the new point the data point is inserted into the given tree and the tree is efficiently (and temporarily) reassembled in a manner equivalent to the training process described above. That is, the resulting tree is as if the input data point were a member of the sample used to construct the tree in the first place. The reported score is inversely proportional to the depth of the input point within the tree.

Choose Hyperparameters

The primary hyperparameters used to tune the RCF model are `num_trees` and `num_samples_per_tree`. Increasing `num_trees` has the effect of reducing the noise observed in anomaly scores since the final score is the average of the scores reported by each tree. While the optimal value is application-dependent we recommend using 100 trees to begin with as a balance between score noise and model complexity. Note that inference time is proportional to the number of trees. Although training time is also affected it is dominated by the reservoir sampling algorithm described above.

The parameter `num_samples_per_tree` is related to the expected density of anomalies in the dataset. In particular, `num_samples_per_tree` should be chosen such that $1/\text{num_samples_per_tree}$ approximates the ratio of anomalous data to normal data. For example, if 256 samples are used in each

tree then we expect our data to contain anomalies 1/256 or approximately 0.4% of the time. Again, an optimal value for this hyperparameter is dependent on the application.

References

1. Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. "Robust random cut forest based anomaly detection on streams." In *International Conference on Machine Learning*, pp. 2712-2721. 2016.
2. Byung-Hoon Park, George Ostrouchov, Nagiza F. Samatova, and Al Geist. "Reservoir-based random sampling with replacement from data stream." In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 492-496. Society for Industrial and Applied Mathematics, 2004.

RCF Hyperparameters

In the [CreateTrainingJob](#) request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker RCF algorithm. For more information, including recommendations on how to choose hyperparameters, see [How RCF Works \(p. 418\)](#).

Parameter Name	Description
<code>feature_dim</code>	The number of features in the data set. (If you are using the client libraries through a notebook, this value is calculated for you and need not be specified.) Required (When the job is run through the console.) Valid values: Positive integer (min: 1, max: 10000)
<code>eval_metrics</code>	A list of metrics used to score a labeled test data set. The following metrics can be selected for output: <ul style="list-style-type: none"> • <code>accuracy</code> - returns fraction of correct predictions. • <code>precision_recall_fscore</code> - returns the positive and negative precision, recall, and F1-scores. Optional Valid values: a list with possible values taken from <code>accuracy</code> or <code>precision_recall_fscore</code> . Default value: Both <code>accuracy</code> , <code>precision_recall_fscore</code> are calculated.
<code>num_samples_per_tree</code>	Number of random samples given to each tree from the training data set. Optional Valid values: Positive integer (min: 1, max: 2048) Default value: 256
<code>num_trees</code>	Number of trees in the forest. Optional Valid values: Positive integer (min: 50, max: 1000)

Parameter Name	Description
	Default value: 100

Tune an RCF Model

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

The Amazon SageMaker RCF algorithm is an unsupervised anomaly-detection algorithm that requires a labeled test dataset for hyperparameter optimization. It calculates anomaly scores for test datapoints and then labels the datapoints as anomalous if their scores are beyond three standard deviations from the mean score. This is known as the three-sigma limit heuristic. The F1 score is emitted based on the difference between calculated labels and actual labels. The hyperparameter tuning job finds the model that maximizes that score. The success of hyperparameter optimization depends on the applicability of the three-sigma limit heuristic to the test dataset.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the RCF Algorithm

The RCF algorithm computes the following metric during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
test:f1	F1 score on the test dataset, based on the difference between calculated labels and actual labels.	Maximize

Tunable RCF Hyperparameters

You can tune a RCF model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
num_samples_per_tree	IntegerParameterRanges	MinValue: 1, MaxValue:2048
num_trees	IntegerParameterRanges	MinValue: 50, MaxValue:1000

RCF Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). Note that Amazon SageMaker Random Cut Forest supports both dense and sparse JSON and RecordIO formats. This topic contains a list of the available output formats for the Amazon SageMaker RCF algorithm.

JSON Response Format

ACCEPT: application/json.

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

JSONLINES Response Format

ACCEPT: application/jsonlines.

```
{"score": 0.02},  
{"score": 0.25}
```

RECORDIO Response Format

ACCEPT: application/x-recordio-protobuf.

```
[  
  
    Record = {  
  
        features = {},  
  
        label = {  
  
            'score': {  
  
                keys: [],  
  
                values: [0.25] # float32  
            }  
        }  
    }  
]
```

```
        }

    }

},
Record = {

    features = {},

    label = {

        'score': {

            keys: [],

            values: [0.23] # float32

        }
    }

}

]
}
```

Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm provides a fine-grained, pixel-level approach to developing computer vision applications. It tags every pixel in an image with a class label from a predefined set of classes. Tagging is fundamental for understanding scenes, which is critical to an increasing number of computer vision applications, such as self-driving vehicles, medical imaging diagnostics, and robot sensing.

For comparison, the Amazon SageMaker [Image Classification Algorithm \(p. 324\)](#) is a supervised learning algorithm that analyzes only whole images, classifying them into one of multiple output categories. The [Object Detection Algorithm \(p. 402\)](#) is a supervised learning algorithm that detects and classifies all instances of an object in an image. It indicates the location and scale of each object in the image with a rectangular bounding box.

Because the semantic segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects contained in the image. The segmentation output is represented as an RGB or grayscale image, called a *segmentation mask*. A segmentation mask is an RGB (or grayscale) image with the same shape as the input image.

Amazon SageMaker semantic segmentation algorithm is built using the [MXNet Gluon framework](#) and the [Gluon CV toolkit](#) provides you with a choice of three build-in algorithms to train a deep neural network. You can use the [Fully-Convolutional Network \(FCN\) algorithm](#), [Pyramid Scene Parsing \(PSP\) algorithm](#), or [DeepLabV3](#).

Each of the three algorithms has two distinct components:

- The *backbone* (or *encoder*)—A network that produces reliable activation maps of features.
- The *decoder*—A network that constructs the segmentation mask from the encoded activation maps.

You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: [ResNet50](#) or [ResNet101](#). These backbones include pretrained artifacts that were originally trained on the [ImageNet](#) classification task. You can fine-tune these backbones for segmentation using your own data. Or, you can initialize and train these networks from scratch using only your own data. The decoders are never pretrained.

To deploy the trained model for inference, use the Amazon SageMaker hosting service. During inference, you can request the segmentation mask either as a PNG image or as a set of probabilities for each class for each pixel. You can use these masks as part of a larger pipeline that includes additional downstream image processing or other applications.

Topics

- [Semantic Segmentation Sample Notebooks \(p. 425\)](#)
- [Input/Output Interface for the Semantic Segmentation Algorithm \(p. 425\)](#)
- [EC2 Instance Recommendation for the Semantic Segmentation Algorithm \(p. 428\)](#)
- [Semantic Segmentation Hyperparameters \(p. 428\)](#)

Semantic Segmentation Sample Notebooks

For a sample Jupyter notebook that uses the Amazon SageMaker semantic segmentation algorithm to train a model and deploy it to perform inferences, see the [Semantic Segmentation Example](#). For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#).

To see a list of all of the Amazon SageMaker samples, create and open a notebook instance, and choose the [SageMaker Examples](#) tab. The example semantic segmentation notebooks are located under [Introduction to Amazon algorithms](#). To open a notebook, choose its **Use** tab, and choose **Create copy**.

Input/Output Interface for the Semantic Segmentation Algorithm

Amazon SageMaker semantic segmentation expects the customer's training dataset to be on [Amazon Simple Storage Service \(Amazon S3\)](#). Once trained, it produces the resulting model artifacts on Amazon S3. The input interface format for the Amazon SageMaker semantic segmentation is similar to that of most standardized semantic segmentation benchmarking datasets. The dataset in Amazon S3 is expected to be presented in two channels, one for **train** and one for **validation** using four directories, two for images and two for annotations. Annotations are expected to be uncompressed PNG images. The dataset might also have a label map that describes how the annotation mappings are established. If not, the algorithm uses a default. It also supports the augmented manifest image format (`application/x-image`) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an `image/jpeg` content type.

How Training Works

The training data is split into four directories: `train`, `train_annotation`, `validation`, and `validation_annotation`. There is a channel for each of these directories. The dataset also expected to have one `label_map.json` file per channel for `train_annotation` and `validation_annotation` respectively. If you don't provide these JSON files, Amazon SageMaker provides the default set label map.

The dataset specifying these files should look similar to the following example:

```
s3://bucket_name
  |
  |- train
  |   |
  |   |- 0000.jpg
  |   |- coffee.jpg
  |- validation
  |   |
  |   |- 00a0.jpg
  |   |- bananna.jpg
  |- train_annotation
  |   |
  |   |- 0000.png
  |   |- coffee.png
  |- validation_annotation
  |   |
  |   |- 00a0.png
  |   |- bananna.png
  |- label_map
      |
      |- train_label_map.json
      |- validation_label_map.json
```

Every JPG image in the `train` and `validation` directories has a corresponding PNG label image with the same name in the `train_annotation` and `validation_annotation` directories. This naming convention helps the algorithm to associate a label with its corresponding image during training. The `train`, `train_annotation`, `validation`, and `validation_annotation` channels are mandatory. The annotations are single-channel PNG images. The format works as long as the metadata (modes) in the image helps the algorithm read the annotation images into a single-channel 8-bit unsigned integer. For more information on our support for modes, see the [Python Image Library documentation](#). We recommend using the 8-bit pixel, true color `P` mode.

The image that is encoded is a simple 8-bit integer when using modes. To get from this mapping to a map of a label, the algorithm uses one mapping file per channel, called the *label map*. The label map is used to map the values in the image with actual label indices. In the default label map, which is provided by default if you don't provide one, the pixel value in an annotation matrix (image) directly index the label. These images can be grayscale PNG files or 8-bit indexed PNG files. The label map file for the unscaled default case is the following:

```
{
  "scale": "1"
}
```

To provide some contrast for viewing, some annotation software scales the label images by a constant amount. To support this, the Amazon SageMaker semantic segmentation algorithm provides a rescaling option to scale down the values to actual label values. When scaling down doesn't convert the value to an appropriate integer, the algorithm defaults to the greatest integer less than or equal to the scale value. The following code shows how to set the scale value to rescale the label values:

```
{
  "scale": "3"
```

}

The following example shows how this "scale" value is used to rescale the `encoded_label` values of the input annotation image when they are mapped to the `mapped_label` values to be used in training. The label values in the input annotation image are 0, 3, 6, with scale 3, so they are mapped to 0, 1, 2 for training:

```
encoded_label = [0, 3, 6]
mapped_label = [0, 1, 2]
```

In some cases, you might need to specify a particular color mapping for each class. Use the `map` option in the label mapping as shown in the following example of a `label_map` file:

```
{
  "map": {
    "0": 5,
    "1": 0,
    "2": 2
  }
}
```

This label mapping for this example is:

```
encoded_label = [0, 5, 2]
mapped_label = [1, 0, 2]
```

With label mappings, you can use different annotation systems and annotation software to obtain data without a lot of preprocessing. You can provide one label map per channel. The files for a label map in the `label_map` channel must follow the naming conventions for the four directory structure. If you don't provide a label map, the algorithm assumes a scale of 1 (the default).

Training with the Augmented Manifest Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. The augmented manifest file contains data objects and should be in [JSON Lines](#) format, as described in the [CreateTrainingJob](#) request. Each line in the manifest is an entry containing the Amazon S3 URI for the image and the URI for the annotation image.

Each JSON object in the manifest file must contain a `source-ref` key. The `source-ref` key should contain the value of the Amazon S3 URI to the image. The labels are provided under the `AttributeNames` parameter value as specified in the [CreateTrainingJob](#) request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the example below, the `AttributeNames` are contained in the list of image and annotation references `["source-ref", "city-streets-ref"]`. These names must have `-ref` appended to them. When using the Semantic Segmentation algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be `"RecordIO"` and value of the `ContentType` parameter must be `application/x-recordio`.

```
{"source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-streets-metadata": {"job-name": "label-city-streets", }}
```

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Incremental Training

You can also seed the training of a new model with a model that you trained previously using Amazon SageMaker. This incremental training saves training time when you want to train a new model with the

same or similar data. Currently, incremental training is supported only for models trained with the built-in Amazon SageMaker Semantic Segmentation.

To use your own pre-trained model, specify the `ChannelName` as "model" in the `InputDataConfig` for the [CreateTrainingJob](#) request. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The `backbone`, `algorithm`, `crop_size`, and `num_classes` input parameters that define the network architecture must be consistently specified in the input hyperparameters of the new model and the pre-trained model that you upload to the model channel. For the pretrained model file, you can use the compressed (.tar.gz) artifacts from Amazon SageMaker outputs. You can use either RecordIO or Image formats for input data. For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 613\)](#).

Produce Inferences

To query a trained model that is deployed to an endpoint, you need to provide an image and an `AcceptType` that denotes the type of output required. The endpoint takes JPEG images with an `image/jpeg` content type. If you request an `AcceptType` of `image/png`, the algorithm outputs a PNG file with a segmentation mask in the same format as the labels themselves. If you request an accept type of `application/x-recordio-protobuf`, the algorithm returns class probabilities encoded in recordio-protobuf format. The latter format outputs a 3D tensor where the third dimension is the same size as the number of classes. This component denotes the probability of each class label for each pixel.

EC2 Instance Recommendation for the Semantic Segmentation Algorithm

The Amazon SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes. The algorithm can be trained using [P2/P3 EC2 Amazon Elastic Compute Cloud \(Amazon EC2\)](#) instances in single machine configurations. It supports the following GPU instances for training:

- `ml.p2.xlarge`
- `ml.p2.8xlarge`
- `ml.p2.16xlarge`
- `ml.p3.2xlarge`
- `ml.p3.8xlarge`
- `ml.p3.16xlarge`

For inference, you can use either CPU instances (such as c5 and m5) and GPU instances (such as p2 and p3) or both. For information about the instance types that provide varying combinations of CPU, GPU, memory, and networking capacity for inference, see [Amazon SageMaker ML Instance Types](#).

Semantic Segmentation Hyperparameters

The following tables list the hyperparameters supported by the Amazon SageMaker semantic segmentation algorithm for network architecture, data inputs, and training. You specify Semantic Segmentation for training in the `AlgorithmName` of the [CreateTrainingJob](#) request.

Network Architecture Hyperparameters

Parameter Name	Description
<code>backbone</code>	<p>The backbone to use for the algorithm's encoder component.</p> <p>Optional</p> <p>Valid values: <code>resnet-50</code>, <code>resnet-101</code></p> <p>Default value: <code>resnet-50</code></p>

Parameter Name	Description
use_pretrained_model	<p>Whether a pretrained model is to be used for the backbone.</p> <p>Optional</p> <p>Valid values: <code>True</code>, <code>False</code></p> <p>Default value: <code>True</code></p>
algorithm	<p>The algorithm to use for semantic segmentation.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>fcn</code>: Fully-Convolutional Network (FCN) algorithm • <code>psp</code>: Pyramid Scene Parsing (PSP) algorithm • <code>deeplab</code>: DeepLab V3 algorithm <p>Default value: <code>fcn</code></p>

Data Hyperparameters

Parameter Name	Description
num_classes	<p>The number of classes to segment.</p> <p>Required</p> <p>Valid values: $2 \leq$ positive integer ≤ 254</p>
num_training_samples	<p>The number of samples in the training data. The algorithm uses this value to set up the learning rate scheduler.</p> <p>Required</p> <p>Valid values: positive integer</p>
crop_size	<p>The image size for input during training. We randomly rescale the input image while preserving the aspect ratio and then take a random square crop with side length <code>crop_size</code>. Input <code>crop_sizes</code> will be automatically rounded up to multiples of 8.</p> <p>Optional</p> <p>Valid values: positive integer > 16</p> <p>Default value: 480</p>

Training Hyperparameters

Parameter Name	Description
early_stopping	Whether to use early stopping logic during training.

Parameter Name	Description
	Optional Valid values: <code>True</code> , <code>False</code> Default value: <code>False</code>
<code>early_stopping_min_epochs</code>	The minimum number of epochs that must be run. Optional Valid values: integer Default value: 5
<code>early_stopping_patient</code>	The number of epochs that meet the tolerance for lower performance before the algorithm enforces an early stop. Optional Valid values: integer Default value: 4
<code>early_stopping_tolerance</code>	If the relative improvement of the score of the training job, the mIOU, is smaller than this value, early stopping considers the epoch as not improved. This is used only when <code>early_stopping = True</code> . Optional Valid values: $0 \leq \text{float} \leq 1$ Default value: 0.0
<code>epochs</code>	The number of epochs with which to train. Optional Valid values: positive integer Default value: 30
<code>gamma1</code>	The decay factor for the moving average of the squared gradient for <code>rmsprop</code> . Used only for <code>rmsprop</code> . Optional Valid values: $0 \leq \text{float} \leq 1$ Default value: 0.9
<code>gamma2</code>	The momentum factor for <code>rmsprop</code> . Optional Valid values: $0 \leq \text{float} \leq 1$ Default value: 0.9

Parameter Name	Description
<code>learning_rate</code>	<p>The initial learning rate.</p> <p>Optional</p> <p>Valid values: $0 < \text{float} \leq 1$</p> <p>Default value: 0.001</p>
<code>lr_scheduler</code>	<p>The shape of the learning rate schedule that controls its decrease over time.</p> <p>Optional</p> <p>Valid values:</p> <ul style="list-style-type: none"> • <code>step</code>: A stepwise decay, where the learning rate is reduced by a factor at certain intervals. • <code>poly</code>: A smooth decay using a polynomial function. • <code>cosine</code>: A smooth decay using a cosine function. <p>Default value: <code>poly</code></p>
<code>mini_batch_size</code>	<p>The batch size for training. Using a large <code>mini_batch_size</code> usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the <code>mini_batch_size</code> and <code>image_shape</code> parameters, and the backbone architecture.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 4</p>
<code>momentum</code>	<p>The momentum for the <code>sgd</code> optimizer. When you use other optimizers, the semantic segmentation algorithm ignores this parameter.</p> <p>Optional</p> <p>Valid values: $0 < \text{float} \leq 1$</p> <p>Default value: 0.9</p>

Parameter Name	Description
optimizer	<p>The type of optimizer. For more information about an optimizer, choose the appropriate link:</p> <ul style="list-style-type: none"> • adam: Adaptive momentum estimation • adagrad: Adaptive gradient descent • nag: Nesterov accelerated gradient • rmsprop: Root mean square propagation • sgd: Stochastic gradient descent <p>Optional</p> <p>Valid values: adam, adagrad, nag, rmsprop, sgd</p> <p>Default value: sgd</p>
validation_mini_batch_size	<p>The batch size for validation. A large mini_batch_size usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the mini_batch_size and image_shape parameters, and the backbone architecture.</p> <ul style="list-style-type: none"> • To score the validation on the entire image without cropping the images, set this parameter to 1. Use this option if you want to measure performance on the entire image as a whole. <p>Note Setting the validation_mini_batch_size parameter to 1 causes the algorithm to create a new network model for every image. This might slow validation and training.</p> <ul style="list-style-type: none"> • To crop images to the size specified in the crop_size parameter, even during evaluation, set this parameter to a value greater than 1. <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 4</p>
weight_decay	<p>The weight decay coefficient for the sgd optimizer. When you use other optimizers, the algorithm ignores this parameter.</p> <p>Optional</p> <p>Valid values: $0 < \text{float} < 1$</p> <p>Default value: 0.0001</p>

Sequence-to-Sequence Algorithm

Amazon SageMaker Sequence to Sequence is a supervised learning algorithm where the input is a sequence of tokens (for example, text, audio) and the output generated is another sequence of tokens. Example applications include: machine translation (input a sentence from one language and predict what that sentence would be in another language), text summarization (input a longer string of words and predict a shorter string of words that is a summary), speech-to-text (audio clips converted into output

sentences in tokens). Recently, problems in this domain have been successfully modeled with deep neural networks that show a significant performance boost over previous methodologies. Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

Topics

- [Input/Output Interface for the Sequence-to-Sequence Algorithm \(p. 433\)](#)
- [EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm \(p. 434\)](#)
- [Sequence-to-Sequence Sample Notebooks \(p. 434\)](#)
- [How Sequence-to-Sequence Works \(p. 434\)](#)
- [Sequence-to-Sequence Hyperparameters \(p. 435\)](#)
- [Tune a Sequence-to-Sequence Model \(p. 443\)](#)

[Input/Output Interface for the Sequence-to-Sequence Algorithm](#)

Training

Amazon SageMaker seq2seq expects data in RecordIO-Protobuf format. However, the tokens are expected as integers, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in [the seq2seq example notebook](#). In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

- **train**: It should contain the training data (for example, the `train.rec` file generated by the preprocessing script).
- **validation**: It should contain the validation data (for example, the `val.rec` file generated by the preprocessing script).
- **vocab**: It should contain two vocabulary files (`vocab.src.json` and `vocab.trg.json`)

If the algorithm doesn't find data in any of these three channels, training results in an error.

Inference

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the `application/json` format. Otherwise, use the `recordio-protobuf` format to work with the integer encoded data. Both mode supports batching of input data. `application/json` format also allows you to visualize the attention matrix.

- `application/json`: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be `application/json`. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:

`configuration: {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 `applications/x-recordio-protobuf`. For this format, the source sequences must be converted into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For batch transform, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be application/jsonlines. The format for input is as follows:

```
content-type: application/jsonlines

{"source": "source_sequence_0"}
{"source": "source_sequence_1"}
```

The format for response is as follows:

```
accept: application/jsonlines

{"target": "predicted_sequence_0"}
{"target": "predicted_sequence_1"}
```

For additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference, see the [Sequence-to-Sequence Sample Notebooks \(p. 434\)](#).

EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm

Currently Amazon SageMaker seq2seq is only supported on GPU instance types and is only set up to train on a single machine. But it does also offer support for multiple GPUs.

Sequence-to-Sequence Sample Notebooks

For a sample notebook that shows how to use the Amazon SageMaker Sequence to Sequence algorithm to train a English-German translation model, see [Machine Translation English-German Example Using SageMaker Seq2Seq](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How Sequence-to-Sequence Works

Typically, a neural network for sequence-to-sequence modeling consists of a few layers, including:

- An **embedding layer**. In this layer, the input matrix, which is input tokens encoded in a sparse way (for example, one-hot encoded) are mapped to a dense feature layer. This is required because a high-dimensional feature vector is more capable of encoding information regarding a particular token (word for text corpora) than a simple one-hot-encoded vector. It is also a standard practice to initialize this embedding layer with a pre-trained word vector like [FastText](#) or [Glove](#) or to initialize it randomly and learn the parameters during training.
- An **encoder layer**. After the input tokens are mapped into a high-dimensional feature space, the sequence is passed through an encoder layer to compress all the information from the input embedding layer (of the entire sequence) into a fixed-length feature vector. Typically, an encoder is made of RNN-type networks like long short-term memory (LSTM) or gated recurrent units (GRU). ([Colah's blog](#) explains LSTM in a great detail.)
- A **decoder layer**. The decoder layer takes this encoded feature vector and produces the output sequence of tokens. This layer is also usually built with RNN architectures (LSTM and GRU).

The whole model is trained jointly to maximize the probability of the target sequence given the source sequence. This model was first introduced by [Sutskever et al.](#) in 2014.

Attention mechanism. The disadvantage of an encoder-decoder framework is that model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain. To tackle this problem, in 2015, Bahdanau et al. proposed the [attention mechanism](#). In an attention mechanism, the decoder tries to find the location in the encoder sequence where the most important information could be located and uses that information and previously decoded words to predict the next token in the sequence.

For more in details, see the whitepaper [Effective Approaches to Attention-based Neural Machine Translation](#) by Luong, et al. that explains and simplifies calculations for various attention mechanisms. Additionally, the whitepaper [Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](#) by Wu, et al. describes Google's architecture for machine translation, which uses skip connections between encoder and decoder layers.

Sequence-to-Sequence Hyperparameters

Parameter Name	Description
batch_size	<p>Mini batch size for gradient descent.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 64</p>
beam_size	<p>Length of the beam for beam search. Used during training for computing bleu and used during inference.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
bleu_sample_size	<p>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).</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
bucket_width	<p>Returns (source,target) buckets up to (max_seq_len_source,max_seq_len_target). The longer side of the data uses steps of bucket_width while the shorter side uses steps scaled down by the average target/source length ratio. If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of max_len.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>

Parameter Name	Description
bucketing_enabled	<p>Set to <code>false</code> to disable bucketing, unroll to maximum length.</p> <p>Optional</p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
checkpoint_frequency_num_batches	<p>Checkpoint and evaluate every x batches.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1000</p>
checkpoint_threshold	<p>Maximum number of checkpoints model is allowed to not improve in <code>optimized_metric</code> on validation dataset before training is stopped.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
clip_gradient	<p>Clip absolute gradient values greater than this. Set to negative to disable.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
cnn_activation_type	<p>The cnn activation type to be used.</p> <p>Optional</p> <p>Valid values: String. One of <code>glu</code>, <code>relu</code>, <code>softrelu</code>, <code>sigmoid</code>, or <code>tanh</code>.</p> <p>Default value: <code>glu</code></p>
cnn_hidden_dropout	<p>Dropout probability for dropout between convolutional layers.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>

Parameter Name	Description
cnn_kernel_width_decoder	<p>Kernel width for the <code>cnn</code> decoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
cnn_kernel_width_encoder	<p>Kernel width for the <code>cnn</code> encoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
cnn_num_hidden	<p>Number of <code>cnn</code> hidden units for encoder and decoder.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
decoder_type	<p>Decoder type.</p> <p>Optional</p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>
embed_dropout_source	<p>Dropout probability for source side embeddings.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
embed_dropout_target	<p>Dropout probability for target side embeddings.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
encoder_type	<p>Encoder type. The <code>rnn</code> architecture is based on attention mechanism by Bahdanau et al. and <code>cnn</code> architecture is based on Gehring et al.</p> <p>Optional</p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>

Parameter Name	Description
<code>fixed_rate_lr_half_life</code>	<p>Half life for learning rate in terms of number of checkpoints for <code>fixed_rate_*</code> schedulers.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
<code>learning_rate</code>	<p>Initial learning rate.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.0003</p>
<code>loss_type</code>	<p>Loss function for training.</p> <p>Optional</p> <p>Valid values: String. <code>cross-entropy</code></p> <p>Default value: <code>cross-entropy</code></p>
<code>lr_scheduler_type</code>	<p>Learning rate scheduler type. <code>plateau_reduce</code> means reduce the learning rate whenever <code>optimized_metric</code> on <code>validation_accuracy</code> plateaus. <code>inv_t</code> is inverse time decay. $\text{learning_rate}/(1+\text{decay_rate}^t)$</p> <p>Optional</p> <p>Valid values: String. One of <code>plateau_reduce</code>, <code>fixed_rate_inv_t</code>, or <code>fixed_rate_inv_sqrt_t</code>.</p> <p>Default value: <code>plateau_reduce</code></p>
<code>max_num_batches</code>	<p>Maximum number of updates/batches to process. -1 for infinite.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: -1</p>
<code>max_num_epochs</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Positive integer and less than or equal to <code>max_num_epochs</code>.</p> <p>Default value: none</p>

Parameter Name	Description
max_seq_len_source	<p>Maximum length for the source sequence length. Sequences longer than this length are truncated to this length.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 100</p>
max_seq_len_target	<p>Maximum length for the target sequence length. Sequences longer than this length are truncated to this length.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 100</p>
min_num_epochs	<p>Minimum number of epochs the training must run before it is stopped via <code>early_stopping</code> conditions.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 0</p>
momentum	<p>Momentum constant used for <code>sgd</code>. Don't pass this parameter if you are using <code>adam</code> or <code>rmsprop</code>.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: none</p>
num_embed_source	<p>Embedding size for source tokens.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
num_embed_target	<p>Embedding size for target tokens.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 512</p>

Parameter Name	Description
<code>num_layers_decoder</code>	<p>Number of layers for Decoder <code>rnn</code> or <code>cnn</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1</p>
<code>num_layers_encoder</code>	<p>Number of layers for Encoder <code>rnn</code> or <code>cnn</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 1</p>
<code>optimized_metric</code>	<p>Metrics to optimize with early stopping.</p> <p>Optional</p> <p>Valid values: String. One of <code>perplexity</code>, <code>accuracy</code>, or <code>bleu</code>.</p> <p>Default value: <code>perplexity</code></p>
<code>optimizer_type</code>	<p>Optimizer to choose from.</p> <p>Optional</p> <p>Valid values: String. One of <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</p> <p>Default value: <code>adam</code></p>
<code>plateau_reduce_lr_factor</code>	<p>Factor to multiply learning rate with (for <code>plateau_reduce</code>).</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.5</p>
<code>plateau_reduce_lr_threshold</code>	<p>For <code>plateau_reduce</code> scheduler, multiply learning rate with reduce factor if <code>optimized_metric</code> didn't improve for this many checkpoints.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 3</p>

Parameter Name	Description
<code>rnn_attention_in_upper_layers</code>	<p>Pass the attention to upper layers of <i>rnn</i>, like Google NMT paper. Only applicable if more than one layer is used.</p> <p>Optional</p> <p>Valid values: boolean (<code>true</code> or <code>false</code>)</p> <p>Default value: <code>true</code></p>
<code>rnn_attention_num_hidden</code>	<p>Number of hidden units for attention layers. defaults to <code>rnn_num_hidden</code>.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: <code>rnn_num_hidden</code></p>
<code>rnn_attention_type</code>	<p>Attention model for encoders. <code>mlp</code> refers to concat and <code>bilinear</code> refers to general from the Luong et al. paper.</p> <p>Optional</p> <p>Valid values: String. One of <code>dot</code>, <code>fixed</code>, <code>mlp</code>, or <code>bilinear</code>.</p> <p>Default value: <code>mlp</code></p>
<code>rnn_cell_type</code>	<p>Specific type of <i>rnn</i> architecture.</p> <p>Optional</p> <p>Valid values: String. Either <code>lstm</code> or <code>gru</code>.</p> <p>Default value: <code>lstm</code></p>
<code>rnn_decoder_state_init</code>	<p>How to initialize <i>rnn</i> decoder states from encoders.</p> <p>Optional</p> <p>Valid values: String. One of <code>last</code>, <code>avg</code>, or <code>zero</code>.</p> <p>Default value: <code>last</code></p>
<code>rnn_first_residual_layer</code>	<p>First <i>rnn</i> layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1.</p> <p>Optional</p> <p>Valid values: positive integer</p> <p>Default value: 2</p>

Parameter Name	Description
rnn_num_hidden	<p>The number of <i>rnn</i> hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional Long Term Short Term Memory (LSTM) by default.</p> <p>Optional</p> <p>Valid values: positive even integer</p> <p>Default value: 1024</p>
rnn_residual_connections	<p>Add residual connection to stacked <i>rnn</i>. Number of layers should be more than 1.</p> <p>Optional</p> <p>Valid values: boolean (<code>true</code> or <code>false</code>)</p> <p>Default value: <code>false</code></p>
rnn_decoder_hidden_dropout	<p>Dropout probability for hidden state that combines the context with the <i>rnn</i> hidden state in the decoder.</p> <p>Optional</p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
training_metric	<p>Metrics to track on training on validation data.</p> <p>Optional</p> <p>Valid values: String. Either <code>perplexity</code> or <code>accuracy</code>.</p> <p>Default value: <code>perplexity</code></p>
weight_decay	<p>Weight decay constant.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0</p>
weight_init_scale	<p>Weight initialization scale (for <code>uniform</code> and <code>xavier</code> initialization).</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 2.34</p>

Parameter Name	Description
weight_init_type	<p>Type of weight initialization.</p> <p>Optional</p> <p>Valid values: String. Either <code>uniform</code> or <code>xavier</code>.</p> <p>Default value: <code>xavier</code></p>
xavier_factor_type	<p>Xavier factor type.</p> <p>Optional</p> <p>Valid values: String. One of <code>in</code>, <code>out</code>, or <code>avg</code>.</p> <p>Default value: <code>in</code></p>

Tune a Sequence-to-Sequence Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the Sequence-to-Sequence Algorithm

The sequence to sequence algorithm reports three metrics that are computed during training. Choose one of them as an objective to optimize when tuning the hyperparameter values.

Metric Name	Description	Optimization Direction
validation:accuracy	Accuracy computed on the validation dataset.	Maximize
validation:bleu	Bleu score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the <code>bleu_sample_size</code> parameter to specify the subsample.	Maximize
validation:perplexity	Perplexity, is a loss function computed on the validation dataset. Perplexity measures the cross-entropy between an empirical sample and the distribution predicted by a model and so provides a measure of how well a model predicts the sample values. Models that are good at predicting a sample have a low perplexity.	Minimize

Tunable Sequence-to-Sequence Hyperparameters

You can tune the following hyperparameters for the Amazon SageMaker Sequence to Sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective

metrics are: `batch_size`, `optimizer_type`, `learning_rate`, `num_layers_encoder`, and `num_layers_decoder`.

Parameter Name	Parameter Type	Recommended Ranges
<code>num_layers_encoder</code>	IntegerParameterRange	[1-10]
<code>num_layers_decoder</code>	IntegerParameterRange	[1-10]
<code>batch_size</code>	CategoricalParameterRange	[16,32,64,128,256,512,1024,2048]
<code>optimizer_type</code>	CategoricalParameterRange	['adam', 'sgd', 'rmsprop']
<code>weight_init_type</code>	CategoricalParameterRange	['xavier', 'uniform']
<code>weight_init_scale</code>	ContinuousParameterRange	For the xavier type: MinValue: 2.0, MaxValue: 3.0 For the uniform type: MinValue: -1.0, MaxValue: 1.0
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 0.00005, MaxValue: 0.2
<code>weight_decay</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.1
<code>momentum</code>	ContinuousParameterRange	MinValue: 0.5, MaxValue: 0.9
<code>clip_gradient</code>	ContinuousParameterRange	MinValue: 1.0, MaxValue: 5.0
<code>rnn_num_hidden</code>	CategoricalParameterRange	Applicable only to recurrent neural networks (RNNs). [128,256,512,1024,2048]
<code>cnn_num_hidden</code>	CategoricalParameterRange	Applicable only to convolutional neural networks (CNNs). [128,256,512,1024,2048]
<code>num_embed_source</code>	IntegerParameterRange	[256-512]
<code>num_embed_target</code>	IntegerParameterRange	[256-512]
<code>embed_dropout_source</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
<code>embed_dropout_target</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
<code>rnn_decoder_hidden_dropout</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
<code>cnn_hidden_dropout</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5

Parameter Name	Parameter Type	Recommended Ranges
lr_scheduler_type	CategoricalParameterRange	['plateau_reduce', 'fixed_rate_inv_t', 'fixed_rate_inv_sqrt_t']
plateau_reduce_lr_factor	ContinuousParameterRange	MinValue: 0.1, MaxValue: 0.5
plateau_reduce_lr_thresold	IntegerParameterRange	[1-5]
fixed_rate_lr_half_life	IntegerParameterRange	[10-30]

XGBoost Algorithm

The [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 an ensemble of estimates from 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 because of the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

This current release of the XGBoost algorithm makes upgrades from the open source XGBoost code base easy to install and use in Amazon SageMaker. Customers can use this release of the XGBoost algorithm either as an Amazon SageMaker built-in algorithm, as with the previous 0.72-based version, or as a framework to run training scripts in their local environments as they would typically do, for example, with a TensorFlow deep learning framework. This implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics than the original 0.72-based version. It also provides an XGBoost estimator that executes a training script in a managed XGBoost environment. The current release of Amazon SageMaker XGBoost is based on version 0.90 and will be upgradeable to future releases. The previous implementation [XGBoost Release 0.72 \(p. 456\)](#) is still available to customers if they need to postpone migrating to the current version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

Topics

- [How to Use Amazon SageMaker XGBoost \(p. 445\)](#)
- [Input/Output Interface for the XGBoost Algorithm \(p. 446\)](#)
- [EC2 Instance Recommendation for the XGBoost Algorithm \(p. 447\)](#)
- [XGBoost Sample Notebooks \(p. 447\)](#)
- [How XGBoost Works \(p. 448\)](#)
- [XGBoost Hyperparameters \(p. 448\)](#)
- [Tune an XGBoost Model \(p. 455\)](#)
- [XGBoost Previous Versions \(p. 456\)](#)

How to Use Amazon SageMaker XGBoost

The XGBoost algorithm can be used as a built-in algorithm or as a framework such as TensorFlow. Using XGBoost as a framework provides more flexibility than using it as a built-in algorithm as it enables more advanced scenarios that allow pre-processing and post-processing scripts to be incorporated into your training script. Using XGBoost as a built-in Amazon SageMaker algorithm is how you had to use the original [XGBoost Release 0.72 \(p. 456\)](#) version and nothing changes here except the version of XGBoost that you use.

- **Use XGBoost as a framework**

Use XGBoost as a framework to run scripts that can incorporate additional data processing into your training jobs. This way of using XGBoost should be familiar to users who have worked with the open source [XGBoost](#) and other Amazon SageMaker frameworks such as Scikit-learn. You use the Amazon SageMaker Python SDK as you would for other frameworks such as TensorFlow. One change from other Amazon SageMaker frameworks is that the `framework_version` field of the estimator for XGBoost is mandatory and is not set by default. Note that the first part of the version refers to the upstream module version (aka, 0.90), while the second part refers to the Amazon SageMaker version for the container. An error is generated if the `framework_version` is not set.

```
import sagemaker.xgboost
estimator = XGBoost(entry_point = 'myscript.py',
                     source_dir, model_dir,
                     train_instance_type,
                     train_instance_count,
                     hyperparameters,
                     role,
                     base_job_name,
                     framework_version = '0.90-2',
                     py_version)
estimator.fit({'train':'s3://my-bucket/training',
               'validation':'s3://my-bucket/validation'})
```

The AWS SDK for Python (Boto 3) and the CLI also require this field.

- **Use XGBoost as a built-in algorithm**

Use XGBoost to train and deploy a model as you would other built-in Amazon SageMaker algorithms. Using the current version of XGBoost as a built-in algorithm will be familiar to users who have used the original [XGBoost Release 0.72 \(p. 456\)](#) version with the Amazon SageMaker Python SDK and want to continue using the same procedures.

```
import sagemaker
from sagemaker.amazon.amazon_estimator import get_image_uri
# get the URI for new container
container = get_image_uri(boto3.Session().region_name,
                          'xgboost',
                          repo_version='0.90-2');
estimator = sagemaker.estimator.Estimator(container, role, instance_count, instance_type,
                                           train_volume_size, output_path, sagemaker.Session());
estimator.fit({'train':'s3://my-bucket/training', 'validation':'s3://my-bucket/
validation'})
```

If customers do not specify `version` in the `get_image_uri` function, they get the [XGBoost Release 0.72 \(p. 456\)](#) version by default. If you want to migrate to the current version, you have to specify `repo_version='0.90-2'` in the `get_image_uri` function. If you use the current version, you must update your code to use the new hyperparameters that are required by the 0.90 version of upstream algorithm. The AWS SDK for Python (Boto 3) and the CLI usage is similar. You also have to choose the version you want to run when using the Console to select the XGBoost algorithm.

Input/Output Interface for the XGBoost Algorithm

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

Note

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

To use a model trained with SageMaker XGBoost in open source XGBoost

- Use the following Python code:

```
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

To differentiate the importance of labelled data points use Instance Weight Supports

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For *text/libsvm* input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...`. For *text/csv* input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

EC2 Instance Recommendation for the XGBoost Algorithm

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M5) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

XGBoost Sample Notebooks

For a sample notebook that shows how to use Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see [Regression with the Amazon SageMaker XGBoost algorithm](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon

SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

How XGBoost Works

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using [gradient boosting](#) for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs 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 following table contains the subset of hyperparameters that are required or most commonly used for the Amazon SageMaker XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source DMLC XGBoost package. Currently Amazon SageMaker supports version 0.90. For details about full set of hyperparameter that can be configured for this version of XGBoost, see [XGBoost Parameters](#).

Parameter Name	Description
num_round	<p>The number of rounds to run the training.</p> <p>Required</p> <p>Valid values: integer</p>
num_class	<p>The number of classes.</p> <p>Required if <code>objective</code> is set to <code>multi:softmax</code> or <code>multi:softprob</code>.</p> <p>Valid values: integer</p> <p>Default value:</p>
alpha	<p>L1 regularization term on weights. Increasing this value makes models more conservative.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0</p>

Parameter Name	Description
<code>base_score</code>	<p>The initial prediction score of all instances, global bias.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 0.5</p>
<code>booster</code>	<p>Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function.</p> <p>Optional</p> <p>Valid values: String. One of <code>gbtree</code>, <code>gblinear</code>, or <code>dart</code>.</p> <p>Default value: <code>gbtree</code></p>
<code>colsample_bylevel</code>	<p>Subsample ratio of columns for each split, in each level.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>colsample_bynode</code>	<p>Subsample ratio of columns from each node.</p> <p>Optional</p> <p>Valid values: Float. Range: (0,1].</p> <p>Default value: 1</p>
<code>colsample_bytree</code>	<p>Subsample ratio of columns when constructing each tree.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>csv_weights</code>	<p>When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>

Parameter Name	Description
<code>early_stopping_rounds</code>	<p>The model trains until the validation score stops improving. Validation error needs to decrease at least every <code>early_stopping_rounds</code> to continue training. Amazon SageMaker hosting uses the best model for inference.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: -</p>
<code>eta</code>	<p>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The <code>eta</code> parameter actually shrinks the feature weights to make the boosting process more conservative.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 0.3</p>
<code>eval_metric</code>	<p>Evaluation metrics for validation data. A default metric is assigned according to the objective:</p> <ul style="list-style-type: none"> • <code>rmse</code>: for regression • <code>error</code>: for classification • <code>map</code>: for ranking <p>For a list of valid inputs, see XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: Default according to objective.</p>
<code>gamma</code>	<p>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 0</p>
<code>grow_policy</code>	<p>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</p> <p>Default value: <code>depthwise</code></p>

Parameter Name	Description
<code>lambda</code>	<p>L2 regularization term on weights. Increasing this value makes models more conservative.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>lambda_bias</code>	<p>L2 regularization term on bias.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0</p>
<code>max_bin</code>	<p>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 256</p>
<code>max_delta_step</code>	<p>Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.</p> <p>Optional</p> <p>Valid values: Integer. Range: $[0, \infty)$.</p> <p>Default value: 0</p>
<code>max_depth</code>	<p>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</p> <p>Optional</p> <p>Valid values: Integer. Range: $[0, \infty)$</p> <p>Default value: 6</p>
<code>max_leaves</code>	<p>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>

Parameter Name	Description
<code>min_child_weight</code>	<p>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 <code>min_child_weight</code>, 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.</p> <p>Optional</p> <p>Valid values: Float. Range: $[0, \infty)$.</p> <p>Default value: 1</p>
<code>normalize_type</code>	<p>Type of normalization algorithm.</p> <p>Optional</p> <p>Valid values: Either <i>tree</i> or <i>forest</i>.</p> <p>Default value: <i>tree</i></p>
<code>nthread</code>	<p>Number of parallel threads used to run <i>xgboost</i>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: Maximum number of threads.</p>
<code>objective</code>	<p>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:logistic</code>, <code>multi:softmax</code>, <code>reg:squarederror</code>. For a full list of valid inputs, refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: <code>reg:squarederror</code></p>
<code>one_drop</code>	<p>When this flag is enabled, at least one tree is always dropped during the dropout.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>process_type</code>	<p>The type of boosting process to run.</p> <p>Optional</p> <p>Valid values: String. Either <code>default</code> or <code>update</code>.</p> <p>Default value: <code>default</code></p>

Parameter Name	Description
<code>rate_drop</code>	<p>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
<code>refresh_leaf</code>	<p>This is a parameter of the 'refresh' updater plug-in. When set to <code>true</code> (1), tree leaves and tree node stats are updated. When set to <code>false</code>(0), only tree node stats are updated.</p> <p>Optional</p> <p>Valid values: 0/1</p> <p>Default value: 1</p>
<code>sample_type</code>	<p>Type of sampling algorithm.</p> <p>Optional</p> <p>Valid values: Either <code>uniform</code> or <code>weighted</code>.</p> <p>Default value: <code>uniform</code></p>
<code>scale_pos_weight</code>	<p>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: <code>sum(negative cases) / sum(positive cases)</code>.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>seed</code>	<p>Random number seed.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>silent</code>	<p>0 means print running messages, 1 means silent mode.</p> <p>Valid values: 0 or 1</p> <p>Optional</p> <p>Default value: 0</p>

Parameter Name	Description
<code>sketch_eps</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float, Range: [0, 1].</p> <p>Default value: 0.03</p>
<code>skip_drop</code>	<p>Probability of skipping the dropout procedure during a boosting iteration.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
<code>subsample</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>tree_method</code>	<p>The tree construction algorithm used in XGBoost.</p> <p>Optional</p> <p>Valid values: One of <code>auto</code>, <code>exact</code>, <code>approx</code>, or <code>hist</code>.</p> <p>Default value: <code>auto</code></p>
<code>tweedie_variance_power</code>	<p>Parameter that controls the variance of the Tweedie distribution.</p> <p>Optional</p> <p>Valid values: Float. Range: (1, 2).</p> <p>Default value: 1.5</p>
<code>updater</code>	<p>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.</p> <p>For a full list of valid inputs, please refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: comma-separated string.</p> <p>Default value: <code>grow_colmaker, prune</code></p>

Tune an XGBoost Model

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

Note

Automatic model tuning for XGBoost 0.90 is only available from the SDKs, not from the Amazon SageMaker console.

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the XGBoost Algorithm

The XGBoost algorithm computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

Metric Name	Description	Optimization Direction
validation:accuracy	Classification rate, calculated as #(right)/#(all cases).	Maximize
validation:auc	Area under the curve.	Maximize
validation:error	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:f1	Indicator of classification accuracy, calculated as the harmonic mean of precision and recall.	Maximize
validation:logloss	Negative log-likelihood.	Minimize
validation:mae	Mean absolute error.	Minimize
validation:map	Mean average precision.	Maximize
validation:merror	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:mlogloss	Negative log-likelihood for multiclass classification.	Minimize
validation:mse	Mean squared error.	Minimize
validation:ndcg	Normalized Discounted Cumulative Gain.	Maximize
validation:rmse	Root mean square error.	Minimize

Tunable XGBoost Hyperparameters

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

Parameter Name	Parameter Type	Recommended Ranges
alpha	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
colsample_bylevel	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
colsample_bynode	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
colsample_bytree	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
eta	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
gamma	ContinuousParameterRanges	MinValue: 0, MaxValue: 5
lambda	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
max_delta_step	IntegerParameterRanges	[0, 10]
max_depth	IntegerParameterRanges	[0, 10]
min_child_weight	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
num_round	IntegerParameterRanges	[1, 4000]
subsample	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

XGBoost Previous Versions

This page contains links to the documentation for previous versions of Amazon SageMaker XGBoost.

Topics

- [XGBoost Release 0.72 \(p. 456\)](#)

XGBoost Release 0.72

This previous release of the Amazon SageMaker XGBoost algorithm is based on the 0.72 release. [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 because of the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

Customers should consider using the new release of [XGBoost Algorithm \(p. 445\)](#). They can use it as an Amazon SageMaker built-in algorithm or as a framework to run scripts in their local environments as they would typically, for example, do with a Tensorflow deep learning framework. The new implementation has a smaller memory footprint, better logging, improved hyperparameter validation,

and an expanded set of metrics. The earlier implementation of XGBoost remains available to customers if they need to postpone migrating to the new version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

Topics

- [Input/Output Interface for the XGBoost Release 0.72 \(p. 457\)](#)
- [EC2 Instance Recommendation for the XGBoost Release 0.72 \(p. 458\)](#)
- [XGBoost Release 0.72 Sample Notebooks \(p. 458\)](#)
- [XGBoost Release 0.72 Hyperparameters \(p. 458\)](#)
- [Tune an XGBoost Release 0.72 Model \(p. 464\)](#)

[Input/Output Interface for the XGBoost Release 0.72](#)

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The Amazon SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

Note

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other Amazon SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count * the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

To use a model trained with SageMaker XGBoost in open source XGBoost

- Use the following Python code:

```
import pickle as pkl
model = pkl.load(open(model_file_path, 'rb'))
# prediction with test data
pred = model.predict(dtest)
```

To differentiate the importance of labelled data points use Instance Weight Supports

- Amazon SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For *text/libsvm* input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight`

`idx_0:val_0 idx_1:val_1...` For `text/csv` input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

EC2 Instance Recommendation for the XGBoost Release 0.72

Amazon SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

XGBoost Release 0.72 Sample Notebooks

For a sample notebook that shows how to use the latest version of Amazon SageMaker XGBoost as a built-in algorithm to train and host a regression model, see [Regression with Amazon SageMaker XGBoost algorithm](#). To use the 0.72 version of XGBoost, you need to change the version in the sample code to 0.72. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

XGBoost Release 0.72 Hyperparameters

The following table contains the hyperparameters for the XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The Amazon SageMaker XGBoost algorithm is an implementation of the open-source XGBoost package. Currently Amazon SageMaker supports version 0.72. For more detail about hyperparameter configuration for this version of XGBoost, see [XGBoost Parameters](#).

Parameter Name	Description
<code>num_class</code>	The number of classes. Required if <code>objective</code> is set to <code>multi:softmax</code> or <code>multi:softprob</code> . Valid values: integer
<code>num_round</code>	The number of rounds to run the training. Required Valid values: integer
<code>alpha</code>	L1 regularization term on weights. Increasing this value makes models more conservative. Optional Valid values: float Default value: 0
<code>base_score</code>	The initial prediction score of all instances, global bias.

Parameter Name	Description
	Optional Valid values: float Default value: 0.5
<code>booster</code>	Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function. Optional Valid values: String. One of <code>gbtree</code> , <code>gblinear</code> , or <code>dart</code> . Default value: <code>gbtree</code>
<code>colsample_bylevel</code>	Subsample ratio of columns for each split, in each level. Optional Valid values: Float. Range: [0,1]. Default value: 1
<code>colsample_bytree</code>	Subsample ratio of columns when constructing each tree. Optional Valid values: Float. Range: [0,1]. Default value: 1
<code>csv_weights</code>	When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights. Optional Valid values: 0 or 1 Default value: 0
<code>early_stopping_rounds</code>	The model trains until the validation score stops improving. Validation error needs to decrease at least every <code>early_stopping_rounds</code> to continue training. Amazon SageMaker hosting uses the best model for inference. Optional Valid values: integer Default value: -

Parameter Name	Description
eta	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 0.3</p>
eval_metric	<p>Evaluation metrics for validation data. A default metric is assigned according to the objective:</p> <ul style="list-style-type: none"> • rmse: for regression • error: for classification • map: for ranking <p>For a list of valid inputs, see XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: Default according to objective.</p>
gamma	<p>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 0</p>
grow_policy	<p>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</p> <p>Default value: <code>depthwise</code></p>
lambda	<p>L2 regularization term on weights. Increasing this value makes models more conservative.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>

Parameter Name	Description
<code>lambda_bias</code>	<p>L2 regularization term on bias.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0</p>
<code>max_bin</code>	<p>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 256</p>
<code>max_delta_step</code>	<p>Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.</p> <p>Optional</p> <p>Valid values: Integer. Range: [0,∞).</p> <p>Default value: 0</p>
<code>max_depth</code>	<p>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</p> <p>Optional</p> <p>Valid values: Integer. Range: [0,∞)</p> <p>Default value: 6</p>
<code>max_leaves</code>	<p>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>min_child_weight</code>	<p>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 <code>min_child_weight</code>, 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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 1</p>

Parameter Name	Description
<code>normalize_type</code>	<p>Type of normalization algorithm.</p> <p>Optional</p> <p>Valid values: Either <i>tree</i> or <i>forest</i>.</p> <p>Default value: <i>tree</i></p>
<code>nthread</code>	<p>Number of parallel threads used to run <i>xgboost</i>.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: Maximum number of threads.</p>
<code>objective</code>	<p>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:logistic</code>, <code>reg:softmax</code>, <code>multi:squarederror</code>. For a full list of valid inputs, refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: string</p> <p>Default value: <code>reg:squarederror</code></p>
<code>one_drop</code>	<p>When this flag is enabled, at least one tree is always dropped during the dropout.</p> <p>Optional</p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>process_type</code>	<p>The type of boosting process to run.</p> <p>Optional</p> <p>Valid values: String. Either <code>default</code> or <code>update</code>.</p> <p>Default value: <code>default</code></p>
<code>rate_drop</code>	<p>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>

Parameter Name	Description
<code>refresh_leaf</code>	<p>This is a parameter of the 'refresh' updater plug-in. When set to <code>true</code>(1), tree leaves and tree node stats are updated. When set to <code>false</code>(0), only tree node stats are updated.</p> <p>Optional</p> <p>Valid values: 0/1</p> <p>Default value: 1</p>
<code>sample_type</code>	<p>Type of sampling algorithm.</p> <p>Optional</p> <p>Valid values: Either <code>uniform</code> or <code>weighted</code>.</p> <p>Default value: <code>uniform</code></p>
<code>scale_pos_weight</code>	<p>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: <code>sum(negative cases) / sum(positive cases)</code>.</p> <p>Optional</p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>seed</code>	<p>Random number seed.</p> <p>Optional</p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>silent</code>	<p>0 means print running messages, 1 means silent mode.</p> <p>Valid values: 0 or 1</p> <p>Optional</p> <p>Default value: 0</p>
<code>sketch_eps</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float, Range: [0, 1].</p> <p>Default value: 0.03</p>

Parameter Name	Description
<code>skip_drop</code>	<p>Probability of skipping the dropout procedure during a boosting iteration.</p> <p>Optional</p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
<code>subsample</code>	<p>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.</p> <p>Optional</p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>tree_method</code>	<p>The tree construction algorithm used in XGBoost.</p> <p>Optional</p> <p>Valid values: One of <code>auto</code>, <code>exact</code>, <code>approx</code>, or <code>hist</code>.</p> <p>Default value: <code>auto</code></p>
<code>tweedie_variance_power</code>	<p>Parameter that controls the variance of the Tweedie distribution.</p> <p>Optional</p> <p>Valid values: Float. Range: (1, 2).</p> <p>Default value: 1.5</p>
<code>updater</code>	<p>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.</p> <p>For a full list of valid inputs, please refer to XGBoost Parameters.</p> <p>Optional</p> <p>Valid values: comma-separated string.</p> <p>Default value: <code>grow_colmaker</code>, <code>prune</code></p>

Tune an XGBoost Release 0.72 Model

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

For more information about model tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).

Metrics Computed by the XGBoost Release 0.72 Algorithm

The XGBoost algorithm based on version 0.72 computes the following nine metrics during training. When tuning the model, choose one of these metrics as the objective to evaluate the model.

Metric Name	Description	Optimization Direction
validation:auc	Area under the curve.	Maximize
validation:error	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:logloss	Negative log-likelihood.	Minimize
validation:mae	Mean absolute error.	Minimize
validation:map	Mean average precision.	Maximize
validation:merror	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:mlogloss	Negative log-likelihood for multiclass classification.	Minimize
validation:ndcg	Normalized Discounted Cumulative Gain.	Maximize
validation:rmse	Root mean square error.	Minimize

Tunable XGBoost Release 0.72 Hyperparameters

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

Parameter Name	Parameter Type	Recommended Ranges
alpha	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
colsample_bylevel	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
colsample_bytree	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
eta	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
gamma	ContinuousParameterRanges	MinValue: 0, MaxValue: 5
lambda	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
max_delta_step	IntegerParameterRanges	[0, 10]
max_depth	IntegerParameterRanges	[0, 10]

Parameter Name	Parameter Type	Recommended Ranges
min_child_weight	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
num_round	IntegerParameterRanges	[1, 4000]
subsample	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

Use Machine Learning Frameworks with Amazon SageMaker

The Amazon SageMaker Python SDK provides open source APIs and containers that make it easy to train and deploy models in Amazon SageMaker with several different machine learning and deep learning frameworks. For general information about the Amazon SageMaker Python SDK, see <https://github.com/aws/sagemaker-python-sdk>. For information about using specific frameworks in Amazon SageMaker, see the following topics:

Topics

- [Use Apache Spark with Amazon SageMaker \(p. 466\)](#)
- [Use TensorFlow with Amazon SageMaker \(p. 475\)](#)
- [Use Apache MXNet with Amazon SageMaker \(p. 476\)](#)
- [Use Scikit-learn with Amazon SageMaker \(p. 476\)](#)
- [Use PyTorch with Amazon SageMaker \(p. 477\)](#)
- [Use Chainer with Amazon SageMaker \(p. 478\)](#)
- [Use SparkML Serving with Amazon SageMaker \(p. 478\)](#)

Use Apache Spark with Amazon SageMaker

This section provides information for developers who want to use Apache Spark for preprocessing data and Amazon SageMaker for model training and hosting. For information about supported versions of Apache Spark, see <https://github.com/aws/sagemaker-spark#getting-sagemaker-spark>.

Amazon SageMaker provides an Apache Spark library, in both Python and Scala, that you can use to easily train models in Amazon SageMaker using `org.apache.spark.sql.DataFrame` data frames in your Spark clusters. After model training, you can also host the model using Amazon SageMaker hosting services.

The Amazon SageMaker Spark library, `com.amazonaws.services.sagemaker.sparksdk`, provides the following classes, among others:

- `SageMakerEstimator`—Extends the `org.apache.spark.ml.Estimator` interface. You can use this estimator for model training in Amazon SageMaker.
- `KMeansSageMakerEstimator`, `PCASageMakerEstimator`, and `XGBoostSageMakerEstimator`—Extend the `SageMakerEstimator` class.
- `SageMakerModel`—Extends the `org.apache.spark.ml.Model` class. You can use this `SageMakerModel` for model hosting and obtaining inferences in Amazon SageMaker.

Download the Amazon SageMaker Spark Library

You have the following options for downloading the Spark library provided by Amazon SageMaker:

- You can download the source code for both PySpark and Scala libraries from GitHub at <https://github.com/aws/sagemaker-spark>.
- For the Python Spark library, you have the following additional options:
 - Use pip install:

```
$ pip install sagemaker_pyspark
```
 - In a notebook instance, create a new notebook that uses either the `Sparkmagic` (PySpark) or the `Sparkmagic` (PySpark3) kernel and connect to a remote Amazon EMR cluster.
- 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:

```
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>sagemaker-spark_2.11</artifactId>
    <version>spark_2.2.0-1.0</version>
</dependency>
```

Integrate Your Apache Spark Application with Amazon SageMaker

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

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

Amazon SageMaker assumes the IAM role that you specified for model training to perform tasks on your behalf. For example, it uses the role to read training data from an S3 bucket and to write model artifacts to a 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: Use Amazon SageMaker for Training and Inference with Apache Spark

Topics

- [Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark \(p. 472\)](#)
- [Use the SageMakerEstimator in a Spark Pipeline \(p. 473\)](#)

Amazon SageMaker provides an Apache Spark library (in both Python and Scala) that you can use to integrate your Apache Spark applications with Amazon SageMaker. For example, you might use Apache Spark for data preprocessing and Amazon SageMaker for model training and hosting. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 466\)](#). 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 \(p. 344\)](#).

This is the example code:

```

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/")

val roleArn = "arn:aws:iam::account-id:role/rolename"

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

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

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

```

The code does the following:

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

```

// Get a Spark session.

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

```

```
val roleArn = "arn:aws:iam::account-id:role/rolename"
trainingData.show()
```

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

```
+-----+-----+
|label|      features|
+-----+-----+
| 5.0|(784,[152,153,154...|
| 0.0|(784,[127,128,129...|
| 4.0|(784,[160,161,162...|
| 1.0|(784,[158,159,160...|
| 9.0|(784,[208,209,210...|
| 2.0|(784,[155,156,157...|
| 1.0|(784,[124,125,126...|
| 3.0|(784,[151,152,153...|
| 1.0|(784,[152,153,154...|
| 4.0|(784,[134,135,161...|
| 3.0|(784,[123,124,125...|
| 5.0|(784,[216,217,218...|
| 3.0|(784,[143,144,145...|
| 6.0|(784,[72,73,74,99...|
| 1.0|(784,[151,152,153...|
| 7.0|(784,[211,212,213...|
| 2.0|(784,[151,152,153...|
| 8.0|(784,[159,160,161...|
| 6.0|(784,[100,101,102...|
| 9.0|(784,[209,210,211...|
+-----+
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`.

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

The constructor parameters provide information that is used for training a model and deploying it on Amazon SageMaker:

- `trainingInstanceType` and `trainingInstanceCount`—Identify the type and number of ML compute instances to use for model training.
- `endpointInstanceType`—Identifies the ML compute instance type to use when hosting the model in Amazon SageMaker. By default, one ML compute instance is assumed.
- `endpointInitialInstanceCount`—Identifies the number of ML compute instances initially backing the endpoint hosting the model in Amazon SageMaker.
- `sagemakerRole`—Amazon SageMaker assumes this IAM role to perform tasks on your behalf. For example, for model training, it reads data from S3 and writes training results (model artifacts) to S3.

Note

This example implicitly creates an Amazon SageMaker client. To create this client, you must provide your credentials. The API uses these credentials to authenticate requests to Amazon SageMaker. For example, it uses the credentials to authenticate requests to create a training job and API calls for deploying the model using Amazon SageMaker hosting services.

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

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

You pass the input `DataFrame` as a parameter. The model does all the work of training the model and deploying it to Amazon SageMaker. For more information see, [Integrate Your Apache Spark Application with Amazon SageMaker \(p. 467\)](#). 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.

```
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	
3.0 (784,[143,144,145...	1736.765869140625	4.0	
6.0 (784,[72,73,74,99...	1473.69384765625	8.0	
1.0 (784,[151,152,153...	944.88720703125	2.0	
7.0 (784,[211,212,213...	1285.9071044921875	3.0	
2.0 (784,[151,152,153...	1635.0125732421875	1.0	
8.0 (784,[159,160,161...	1436.3162841796875	6.0	
6.0 (784,[100,101,102...	1499.7366943359375	7.0	
9.0 (784,[209,210,211...	1364.6319580078125	6.0	

You can interpret the data, as follows:

- A handwritten number with the `label` 5 belongs to cluster 4 (`closest_cluster`).
- A handwritten number with the `label` 0 belongs to cluster 5.
- A handwritten number with the `label` 4 belongs to cluster 9.
- A handwritten number with the `label` 1 belongs to cluster 6.

For more information on how to run these examples, see <https://github.com/aws/sagemaker-spark/blob/master/README.md> on GitHub.

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

In [Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark \(p. 468\)](#), you use the `kMeansSageMakerEstimator` because the example uses the k-means algorithm provided by Amazon SageMaker for model training. You might choose to use your own custom algorithm for model training instead. Assuming that you have already created a Docker image, you can create your own `SageMakerEstimator` and specify the Amazon Elastic Container Registry path for your custom image.

The following example shows how to create a `KMeansSageMakerEstimator` from the `SageMakerEstimator`. In the new estimator, you explicitly specify the Docker registry path to your training and inference code images.

```
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.KMeansProtobufResponseRowDeseri

val estimator = new SageMakerEstimator(
  trainingImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  modelImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  requestRowSerializer = new ProtobufRequestRowSerializer(),
  responseRowDeserializer = new KMeansProtobufResponseRowDeserializer(),
  hyperParameters = Map("k" -> "10", "feature_dim" -> "784"),
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1,
  trainingSparkDataFormat = "sagemaker")

```

In the code, the parameters in the `SageMakerEstimator` constructor include:

- `trainingImage` —Identifies the Docker registry path to the training image containing your custom code.
- `modelImage` —Identifies the Docker registry path to the image containing inference code.
- `requestRowSerializer` —Implements `com.amazonaws.services.sagemaker.sparksdk.transformation.RequestRowSerializer`.

This parameter serializes rows in the input `DataFrame` to send them to the model hosted in Amazon SageMaker for inference.

- `responseRowDeserializer` —Implements

```
com.amazonaws.services.sagemaker.sparksdk.transformation.ResponseRowDeserializer.
```

This parameter deserializes responses from the model, hosted in Amazon SageMaker, back into a `DataFrame`.

- `trainingSparkDataFormat` —Specifies the data format that Spark uses when uploading training data from a `DataFrame` to S3. For example, "sagemaker" for protobuf format, "csv" for comma-separated values, and "libsvm" for LibSVM format.

You can implement your own `RequestRowSerializer` and `ResponseRowDeserializer` to serialize and deserialize rows from a data format that your inference code supports, such as .libsvm or ..csv.

Use the `SageMakerEstimator` in a Spark Pipeline

You can use `org.apache.spark.ml.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:

```

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)
.setK(10).setFeatureDim(50)

val pipeline = new Pipeline().setStages(Array(pcaEstimator, kMeansSageMakerEstimator))

// train
val pipelineModel = pipeline.fit(trainingData)

val transformedData = pipelineModel.transform(testData)
transformedData.show()

```

The parameter `trainingSparkDataFormatOptions` configures Spark to serialize to protobuf the "projectedFeatures" column for model training. Additionally, Spark serializes to protobuf the "label" column by default.

Because we want to make inferences using the "projectedFeatures" column, we pass the column name into the `ProtobufRequestRowSerializer`.

The following example shows a transformed DataFrame:

label	features	projectedFeatures	distance_to_cluster	closest_cluster
5.0	[784,[152,153,154...]	[880.731433034386...]	1500.470703125	0.0
0.0	[784,[127,128,129...]	[1768.51722024166...]	1142.18359375	4.0
4.0	[784,[160,161,162...]	[704.949236329314...]	1386.246826171875	9.0
1.0	[784,[158,159,160...]	[-42.328192193771...]	1277.0736083984375	5.0
9.0	[784,[208,209,210...]	[374.043902028333...]	1211.00927734375	3.0
2.0	[784,[155,156,157...]	[941.267714528850...]	1496.157958984375	8.0
1.0	[784,[124,125,126...]	[30.2848596410594...]	1327.6766357421875	5.0
3.0	[784,[151,152,153...]	[1270.14374062052...]	1570.7674560546875	0.0
1.0	[784,[152,153,154...]	[-112.10792566485...]	1037.568359375	5.0
4.0	[784,[134,135,161...]	[452.068280676606...]	1165.1236572265625	3.0
3.0	[784,[123,124,125...]	[610.596447285397...]	1325.953369140625	7.0
5.0	[784,[216,217,218...]	[142.959601818422...]	1353.4930419921875	5.0
3.0	[784,[143,144,145...]	[1036.71862533658...]	1460.4315185546875	7.0
6.0	[784,[72,73,74,99...]	[996.740157435754...]	1159.8631591796875	2.0
1.0	[784,[151,152,153...]	[-107.26076167417...]	960.963623046875	5.0
7.0	[784,[211,212,213...]	[619.771820430940...]	1245.13623046875	6.0
2.0	[784,[151,152,153...]	[850.152101817161...]	1304.437744140625	8.0
8.0	[784,[159,160,161...]	[370.041887230547...]	1192.4781494140625	0.0
6.0	[784,[100,101,102...]	[546.674328209335...]	1277.0908203125	2.0
9.0	[784,[209,210,211...]	[-29.259112927426...]	1245.8182373046875	6.0

Additional Examples: Use 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>.

Use TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom TensorFlow code. The Amazon SageMaker Python SDK TensorFlow estimators and models and the Amazon SageMaker open-source TensorFlow containers make writing a TensorFlow script and running it in Amazon SageMaker easier.

Use TensorFlow Version 1.11 and Later

For TensorFlow versions 1.11 and later, the Amazon SageMaker Python SDK supports script mode training scripts.

What do you want to do?

I want to train a custom TensorFlow model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/tensorflow_distributed_mnist.

For documentation, see [Train a Model with TensorFlow](#).

I have a TensorFlow model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

[Deploy TensorFlow Serving models](#).

I have a TensorFlow model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

[Deploying directly from model artifacts](#).

I want to see the API documentation for Amazon SageMaker Python SDK TensorFlow classes.

[TensorFlow Estimator](#)

I want to see information about Amazon SageMaker TensorFlow containers.

<https://github.com/aws/sagemaker-tensorflow-container>.

For general information about writing TensorFlow script mode training scripts and using TensorFlow script mode estimators and models with Amazon SageMaker, see [Using TensorFlow with the SageMaker Python SDK](#).

For information about TensorFlow versions supported by the Amazon SageMaker TensorFlow container, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst>.

Use TensorFlow Legacy Mode for Versions 1.11 and Earlier

The Amazon SageMaker Python SDK provides a legacy mode that supports TensorFlow versions 1.11 and earlier. Use legacy mode TensorFlow training scripts to run TensorFlow jobs in Amazon SageMaker if:

- You have existing legacy mode scripts that you do not want to convert to script mode.
- You want to use a TensorFlow version earlier than 1.11.

For information about writing legacy mode TensorFlow scripts to use with the Amazon SageMaker Python SDK, see <https://github.com/aws/sagemaker-python-sdk/tree/v1.12.0/src/sagemaker/tensorflow#tensorflow-sagemaker-estimators-and-models>.

Use Apache MXNet with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom MXNet code. The Amazon SageMaker Python SDK MXNet estimators and models and the Amazon SageMaker open-source MXNet container make writing a MXNet script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom MXNet model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/mxnet_mnist.

For documentation, see [Train a Model with MXNet](#).

I have an MXNet model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

[Deploy MXNet models](#).

I have an MXNet model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint.

[Deploy Endpoints from Model Data](#).

I want to see the API documentation for Amazon SageMaker Python SDK MXNet classes.

[MXNet Classes](#)

I want to see information about Amazon SageMaker MXNet containers.

<https://github.com/aws/sagemaker-mxnet-container>.

For general information about writing MXNet script mode training scripts and using MXNet script mode estimators and models with Amazon SageMaker, see [Using MXNet with the SageMaker Python SDK](#).

For information about MXNet versions supported by the Amazon SageMaker MXNet container, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst>.

Use Scikit-learn with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Scikit-learn code. The Amazon SageMaker Python SDK Scikit-learn estimators and models and the Amazon SageMaker open-source Scikit-learn container make writing a Scikit-learn script and running it in Amazon SageMaker easier.

What do you want to do?

I want to use Scikit-learn for data processing, feature engineering, or model evaluation in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_processing/scikit_learn_data_processing_and_model_evaluation.

For documentation, see [Amazon SageMaker Python SDK ReadTheDocs](#)

I want to train a custom Scikit-learn model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/scikit_learn_iris.

For documentation, see [Train a Model with Scikit-learn](#).

I have a Scikit-learn model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

[Deploy Scikit-learn models](#).

I have a Scikit-learn model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

[Deploy Endpoints from Model Data](#).

I want to see the API documentation for Amazon SageMaker Python SDK Scikit-learn classes.

[Scikit-learn Classes](#)

I want to see information about Amazon SageMaker Scikit-learn containers.

<https://github.com/aws/sagemaker-scikit-learn-container>.

For general information about writing Scikit-learn training scripts and using Scikit-learn estimators and models with Amazon SageMaker, see [Using Scikit-learn with the SageMaker Python SDK](#).

For information about Scikit-learn versions supported by the Amazon SageMaker Scikit-learn container, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/sklearn/README.rst>.

Use PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. The Amazon SageMaker Python SDK PyTorch estimators and models and the Amazon SageMaker open-source PyTorch container make writing a PyTorch script and running it in Amazon SageMaker easier.

[What do you want to do?](#)

I want to train a custom PyTorch model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/pytorch_mnist.

For documentation, see [Train a Model with PyTorch](#).

I have a PyTorch model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

[Deploy PyTorch models](#).

I have a PyTorch model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

[Deploy Endpoints from Model Data](#).

I want to see the API documentation for Amazon SageMaker Python SDK PyTorch classes.

[PyTorch Classes](#)

I want to see information about Amazon SageMaker PyTorch containers.

<https://github.com/aws/sagemaker-pytorch-container>.

For general information about writing PyTorch training scripts and using PyTorch estimators and models with Amazon SageMaker, see [Using PyTorch with the SageMaker Python SDK](#).

For information about PyTorch versions supported by the Amazon SageMaker PyTorch container, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/pytorch/README.rst>.

Use Chainer with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Chainer code. The Amazon SageMaker Python SDK Chainer estimators and models and the Amazon SageMaker open-source Chainer container make writing a Chainer script and running it in Amazon SageMaker easier.

What do you want to do?

I want to train a custom Chainer model in Amazon SageMaker.

For a sample Jupyter notebook, see https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/chainer_minist.

For documentation, see [Train a Model with Chainer](#).

I have a Chainer model that I trained in Amazon SageMaker, and I want to deploy it to a hosted endpoint.

[Deploy Chainer models](#).

I have a Chainer model that I trained outside of Amazon SageMaker, and I want to deploy it to an Amazon SageMaker endpoint

[Deploy Endpoints from Model Data](#).

I want to see the API documentation for Amazon SageMaker Python SDK Chainer classes.

[Chainer Classes](#)

I want to see information about Amazon SageMaker Chainer containers.

<https://github.com/aws/sagemaker-chainer-container>.

For general information about writing Chainer training scripts and using Chainer estimators and models with Amazon SageMaker, see [Using Chainer with the SageMaker Python SDK](#).

For information about Chainer versions supported by the Amazon SageMaker Chainer container, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/chainer/README.rst>.

Use SparkML Serving with Amazon SageMaker

The Amazon SageMaker Python SDK SparkML Serving model and predictor and the Amazon SageMaker open-source SparkML Serving container support deploying Apache Spark ML pipelines serialized with MLeap in Amazon SageMaker to get inferences.

For information about using the SparkML Serving container to deploy models to Amazon SageMaker, see <https://github.com/aws/sagemaker-sparkml-serving-container>. For information about the Amazon SageMaker Python SDK SparkML Serving model and predictors, see <https://sagemaker.readthedocs.io/en/stable/sagemaker.sparkml.html>.

Use Your Own Algorithms or Models with Amazon SageMaker

Amazon SageMaker makes extensive use of *Docker containers* for build and runtime tasks. Before using your own algorithm or model with Amazon SageMaker, you need to understand how Amazon SageMaker manages and runs them. Amazon SageMaker provides pre-built Docker images for its built-in algorithms and the supported deep learning frameworks used for training and inference. By using containers, you can train machine learning algorithms and deploy models quickly and reliably at any scale. Docker is a program that performs operating-system-level virtualization for installing, distributing, and managing software. It packages applications and their dependencies into virtual containers that provide isolation, portability, and security.

You can put scripts, algorithms, and inference code for your machine learning models into containers. The container includes the runtime, system tools, system libraries, and other code required to train your algorithms or deploy your models. This gives you the flexibility to use almost any script or algorithm code with Amazon SageMaker, regardless of runtime or implementation language. The code that runs in containers is effectively isolated from its surroundings, ensuring a consistent runtime, regardless of where the container is deployed. After packaging your training code, inference code, or both into Docker containers, you can create algorithm resources and model package resources for use in Amazon SageMaker or to publish on AWS Marketplace. With Docker, you can ship code faster, standardize application operations, seamlessly move code, and economize by improving resource utilization.

You create Docker containers from *images* that are saved in a *repository*. You build the images from scripted instructions provided in a *Dockerfile*. To use Docker containers in Amazon SageMaker, the scripts that you use must satisfy certain requirements. For information about the requirements, see [Use Your Own Training Algorithms \(p. 496\)](#) and [Use Your Own Inference Code \(p. 501\)](#).

Scenarios for Running Scripts, Training Algorithms, or Deploying Models with Amazon SageMaker

Amazon SageMaker always uses Docker containers when running scripts, training algorithms or deploying models. However, your level of engagement with containers varies depending on whether you are using a built-in algorithm provided by Amazon SageMaker or a script or model that you have developed yourself. If you're using your own code, it also depends on the language and framework or environment used to develop it, and any other of the dependencies it requires to run. In particular, it depends on whether you use the Amazon SageMaker Python SDK or AWS SDK for Python (Boto3) or some other SDK. Amazon SageMaker provides containers for its built-in algorithms and pre-built Docker images for some of the most common machine learning frameworks. You can use the containers and images as provided or extend them to cover more complicated use cases. You can also create your own container images to manage more advanced use cases not addressed by the containers provided by Amazon SageMaker.

There are four main scenarios for running scripts, algorithms, and models in the Amazon SageMaker environment. The last three describe the scenarios covered here: the ways you can use containers to *bring your own script, algorithm or model*.

- **Use a built-in algorithm.** Containers are used behind the scenes when you use one of the Amazon SageMaker built-in algorithms, but you do not deal with them directly. You can train and deploy these algorithms from the Amazon SageMaker console, the AWS Command Line Interface (AWS CLI), a Python notebook, or the Amazon SageMaker Python SDK. The built-in algorithms available are itemized and described in the [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#) topic. For an example of how to train and deploy a built-in algorithm using Jupyter Notebook running in an Amazon SageMaker notebook instance, see the [Get Started with Amazon SageMaker \(p. 22\)](#) topic.
- **Use pre-built container images.** Amazon SageMaker provides pre-built containers to support deep learning frameworks such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports machine learning libraries such as scikit-learn and SparkML by providing pre-built Docker images. If you use the Amazon SageMaker Python SDK, they are deployed using their respective Amazon SageMaker SDK `Estimator` class. In this case, you can supply the Python code that implements your algorithm and configure the pre-built image to access your code as an entry point. For a list of deep learning frameworks currently supported by Amazon SageMaker and samples that show how to use their pre-build container images, see [Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch \(p. 493\)](#). For information on the scikit-learn and SparkML pre-built container images, see [Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML \(p. 494\)](#). For more information about using frameworks with the Amazon SageMaker Python SDK, see their respective topics in [Use Machine Learning Frameworks with Amazon SageMaker \(p. 466\)](#).
- **Extend a pre-built container image.** If you have additional functional requirements for an algorithm or model that you developed in a framework that a pre-built Amazon SageMaker Docker image doesn't support, you can modify an Amazon SageMaker image to satisfy your needs. For an example, see [Extending our PyTorch containers](#).

- **Build your own custom container image:** If there is no pre-built Amazon SageMaker container image that you can use or modify for an advanced scenario, you can package your own script or algorithm to use with Amazon SageMaker. You can use any programming language or framework to develop your container. For an example that shows how to build your own containers to train and host an algorithm, see [Bring Your Own R Algorithm](#).

The next topic provides a brief introduction to Docker containers. Amazon SageMaker has certain contractual requirements that a container must satisfy to be used with it. The following topic describes the Amazon SageMaker Containers library that can be used to create Amazon SageMaker-compatible containers, including a list of the environmental variables it defines and may require. Then a tutorial that shows how to get started by using Amazon SageMaker Containers to train a Python script. After the tutorial, topics:

- Describe the pre-built Docker containers provided by Amazon SageMaker for deep learning frameworks and other libraries.
- Provide examples of how to deploy containers for the various scenarios.

Subsequent sections describe in more detail the contractual requirements to use Docker with Amazon SageMaker to train your custom algorithms and to deploy your inference code to make predictions. There are two ways to make predictions when deploying a model. First, to get individual, real-time predictions, you can make inferences with a hosting service. Second, to get predictions for an entire dataset, you can use a batch transform. The final sections describe how to create algorithm and model package resources for use in your Amazon SageMaker account or to publish on AWS Marketplace.

Topics

- [Docker Container Basics \(p. 480\)](#)
- [Amazon SageMaker Containers: a Library to Create Docker Containers \(p. 481\)](#)
- [Get Started: Use Amazon SageMaker Containers to Run a Python Script \(p. 490\)](#)
- [Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch \(p. 493\)](#)
- [Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML \(p. 494\)](#)
- [Example Notebooks: Use Your Own Algorithm or Model \(p. 495\)](#)
- [Use Your Own Training Algorithms \(p. 496\)](#)
- [Use Your Own Inference Code \(p. 501\)](#)
- [Create Algorithm and Model Package Resources \(p. 506\)](#)
- [Use Algorithm and Model Package Resources \(p. 511\)](#)

Docker Container Basics

Docker containers provide isolation, portability, and security. They simplify the creation of highly distributed systems and save money by improving resource utilization. Docker relies on Linux kernel functionality to provide a lightweight virtualization to package applications into an image that is totally self-contained. Docker uses a file, called a Dockerfile, to specify how the image is assembled. When you have an image, you use Docker to build and run a container based on that image.

You can build your Docker images from scratch or base them on other Docker images that you or others have built. Images are stored in repositories that are indexed and maintained by registries. An image can be pushed into or pulled out of a repository using its registry address, which is similar to a URL. Docker Hub is a registry hosted by Docker, Inc. that provides publicly available repositories. AWS provides the [Amazon Elastic Container Service \(Amazon ECS\)](#), a highly scalable, fast container management service. With Amazon ECS, you can deploy any kind of code in Amazon SageMaker. You can also create a logical division of labor by creating a deployment team that handles DevOps and infrastructure, and that maintains the container, and a data science team that creates the algorithms and models that are later added to a container.

Docker builds images by reading the instructions from a Dockerfile text file that contains all of the commands, in order, that are needed to build the image. A Dockerfile adheres to a specific format and set of instructions. For more information, see [Dockerfile reference](#). Dockerfiles used in Amazon SageMaker must also satisfy additional requirements regarding the environmental variables, directory structure, timeouts, and other common functionality. For information, see [Use Your Own Training Algorithms \(p. 496\)](#) and [Use Your Own Inference Code \(p. 501\)](#).

For general information about Docker containers managed by Amazon ECS, see [Docker Basics for Amazon ECS](#) in the *Amazon Elastic Container Service Developer Guide*.

For more information about writing Dockerfiles to build images, see [Best practices for writing Dockerfiles](#).

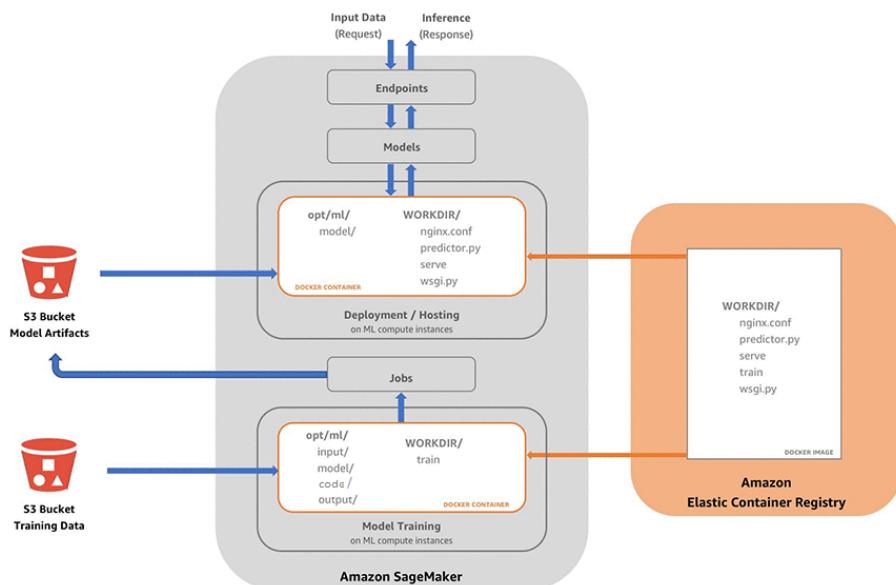
For general information about Docker, see the following:

- [Docker home page](#)
- [Docker overview](#)
- [Getting Started with Docker](#)
- [Dockerfile reference](#)

Amazon SageMaker Containers: a Library to Create Docker Containers

[Amazon SageMaker Containers](#) is a library that implements the functionality that you need to create containers to run scripts, train algorithms, or deploy models that are compatible with Amazon SageMaker. To install this library, use a `RUN pip install sagemaker-containers` command in your Dockerfile. The library defines the locations for storing code and other resources when you install it. Your Dockerfile must also copy the code to be run into the location expected by an Amazon SageMaker-compatible container and define the entry point containing the code to run when the container is started. The library also defines other information that a container needs to manage deployments for training and inference. After you build a Docker image, you can push it to the Amazon Elastic Container Registry (Amazon ECR). To create a container, you can pull the image from Amazon ECR and build the container using the `docker build` command.

The following high-level schematic shows how the files are organized in an Amazon SageMaker-compatible container created with the Amazon SageMaker Containers library.



When Amazon SageMaker trains a model, it creates a number of files in the container's `/opt/ml` directory.

```
/opt/ml
### input
#   ### config
#   #   ### hyperparameters.json
#   #   ### resourceConfig.json
#   ### data
#       ### <channel_name>
#           ### <input data>
### model
#
### code
#   ### <script files>
#
### output
### failure
```

When you run a model *training* job, the Amazon SageMaker container has a `/opt/ml/input/` directory that contains JSON files that configure the hyperparameters for the algorithm and the network layout used for distributed training. The directory also contains files that specify the channels through which Amazon SageMaker accesses the data in Amazon Simple Storage Service (Amazon S3). Place scripts to run in the `/opt/ml/code/` directory. The `/opt/ml/model/` directory contains the model generated by your algorithm in a single file or an entire directory tree in any format. You can also send information about why a training job failed to the `/opt/ml/output/` directory. Amazon SageMaker packages files in this directory into a compressed tar archive file.

When you *host* a trained model on Amazon SageMaker to make inferences, you deploy the model to an HTTP endpoint. The model makes realtime predictions in response to inference requests. The container must contain a serving stack to process these requests. The five files used in the standard Python serving stack by Amazon SageMaker are installed in the container's `WORKDIR`. You can choose a different toolset to deploy an HTTP endpoint and, therefore, could have a different layout. If you're writing in a programming language other than Python, you will have a different layout, the nature of which will depend on the frameworks and tools that you choose. The Python serving stack in the `WORKDIR` directory contains the following files:

- **`nginx.conf`** – The configuration file for the nginx front end.
- **`predictor.py`** – The program that implements the `Flask` web server and the decision tree predictions for this application. You need to customize the code that performs prediction for your application.
- **`serve`** – The program started when the container is started for hosting. This file simply launches the Gunicorn server, which runs multiple instances of the Flask application defined in `predictor.py`.
- **`train`** – The program that is invoked when you run the container for training. To implement your training algorithm, you modify this program.
- **`wsgi.py`** – A small wrapper used to invoke the Flask application.

In the container, the model files are in the same place that they were written to during training.

```
/opt/ml
### model
### <model files>
```

For more information, see [Use Your Own Inference Code \(p. 501\)](#)

You can provide separate Docker images for the training algorithm and inference code, as shown in the figure. Or you can use a single Docker image for both. 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.

Topics

- [Environmental Variables used by Amazon SageMaker Containers to Define Entry Points \(p. 483\)](#)
- [Environmental Variables used by Amazon SageMaker Containers Important for Running User Scripts \(p. 484\)](#)
- [Reference: Amazon SageMaker Containers Environmental Variables \(p. 486\)](#)
- [Additional Information for Scripts \(p. 490\)](#)

Environmental Variables used by Amazon SageMaker Containers to Define Entry Points

When creating a Dockerfile, you must define an entry point that specifies the location of the code to run when the container starts. Amazon SageMaker Containers does this by setting an `ENV` environment variable. The environment variable that you need to set depends on the job you want to do:

- To run a script, specify the `SAGEMAKER_PROGRAM` `ENV` variable.
- To train an algorithm, specify the `SAGEMAKER_TRAINING_MODULE` `ENV` variable.
- To host a model, specify the `SAGEMAKER_SERVING_MODULE` `ENV` variable.

You can use the Amazon SageMaker containers SDK package to set environment variables.

SAGEMAKER_PROGRAM

Train scripts similar to those you would use outside Amazon SageMaker using Amazon SageMaker Script Mode. It supports Python and Shell scripts: Amazon SageMaker uses the Python interpreter for any script with the `.py` suffix. Amazon SageMaker uses the Shell interpreter to execute any other script.

When running a program to specify the entry point for Script Mode, set the `SAGEMAKER_PROGRAM` environmental variable. The script must be located in the `/opt/ml/code` folder.

For example, the container used in the example in [Get Started: Use Amazon SageMaker Containers to Run a Python Script \(p. 490\)](#) sets this `ENV` as follows.

```
ENV SAGEMAKER_PROGRAM train.py
```

The Amazon SageMaker PyTorch container sets the `ENV` variable as follows.

```
ENV SAGEMAKER_PROGRAM cifar10.py
```

In the example, `cifar10.py` is the program that implements the training algorithm and handles loading the model for inferences. For more information, see the [Extending our PyTorch containers](#) notebook.

SAGEMAKER_TRAINING_MODULE

When training an algorithm, specify the location of the module that contains the training logic by setting the **SAGEMAKER_TRAINING_MODULE** environment variable. An Amazon SageMaker container invokes this module when the container starts training. For example, you set this environment variable in MXNet as follows.

```
ENV SAGEMAKER_TRAINING_MODULE sagemaker_mxnet_container.training:main
```

For [TensorFlow](#), set this environment variable as follows.

```
ENV SAGEMAKER_TRAINING_MODULE sagemaker_tensorflow_container.training:main
```

The code that implements this logic is in [Amazon SageMaker Containers](#).

SAGEMAKER_SERVING_MODULE

To locate the module that contains the hosting logic when deploying a model, set the **SAGEMAKER_SERVING_MODULE** environmental variable. An Amazon SageMaker container invokes this module when it starts hosting.

```
ENV SAGEMAKER_SERVING_MODULE sagemaker_mxnet_container.serving:main
```

The code that implements this logic is in the: [Amazon SageMaker Containers](#).

Environmental Variables used by Amazon SageMaker Containers Important for Running User Scripts

When you write a script to run in a container, you are likely to use the following build-time environment variables. [Amazon SageMaker Containers](#) sets some of these variable values by default.

- **SM_MODEL_DIR**

```
SM_MODEL_DIR=/opt/ml/model
```

When the training job finishes, Amazon SageMaker deletes the container, including its file system, except for the files in the /opt/ml/model and /opt/ml/output folders. Use /opt/ml/model to save the model checkpoints. Amazon SageMaker uploads these checkpoints to the default S3 bucket. Examples:

```
# Using it in argparse
parser.add_argument('model_dir', type=str, default=os.environ['SM_MODEL_DIR'])

# Using it as a variable
model_dir = os.environ['SM_MODEL_DIR']

# Saving checkpoints to the model directory in Chainer
serializers.save_npz(os.path.join(os.environ['SM_MODEL_DIR'], 'model.npz'), model)
```

For more information, see [How Amazon SageMaker Processes Training Output](#).

- **SM_CHANNELS**

```
SM_CHANNELS='["testing", "training"]'
```

The **SM_CHANNELS** environmental variable contains the list of input data channels for the container. When you train a model, you can partition your training data into different logical "channels". Common

channels are: training, testing, and evaluation, or images and labels. **SM_CHANNELS** includes the name of the channels that are in the container as a JSON encoded list.

Examples:

```
import json

# Using it in argparse
parser.add_argument('channel_names', type=int,
    default=json.loads(os.environ['SM_CHANNELS']))

# Using it as a variable
channel_names = json.loads(os.environ['SM_CHANNELS'])
```

- **SM_CHANNEL_{channel_name}**

```
SM_CHANNEL_TRAINING='/opt/ml/input/data/training'
SM_CHANNEL_TESTING='/opt/ml/input/data/testing'
```

The **SM_CHANNEL_{channel_name}** environmental variable contains the directory where the channel named `channel_name` is located in the container.

Examples:

```
import json

parser.add_argument('--train', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
parser.add_argument('--test', type=str, default=os.environ['SM_CHANNEL_TESTING'])

args = parser.parse_args()

train_file = np.load(os.path.join(args.train, 'train.npz'))
test_file = np.load(os.path.join(args.test, 'test.npz'))
```

- **SM_HPS**

```
SM_HPS='{"batch-size": "256", "learning-rate": "0.0001", "communicator": "pure_nccl"}'
```

The **SM_HPS** environmental variable contains a JSON encoded dictionary with the hyperparameters that you have provided.

Example:

```
import json

hyperparameters = json.loads(os.environ['SM_HPS'])
# {"batch-size": 256, "learning-rate": 0.0001, "communicator": "pure_nccl"}
```

- **SM_HP_{hyperparameter_name}**

```
SM_HP_LEARNING-RATE=0.0001
SM_HP_BATCH-SIZE=10000
SM_HP_COMMUNICATOR=pure_nccl
```

The **SM_HP_{hyperparameter_name}** environmental variable contains the value of the hyperparameter named `hyperparameter_name`.

Examples:

```
learning_rate = float(os.environ['SM_HP_LEARNING-RATE'])
batch_size = int(os.environ['SM_HP_BATCH-SIZE'])
communicator = os.environ['SM_HP_COMMUNICATOR']
```

- **SM_CURRENT_HOST**

```
SM_CURRENT_HOST=algo-1
```

The `SM_CURRENT_HOST` contains the name of the current container on the container network.

Examples:

```
# Using it in argparse
parser.add_argument('current_host', type=str, default=os.environ['SM_CURRENT_HOST'])

# Using it as a variable
current_host = os.environ['SM_CURRENT_HOST']
```

- **SM_HOSTS**

```
SM_HOSTS='["algo-1","algo-2"]'
```

The `SM_HOSTS` environmental variable contains a JSON-encoded list of all of the hosts.

Example:

```
import json

# Using it in argparse
parser.add_argument('hosts', type=nargs, default=json.loads(os.environ['SM_HOSTS']))

# Using it as variable
hosts = json.loads(os.environ['SM_HOSTS'])
```

- **SM_NUM_GPUS**

```
SM_NUM_GPUS=1
```

The `SM_NUM_GPUS` environmental variable contains the number of GPUs available in the current container.

Examples:

```
# Using it in argparse
parser.add_argument('num_gpus', type=int, default=os.environ['SM_NUM_GPUS'])

# Using it as a variable
num_gpus = int(os.environ['SM_NUM_GPUS'])
```

Reference: Amazon SageMaker Containers Environmental Variables

The following build-time environment variables are also defined by default when you use the [Amazon SageMaker Containers](#).

- **SM_NUM_CPUS**

```
SM_NUM_CPUS=32
```

The `SM_NUM_CPUS` environment variable contains the number of CPUs available in the current container.

Example:

```
# Using it in argparse
parser.add_argument('num_cpus', type=int, default=os.environ['SM_NUM_CPUS'])

# Using it as a variable
num_cpus = int(os.environ['SM_NUM_CPUS'])
```

- **SM_LOG_LEVEL**

```
SM_LOG_LEVEL=20
```

The `SM_LOG_LEVEL` environment variable contains the current log level in the container.

Example:

```
import logging

logger = logging.getLogger(__name__)

logger.setLevel(int(os.environ.get('SM_LOG_LEVEL', logging.INFO)))
```

- **SM_NETWORK_INTERFACE_NAME**

```
SM_NETWORK_INTERFACE_NAME=ethwe
```

The `SM_NETWORK_INTERFACE_NAME` environment variable contains the name of the network interface, which is used for distributed training.

Example:

```
# Using it in argparse
parser.add_argument('network_interface', type=str,
    default=os.environ['SM_NETWORK_INTERFACE_NAME'])

# Using it as a variable
network_interface = os.environ['SM_NETWORK_INTERFACE_NAME']
```

- **SM_USER_ARGS**

```
SM_USER_ARGS='["--batch-size","256","--learning_rate","0.0001","--communicator","pure_nccl"]'
```

The `SM_INPUT_DIR` environment variable contains a JSON-encoded list of the script arguments provided for training.

- **SM_INPUT_DIR**

```
SM_INPUT_DIR=/opt/ml/input/
```

The `SM_INPUT_DIR` environment variable contains the path of the input directory, `/opt/ml/input/`. This is the directory where Amazon SageMaker saves input data and configuration files before and during training.

- **`SM_INPUT_CONFIG_DIR`**

```
SM_INPUT_DIR=/opt/ml/input/config
```

The `SM_INPUT_CONFIG_DIR` environment variable contains the path of the input config directory, `/opt/ml/input/config/`. This is the directory where standard Amazon SageMaker configuration files are located.

When training starts, Amazon SageMaker creates the following files in this directory:

- `hyperparameters.json` – Contains the hyperparameters specified in the `CreateTrainingJob` request.
- `inputdataconfig.json` – Contains the data channel information that you specified in the `InputDataConfig` parameter in a `CreateTrainingJob` request.
- `resourceconfig.json` – Contains the name of the current host and all host containers used in the training.

For more information, see: [Using Your Own Training Algorithms](#).

- **`SM_OUTPUT_DATA_DIR`**

```
SM_OUTPUT_DATA_DIR=/opt/ml/output/data/algo-1
```

The `SM_OUTPUT_DATA_DIR` environment variable contains the directory where the algorithm writes non-model training artifacts, such as evaluation results. Amazon SageMaker retains these artifacts. As it runs in a container, your algorithm generates output, including the status of the training job and model, and the output artifacts. Your algorithm should write this information to this directory.

- **`SM_RESOURCE_CONFIG`**

```
SM_RESOURCE_CONFIG='{"current_host":"algo-1","hosts":["algo-1","algo-2"]}'
```

The `SM_RESOURCE_CONFIG` environment variable contains the contents of the `resourceconfig.json` file located in the `/opt/ml/input/config/` directory. It has the following keys:

- `current_host` – The name of the current container on the container network. For example, `"algo-1"`.
- `hosts` – The list of names of all of the containers on the container network, sorted lexicographically. For example, `["algo-1", "algo-2", "algo-3"]` for a three-node cluster.

For more information about the `resourceconfig.json` file, see: [Distributed Training Configuration](#).

- **`SM_INPUT_DATA_CONFIG`**

```
SM_INPUT_DATA_CONFIG='{
  "testing": {
    "RecordWrapperType": "None",
    "S3DistributionType": "FullyReplicated",
    "TrainingInputMode": "File"
  },
  "training": {
    "RecordWrapperType": "None",
    "S3DistributionType": "FullyReplicated",
    "TrainingInputMode": "File"
  }
}'
```

```
}'
```

The `SM_INPUT_DATA_CONFIG` environment variable contains the input data configuration of the `inputdataconfig.json` file located in the `/opt/ml/input/config/` directory.

For more information about the `resourceconfig.json` file, see [Distributed Training Configuration](#).

- **`SM_TRAINING_ENV`**

```
SM_TRAINING_ENV='
{
    "channel_input_dirs": {
        "test": "/opt/ml/input/data/testing",
        "train": "/opt/ml/input/data/training"
    },
    "current_host": "algo-1",
    "framework_module": "sagemaker_chainer_container.training:main",
    "hosts": [
        "algo-1",
        "algo-2"
    ],
    "hyperparameters": {
        "batch-size": 10000,
        "epochs": 1
    },
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "test": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        },
        "train": {
            "RecordWrapperType": "None",
            "S3DistributionType": "FullyReplicated",
            "TrainingInputMode": "File"
        }
    },
    "input_dir": "/opt/ml/input",
    "job_name": "preprod-chainer-2018-05-31-06-27-15-511",
    "log_level": 20,
    "model_dir": "/opt/ml/model",
    "module_dir": "s3://sagemaker-{aws-region}-{aws-id}/{training-job-name}/source/
sourcedir.tar.gz",
    "module_name": "user_script",
    "network_interface_name": "ethwe",
    "num_cpus": 4,
    "num_gpus": 1,
    "output_data_dir": "/opt/ml/output/data/algo-1",
    "output_dir": "/opt/ml/output",
    "resource_config": {
        "current_host": "algo-1",
        "hosts": [
            "algo-1",
            "algo-2"
        ]
    }
}'
```

The `SM_TRAINING_ENV` environment variable provides all of the training information as a JSON-encoded dictionary.

Additional Information for Scripts

Scripts can assign values for the hyperparameters of an algorithm. The interpreter passes all hyperparameters specified in the training job to the entry point as script arguments. For example, it passes the training job hyperparameters as follow.:

```
{"HyperParameters": {"batch-size": 256, "learning-rate": 0.0001, "communicator": "pure_nccl"}}
```

When an entry point needs additional information from the container that isn't available in hyperparameters, Amazon SageMaker Containers writes this information as environment variables that are available in the script. For example, the following training job includes the `training` and `testing` channels.

```
from sagemaker.pytorch import PyTorch

estimator = PyTorch(entry_point='train.py', ...)

estimator.fit({'training': 's3://bucket/path/to/training/data',
               'testing': 's3://bucket/path/to/testing/data'})
```

The environment variable `SM_CHANNEL_{channel_name}` provides the path where the channel is located.

```
import argparse
import os

if __name__ == '__main__':
    parser = argparse.ArgumentParser()

    ...

    # reads input channels training and testing from the environment variables
    parser.add_argument('--training', type=str, default=os.environ['SM_CHANNEL_TRAINING'])
    parser.add_argument('--testing', type=str, default=os.environ['SM_CHANNEL_TESTING'])

    args = parser.parse_args()
    ...
```

Get Started: Use Amazon SageMaker Containers to Run a Python Script

To run an arbitrary script-based program in a Docker container using the Amazon SageMaker Containers, build a Docker container with an Amazon SageMaker notebook instance, as follows:

1. Create the notebook instance.
2. Create and upload the Dockerfile and Python scripts.
3. Build the container.
4. Test the container locally.
5. Clean up the resources.

To create an Amazon SageMaker notebook instance

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.

2. Choose **Notebook**, **Notebook instances**, and **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information:
 - a. For **Notebook instance name**, enter **RunScriptNotebookInstance**.
 - b. For **Notebook Instance type**, choose **ml.t2.medium**.
 - c. For **IAM role**, choose **Create a new role**.
 - i. Choose **Create a new role**.
 - ii. On the **Create an IAM role** page, choose **Specific S3 buckets**, specify an S3 bucket named **sagemaker-run-script**, and then choose **Create role**.

Amazon SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmSS`. For example, `AmazonSageMaker-ExecutionRole-20190429T110788`. Note that the execution role naming convention uses the date and time when the role was created, separated by a `T`. Record the role name because you'll need it later.

- d. For **Root Access**, accept the default, **Enabled**.
- e. Choose **Create notebook instance**.

It takes a few minutes for Amazon SageMaker to launch an ML compute instance—in this case, a notebook instance—and attach an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the [CreateNotebookInstance API](#).

4. When the status of the notebook instance is **InService**, from **Actions**, choose **Open Jupyter**.
For **New**, choose **conda_tensorflw_p36**. This is the kernel you need.
5. To name the notebook, choose **File**, **Rename**, enter **tRun-Python-Script**, and then choose **Rename**.

To create and upload the Dockerfile and Python scripts

1. In the editor of your choice, create the following Dockerfile text file locally and save it with the file name "Dockerfile" without an extension. The `docker build` command expects by default to find a file with precisely this name in the dockerfile directory. For example, in Notepad, you can save a text file without an extension by choosing **File**, **Save As** and choosing **All types (*.*)**.

```
FROM tensorflow/tensorflow:2.0.0a0
RUN pip install sagemaker-containers
# Copies the training code inside the container
COPY train.py /opt/ml/code/train.py
# Defines train.py as script entrypoint
ENV SAGEMAKER_PROGRAM train.py
```

The Dockerfile script performs the following tasks:

- `FROM tensorflow/tensorflow:2.0.0a0` downloads the TensorFlow library used to run the Python script.
- `RUN pip install sagemaker-containers` [Amazon SageMaker Containers](#) contains the common functionality necessary to create a container compatible with Amazon SageMaker.
- `COPY train.py /opt/ml/code/train.py` copies the script to the location inside the container that is expected by Amazon SageMaker. The script must be located in this folder.

- ENV SAGEMAKER_PROGRAM train.py defines train.py as the name of the entrypoint script that is located in the /opt/ml/code folder for the container. This is the only environmental variable that you must specify when you are using your own container.
2. In the editor of your choice, create and save the following train.py text file locally.

```

import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=1)

model.evaluate(x_test, y_test)

```

3. To upload the Dockerfile to a dockerfile directory, choose **Open JupyterLab**, choose the **File Browser** icon, and then choose the **New Folder** icon. This creates a new directory named **dockerfile**.
4. Double-click the new dockerfile folder, choose the **Upload Files** icon, navigate to where you saved your Dockerfile and train.py script files, and upload them to the dockerfile folder.

To build the container

1. The Jupyter Notebook opens in the SageMaker directory. The Docker build command must be run from the dockerfile directory you created. Run the following command to change into the dockerfile directory:

```
cd dockerfile
```

This returns your current directory: /home/ec2-user/SageMaker/dockerfile

2. To build the Docker container, run the following Docker build command, including the final period.

```
!docker build -t tf-2.0 .
```

To test the container locally

1. Use Local Mode to test the container locally. Replace the 'SageMakerRole' value with the ARN for the role with the IAM role you created when configuring the notebook instance. The ARN should look like: 'arn:aws:iam::109225375568:role/service-role/AmazonSageMaker-ExecutionRole-20190429T110788'.

```

from sagemaker.estimator import Estimator

estimator = Estimator(image_name='tf-2.0',
                      role='SageMakerRole',

```

```
train_instance_count=1,
train_instance_type='local')

estimator.fit()
```

This test outputs the training environment configuration, the values used for the environmental variables, the source of the data, and the loss and accuracy obtained during training.

2. After using Local Mode, you can push the image to Amazon Elastic Container Registry and use it to run training jobs. For an example that shows how to complete these tasks, see [Building Your Own TensorFlow Container](#)

To clean up resources when done with the get started example

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>, stop and then delete the notebook instance.
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.

Note

The Docker container shuts down automatically after it has run. You don't need to delete it.

Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch

Amazon SageMaker provides prebuilt Docker images that include deep learning framework libraries and other dependencies needed for training and inference. With the [Amazon SageMaker SageMaker Python SDK](#), you can train and deploy models using these popular deep learning frameworks. For instructions on installing and using the SDK, see [Amazon SageMaker Python SDK](#).

The following table provides links to the GitHub repositories that contain the source code and Dockerfiles for each framework and for TensorFlow and MXNet Serving. The instructions linked are for using the Python SDK to run training algorithms and host models on Amazon SageMaker.

Framework	Prebuilt Docker Image Source Code	Instructions
TensorFlow	Amazon SageMaker TensorFlow Containers	Using TensorFlow with the SageMaker Python SDK
	Amazon SageMaker TensorFlow Serving Containers	
MXNet	Amazon SageMaker MXNet Containers	Using MXNet with the SageMaker Python SDK
	Amazon SageMaker MXNet Serving Containers	
PyTorch	Amazon SageMaker PyTorch Containers	Using PyTorch with the SageMaker Python SDK
	Amazon SageMaker PyTorch Serving Containers	
Chainer	Amazon SageMaker Chainer SageMaker Containers	Using Chainer with the SageMaker Python SDK

If you are not using the Amazon SageMaker Python SDK and one of its estimators to retrieve the pre-built images, you have to retrieve them yourself. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). For a complete list of the available pre-built Docker containers, see [Deep Learning Containers Images](#).

Amazon SageMaker also provides prebuilt Docker images for scikit-learn and Spark ML. For information about Docker images that enable using scikit-learn and Spark ML solutions in Amazon SageMaker, see [Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML \(p. 494\)](#).

You can use prebuilt containers to deploy your custom models or models that have been trained in a framework other than Amazon SageMaker. For an overview of the process of bringing the trained model artifacts into Amazon SageMaker and hosting them at an endpoint, see [Bring Your Own Pretrained MXNet or TensorFlow Models into Amazon SageMaker](#).

You can customize these prebuilt containers or extend them to handle any additional functional requirements for your algorithm or model that the prebuilt Amazon SageMaker Docker image doesn't support. For an example, see [Extending Our PyTorch Containers](#).

Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML

Amazon SageMaker provides prebuilt Docker images that install the scikit-learn and Spark ML libraries and the dependencies they need to build Docker images that are compatible with Amazon SageMaker using the Amazon SageMaker Python SDK. With the SDK, you can use scikit-learn for machine learning tasks and use Spark ML to create and tune machine learning pipelines. For instructions on installing and using the SDK, see [SageMaker Python SDK](#). The following table contains links to the GitHub repositories with the source code and the Dockerfiles for scikit-learn and Spark ML frameworks and to instructions that show how use the Python SDK estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker Hosting.

Library	Prebuilt Docker Image Source Code	Instructions
scikit-learn	SageMaker Scikit-learn Containers	Using Scikit-learn with the Amazon SageMaker Python SDK
Spark ML	SageMaker Spark ML Serving Containers	SparkML Serving

If you are not using the SM Python SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-build container. The Amazon SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). You can push or pull them using their fullname registry addresses. Amazon SageMaker uses the following Docker Image URL patterns for scikit-learn and Spark M:

- <ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-scikit-learn
 - For example, 746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-scikit-learn
- <ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-sparkml-serving
 - For example, 341280168497.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-sparkml-serving

The following table lists the supported values for account IDs and corresponding AWS Region names.

ACCOUNT_ID	REGION_NAME
746614075791	us-west-1
246618743249	us-west-2
683313688378	us-east-1
257758044811	us-east-2
354813040037	ap-northeast-1
366743142698	ap-northeast-2
121021644041	ap-southeast-1
783357654285	ap-southeast-2
720646828776	ap-south-1
141502667606	eu-west-1
764974769150	eu-west-2
492215442770	eu-central-1
341280168497	ca-central-1
414596584902	us-gov-west-1

The supported values listed in the table are also available on the [fw_registry.py](#) page of the Amazon SageMaker Python SDK GitHub repository.

Amazon SageMaker also provides prebuilt Docker images for popular deep learning frameworks. For information about Docker images that enable using deep learning frameworks in Amazon SageMaker, see [Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch \(p. 493\)](#).

For information on Docker images for developing reinforcement learning (RL) solutions in Amazon SageMaker, see [Amazon SageMaker RL Containers](#).

Example Notebooks: Use Your Own Algorithm or Model

The following sample notebooks show how to use your own algorithms or pretrained models from an Amazon SageMaker notebook instance. After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab for a list of all Amazon SageMaker example notebooks. You can open the sample notebooks from the **Advanced Functionality** section in your notebook instance or in GitHub at the provided links. To open a notebook, choose its **Use** tab, then choose **Create copy**.

For instructions on how to create and access Jupyter notebook instances, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#)

To learn how to **host models trained in scikit-learn** for making predictions in Amazon SageMaker by injecting them first-party k-means and XGBoost containers, see the following sample notebooks.

- kmeans Bring Your Own Model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/kmeansBringYourOwnModel
- xgboost Bring Your Own Model - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/xgboostBringYourOwnModel

To learn how to **package algorithms that you have developed in TensorFlow and scikit-learn frameworks for training and deployment in the Amazon SageMaker environment**, see the following notebooks. They show you how to build, register, and deploy your own Docker containers using Dockerfiles.

- tensorflow_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/tensorflow_bring_your_own
- scikit_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/scikit_bring_your_own

To learn how to **train a neural network locally using MXNet or TensorFlow**, and then create an endpoint from the trained model and **deploy it on Amazon SageMaker**, see the following notebooks. The MXNet model is trained to recognize handwritten numbers from the MNIST dataset. The TensorFlow model is trained to classify irises.

- mxnet_mnist_byom - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/mxnet_mnist_byom
- tensorflow_iris_byom - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/tensorflow_iris_byom

To learn how to **use a Dockerfile to build a container that calls the `train.py` script** and uses pipe mode to custom train an algorithm, see the following notebook. In pipe mode, the input data is transferred to the algorithm while it is training. This can decrease training time compared to using file-mode.

- pipe_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/pipe_bring_your_own

To learn how to **use an R container to train and host a model** with the R kernel installed in a notebook , see the following notebook. To take advantage of the AWS SDK for Python (Boto 3), we use Python within the notebook. You can achieve the same results completely in R by invoking command line arguments.

- r_bring_your_own - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/r_bring_your_own

To learn how to **extend a prebuilt Amazon SageMaker PyTorch container image** when you have additional functional requirements for your algorithm or model that the pre-built Docker image doesn't support, see the following notebook.

- pytorch_extending_our_containers - https://github.com/awslabs/amazon-sagemaker-examples/tree/master/advanced_functionality/pytorch_extending_our_containers

For links to the GitHub repositories with the prebuilt Dockerfiles for the TensorFlow, MXNet, Chainer, and PyTorch frameworks and instructions on use the AWS SDK for Python (Boto 3) estimators to run your own training algorithms on Amazon SageMaker Learner and your own models on Amazon SageMaker hosting, see [Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch \(p. 493\)](#)

Use 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 \(p. 497\)](#)
- [How Amazon SageMaker Provides Training Information \(p. 498\)](#)
- [How Amazon SageMaker Signals Algorithm Success and Failure \(p. 500\)](#)
- [How Amazon SageMaker Processes Training Output \(p. 500\)](#)

How Amazon SageMaker Runs Your Training Image

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

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

```
docker run image train
```

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

- Use the `exec` form of the `ENTRYPOINT` instruction:

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

For example:

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

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

- The [CreateTrainingJob](#) API has a stopping condition that directs Amazon SageMaker to stop model training after a specific time.
- The [StopTrainingJob](#) API issues the equivalent of the `docker stop`, with a 2 minute timeout, command to gracefully stop the specified container:

```
docker stop -t120
```

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

Note

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

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

- You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the train and serve arguments.
- `/opt/ml` and all sub-directories are reserved by Amazon SageMaker training. When building your algorithm's docker image, please ensure you don't place any data required by your algorithm under them as the data may no longer be visible during training.

How Amazon SageMaker Provides Training Information

This section explains how Amazon SageMaker makes training information, such as training data, hyperparameters, and other configuration information, available to your Docker container.

When you send a `CreateTrainingJob` request to Amazon SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contains the training algorithm. You also specify the Amazon Simple Storage Service (Amazon S3) location where training data is stored and algorithm-specific parameters. Amazon SageMaker makes this information available to the Docker container so that your training algorithm can use it. This section explains how we make this information available to your Docker container. For information about creating a training job, see `CreateTrainingJob`.

Topics

- [Hyperparameters \(p. 498\)](#)
- [Environment Variables \(p. 498\)](#)
- [Input Data Configuration \(p. 498\)](#)
- [Training Data \(p. 499\)](#)
- [Distributed Training Configuration \(p. 499\)](#)

Hyperparameters

Amazon SageMaker makes the hyperparameters in a `CreateTrainingJob` request available in the Docker container in the `/opt/ml/input/config/hyperparameters.json` file.

Environment Variables

- `TRAINING_JOB_NAME`—The training job name stored in the `TrainingJobName` parameter in a `CreateTrainingJob` request.
- `TRAINING_JOB_ARN`—The Amazon Resource Name (ARN) of the training job returned as the `TrainingJobArn` response element for `CreateTrainingJob`.

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:

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

```
    "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. `S3DistributionType` will be set as `FullyReplicated` if specify EFS or FSxLustre as input data sources.

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`
- **Note**
Channels that use file system data sources such as Amazon Elastic File System (EFS) and Amazon FSx must use FILE mode. Also to utilize an FSx file server, you must specify a path that begins with `/fsx`. If a file system is specified, the directory path provided in the channel is mounted at `/opt/ml/input/data/channel_name`.
- **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 end-of-file (EOF), or if you are done with the first epoch, close the pipe file early.
2. After closing the first pipe file, look for `/opt/ml/input/data/training_1` and read it until you have completed the second epoch, and so on.

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

Distributed Training Configuration

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

To enable inter-container communication, this JSON file contains information for all containers. Amazon SageMaker makes this file available for both FILE and PIPE mode algorithms. The file provides the following information:

- `current_host`—The name of the current container on the container network. For example, `algo-1`. Host values can change at any time. Don't write code with specific values for this variable.
- `hosts`—The list of names of all containers on the container network, sorted lexicographically. For example, `["algo-1", "algo-2", "algo-3"]` for a three-node cluster. Containers can use these names to address other containers on the container network. Host values can change at any time. Don't write code with specific values for these variables.
- `network_interface_name`—The name of the network interface that is exposed to your container. For example, containers running the Message Passing Interface (MPI) can use this information to set the network interface name.
- Do not use the information in `/etc/hostname` or `/etc/hosts` because it might be inaccurate.
- Hostname information may not be immediately available to the algorithm container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

The following is an example file on node 1 in a three-node cluster:

```
{  
  "current_host": "algo-1",  
  "hosts": ["algo-1", "algo-2", "algo-3"],  
  "network_interface_name": "eth1"  
}
```

How Amazon SageMaker Signals Algorithm Success and Failure

A training algorithm indicates whether it succeeded or failed using the exit code of its process.

A successful training execution should exit with an exit code of 0 and an unsuccessful training execution should exit with a non-zero exit code. These will be converted to "Completed" and "Failed" in the `TrainingJobStatus` returned by `DescribeTrainingJob`. This exit code convention is standard and is easily implemented in all languages. For example, in Python, you can use `sys.exit(1)` to signal a failure exit and simply running to the end of the main routine will cause Python to exit with code 0.

In the case of failure, the algorithm can write a description of the failure to the failure file. See next section for details.

How Amazon SageMaker Processes Training Output

As your algorithm runs in a container, it generates output including the status of the training job and model and output artifacts. Your algorithm should write this information to the following files, which are located in the container's `/output` directory. Amazon SageMaker processes the information contained in this directory as follows:

- `/opt/ml/output/failure`—If training fails, after all algorithm output (for example, logging) completes, your algorithm should write the failure description to this file. In a `DescribeTrainingJob` response, Amazon SageMaker returns the first 1024 characters from this file as `FailureReason`.
- `/opt/ml/model`—Your algorithm should write all final model artifacts to this directory. Amazon SageMaker copies this data as a single object in compressed tar format to the S3 location that you specified in the `CreateTrainingJob` request. If multiple containers in a single training job write to this directory they should ensure no file/directory names clash. Amazon SageMaker aggregates the result in a tar file and uploads to s3.

Use Your Own Inference Code

You can use Amazon SageMaker to interact with Docker containers and run your own inference code in one of two ways:

- To use your own inference code with a persistent endpoint to get one prediction at a time, use Amazon SageMaker hosting services.
- To use your own inference code to get predictions for an entire dataset, use Amazon SageMaker batch transform.

Topics

- [Use Your Own Inference Code with Hosting Services \(p. 501\)](#)
- [Use Your Own Inference Code with Batch Transform \(p. 503\)](#)

Use Your Own Inference Code with Hosting Services

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

Topics

- [How Amazon SageMaker Runs Your Inference Image \(p. 501\)](#)
- [How Amazon SageMaker Loads Your Model Artifacts \(p. 502\)](#)
- [How Containers Serve Requests \(p. 503\)](#)
- [How Your Container Should Respond to Inference Requests \(p. 503\)](#)
- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 503\)](#)

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](#) API to create an endpoint, Amazon SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. Amazon SageMaker runs the Docker container on those instances.

If you reduce the number of instances backing the endpoint (by calling the [UpdateEndpointWeightsAndCapacities](#) APIs), Amazon SageMaker runs a command to stop the Docker container on the instances being terminated. The command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

If you update the endpoint (by calling the [UpdateEndpoint](#) API), Amazon SageMaker launches another set of ML compute instances and runs the Docker containers that contain your inference code on them. Then it runs a command to stop the previous Docker containers. To stop a Docker container, command sends the `SIGTERM` signal, then it sends the `SIGKILL` signal thirty seconds later.

- Amazon SageMaker uses the container definition that you provided in your [CreateModel](#) request to set environment variables and the DNS hostname for the container as follows:
 - It sets environment variables using the `ContainerDefinition.Environment` string-to-string map.
 - It sets the DNS hostname using the `ContainerDefinition.ContainerHostname`.
- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateEndpointConfig` request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see [NVIDIA/nvidia-docker](#).
- You can't use the `tini` initializer as your entry point in Amazon SageMaker containers because it gets confused by the `train` and `serve` arguments.

How Amazon SageMaker Loads Your Model Artifacts

In your [CreateModel](#) request, the container definition includes the `ModelDataUrl` parameter, which identifies the S3 location where model artifacts are stored. Amazon SageMaker uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory for use by your inference code.

The `ModelDataUrl` must point to a tar.gz file. Otherwise, Amazon SageMaker won't download the file.

If you trained your model in Amazon SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside Amazon SageMaker, you need to create this single compressed tar file and save it in a S3 location. Amazon SageMaker decompresses this tar file into `/opt/ml/model` directory before your container starts.

How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

How Your Container Should Respond to Inference Requests

To obtain inferences, the client application sends a POST request to the Amazon SageMaker endpoint. For more information, see the [InvokeEndpoint](#) API. Amazon SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

- Amazon SageMaker strips all POST headers except those supported by `InvokeEndpoint`. Amazon SageMaker might add additional headers. Inference containers must be able to safely ignore these additional headers.
- To receive inference requests, the container must have a web server listening on port 8080 and must accept POST requests to the `/invocations` endpoint.
- A customer's model containers must accept socket connection requests within 250 ms.
- A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the `/invocations`. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

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

The `CreateEndpoint` and `UpdateEndpoint` API calls result in Amazon SageMaker starting new inference containers. Soon after container startup, Amazon SageMaker starts sending periodic GET requests to the `/ping` endpoint.

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the `/invocations` endpoint.

If the container does not begin to pass health checks, by consistently responding with 200s, during the 4 minutes after startup, `CreateEndpoint` will fail, leaving Endpoint in a failed state, and the update requested by `UpdateEndpoint` will not be completed.

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.

Use Your Own Inference Code with Batch Transform

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

Topics

- [How Amazon SageMaker Runs Your Inference Image \(p. 503\)](#)
- [How Amazon SageMaker Loads Your Model Artifacts \(p. 504\)](#)
- [How Containers Serve Requests \(p. 505\)](#)
- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 505\)](#)

How Amazon SageMaker Runs Your Inference Image

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

- For batch transforms, 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"]
```

- Amazon SageMaker sets environment variables specified in [CreateModel](#) and [CreateTransformJob](#) on your container. Additionally, the following environment variables will be populated:
 - `SAGEMAKER_BATCH` is always set to `true` when the container runs in Batch Transform.
 - `SAGEMAKER_MAX_PAYLOAD_IN_MB` is set to the largest size payload that will be sent to the container via HTTP.
 - `SAGEMAKER_BATCH_STRATEGY` will be set to `SINGLE_RECORD` when the container will be sent a single record per call to invocations and `MULTI_RECORD` when the container will get as many records as will fit in the payload.
 - `SAGEMAKER_MAX_CONCURRENT_TRANSFORMS` is set to the maximum number of /invocations requests that can be opened simultaneously.

Note

The last three environment variables come from the API call made by the user. If the user doesn't set values for them, they aren't passed. In that case, either the default values or the values requested by the algorithm (in response to the `/execution-parameters`) are used.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your [CreateTransformJob](#) request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see [NVIDIA/nvidia-docker](#).
- You can't use the `init` 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 a [CreateModel](#) request, container definitions includes the `ModelDataUrl` parameter, which identifies the location in Amazon S3 where model artifacts are stored. When you use Amazon SageMaker to run inferences, it uses this information to determine where to copy the model artifacts from. It copies the artifacts to the `/opt/ml/model` directory in the Docker container for use by your inference code.

The `ModelDataUrl` parameter must point to a tar.gz file. Otherwise, Amazon SageMaker can't download the file. If you train a model in Amazon SageMaker, it saves the artifacts as a single compressed tar file in Amazon S3. If you train a model in another framework, you need to store the

model artifacts in Amazon S3 as a compressed tar file. Amazon SageMaker decompresses this tar file and saves it in the `/opt/ml/model` directory in the container before the batch transform job starts.

How Containers Serve Requests

Containers must implement a web server that responds to invocations and ping requests on port 8080. For batch transforms, you have the option to set algorithms to implement execution-parameters requests to provide a dynamic runtime configuration to Amazon SageMaker. Amazon SageMaker uses the following endpoints:

- `ping`—Used to periodically check the health of the container. Amazon SageMaker waits for an HTTP 200 status code and an empty body for a successful ping request before sending an invocations request. You might use a ping request to load a model into memory to generate inference when invocations requests are sent.
- (Optional) `execution-parameters`—Allows the algorithm to provide the optimal tuning parameters for a job during runtime. Based on the memory and CPUs available for a container, the algorithm chooses the appropriate `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` values for the job.

Before calling the invocations request, Amazon SageMaker attempts to invoke the execution-parameters request. When you create a batch transform job, you can provide values for the `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. Amazon SageMaker determines the values for these parameters using this order of precedence:

1. The parameter values that you provide when you create the `CreateTransformJob` request,
2. The values that the model container returns when Amazon SageMaker invokes the execution-parameters endpoint
3. The parameters default values, listed in the following table.

Parameter	Default Values
<code>MaxConcurrentTransforms</code>	1
<code>BatchStrategy</code>	<code>MULTI_RECORD</code>
<code>MaxPayloadInMB</code>	6

The response for a `GET` execution-parameters request is a JSON object with keys for `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. This is an example of a valid response:

```
{
  "MaxConcurrentTransforms": 8,
  "BatchStrategy": "MULTI_RECORD",
  "MaxPayloadInMB": 6
}
```

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

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to Amazon SageMaker that the container is ready to accept inference requests at the `/invocations` endpoint.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on `/ping` attempts is 2 seconds.

Create Algorithm and Model Package Resources

After your training and/or inference code is packaged in Docker containers, create algorithm and model package resources that you can use in your Amazon SageMaker account and, optionally, publish on AWS Marketplace.

Topics

- [Create an Algorithm Resource \(p. 506\)](#)
- [Create a Model Package Resource \(p. 509\)](#)

Create an Algorithm Resource

To create an algorithm resource that you can use to run training jobs in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker containers that contains the training and, optionally, inference code.
- The configuration of the input data that your algorithm expects for training.
- The hyperparameters that your algorithm supports.
- Metrics that your algorithm sends to Amazon CloudWatch during training jobs.
- The instance types that your algorithm supports for training and inference, and whether it supports distributed training across multiple instances.
- Validation profiles, which are training jobs that Amazon SageMaker uses to test your algorithm's training code and batch transform jobs that Amazon SageMaker runs to test your algorithm's inference code.

To ensure that buyers and sellers can be confident that products work in Amazon SageMaker, we require that you validate your algorithms before listing them on AWS Marketplace. You can list products in the AWS Marketplace only if validation succeeds. To validate your algorithms, Amazon SageMaker uses your validation profile and sample data to run the following validations tasks:

1. Create a training job in your account to verify that your training image works with Amazon SageMaker.
2. If you included inference code in your algorithm, create a model in your account using the algorithm's inference image and the model artifacts produced by the training job.
3. If you included inference code in your algorithm, create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.

When you list your product on AWS Marketplace, the inputs and outputs of this validation process persist as part of your product and are made available to your buyers. This helps buyers understand and evaluate the product before they buy it. For example, buyers can inspect the input data that you used, the outputs generated, and the logs and metrics emitted by your code. The more comprehensive your validation specification, the easier it is for customers to evaluate your product.

Note

In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the **Training jobs** and **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. If any issues are found, you will have to create the algorithm again.

Note

To publish your algorithm on AWS Marketplace, at least one validation profile is required.

You can create an algorithm by using either the Amazon SageMaker console or the Amazon SageMaker API.

Topics

- [Create an Algorithm Resource \(Console\) \(p. 507\)](#)
- [Create an Algorithm Resource \(API\) \(p. 509\)](#)

Create an Algorithm Resource (Console)

To create an algorithm resource (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Algorithms**, then choose **Create algorithm**.
3. On the **Training specifications** page, provide the following information:
 - a. For **Algorithm name**, type a name for your algorithm. The algorithm name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
 - b. Type a description for your algorithm. This description appears in the Amazon SageMaker console and in the AWS Marketplace.
 - c. For **Training image**, type the path in Amazon ECR where your training container is stored.
 - d. For **Support distributed training**, Choose **Yes** if your algorithm supports training on multiple instances. Otherwise, choose **No**.
 - e. For **Support instance types for training**, choose the instance types that your algorithm supports.
 - f. For **Channel specification**, specify up to 8 channels of input data for your algorithm. For example, you might specify 3 input channels named `train`, `validation`, and `test`. For each channel, specify the following information:
 - i. For **Channel name**, type a name for the channel. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
 - ii. To require the channel for your algorithm, choose **Channel required**.
 - iii. Type a description for the channel.
 - iv. For **Supported input modes**, choose **Pipe mode** if your algorithm supports streaming the input data, and **File mode** if your algorithm supports downloading the input data as a file. You can choose both.
 - v. For **Supported content types**, type the MIME type that your algorithm expects for input data.
 - vi. For **Supported compression type**, choose **Gzip** if your algorithm supports Gzip compression. Otherwise, choose **None**.
 - vii. Choose **Add channel** to add another data input channel, or choose **Next** if you are done adding channels.
4. On the **Tuning specifications** page, provide the following information:
 - a. For **Hyperparameter specification**, specify the hyperparameters that your algorithm supports by editing the JSON object. For each hyperparameter that your algorithm supports, construct a JSON block similar to the following:

```
{  
  "DefaultValue": "5",  
  "Description": "The first hyperparameter",  
  "IsRequired": true,  
  "IsTunable": false,  
  "Name": "intRange",  
  "Range": {  
    "IntegerParameterRangeSpecification": {  
      "MaxValue": "10",  
      "MinValue": "1"  
    }  
  }  
}
```

```
"MinValue": "1"
}
},
"Type": "Integer"
```

In the JSON, supply the following:

- i. For **DefaultValue**, specify a default value for the hyperparameter, if there is one.
 - ii. For **Description**, specify a description for the hyperparameter.
 - iii. For **IsRequired**, specify whether the hyperparameter is required.
 - iv. For **IsTunable**, specify `true` if this hyperparameter can be tuned when a user runs a hyperparameter tuning job that uses this algorithm. For information, see [Perform Automatic Model Tuning \(p. 575\)](#).
 - v. For **Name**, specify a name for the hyperparameter.
 - vi. For **Range**, specify one of the following:
 - **IntegerParameterRangeSpecification** - the values of the hyperparameter are integers. Specify minimum and maximum values for the hyperparameter.
 -
 - **ContinuousParameterRangeSpecification** - the values of the hyperparameter are floating-point values. Specify minimum and maximum values for the hyperparameter.
 - **CategoricalParameterRangeSpecification** - the values of the hyperparameter are categorical values. Specify a list of all of the possible values.
 - vii. For **Type**, specify `Integer`, `Continuous`, or `Categorical`. The value must correspond to the type of **Range** that you specified.
- b. For **Metric definitions**, specify any training metrics that you want your algorithm to emit. Amazon SageMaker uses the regular expression that you specify to find the metrics by parsing the logs from your training container during training. Users can view these metrics when they run training jobs with your algorithm, and they can monitor and plot the metrics in Amazon CloudWatch. For information, see [Monitor and Analyze Training Jobs Using Metrics \(p. 606\)](#). For each metric, provide the following information:
 - i. For **Metric name**, type a name for the metric.
 - ii. For **Regex**, type the regular expression that Amazon SageMaker uses to parse training logs so that it can find the metric value.
 - iii. For **Objective metric support** choose `Yes` if this metric can be used as the objective metric for a hyperparameter tuning job. For information, see [Perform Automatic Model Tuning \(p. 575\)](#).
 - iv. Choose **Add metric** to add another metric, or choose **Next** if you are done adding metrics.
5. On the **Inference specifications** page, provide the following information if your algorithm supports inference:
 - a. For **Container definition**, type path in Amazon ECR where your inference container is stored.
 - b. For **Container DNS host name**, type the name of a DNS host for your image.
 - c. For **Supported instance types for real-time inference**, choose the instance types that your algorithm supports for models deployed as hosted endpoints in Amazon SageMaker. For information, see [Deploy a Model on Amazon SageMaker Hosting Services \(p. 8\)](#).
 - d. For **Supported instance types for batch transform jobs**, choose the instance types that your algorithm supports for batch transform jobs. For information, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#).
 - e. For **Supported content types**, type the type of input data that your algorithm expects for inference requests.

- f. For **Supported response MIME types**, type the MIME types that your algorithm supports for inference responses.
 - g. Choose **Next**.
6. On the **Validation specifications** page, provide the following information:
- a. For **Publish this algorithm on AWS Marketplace**, choose **Yes** to publish the algorithm on AWS Marketplace.
 - b. For **Validate this algorithm**, choose **Yes** if you want Amazon SageMaker to run training jobs and/or batch transform jobs that you specify to test the training and/or inference code of your algorithm.

Note

To publish your algorithm on AWS Marketplace, your algorithm must be validated.

- c. For **IAM role**, choose an IAM role that has the required permissions to run training jobs and batch transform jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [Amazon SageMaker Roles \(p. 776\)](#).
- d. For **Validation profile**, specify the following:
 - A name for the validation profile.
 - A **Training job definition**. This is a JSON block that describes a training job. This is in the same format as the `TrainingJobDefinition` input parameter of the `CreateAlgorithm` API.
 - A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition` input parameter of the `CreateAlgorithm` API.
- e. Choose **Create algorithm**.

[Create an Algorithm Resource \(API\)](#)

To create an algorithm resource by using the Amazon SageMaker API, call the `CreateAlgorithm` API.

[Create a Model Package Resource](#)

To create a model package resource that you can use to create deployable models in Amazon SageMaker and publish on AWS Marketplace specify the following information:

- The Docker container that contains the inference code, or the algorithm resource that was used to train the model.
- The location of the model artifacts. Model artifacts can either be packaged in the same Docker container as the inference code or stored in Amazon S3.
- The instance types that your model package supports for both real-time inference and batch transform jobs.
- Validation profiles, which are batch transform jobs that Amazon SageMaker runs to test your model package's inference code.

Before listing model packages on AWS Marketplace, you must validate them. This ensures that buyers and sellers can be confident that products work in Amazon SageMaker. You can list products on AWS Marketplace only if validation succeeds.

The validation procedure uses your validation profile and sample data to run the following validations tasks:

1. Create a model in your account using the model package's inference image and the optional model artifacts that are stored in Amazon S3.

Note

A model package is specific to the region in which you create it. The S3 bucket where the model artifacts are stored must be in the same region where your created the model package.

2. Create a transform job in your account using the model to verify that your inference image works with Amazon SageMaker.
3. Create a validation profile.

Note

In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the Amazon SageMaker console, see the **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the Amazon SageMaker console. After fixing issues, recreate the algorithm. When the status of the algorithm is **COMPLETED**, find it in the Amazon SageMaker console and start the listing process

Note

To publish your model package on AWS Marketplace, at least one validation profile is required.

You can create an model package either by using the Amazon SageMaker console or by using the Amazon SageMaker API.

Topics

- [Create a Model Package Resource \(Console\) \(p. 510\)](#)
- [Create a Model Package Resource \(API\) \(p. 511\)](#)

Create a Model Package Resource (Console)

To create a model package in the Amazon SageMaker console:

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Model packages**, then choose **Create model package**.
3. On the **Inference specifications** page, provide the following information:
 - a. For **Model package name**, type a name for your model package. The model package name must be unique in your account and in the AWS region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
 - b. Type a description for your model package. This description appears in the Amazon SageMaker console and in the AWS Marketplace.
 - c. For **Inference specification options**, choose **Provide the location of the inference image and model artifacts** to create a model package by using an inference container and model artifacts. Choose **Provide the algorithm used for training and its model artifacts** to create a model package from an algorithm resource that you created or subscribe to from AWS Marketplace.
 - d. If you chose **Provide the location of the inference image and model artifacts** for **Inference specification options**, provide the following information for **Container definition** and **Supported resources**:
 - i. For **Location of inference image**, type the path to the image that contains your inference code. The image must be stored as a Docker container in Amazon ECR.
 - ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.
 - iii. For **Container DNS host name**, type the name of the DNS host to use for your container.

- iv. For **Supported instance types for real-time inference**, choose the instance types that your model package supports for real-time inference from Amazon SageMaker hosted endpoints.
 - v. For **Supported instance types for batch transform jobs**, choose the instance types that your model package supports for batch transform jobs.
 - vi. **Supported content types**, type the content types that your model package expects for inference requests.
 - vii. For **Supported response MIME types**, type the MIME types that your model package uses to provide inferences.
- e. If you chose **Provide the algorithm used for training and its model artifacts for Inference specification options**, provide the following information:
 - i. For **Algorithm ARN**, type the Amazon Resource Name (ARN) of the algorithm resource to use to create the model package.
 - ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.
- f. Choose **Next**.
4. On the **Validation and scanning** page, provide the following information:
 - a. For **Publish this model package on AWS Marketplace**, choose **Yes** to publish the model package on AWS Marketplace.
 - b. For **Validate this model package**, choose **Yes** if you want Amazon SageMaker to run batch transform jobs that you specify to test the inference code of your model package.
- Note**
To publish your model package on AWS Marketplace, your model package must be validated.
- c. For **IAM role**, choose an IAM role that has the required permissions to run batch transform jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [Amazon SageMaker Roles \(p. 776\)](#).
 - d. For **Validation profile**, specify the following:
 - A name for the validation profile.
 - A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition` input parameter of the [CreateAlgorithm API](#).
5. Choose **Create model package**.

[Create a Model Package Resource \(API\)](#)

To create a model package by using the Amazon SageMaker API, call the [CreateModelPackage API](#).

Use Algorithm and Model Package Resources

You can create algorithms and model packages as resources in your Amazon SageMaker account, and you can find and subscribe to algorithms and model packages on AWS Marketplace.

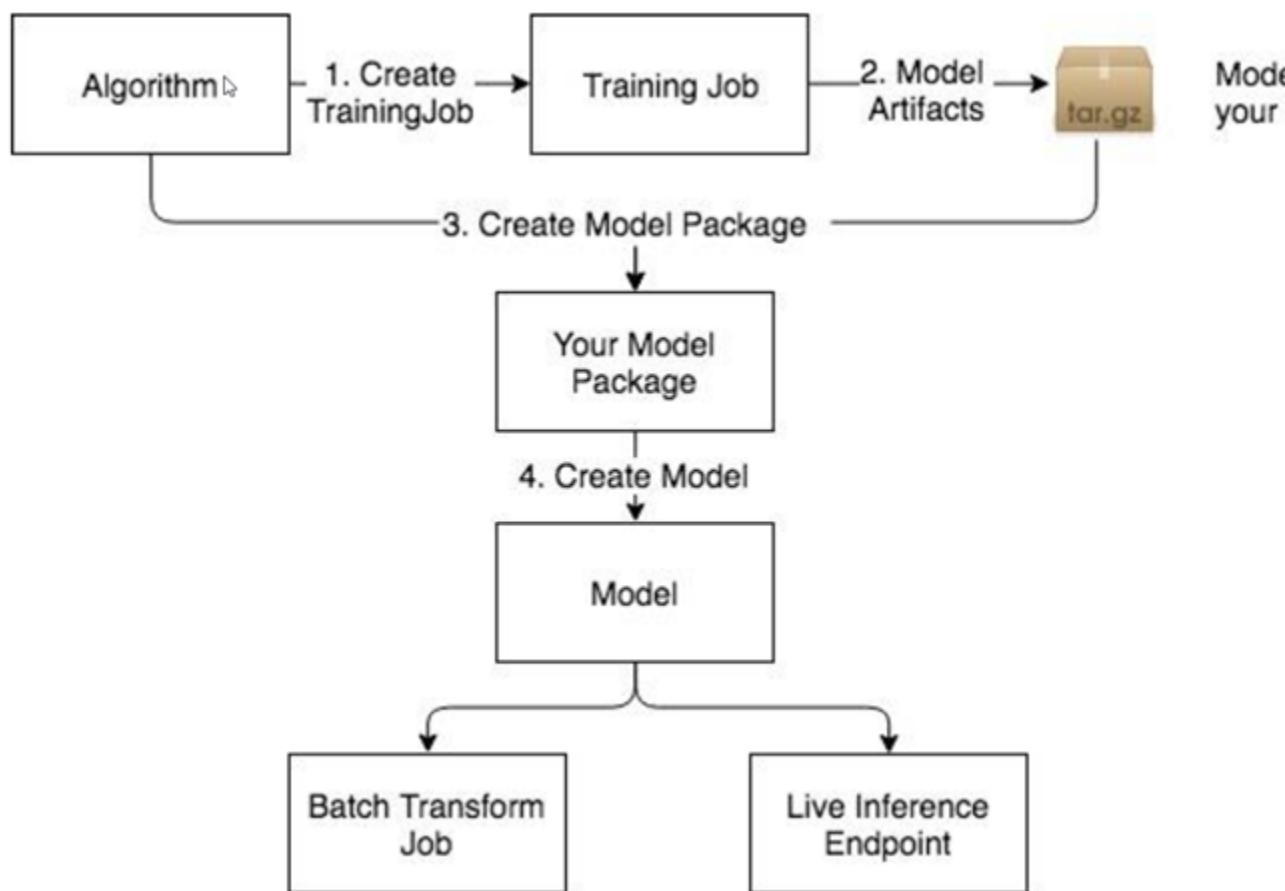
Use algorithms to:

- Run training jobs. For information, see [Use an Algorithm to Run a Training Job \(p. 513\)](#).
- Run hyperparameter tuning jobs. For information, see [Use an Algorithm to Run a Hyperparameter Tuning Job \(p. 514\)](#).

- Create model packages. After you use an algorithm resource to run a training job or a hyperparameter tuning job, you can use the model artifacts that these jobs output along with the algorithm to create a model package. For information, see [Create a Model Package Resource \(p. 509\)](#).

Note

If you subscribe to an algorithm on AWS Marketplace, you must create a model package before you can use it to get inferences by creating hosted endpoint or running a batch transform job.



Use model packages to:

- Create models that you can use to get real-time inference or run batch transform jobs. For information, see [Use a Model Package to Create a Model \(p. 517\)](#).
- Create hosted endpoints to get real-time inference. For information, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#).
- Create batch transform jobs. For information, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Topics

- [Use an Algorithm to Run a Training Job \(p. 513\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(p. 514\)](#)
- [Use a Model Package to Create a Model \(p. 517\)](#)

Use an Algorithm to Run a Training Job

You can create use an algorithm resource to create a training job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics

- [Use an Algorithm to Run a Training Job \(Console\) \(p. 513\)](#)
- [Use an Algorithm to Run a Training Job \(API\) \(p. 514\)](#)
- [Use an Algorithm to Run a Training Job \(Amazon SageMaker Python SDK\) \(p. 514\)](#)

Use an Algorithm to Run a Training Job (Console)

To use an algorithm to run a training job (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Algorithms**.
3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create training job**.

The algorithm you chose will automatically be selected.

5. On the **Create training job** page, provide the following information:
 - a. For **Job name**, type a name for the training job.
 - b. For **IAM role**, choose an IAM role that has the required permissions to run training jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [Amazon SageMaker Roles \(p. 776\)](#).
 - c. For **Resource configuration**, provide the following information:
 - i. For **Instance type**, choose the instance type to use for training.
 - ii. For **Instance count**, type the number of ML instances to use for the training job.
 - iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision. ML storage volumes store model artifacts and incremental states.
 - iv. For **Encryption key**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instance, specify the key.
 - v. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want the training job to run.
 - d. For **VPC**, choose a Amazon VPC that you want to allow your training container to access. For more information, see [Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 808\)](#).
 - e. For **Hyperparameters**, specify the values of the hyperparameters to use for the training job.
 - f. For **Input data configuration**, specify the following values for each channel of input data to use for the training job. You can see what channels the algorithm you're using for training supports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.
 - i. For **Channel name**, type the name of the input channel.
 - ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.
 - iii. For **Compression type**, choose the data compression type to use, if any.

- iv. For **Record wrapper**, choose RecordIO if the algorithm expects data in the RecordIO format.
- v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see [S3DataSource](#).
- vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **PipeTo** stream data directly from Amazon S3 to the container.
- vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.
- g. For **Output** location, specify the following values:
 - i. For **S3 output path**, choose the S3 location where the training job stores output, such as model artifacts.

Note
You use the model artifacts stored at this location to create a model or model package from this training job.
 - ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.
 - h. For **Tags**, specify one or more tags to manage the training job. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
 - i. Choose **Create training job** to run the training job.

Use an Algorithm to Run a Training Job (API)

To use an algorithm to run a training job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) as the `AlgorithmName` field of the `AlgorithmSpecification` object that you pass to [CreateTrainingJob](#). For information about training models in Amazon SageMaker, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a training job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then call the `fit` method of the estimator. For example:

```
from sagemaker import AlgorithmEstimator
data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    algorithm_arn='arn:aws:sagemaker:us-east-2:012345678901:algorithm/my-algorithm',
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

Use an Algorithm to Run a Hyperparameter Tuning Job

A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the

hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. For more information, see [Perform Automatic Model Tuning \(p. 575\)](#).

You can create use an algorithm resource to create a hyperparameter tuning job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics

- [Use an Algorithm to Run a Hyperparameter Tuning Job \(Console\) \(p. 515\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(API\) \(p. 517\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(Amazon SageMaker Python SDK\) \(p. 517\)](#)

[Use an Algorithm to Run a Hyperparameter Tuning Job \(Console\)](#)

To use an algorithm to run a hyperparameter tuning job (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
 2. Choose **Algorithms**.
 3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **AWS Marketplace subscriptions** tab.
 4. Choose **Create hyperparameter tuning job**.
- The algorithm you chose will automatically be selected.
5. On the **Create hyperparameter tuning job** page, provide the following information:
 - a. For **Warm start**, choose **Enable warm start** to use the information from previous hyperparameter tuning jobs as a starting point for this hyperparameter tuning job. For more information, see [Run a Warm Start Hyperparameter Tuning Job \(p. 590\)](#).
 - i. Choose **Identical data and algorithm** if your input data is the same as the input data for the parent jobs of this hyperparameter tuning job, or choose **Transfer learning** to use additional or different input data for this hyperparameter tuning job.
 - ii. For **Parent hyperparameter tuning job(s)**, choose up to 5 hyperparameter tuning jobs to use as parents to this hyperparameter tuning job.
 - b. For **Hyperparameter tuning job name**, type a name for the tuning job.
 - c. For **IAM role**, choose an IAM role that has the required permissions to run hyperparameter tuning jobs in Amazon SageMaker, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [Amazon SageMaker Roles \(p. 776\)](#).
 - d. For **VPC**, choose a Amazon VPC that you want to allow the training jobs that the tuning job launches to access. For more information, see [Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 808\)](#).
 - e. Choose **Next**.
 - f. For **Objective metric**, choose the metric that the hyperparameter tuning job uses to determine the best combination of hyperparameters, and choose whether to minimize or maximize this metric. For more information, see [View the Best Training Job \(p. 587\)](#).
 - g. For **Hyperparameter configuration**, choose ranges for the tunable hyperparameters that you want the tuning job to search, and set static values for hyperparameters that you want to remain constant in all training jobs that the hyperparameter tuning job launches. For more information, see [Define Hyperparameter Ranges \(p. 578\)](#).
 - h. Choose **Next**.
 - i. For **Input data configuration**, specify the following values for each channel of input data to use for the hyperparameter tuning job. You can see what channels the algorithm you're using for hyperparameter tuning supports, and the content type, supported compression type, and

supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.

- i. For **Channel name**, type the name of the input channel.
 - ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.
 - iii. For **Compression type**, choose the data compression type to use, if any.
 - iv. For **Record wrapper**, choose RecordIO if the algorithm expects data in the RecordIO format.
 - v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see [S3DataSource](#).
 - vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe** to stream data directly from Amazon S3 to the container.
 - vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.
- j. For **Output** location, specify the following values:
- i. For **S3 output path**, choose the S3 location where the training jobs that this hyperparameter tuning job launches store output, such as model artifacts.
- Note**
You use the model artifacts stored at this location to create a model or model package from this hyperparameter tuning job.
- ii. For **Encryption key**, if you want Amazon SageMaker to use a AWS KMS key to encrypt output data at rest in the S3 location.
- k. For **Resource configuration**, provide the following information:
- i. For **Instance type**, choose the instance type to use for each training job that the hyperparameter tuning job launches.
 - ii. For **Instance count**, type the number of ML instances to use for each training job that the hyperparameter tuning job launches.
 - iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision each training job that the hyperparameter tuning job launches. ML storage volumes store model artifacts and incremental states.
 - iv. For **Encryption key**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the training instances, specify the key.
- l. For **Resource limits**, provide the following information:
- i. For **Maximum training jobs**, specify the maximum number of training jobs that you want the hyperparameter tuning job to launch. A hyperparameter tuning job can launch a maximum of 500 training jobs.
 - ii. For **Maximum parallel training jobs**, specify the maximum number of concurrent training jobs that the hyperparameter tuning job can launch. A hyperparameter tuning job can launch a maximum of 10 concurrent training jobs.
 - iii. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want each training job that the hyperparameter tuning job launches to run.
 - m. For **Tags**, specify one or more tags to manage the hyperparameter tuning job. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
 - n. Choose **Create jobs** to run the hyperparameter tuning job.

Use an Algorithm to Run a Hyperparameter Tuning Job (API)

To use an algorithm to run a hyperparameter tuning job by using the Amazon SageMaker API, specify either the name or the Amazon Resource Name (ARN) of the algorithm as the `AlgorithmName` field of the `AlgorithmSpecification` object that you pass to `CreateHyperParameterTuningJob`. For information about hyperparameter tuning in Amazon SageMaker, see [Perform Automatic Model Tuning \(p. 575\)](#).

Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on AWS Marketplace to create a hyperparameter tuning job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then initialize a `HyperparameterTuner` object with the `AlgorithmEstimator` you created as the value of the `estimator` argument. Finally, call the `fit` method of the `AlgorithmEstimator`. For example:

```
from sagemaker import AlgorithmEstimator
from sagemaker.tuner import HyperparameterTuner

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    algorithm_arn='arn:aws:sagemaker:us-east-2:764419575721:algorithm/scikit-decision-trees-1542410022',
    role='SageMakerRole',
    train_instance_count=1,
    train_instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.set_hyperparameters(max_leaf_nodes=10)
tuner = HyperparameterTuner(estimator=algo, base_tuning_job_name='some-name',
                             objective_metric_name='validation:accuracy',
                             hyperparameter_ranges=hyperparameter_ranges,
                             max_jobs=2, max_parallel_jobs=2)

tuner.fit({'training': train_input}, include_cls_metadata=False)
tuner.wait()
```

Use a Model Package to Create a Model

Use a model package to create a deployable model that you can use to get real-time inferences by creating a hosted endpoint or to run batch transform jobs. You can create a deployable model from a model package by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the Amazon SageMaker Python SDK.

Topics

- [Use a Model Package to Create a Model \(Console\) \(p. 517\)](#)
- [Use a Model Package to Create a Model \(API\) \(p. 518\)](#)
- [Use a Model Package to Create a Model \(Amazon SageMaker Python SDK\) \(p. 518\)](#)

Use a Model Package to Create a Model (Console)

To create a deployable model from a model package (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.

2. Choose **Model packages**.
3. Choose a model package that you created from the list on the **My model packages** tab or choose a model package that you subscribed to on the **AWS Marketplace subscriptions** tab.
4. Choose **Create model**.
5. For **Model name**, type a name for the model.
6. For **IAM role**, choose an IAM role that has the required permissions to call other services on your behalf, or choose **Create a new role** to allow Amazon SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [Amazon SageMaker Roles \(p. 776\)](#).
7. For **VPC**, choose a Amazon VPC that you want to allow the model to access. For more information, see [Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 811\)](#).
8. Leave the default values for **Container input options** and **Choose model package**.
9. For environment variables, provide the names and values of environment variables you want to pass to the model container.
10. For **Tags**, specify one or more tags to manage the model. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
11. Choose **Create model**.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#). For information about batch transform jobs, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Use a Model Package to Create a Model (API)

To use a model package to create a deployable model by using the Amazon SageMaker API, specify the name or the Amazon Resource Name (ARN) of the model package as the `ModelPackageName` field of the `ContainerDefinition` object that you pass to the `CreateModel` API.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#). For information about batch transform jobs, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Use a Model Package to Create a Model (Amazon SageMaker Python SDK)

To use a model package to create a deployable model by using the Amazon SageMaker Python SDK, initialize a `ModelPackage` object, and pass the Amazon Resource Name (ARN) of the model package as the `model_package_arn` argument. For example:

```
from sagemaker import ModelPackage
model = ModelPackage(role='SageMakerRole',
                      model_package_arn='training-job-scikit-decision-trees-1542660466-6f92',
                      sagemaker_session=sagemaker_session)
```

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in Amazon SageMaker, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#). For information about batch transform jobs, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Buy and Sell Amazon SageMaker Algorithms and Models in AWS Marketplace

Amazon SageMaker integrates with AWS Marketplace, enabling developers to charge other Amazon SageMaker users for the use of their algorithms and model packages. AWS Marketplace is a curated digital catalog that makes it easy for customers to find, buy, deploy, and manage third-party software and services that customers need to build solutions and run their businesses. AWS Marketplace includes thousands of software listings in popular categories, such as security, networking, storage, machine learning, business intelligence, database, and DevOps. It simplifies software licensing and procurement with flexible pricing options and multiple deployment methods. For information, see [AWS Marketplace Documentation](#).

Topics

- [Amazon SageMaker Algorithms \(p. 519\)](#)
- [Amazon SageMaker Model Packages \(p. 519\)](#)
- [Sell Amazon SageMaker Algorithms and Model Packages \(p. 519\)](#)
- [Find and Subscribe to Algorithms and Model Packages on AWS Marketplace \(p. 522\)](#)
- [Use Algorithm and Model Package Resources \(p. 511\)](#)

Amazon SageMaker Algorithms

An algorithm enables you to perform end-to-end machine learning. It has two logical components: training and inference. Buyers can use the training component to create training jobs in Amazon SageMaker and build a machine learning model. Amazon SageMaker saves the model artifacts generated by the algorithm during training to an Amazon S3 bucket. For more information, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

Buyers use the inference component with the model artifacts generated during a training job to create a deployable model in their Amazon SageMaker account. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see [Deploy a Model in Amazon SageMaker \(p. 8\)](#).

Amazon SageMaker Model Packages

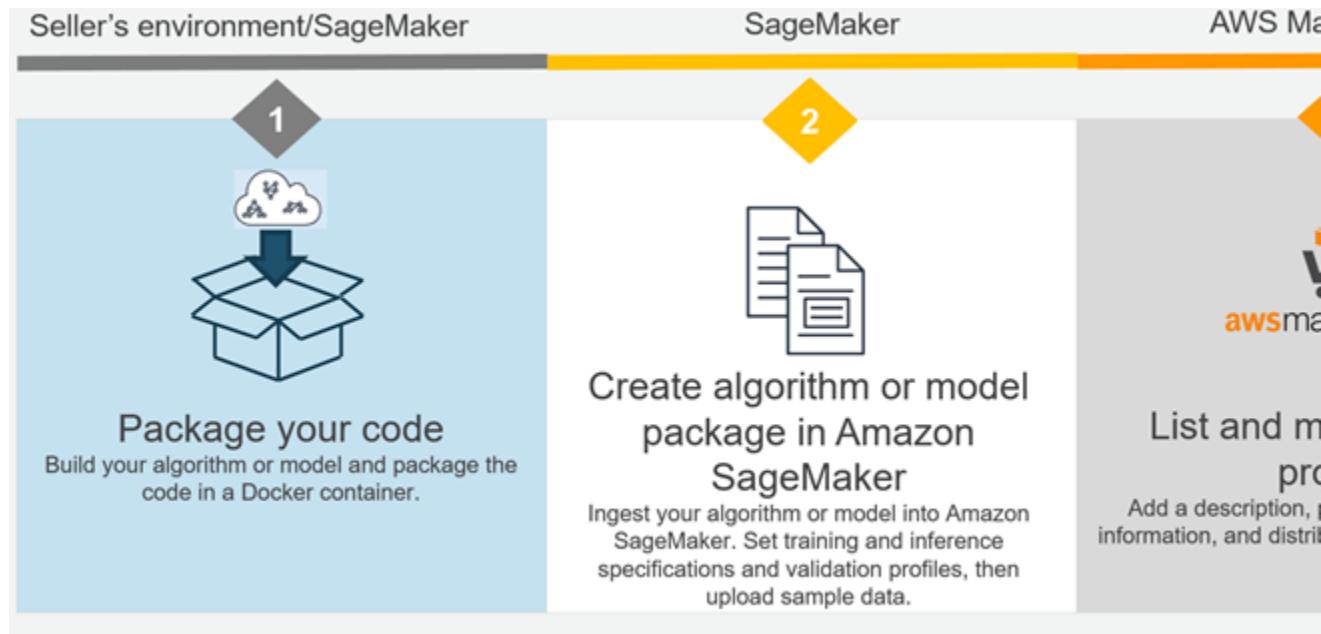
Buyers use a model package to build a deployable model in Amazon SageMaker. They can use the deployable model for real-time inference by using Amazon SageMaker hosting services. Or, they can get inferences for an entire dataset by running batch transform jobs. For more information, see [Deploy a Model in Amazon SageMaker \(p. 8\)](#). As a seller, you can build your model artifacts by training in Amazon SageMaker, or you can use your own model artifacts from a model that you trained outside of Amazon SageMaker. You can charge buyers for inference.

Sell Amazon SageMaker Algorithms and Model Packages

Selling Amazon SageMaker algorithms and model packages is a three-step process:

1. Develop your algorithm or model, and package it in a Docker container. For information, see [Develop Algorithms and Models in Amazon SageMaker \(p. 520\)](#).

2. Create an algorithm or model package resource in Amazon SageMaker. For information, see [Create Algorithm and Model Package Resources \(p. 506\)](#).
3. Register as a seller on AWS Marketplace and list your algorithm or model package on AWS Marketplace. For information about registering as a seller, see [Getting Started as a Seller](#) in the *User Guide for AWS Marketplace Providers*. For information about listing and monetizing your algorithms and model packages, see [Listing Algorithms and Model Packages in AWS Marketplace for Machine Learning](#) in the *User Guide for AWS Marketplace Providers*.



Topics

- [Develop Algorithms and Models in Amazon SageMaker \(p. 520\)](#)
- [Create Algorithm and Model Package Resources \(p. 506\)](#)
- [List Your Algorithm or Model Package on AWS Marketplace \(p. 521\)](#)

Develop Algorithms and Models in Amazon SageMaker

Before you can create algorithm and model package resources to use in Amazon SageMaker or list on AWS Marketplace, you have to develop them and package them in Docker containers.

Note

When algorithms and model packages are created for listing on AWS Marketplace, Amazon SageMaker scans the containers for security vulnerabilities on supported operating systems. Only the following operating system versions are supported:

- Debian: 6.0, 7, 8, 9, 10
- Ubuntu: 12.04, 12.10, 13.04, 14.04, 14.10, 15.04, 15.10, 16.04, 16.10, 17.04, 17.10, 18.04, 18.10
- CentOS: 5, 6, 7
- Oracle Linux: 5, 6, 7
- Alpine: 3.3, 3.4, 3.5
- Amazon Linux

Topics

- [Develop Algorithms in Amazon SageMaker \(p. 521\)](#)
- [Develop Models in Amazon SageMaker \(p. 521\)](#)

Develop Algorithms in Amazon SageMaker

An algorithm should be packaged as a docker container and stored in Amazon ECR to use it in Amazon SageMaker. The Docker container contains the training code used to run training jobs and, optionally, the inference code used to get inferences from models trained by using the algorithm.

For information about developing algorithms in Amazon SageMaker and packaging them as containers, see [Use Your Own Algorithms or Models with Amazon SageMaker \(p. 478\)](#). For a complete example of how to create an algorithm container, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/scikit Bring Your Own/scikit Bring Your Own.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the **Advanced Functionality** section, and is named `scikit Bring Your Own.ipynb`. For information about using the sample notebooks in a notebook instance, see [Use Example Notebooks \(p. 260\)](#).

Always thoroughly test your algorithms before you create algorithm resources to publish on AWS Marketplace.

Note

When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

Develop Models in Amazon SageMaker

A deployable model in Amazon SageMaker consists of inference code, model artifacts, an IAM role that is used to access resources, and other information required to deploy the model in Amazon SageMaker. Model artifacts are the results of training a model by using a machine learning algorithm. The inference code must be packaged in a Docker container and stored in Amazon ECR. You can either package the model artifacts in the same container as the inference code, or store them in Amazon S3.

You create a model by running a training job in Amazon SageMaker, or by training a machine learning algorithm outside of Amazon SageMaker. If you run a training job in Amazon SageMaker, the resulting model artifacts are available in the `ModelArtifacts` field in the response to a call to the [DescribeTrainingJob](#) operation. For information about how to develop an Amazon SageMaker model container, see [Use Your Own Inference Code \(p. 501\)](#). For a complete example of how to create a model container from a model trained outside of Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/advanced_functionality/xgboost Bring Your Own Model/xgboost Bring Your Own Model.ipynb. You can also find the sample notebook in an Amazon SageMaker notebook instance. The notebook is in the **Advanced Functionality** section, and is named `xgboost Bring Your Own Model.ipynb`. For information about using the sample notebooks in a notebook instance, see [Use Example Notebooks \(p. 260\)](#).

Always thoroughly test your models before you create model packages to publish on AWS Marketplace.

Note

When a buyer subscribes to your containerized product, the Docker containers run in an isolated (internet-free) environment. When you create your containers, do not rely on making outgoing calls over the internet. Calls to AWS services are also not allowed.

List Your Algorithm or Model Package on AWS Marketplace

After creating and validating your algorithm or model in Amazon SageMaker, list your product on AWS Marketplace. The listing process makes your products available in the AWS Marketplace and the Amazon SageMaker console.

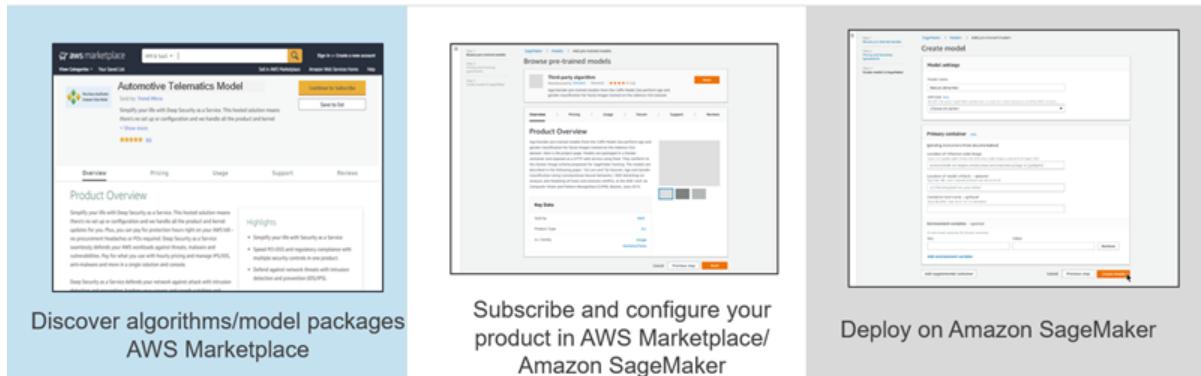
To list products on AWS Marketplace, you must be a registered seller. To register, use the self-registration process from the AWS Marketplace Management Portal (AMMP). For information, see [Getting Started as a Seller](#) in the *User Guide for AWS Marketplace Providers*. When you start the product listing process from the Amazon SageMaker console, we check your seller registration status. If you have not registered, we direct you to do so.

To start the listing process, do one of the following:

- From the Amazon SageMaker console, choose the product, choose **Actions**, and choose **Publish new ML Marketplace listing**. This carries over your product reference, the Amazon Resource Name (ARN), and directs you to the AMMP to create the listing.
- Go to [ML listing process](#), manually enter the Amazon Resource Name (ARN), and start your product listing. This process carries over the product metadata that you entered when creating the product in Amazon SageMaker. For an algorithm listing, the information includes the supported instance types and hyperparameters. In addition, you can enter a product description, promotional information, and support information as you would with other AWS Marketplace products.

Find and Subscribe to Algorithms and Model Packages on AWS Marketplace

With AWS Marketplace, you can browse and search for hundreds of machine learning algorithms and models in a broad range of categories, such as computer vision, natural language processing, speech recognition, text, data, voice, image, video analysis, fraud detection, predictive analysis, and more.



Discover algorithms/model packages AWS Marketplace

Subscribe and configure your product in AWS Marketplace/Amazon SageMaker

Deploy on Amazon SageMaker

To find algorithms on AWS Marketplace

- Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
- Choose **Algorithms**, then choose **Find algorithms**.

This takes you to the AWS Marketplace algorithms page. For information about finding and subscribing to algorithms on AWS Marketplace, see [Machine Learning Products](#) in the *AWS Marketplace User Guide for AWS Consumers*.

To find model packages on AWS Marketplace

- Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
- Choose **Model packages**, then choose **Find model packages**.

This takes you to the AWS Marketplace model packages page. For information about finding and subscribing to model packages on AWS Marketplace, see [Machine Learning Products](#) in the *AWS Marketplace User Guide for AWS Consumers*.

Use Algorithms and Model Packages

For information about using algorithms and model packages that you subscribe to in Amazon SageMaker, see [Use Algorithm and Model Package Resources \(p. 511\)](#).

Note

When you create a training job, inference endpoint, and batch transform job from an algorithm or model package that you subscribe to on AWS Marketplace, the training and inference containers do not have access to the internet. Because the containers do not have access to the internet, the seller of the algorithm or model package does not have access to your data.

Train Models

For an overview on training models with Amazon SageMaker, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

Amazon SageMaker provides features to monitor and manage the training and validation of machine learning models. For guidance on metrics available, incremental training, automatic model tuning, and the use of augmented manifest files to label training data, see the following topics.

- For guidance on debugging the training of machine learning models, see [Amazon SageMaker Debugger \(p. 536\)](#).
- For guidance on metrics used to monitor and train models, see [Monitor and Analyze Training Jobs Using Metrics \(p. 606\)](#).
- For guidance on incremental training in Amazon SageMaker, see [Incremental Training in Amazon SageMaker \(p. 613\)](#).
- For guidance on using managed spot training in Amazon SageMaker, see [Managed Spot Training in Amazon SageMaker \(p. 618\)](#).
- For guidance on using training checkpoints in Amazon SageMaker, see [Use Checkpoints in Amazon SageMaker \(p. 619\)](#).
- For guidance on automatic model tuning, also known as hyperparameter tuning, see [Perform Automatic Model Tuning \(p. 575\)](#).
- For guidance on using an augmented manifest file to label training data, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#).

Topics

- [Manage Machine Learning with Amazon SageMaker Experiments \(p. 524\)](#)
- [Amazon SageMaker Debugger \(p. 536\)](#)
- [Perform Automatic Model Tuning \(p. 575\)](#)
- [Tune Multiple Algorithms to Find the Best Model \(p. 595\)](#)
- [Use Reinforcement Learning with Amazon SageMaker \(p. 598\)](#)
- [Train a Deep Graph Network \(p. 604\)](#)
- [Monitor and Analyze Training Jobs Using Metrics \(p. 606\)](#)
- [Incremental Training in Amazon SageMaker \(p. 613\)](#)
- [Managed Spot Training in Amazon SageMaker \(p. 618\)](#)
- [Use Checkpoints in Amazon SageMaker \(p. 619\)](#)
- [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 619\)](#)

Manage Machine Learning with Amazon SageMaker Experiments

Amazon SageMaker Experiments is a capability of Amazon SageMaker that lets you organize, track, compare, and evaluate your machine learning experiments.

Machine learning is an iterative process. You need to experiment with multiple combinations of data, algorithm and parameters, all the while observing the impact of incremental changes on model accuracy. Over time this iterative experimentation can result in thousands of model training runs and model versions. This makes it hard to track the best performing models and their input configurations. It's

also difficult to compare active experiments with past experiments to identify opportunities for further incremental improvements.

Amazon SageMaker Experiments automatically tracks the inputs, parameters, configurations, and results of your iterations as *trials*. You can assign, group, and organize these trials into *experiments*. Experiments is integrated with Amazon SageMaker Studio providing a visual interface to browse your active and past experiments, compare trials on key performance metrics, and identify the best performing models.

Amazon SageMaker Experiments comes with the Amazon SageMaker Python SDK which makes the search and analytics capabilities easily accessible in Amazon SageMaker Notebooks. Because Experiments enables tracking of all the steps and artifacts that went into creating a model, you can quickly revisit the origins of a model when you are troubleshooting issues in production, or auditing your models for compliance verifications.

Topics

- [Organize Experiments \(p. 525\)](#)
- [Track Experiments \(p. 525\)](#)
- [Compare and Evaluate Experiments \(p. 526\)](#)
- [Amazon SageMaker Autopilot \(p. 526\)](#)
- [Track and Evaluate a Model Training Experiment \(p. 526\)](#)
- [Search \(p. 531\)](#)

Organize Experiments

Amazon SageMaker Experiments offers a structured organization scheme to help users group and organize their machine learning iterations. The top level entity, an Amazon SageMaker *experiment*, is a collection of *trials* that are observed, compared, and evaluated as a group. A trial is a set of steps called *trial components*. Each trial component can include a combination of inputs such as datasets, algorithms, and parameters, and produce specific outputs such as models, metrics, datasets, and checkpoints. Examples of trial components are data pre-processing jobs, training jobs, and batch transform jobs.

The goal of an experiment is to determine the trial that produces the best model. Multiple trials are performed, each one isolating and measuring the impact of a change to one or more inputs, while keeping the remaining inputs constant. By analyzing the trials, you can determine which features have the most effect on the model.

Track Experiments

Amazon SageMaker Experiments enables tracking of experiments.

Automated Tracking

Amazon SageMaker Experiments automatically tracks Amazon SageMaker Autopilot jobs as experiments with their underlying training jobs tracked as trials. Experiments also automatically tracks Amazon SageMaker independently executed training, batch transform, and processing jobs as trial components, whether assigned to a trial or left unassigned. Unassigned trial components can be associated with a trial at a later time. All experiment artifacts including datasets, algorithms, hyperparameters, and model metrics are tracked and recorded. This data allows customers to trace the complete lineage of a model which helps with model governance, auditing, and compliance verifications.

Manual Tracking

Amazon SageMaker Experiments provides tracking APIs in the Amazon SageMaker Python SDK for recording and tracking machine learning workflows running locally on Amazon SageMaker Studio

notebooks, including classic Amazon SageMaker notebooks. These experiments must be part of an Amazon SageMaker training, batch transform, or processing job.

Compare and Evaluate Experiments

Amazon SageMaker Experiments is integrated with Amazon SageMaker Studio. When you use Studio, Experiments automatically tracks your experiments and trials, and presents visualizations of the tracked data and an interface to search the data.

Amazon SageMaker Experiments automatically organizes, ranks, and sorts trials based on a chosen metric using the concept of a trial leaderboard. Amazon SageMaker Studio produces real-time data visualizations, such as metric charts and graphs, to quickly compare and identify the best performing models. These are updated in real-time as the experiment progresses.

Amazon SageMaker Autopilot

Amazon SageMaker Experiments is integrated with Amazon SageMaker Autopilot. When you perform an automated machine learning job using Autopilot, Experiments creates an experiment for the job, and trials for each of the different combinations of trial components, parameters, and artifacts that Autopilot tries for the job. You can visually drill into all trials and components using Amazon SageMaker Studio.

Track and Evaluate a Model Training Experiment

This tutorial demonstrates how to visually track, compare, and evaluate a model training experiment using Amazon SageMaker Studio. The basis of the tutorial is the MNIST Handwritten Digits Classification Experiment (MNIST) example notebook.

It is intended that this topic be viewed alongside Studio with the MNIST notebook open. As you run through the cells, the sections highlight the relevant code and show you how to observe the results in Studio. Some of the code snippets have been edited for brevity.

Prerequisites

- A local copy of the [MNIST](#) example notebook and the companion [mnist.py](#) file. Both are available from the [awslabs/amazon-sagemaker-examples](#) repository. To download the files, select each link, right-click on the **Raw** button, and then choose **Save as**.
- An AWS SSO or IAM account to sign-on to Amazon SageMaker Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

Topics

- [Open the Notebook in Studio \(p. 526\)](#)
- [Set Up Amazon SageMaker Experiments \(p. 527\)](#)
- [Create and Track an Experiment \(p. 527\)](#)
- [Compare Trials and Analyze \(p. 529\)](#)
- [Deploy the Top Model \(p. 530\)](#)
- [Clean Up Resources \(p. 530\)](#)

Open the Notebook in Studio

To open the notebook

1. Sign-on to Studio.
2. In the left sidebar, select the **File Browser** icon.

3. At the top of the File Browser, select the **Up arrow** icon and then a **File Upload** dialog opens. Browse to and select your local versions of the *mnist-handwritten-digits-classification-experiment.ipynb* and *mnist.py* files, and then choose **Open**.
4. Double-click the uploaded notebook file to open the notebook in a new tab.

Set Up Amazon SageMaker Experiments

The Amazon SageMaker Experiments SDK is separate from the Amazon SageMaker Python SDK. In the following steps you install the Experiments SDK and import the relevant modules. For more information on the Experiments SDK, see [sagemaker-experiments](#).

To install the SDK and import modules

1. Install the Experiments SDK.

```
!{sys.executable} -m pip install sagemaker-experiments
```

2. Import the Amazon SageMaker and Experiments modules.

```
import sagemaker
from sagemaker import get_execution_role
from sagemaker.session import Session
from sagemaker.analytics import ExperimentAnalytics

from smexperiments.experiment import Experiment
from smexperiments.trial import Trial
from smexperiments.trial_component import TrialComponent
from smexperiments.tracker import Tracker
```

3. Run the remaining cells until you come to the last cell in the **Dataset** section.

Create and Track an Experiment

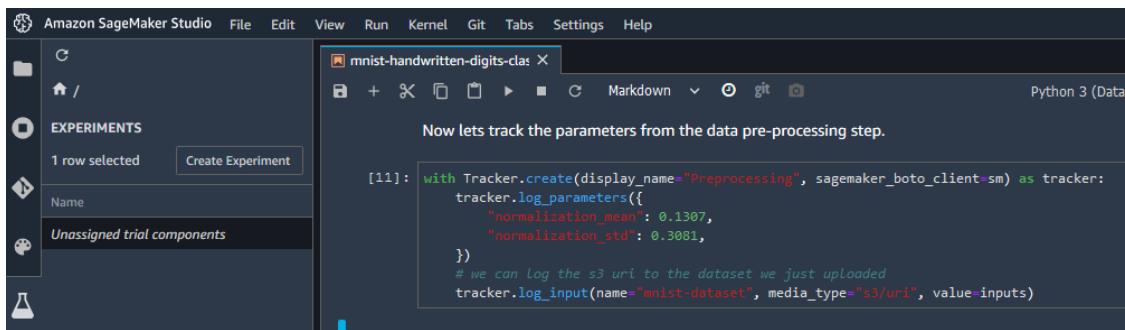
The following code creates and tracks an experiment. First, a **Tracker** is created and used to track the dataset transform job. Next, an experiment is created and then 5 trials are created inside a loop, one for each value of the `num_hidden_channel` hyperparameter you're testing.

1. In the left sidebar, select the **SageMaker Experiment List** icon to display the experiments browser.
2. In the notebook, create a **Tracker** as a preprocessing step to track the transform job for the dataset. The tracker logs the normalization parameters and the URI to the Amazon S3 bucket where the transformed dataset is uploaded.

```
with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
    tracker.log_parameters({
        "normalization_mean": 0.1307,
        "normalization_std": 0.3081,
    })
    tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
```

```
preprocessing_trial_component = tracker.trial_component
```

After the previous code runs, the experiments list contains an entry named **Unassigned trial components**. This is the preprocessing step just created. Your screen should look similar to the following:



The screenshot shows the Amazon SageMaker Studio interface. On the left, the 'EXPERIMENTS' section is open, showing a single row selected and a 'Create Experiment' button. In the main area, a code editor displays Python code for tracking parameters from data pre-processing. The code uses the Tracker.create() method to log parameters like normalization_mean and normalization_std, and logs an input named 'mnist-dataset' with media type 's3/url'.

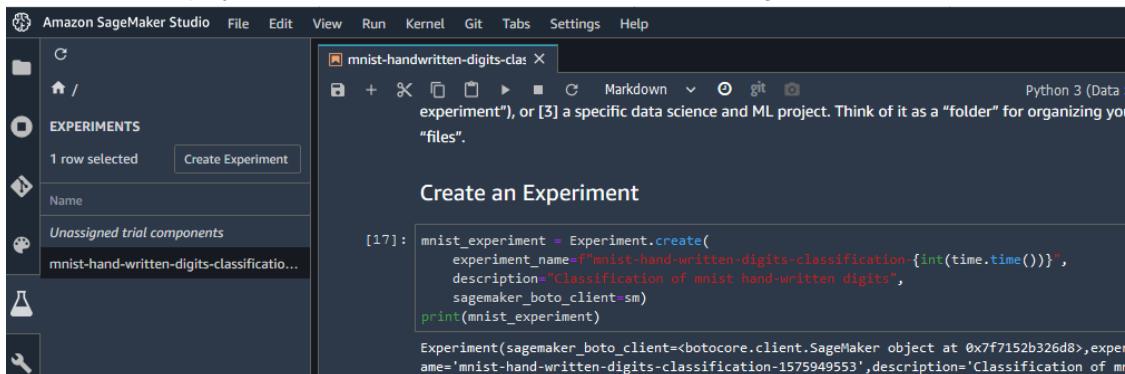
```
[11]: with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
    tracker.log_parameters({
        "normalization_mean": 0.1307,
        "normalization_std": 0.3081,
    })
    # we can log the s3 uri to the dataset we just uploaded
    tracker.log_input(name="mnist-dataset", media_type="s3/url", value=inputs)
```

3. Create and start tracking an experiment.

```
mnist_experiment = Experiment.create(
    experiment_name=f"mnist-hand-written-digits-classification-{int(time.time())}",
    description="Classification of mnist hand-written digits",
    sagemaker_boto_client=sm)
print(mnist_experiment)
```

```
Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f7152b326d8>,
           experiment_name='mnist-hand-written-digits-classification-1575947870',
           description='Classification of mnist hand-written digits',
           experiment_arn='arn:aws:sagemaker:us-east-2:acct-id:experiment/mnist-hand-
written-digits-classification-1575947870')
```

After the previous code runs, the experiments list contains an entry for the experiment. It might take a moment to display. Your screen should look similar to the following:



The screenshot shows the Amazon SageMaker Studio interface. The 'EXPERIMENTS' section now lists the created experiment. In the code editor, a new experiment is being created with the same details as the previous one. The code defines the experiment name, description, and boto client, then prints the resulting Experiment object.

```
[17]: mnist_experiment = Experiment.create(
    experiment_name=f"mnist-hand-written-digits-classification-{int(time.time())}",
    description="Classification of mnist hand-written digits",
    sagemaker_boto_client=sm)
print(mnist_experiment)

Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f7152b326d8>,exp
ame='mnist-hand-written-digits-classification-1575949553',description='Classification of mn
ist hand-written digits')
```

4. Double-click on your experiment to display a list of the trials in the experiment. Initially the list is empty.
5. Create trials for the experiment. Each trial trains a model using a different number for the `hidden_channels` hyperparameter. The preprocessing trial component is added to each trial for complete tracking (for example, for auditing purposes). The code also specifies definitions for the following metrics:
 - Train loss
 - Test loss
 - Test accuracy

The definitions tell Amazon SageMaker to capture those metrics from the algorithm's log output. The metrics are used later to evaluate and compare the models.

```

preprocessing_trial_component = tracker.trial_component

for i, num_hidden_channel in enumerate([2, 5, 10, 20, 32]):
    trial_name = f"cnn-training-job-{num_hidden_channel}-hidden-channels-{int(time.time())}"
    cnn_trial = Trial.create(
        trial_name=trial_name,
        experiment_name=mnist_experiment.experiment_name,
        sagemaker_boto_client=sm,
    )
    hidden_channel_trial_name_map[num_hidden_channel] = trial_name

    cnn_trial.add_trial_component(preprocessing_trial_component)

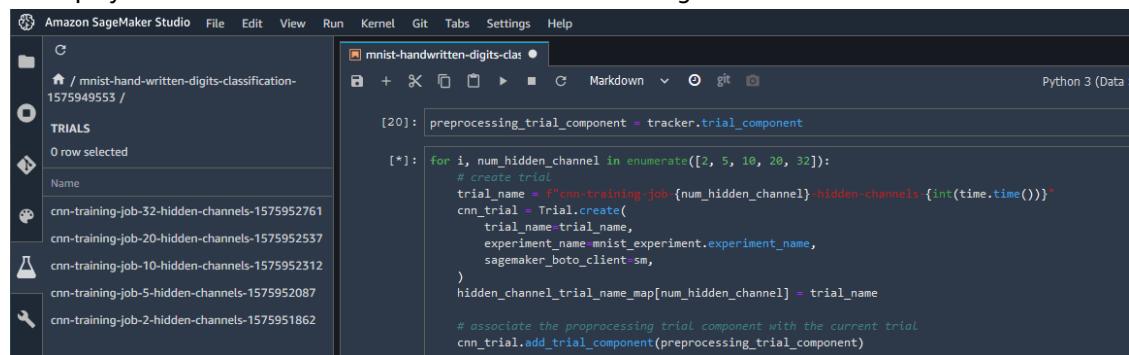
estimator = PyTorch(
    hyperparameters={
        'hidden_channels': num_hidden_channel,
        ...
    },
    metric_definitions=[
        {'Name':'train:loss', 'Regex':'Train Loss: (.*)%;"},
        {'Name':'test:loss', 'Regex':'Test Average loss: (.*)%;"},
        {'Name':'test:accuracy', 'Regex':'Test Accuracy: (.*)%%;"'}
    ],
    enable_sagemaker_metrics=True,
)

cnn_training_job_name = "cnn-training-job-{}".format(int(time.time()))

estimator.fit(
    inputs={'training': inputs},
    job_name=cnn_training_job_name,
    experiment_config={
        "TrialName": cnn_trial.trial_name,
        "TrialComponentDisplayName": "Training",
    },
)

```

The trial list automatically updates as each training job runs. It takes a few minutes for each trial to be displayed. Your screen should look similar to the following:



The screenshot shows the Amazon SageMaker Studio interface. On the left, there's a sidebar with icons for projects, trials, and a search bar. Under 'TRIALS', it says '0 row selected' and lists several training jobs: 'cnn-training-job-32-hidden-channels-1575952761', 'cnn-training-job-20-hidden-channels-1575952537', 'cnn-training-job-10-hidden-channels-1575952312', 'cnn-training-job-5-hidden-channels-1575952087', and 'cnn-training-job-2-hidden-channels-1575951862'. On the right, there's a code editor window titled 'mnist-handwritten-digits-clas'. The code in the editor is identical to the one shown above, used to create multiple trials for different hidden channel counts.

Compare Trials and Analyze

This section deviates from the notebook and show you how to compare and analyze the trained models using the Amazon SageMaker Studio UI.

To view a list of training jobs ordered by test:accuracy

1. Select all 5 trials, right-click the selection, and then choose **Open in trial component list**. A new tab opens that displays a list of the components for all trials. There's a preprocessing job and training job for each trial.
2. If the **TABLE PROPERTIES** pane isn't open, select the gear icon in the upper right corner to open it. Unselect the **Created on** and **Last modified** checkboxes.
3. Right-click the **test:accuracy** column header, choose **Data aggregation**, and then choose **Maximum**. Choose the **test:accuracy** column header to sort the list by decreasing maximum test accuracy. You can see that the models trained with the `hidden_channels` hyperparameter set to 2 and 20 give the highest test accuracy. Due to the randomness of model training and the closeness of the accuracies, your results might differ. Your screen should look similar to the following:

The screenshot shows the 'Trial Component List' page for an experiment named 'mnist-handwritten-digits-clas'. The table lists 10 trial components, all of which are completed. The columns are: Status, Experiment, Type, Trial, Trial component, test:loss, test:accuracy, and train:loss. The 'test:accuracy' column is sorted in descending order, with values ranging from 0.0965 to 0.1158. The 'test:accuracy' header has a dropdown arrow indicating it can be aggregated. The 'TABLE PROPERTIES' pane on the right shows global filters and column visibility settings for various columns like Status, Experiment, Type, Trial, Trial component, Created on, Last modified, Metrics, Parameters, and Tags. The 'Auto refresh enabled' toggle is turned on.

Status	Experiment	Type	Trial	Trial component	test:loss	test:accuracy	train:loss
Completed	mnist-hand-written-d...	Training job	cnn-training-job-2-hidden-channels-1575951862	Training	0.1158	97	0.157259
Completed	mnist-hand-written-d...	Training job	cnn-training-job-20-hidden-channels-1575952537	Training	0.0965	97	0.2272
Completed	mnist-hand-written-d...	Training job	cnn-training-job-5-hidden-channels-1575952087	Training	0.1157	96	0.197844
Completed	mnist-hand-written-d...	Training job	cnn-training-job-32-hidden-channels-1575952761	Training	0.1531	95	0.101494
Completed	mnist-hand-written-d...	Training job	cnn-training-job-10-hidden-channels-1575952312	Training	0.1679	95	0.21003
Completed	mnist-hand-written-d...		cnn-training-job-5-hidden-channels-1575952087	Preprocessing			
Completed	mnist-hand-written-d...		cnn-training-job-32-hidden-channels-1575952761	Preprocessing			
Completed	mnist-hand-written-d...		cnn-training-job-20-hidden-channels-1575952537	Preprocessing			
Completed	mnist-hand-written-d...		cnn-training-job-2-hidden-channels-1575951862	Preprocessing			
Completed	mnist-hand-written-d...		cnn-training-job-10-hidden-channels-1575952312	Preprocessing			

Deploy the Top Model

Now that you've determined the most accurate training job, deploy the associated model to an Amazon SageMaker endpoint.

To deploy the model

1. In the left sidebar, select the **SageMaker Endpoint List** icon to display the endpoints browser.
2. Right-click the top training job in the trial components list and choose **Deploy model**. A new tab opens that displays the **Deploy model** page.
3. Under **REQUIRED SETTINGS**, enter a name for the endpoint. Keep the default values of `m1 . m5 . xlarge` for **Instance type** and `1` for **Instance count**.
4. Choose **Deploy model**.

Clean Up Resources

To avoid incurring unnecessary charges, delete the resources you created after you're done with the tutorial. You can't delete the Experiment resources through the Amazon SageMaker Management Console or the Studio UI. The following code shows how to clean up these resources.

To delete the experiment, first you must delete all trials in the experiment. To delete a trial, first you must remove all trial components from the trial. To delete a trial component, first you must remove the component from all trials.

Note

Trial components can exist independent of trials and experiments. You might want keep them if you plan on further exploration. If so, comment out `tc.delete()`

```
def cleanup(experiment):
    for trial_summary in experiment.list_trials():
        trial = Trial.load(sagemaker_boto_client=sm, trial_name=trial_summary.trial_name)
        for trial_component_summary in trial.list_trial_components():
            tc = TrialComponent.load(
                sagemaker_boto_client=sm,
                trial_component_name=trial_component_summary.trial_component_name)
            trial.remove_trial_component(tc)
        try:
            # comment out to keep trial components
            tc.delete()
        except:
            # tc is associated with another trial
            continue
        # to prevent throttling
        time.sleep(.5)
        trial.delete()
    experiment.delete()

cleanup(mnist_experiment)
```

For information on deleting your Amazon S3 buckets, see [How Do I Delete an S3 Bucket?](#).

To delete the notebook

1. Select the file browser.
2. Right-click the notebook file and choose **Shut Down Kernel**.
3. Right-click the notebook file and choose **Delete**.

Search

Developing a machine learning model typically requires extensive experimenting with different datasets, algorithms, and hyperparameter values. To manage up to thousands of machine learning model experiments, use Amazon SageMaker's search capabilities.

You can use Amazon SageMaker search to:

- Organize, find, and evaluate training jobs using properties, hyperparameters, performance metrics, or any metadata.
- Find the best performing model by reviewing training job and model metrics, such as training loss or validation accuracy.
- Trace a model's lineage to the training job and its related resources, such as the training datasets.

You can organize, find, and evaluate training jobs and models using the Amazon SageMaker console or the API.

Topics

- [Sample Notebooks for Managing ML Experiments \(p. 532\)](#)
- [Organize, Find, and Evaluate Training Jobs \(Console\) \(p. 532\)](#)
- [Find and Evaluate Training Jobs \(API\) \(p. 534\)](#)
- [Verify the Datasets Used by Your Training Jobs \(p. 535\)](#)

- [Trace Model Lineage \(p. 535\)](#)

Sample Notebooks for Managing ML Experiments

For a sample notebook that uses Amazon SageMaker model tracking capability to manage ML experiments, see [Managing ML Experimentation using Amazon SageMaker Model Tracking Capability](#).

For instructions on how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. The notebook for managing ML experiments is located in the **Advanced Functionality** section. To open a notebook, choose its **Use** tab, and choose **Create copy**. If you have questions, post them on the [Amazon Machine Learning Developer Forum](#).

Organize, Find, and Evaluate Training Jobs (Console)

To organize training jobs, assign one or more tags to them.

To find a specific training job, model, or resource, use model tracking to search on keywords assigned to any searchable items. *Searchable items* include training jobs, models, hyperparameters, metadata, tags, and URLs. To refine your tracking results, you can search using multiple criteria.

To choose the best model for deployment, evaluate how all models performed against one or more metrics. You can use model tracking results to list, sort, and evaluate the performance of the models in your experiments.

Topics

- [Use Tags to Track Training Jobs \(Console\) \(p. 532\)](#)
- [Find Training Jobs \(Console\) \(p. 533\)](#)
- [Evaluate Models \(Console\) \(p. 533\)](#)

Use Tags to Track Training Jobs (Console)

To group training jobs, create tags with descriptive keys and a value. For example, create tag keys for: project, owner, customer, and industry.

Add tags to training jobs (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Training jobs** and **Create training job**.
3. Scroll to the bottom of the page and enter a key and value for the tag.

Key	Value	Remove
Project	Project_Binary_Classifier	Remove

[Add tag](#)

Cancel Create training job

4. To add another tag, choose **Add tag**, and add another key-value pair.

Find Training Jobs (Console)

You can search for training jobs using a variety of job attributes. Note that some search parameters appear only if you have created a training job with that attribute. For example, **Tags** appears only if you have added a tag for a training job.

To find training jobs (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Search**.
3. Add **Parameters**.
 - a. In the search box, enter a parameter and choose a parameter type, for example **TrainingJobName**.
 - b. Choose a conditional operation. For numeric values, use operators such as **is equals to**, **lesser than**, or **or greater than**. For text-based values, use operators such as **equals to** or **contains**.
 - c. Enter a value for the parameter.
4. (Optional) To refine your search, add additional search criteria. Choose **Add row** and enter the parameter values.
5. Choose **Search**.

Evaluate Models (Console)

To evaluate a model's performance, review its metadata, hyperparameters, and metrics. To highlight metrics, adjust the view to show only metrics and important hyperparameters.

To evaluate a model (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Search** and search for training jobs by specifying relevant parameters. The results are displayed in a table.

Results: Training jobs

	HyperParameter mini_batch_size	HyperParameter predictor_type	Metric train:binary_f_beta	Metric train:progress	Metric train:objective_loss
	300	binary_classifier	0.966639518737793	100	0.02381423674523
	100	binary_classifier	0.9652714133262634	100	0.02350491285324
	200	binary_classifier	0.9647442698478699	100	0.02325980737805

3. Open the preferences window by choosing the settings icon in the search results table.

4. To show or hide a hyperparameter or metric, turn it on or off by choosing **Hyperparameter** or **Metric**.
5. Make necessary changes, then choose **Update view**.
6. After viewing metrics and important hyperparameters, you can compare and contrast the result. Then, you can choose the best model to host or investigate the models that are performing poorly.

Find and Evaluate Training Jobs (API)

To find and evaluate training jobs or to get suggestions for items used in experiments that are searchable, you can use the [Search API](#).

Topics

- [Find Training Jobs \(API\) \(p. 534\)](#)
- [Evaluate Models \(API\) \(p. 534\)](#)
- [Get Suggestions for a Search \(API\) \(p. 535\)](#)

Find Training Jobs (API)

To find training jobs, create a search parameter using the `search_params` parameter. Then use the `search` function in the `smclient` subprocess in the AWS SDK for Python (Boto 3).

The following example shows how to use the [Search API](#) to find training jobs.

```
import boto3

search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "Tags.Project",
                "Operator": "Equals",
                "Value": "Project_Binary_Classifier"
            }]
    },
    "SortBy": "Metrics.train:binary_classification_accuracy",
    "SortOrder": "Descending"
}

smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)
```

Evaluate Models (API)

To evaluate models, run a search as described in [Find Training Jobs \(API\) \(p. 534\)](#), review model metrics, then, use the AWS SDK for Python (Boto 3) to create a table and plot it.

The following example shows how to evaluate models and to display the results in a table.

```
import pandas

headers=["Training Job Name", "Training Job Status", "Batch Size", "Binary Classification Accuracy"]
rows=[ ]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
    rows.append([trainingJob['TrainingJobName'],
```

```
    trainingJob['TrainingJobStatus'],
    trainingJob['HyperParameters']['mini_batch_size'],
    metrics[[x['MetricName'] for x in
    metrics].index('train:binary_classification_accuracy')]['Value']
])

df = pandas.DataFrame(data=rows,columns=headers)

from IPython.display import display, HTMLdisplay(HTML(df.to_html()))
```

Get Suggestions for a Search (API)

To get suggestions for a search, use the [GetSearchSuggestions API](#).

The following example for AWS SDK for Python (Boto 3) is a `get_search_suggestions` request for items containing "linear".

```
search_suggestion_params={
    "Resource": "TrainingJob",
    "SuggestionQuery": {
        "PropertyNameQuery": {
            "PropertyNameHint": "linear"
        }
    }
}
```

The following is an example response for a `get_search_suggestions` request.

```
{
    'PropertyNameSuggestions': [
        {'PropertyName': 'hyperparameters.linear_init_method'},
        {'PropertyName': 'hyperparameters.linear_init_value'},
        {'PropertyName': 'hyperparameters.linear_init_sigma'},
        {'PropertyName': 'hyperparameters.linear_lr'},
        {'PropertyName': 'hyperparameters.linear_wd'}
    ]
}
```

After getting search suggestions, you can use one of the property names in a search.

Verify the Datasets Used by Your Training Jobs

You can use model tracking capability to verify which datasets were used in training, where holdout datasets were used, and other details about training jobs. For example, use model tracking capability to verify that a specific dataset was used in a training job for an audit or to verify compliance.

To check whether a specific dataset was used in a training job, you search for the URL to its location in Amazon Simple Storage Service (Amazon S3). Model tracking capability returns the training jobs that used the dataset that you specify. If your search doesn't return the dataset (the result is empty), the dataset wasn't used in a training job. An empty result confirms, for example, that a holdout dataset wasn't used.

Trace Model Lineage

You can use model tracking capability to get information about the lineage of training jobs and the model resources that were used for them, including the dataset, algorithm, hyperparameters, and metrics. For example, if you find that the performance of a hosted model has declined, you can review its training job and the resources it used to determine what's causing the problem. You can use the Amazon SageMaker console or the API to trace lineage.

Topics

- [Trace Model Lineage \(Console\) \(p. 536\)](#)
- [Trace Model Lineage \(API\) \(p. 536\)](#)

Trace Model Lineage (Console)

To trace a model's lineage (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Endpoints**, and choose the relevant endpoint.
3. Scroll to the **Endpoint configuration settings** section. This section lists all of the model versions deployed at the endpoint, with a hyperlink to the training job that created each.

Trace Model Lineage (API)

To trace a model's lineage, get the model's name, then use it to search for training jobs.

The following example shows how to trace a model's lineage using the API.

```
# Get the name of model deployed at endpoint
endpoint_config = smclient.describe_endpoint_config(EndpointConfigName=endpointName)
model_name = endpoint_config['ProductionVariants'][0]['ModelName']

# Get the model's name
model = smclient.describe_model(ModelName=model_name)

# Search the training job by the location of model artifacts in Amazon S3
search_params={
    "MaxResults": 1,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "ModelArtifacts.S3ModelArtifacts",
                "Operator": "Equals",
                "Value": model['PrimaryContainer']['ModelDataUrl']
            }]
    }
}
results = smclient.search(**search_params)
```

After finding the training job, you can review the resources used to train the model.

Amazon SageMaker Debugger

Amazon SageMaker Debugger provides full visibility into the training of machine learning models by monitoring, recording, and analyzing the tensor data that captures the state of a machine learning training job at each instance in its lifecycle. It provides a rich set of alerts when detecting errors for the steps of a machine learning training trial, and the ability to perform interactive explorations. It can automatically detect and alert you to commonly occurring errors such as gradient values getting too large or too small. When starting a training job that uses Debugger, you configure what tensors to save, where to save the tensors, and the trials to run on your dataset. The data collected remains in your account to use in subsequent analyses, securing its use for the most privacy-sensitive applications. Overall, Debugger can reduce the time to debug the training of models dramatically.

You can use Amazon SageMaker Debugger from Amazon SageMaker Studio or from Amazon SageMaker notebooks. Studio makes the inspection of training job issues easier by providing a visual interface for

you to use when analyzing your monitoring tensor data. You can use the Amazon SageMaker Debugger SDK to instrument your code to save tensors if necessary and to build customized rules. You can then use the Amazon SageMaker Python SDK to configure the debugger to save the required tensors and deploy built-in or custom rules monitoring these tensors. The open source *smdebug* Python library at [Amazon SageMaker Debugger](#) implements its core debugging functionality. Debugger also provides the parameters needed to hook up the *smdebug* functionality into the Amazon SageMaker training job APIs that enable it to be deployed on our machine learning platform.

Using Debugger is a two-step process:

- **Saving tensors (and scalars):** In deep learning algorithms, tensors define the state of the training job at any particular instant in its lifecycle. Amazon SageMaker Debugger exposes a library that allows you to capture these tensors and save them for analysis. Debugger is highly customizable and can help provide interoperability by saving performance and other feature-related metrics at specified frequencies.
- **Analysis:** The analysis of the tensors emitted is captured by the Amazon SageMaker Debugger concept called **rules**. On a very broad level, a rule is Python code used to detect certain conditions during training. Some of the conditions that a data scientist training an algorithm may care about are monitoring for gradients getting too large or too small or detecting overfitting. Debugger comes pre-packaged with certain rules. Users can write their own rules using the Debugger APIs. You can also analyze raw tensor data outside of the rules construct in an Amazon SageMaker notebook, using Debugger's complete set of APIs.

Amazon SageMaker Debugger Sample Notebooks

Amazon SageMaker Debugger provides sample notebooks that show how to use four learning frameworks to emit tensors in an Amazon SageMaker training job and then how to apply rules over the tensors to monitor the status of these jobs. It also provides sample notebooks that you can run interactive analysis with using the open source *smdebug* Python library at [Amazon SageMaker Debugger](#). The following notebooks are listed in the order we recommend you review them.

- [Using a built-in rule with TensorFlow](#)
- [Using a custom rule with TensorFlow Keras](#)
- [Interactive tensor analysis in notebook with MXNet](#)
- [Visualizing Debugging Tensors of MXNet training](#)
- [Real-time analysis in notebook with MXNet](#)
- [Using a built in rule with XGBoost](#)
- [Real-time analysis in notebook with XGBoost](#)
- [Using SageMaker Debugger with Managed Spot Training and MXNet](#)
- [Reacting to CloudWatch Events from Rules to take an action based on status with TensorFlow](#)
- [Using SageMaker Debugger with a custom PyTorch container](#)

The links to the Debugger sample notebooks is also available at [Amazon SageMaker Debugger Examples](#).

Note

Although each of these notebooks focuses on a specific framework, the same approach works with all the other frameworks that Amazon SageMaker Debugger supports

Learning Frameworks and Built-in Algorithms Supported by Amazon SageMaker Debugger

Amazon SageMaker Debugger supports using the following learning frameworks:

- Apache MXNet: [Use Apache MXNet with Amazon SageMaker \(p. 476\)](#)
- PyTorch: [Use PyTorch with Amazon SageMaker \(p. 477\)](#)
- TensorFlow: [Use TensorFlow with Amazon SageMaker \(p. 475\)](#)
- XGBoost: [XGBoost Algorithm \(p. 445\)](#)

Amazon SageMaker Debugger supports the following Amazon SageMaker built-in algorithm:

- [XGBoost Release 0.72 \(p. 456\)](#)—a supervised learning algorithm that is an open-source implementation of the gradient boosted trees algorithm.

There are two ways to enable Debugger while training models on Amazon SageMaker:

- Use framework containers we provide that do not require any script changes.
- Bring your own container and make the script changes needed to enable Debugger

Topics

- [Zero Script Change \(p. 538\)](#)
- [Bring Your Own Training Container \(p. 538\)](#)

Zero Script Change

We have provided custom versions of the framework containers we support that enable you to use Amazon SageMaker Debugger with no changes to your training script, by automatically adding Debugger's Hook. We support the following frameworks and versions for this experience.

Supported Frameworks and Versions

Framework	Version
TensorFlow	1.15
MXNet	1.6
PyTorch	1.3
XGBoost	>+0.90 (As built-in algorithm)

For more information on the deep learning frameworks containers, see [Prebuilt Amazon SageMaker Docker Images for TensorFlow, MXNet, Chainer, and PyTorch \(p. 493\)](#).

Bring Your Own Training Container

This library *smdebug* itself supports versions other than the ones listed above. If you want to use SageMaker Debugger with a version different from the above, you will have to orchestrate your training script with a few lines. Before we discuss how these changes look like, let us take a look at the versions supported.

Supported Frameworks and Versions

Framework	Version
TensorFlow	1.13, 1.14, 1.15

Framework	Version
Keras (with TensorFlow backend)	2.3
MXNet	1.4, 1.4, 1.6
PyTorch	1.2, 1.3
XGBoost	(As a framework)

To set up Debugger with your script on your container:

- Ensure that you are using Python 3+ runtime as `smdebug` only supports Python 3 or higher.
- Install `smdebug` binary through `pip install smdebug`
- Make some minimal modifications to your training script to add the hook for SageMaker Debugger. Refer to the framework pages linked in the table for instructions.

Amazon CloudWatch Metrics for Amazon SageMaker Debugger

Amazon CloudWatch collects model training metrics so you can monitor training jobs. See the **Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics** sections in the in [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#) topic.

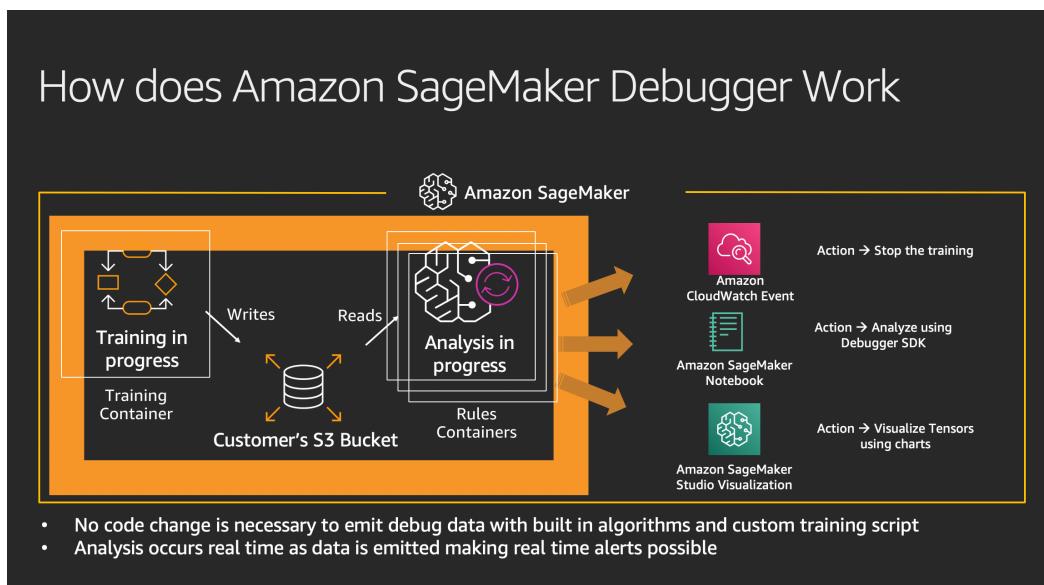
Topics

- [How Debugger Works \(p. 539\)](#)
- [Save Tensor Data for Debugger \(p. 541\)](#)
- [Prebuilt Amazon SageMaker Docker Images for Rules \(p. 543\)](#)
- [Built-in Rules Provided by Amazon SageMaker Debugger \(p. 545\)](#)
- [How to Use Built-in Rules for Model Analysis \(p. 568\)](#)
- [Programming Model for Debugger \(p. 569\)](#)
- [How to Use Custom Rules \(p. 570\)](#)
- [Amazon SageMaker Studio Visualizations of Model Analysis Results \(p. 572\)](#)
- [Amazon SageMaker Debugger Reference Documentation \(p. 573\)](#)

How Debugger Works

Amazon SageMaker Debugger enables you go beyond just looking at scalars like losses and accuracies when evaluating model training. It gives you full visibility into a training job by using a hook to capture tensors that define the state of the training process at each instance in its lifecycle. It also provides the capability of defining 'rules' to analyze the captured tensors. Built-in rules monitor the training flow and alert you to problems with various common conditions that are critical for the success of the training job. You can also create your own custom rules to watch for any issues specific to your model. You can monitor the results of the analysis done by rules with Amazon CloudWatch events, using an Amazon SageMaker notebook, or in visualization provided by Amazon SageMaker Studio. Here is a depiction of the flow for the model training process with Amazon SageMaker Debugger .

The following diagram shows the flow for the model training process with Debugger.



Amazon SageMaker Debugger automatically debugs your machine learning training process. It helps you develop better, faster, cheaper models by catching common errors quickly. While training, it automatically saves tensors and the network state (no script changes needed). By constantly monitoring the training process, a separate job detects anomalies, such as a vanishing gradient, poor weight initialization, or other warning signals. That way, if a training job fails, you'll know what happened and how to fix it. There are over 15 built-in "rules," or state assertions to ensure that training happens efficiently.

Amazon SageMaker Debugger supports all popular machine learning frameworks (TensorFlow, PyTorch, and Apache MXNet) and XGBoost. It provides both an automated and a configurable experience. In the automated experience, it provides custom framework forks in the deep learning containers that automatically detect your training job and save tensors, without any changes to your training script. There is also an advanced configurable mode where you choose precisely which tensors to save, how to organize them, and which custom rules to use.

The data can be saved to an S3 bucket so you can run your own analysis afterwards by creating a "trial." This provides full insight into the training process.

Amazon SageMaker Debugger does the following:

- Creates a "hook" object.
- Passes this hook as a callback inside the training process.
- The hook listens to various events, such as the forward and backward pass through the network. Upon registering a "step," or forward and backward pass, it writes tensors and other data to your S3 bucket.
- The hook can also detect when your mode has switched between training and validation, letting you easily segment your data.
- The tensors are first saved locally on the training instance and then moved to the S3 location you specified.
- A separate instance runs the rule monitoring job which fetches tensors from the S3 location locally and invokes the rule logic on the tensors saved for the training job. If something is amiss, it raises an exception and triggers an Amazon CloudWatch event. You can run multiple rules simultaneously.

Save Tensor Data for Debugger

Tensors define the state of the training job at any particular instant in its lifecycle. Amazon SageMaker Debugger provides the `smdebug` library, which allows you to monitor these tensors, save them, and analyze them to evaluate model training. Tensors can be grouped into collections to help manage them. Debugger gives you a powerful and flexible API to save the tensors you choose at the frequencies you want. These configurations are made available in the Amazon SageMaker Python SDK through the `DebuggerHookConfig` class.

Topics

- [Save Built-in Party Collections \(p. 541\)](#)
- [Save Reductions for a Custom Collection \(p. 541\)](#)
- [Enable TensorBoard Summaries \(p. 542\)](#)

Save Built-in Party Collections

Learn more about these first party collections [Common API](#).

```
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig
hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    hook_parameters={
        "save_interval": 100
    },
    collection_configs=[
        CollectionConfig("weights"),
        CollectionConfig("gradients"),
        CollectionConfig("losses"),
        CollectionConfig(
            name="biases",
            parameters={
                "save_interval": 10,
                "end_step": 500
            }
        ),
    ],
)
import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    debugger_hook_config=hook_config
)
sagemaker_estimator.fit()
```

Save Reductions for a Custom Collection

You can define your collection of tensors. You can also choose to save certain reductions of tensors only instead of saving the full tensor. You may choose to do this to reduce the amount of data saved.

Note

When you save reductions, unless you pass the flag `save_raw_tensor`, only these reductions will be available for analysis. The raw tensor will not be saved.

```

from sagemaker.debugger import DebuggerHookConfig, CollectionConfig
hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    collection_configs=[
        CollectionConfig(
            name="activations",
            parameters={
                "include_regex": "relu|tanh",
                "reductions": "mean,variance,max,abs_mean,abs_variance,abs_max"
            })
    ]
)
import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    debugger_hook_config=hook_config
)
sagemaker_estimator.fit()

```

Enable TensorBoard Summaries

Amazon SageMaker Debugger can automatically generate TensorBoard scalar summaries, distributions and histograms for tensors saved. This can be enabled by passing a `TensorBoardOutputConfig` object when creating an `Estimator` as follows. You can also choose to disable or enable histograms specifically for different collections. By default a collection has `save_histogram` flag set to `True`. Note that scalar summaries are added to TensorBoard for all `ScalarCollections` and any scalar saved through `hook.save_scalar`. For more information on scalar collections and `save_scalar` method, see the [Common API](#).

The following example saves weights and gradients as full tensors, and also saves the gradients as histograms and distributions to visualize in TensorBoard. These are saved to the location passed in `TensorBoardOutputConfig` object.

```

from sagemaker.debugger import DebuggerHookConfig, CollectionConfig,
    TensorBoardOutputConfig
hook_config = DebuggerHookConfig(
    s3_output_path='s3://smdebug-dev-demo-pdx/mnist',
    collection_configs=[
        CollectionConfig(
            name="weights",
            parameters={"save_histogram": False}),
        CollectionConfig(name="gradients"),
    ]
)

tb_config = TensorBoardOutputConfig('s3://smdebug-dev-demo-pdx/tensorboard')

import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",

```

```
    py_version="py3",
    # smdebug-specific arguments below
    debugger_hook_config=hook_config,
    tensorboard_output_config=tb_config
)
sagemaker_estimator.fit()
```

Prebuilt Amazon SageMaker Docker Images for Rules

Amazon SageMaker provides two kinds of prebuilt Docker images for rules, one for evaluating Amazon SageMaker owned rules and other for evaluating your custom rules as Python source files.

If you're using the Amazon SageMaker Python SDK, Amazon SageMaker rules can be used with an estimator, without having to retrieve a Docker image. If you are not using the SDK and one of its estimators to manage the container, you have to retrieve the relevant pre-built container. The Amazon SageMaker prebuilt Rule Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). To pull an image from an Amazon ECR repo (or to push an image to an Amazon ECR repo), use the full name registry address of the image. Amazon SageMaker uses the following URL patterns for the container image registry addresses.

```
<account_id>.dkr.ecr.<region>.amazonaws.com/<ECR repo name>:<tag>
```

Topics

- [Amazon SageMaker Built-in Rules Registry IDs \(p. 543\)](#)
- [Amazon SageMaker Custom Rule Evaluator IDs \(p. 544\)](#)

Amazon SageMaker Built-in Rules Registry IDs

The following table itemizes the supported values for the URL components of the registry addresses for the images providing built-in rules.

ECR Repository Name: sagemaker-debugger-rules

Tag: latest

Example full name registry address:

```
904829902805.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rules:latest
```

Table: Regions and Build-in Rules Registry IDs

Regions	Registry IDs
ap-east-1	199566480951
ap-northeast-1	430734990657
ap-northeast-2	578805364391
ap-south-1	904829902805
ap-southeast-1	972752614525
ap-southeast-2	184798709955
ca-central-1	519511493484
eu-central-1	482524230118

Regions	Registry IDs
eu-north-1	314864569078
eu-west-1	929884845733
eu-west-2	250201462417
eu-west-3	447278800020
me-south-1	986000313247
sa-east-1	818342061345
us-east-1	503895931360
us-east-2	915447279597
us-west-1	685455198987
us-west-2	895741380848

Amazon SageMaker Custom Rule Evaluator Registry IDs

The following table itemizes the supported values for the URL components of the registry addresses for the images providing custom rule evaluators.

ECR Repository Name: sagemaker-debugger-rule-evaluator

Tag: latest

Example full name registry address:

552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rule-evaluator:latest

Table: Regions and Custom Rule Evaluator Registry IDs

Regions	Registry IDs
ap-east-1	645844755771
ap-northeast-1	670969264625
ap-northeast-2	326368420253
ap-south-1	552407032007
ap-southeast-1	631532610101
ap-southeast-2	445670767460
ca-central-1	105842248657
eu-central-1	691764027602
eu-north-1	091235270104
eu-west-1	606966180310
eu-west-2	074613877050

Regions	Registry IDs
eu-west-3	224335253976
me-south-1	050406412588
sa-east-1	466516958431
us-east-1	864354269164
us-east-2	840043622174
us-west-1	952348334681
us-west-2	759209512951

Built-in Rules Provided by Amazon SageMaker Debugger

Use the built-in rules provided by Amazon SageMaker Debugger to analyze tensors emitted during the training of machine learning models. These rules monitor various common conditions that are critical for the success of a training job. There are four scopes of validity for the built-in rules.

Scopes of Validity for Build-in Rules

Scope of Validity	Built-in Rules
Deep learning frameworks (TensorFlow, Apache MXNet, and PyTorch)	<ul style="list-style-type: none"> • DeadRelu • ExplodingTensor • PoorWeightInitialization • SaturatedActivation • VanishingGradient • WeightUpdateRatio
Deep learning frameworks (TensorFlow, MXNet, and PyTorch) and the XGBoost algorithm	<ul style="list-style-type: none"> • AllZero • ClassImbalance • Confusion • LossNotDecreasing • Overfit • Overtraining • SimilarAcrossRuns • TensorVariance • UnchangedTensor
Deep learning applications	<ul style="list-style-type: none"> • CheckInputImages • NLPSequenceRatio
XGBoost algorithm	<ul style="list-style-type: none"> • TreeDepth

Topics

- [DeadRelu Rule \(p. 546\)](#)
- [ExplodingTensor Rule \(p. 547\)](#)

- [PoorWeightInitialization Rule \(p. 548\)](#)
- [SaturatedActivation Rule \(p. 549\)](#)
- [VanishingGradient Rule \(p. 551\)](#)
- [WeightUpdateRatio Rule \(p. 552\)](#)
- [AllZero Rule \(p. 553\)](#)
- [ClassImbalance Rule \(p. 554\)](#)
- [Confusion Rule \(p. 556\)](#)
- [LossNotDecreasing Rule \(p. 557\)](#)
- [Overfit Rule \(p. 559\)](#)
- [Overtraining Rule \(p. 560\)](#)
- [SimilarAcrossRuns Rule \(p. 561\)](#)
- [TensorVariance Rule \(p. 562\)](#)
- [UnchangedTensor Rule \(p. 564\)](#)
- [CheckInputImages Rule \(p. 565\)](#)
- [NLPSequenceRatio Rule \(p. 566\)](#)
- [TreeDepth Rule \(p. 567\)](#)

DeadRelu Rule

This rule detects when the percentage of rectified linear unit (ReLU) activation functions in a trial are considered dead because their activation activity has dropped below a threshold. If the percent of inactive ReLUs in a layer is greater than the `threshold_layer` value of inactive ReLUs, the rule returns `True`.

Parameter Descriptions for the DeadRelu Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: <code>None</code></p>
<code>threshold_inactivity</code>	<p>Defines a level of activity below which a ReLU is considered to be dead. A ReLU might be active in the beginning of a trial and then slowly die during the training process. If the ReLU is active less than</p>

Parameter Name	Description
	<p>the <code>threshold_inactivity</code>, it is considered to be dead.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default values: 1 . 0</p>
<code>threshold_layer</code>	<p>Returns <code>True</code> if the percentage of inactive ReLUs in a layer is greater than <code>threshold_layer</code>.</p> <p>Returns <code>False</code> if the percentage of inactive ReLUs in a layer is less than <code>threshold_layer</code>.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default values: 50 . 0</p>

For an example of to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

ExplodingTensor Rule

This rule detects whether the tensors emitted during training have non-finite values, either infinite or NaN (not a number). If a non-finite value is detected, the rule returns `True`.

Parameter Descriptions for the ExplodingTensor Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered

Parameter Name	Description
	<p>in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: <code>None</code></p>
<code>only_nan</code>	<p><code>True</code> to monitor the <code>base_trial</code> tensors only for <code>Nan</code> values and not for infinity.</p> <p><code>False</code> to treat both <code>Nan</code> and infinity as exploding values and to monitor for both.</p> <p>Optional</p> <p>Valid values: <code>False</code></p>

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

PoorWeightInitialization Rule

This rule detects if your model parameters have been poorly initialized.

Good initialization breaks the symmetry of the weights and gradients in a neural network and maintains commensurate activation variances across layers. Otherwise, the neural network doesn't learn effectively. Initializers like Xavier aim to keep variance constant across activations, which is especially relevant for training very deep neural nets. Too small an initialization can lead to vanishing gradients. Too large an initialization can lead to exploding gradients. This rule checks the variance of activation inputs across layers, the distribution of gradients, and the loss convergence for the initial steps to determine if a neural network has been poorly initialized.

Parameter Descriptions for the PoorWeightInitialization Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>activation_inputs_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p>

Parameter Name	Description
	<p>Valid values: List of strings or a comma-separated string</p> <p>Default value: ". *relu_input"</p>
threshold	<p>If the ratio between minimum and maximum variance of weights per layer exceeds the threshold at a step, the rule returns True.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 10 . 0</p>
distribution_range	<p>If the minimum difference between 5th and 95th percentiles of the gradient distribution is less than the distribution_range, the rule returns True.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 0 . 001</p>
patience	<p>The number of steps to wait until the loss is considered to be no longer decreasing.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: 5</p>
steps	<p>The number of steps this rule analyzes. You typically want to check only the first few iterations.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 10</p>

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

SaturatedActivation Rule

This rule detects if the tanh and sigmoid activation layers are becoming saturated. An activation layer is saturated when the input of the layer is close to the maximum or minimum of the activation function. The minimum and maximum of the tanh and sigmoid activation functions are defined by their respective `min_threshold` and `max_thresholds` values. If the activity of a node drops below the

`threshold_inactivity` percentage, it is considered saturated. If more than a `threshold_layer` percent of the nodes are saturated, the rule returns `True`.

Parameter Descriptions for the SaturatedActivation Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: <code>".*tanh_input, .*sigmoid_input".</code></p>
<code>threshold_tanh</code>	<p>The minimum and maximum thresholds that define the extremes of the input for a tanh activation function, defined as: <code>(min_threshold, max_threshold)</code>. The default values are determined based on a vanishing gradient threshold of 0.0000001.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default values: <code>(-9.4999, 9.4999)</code></p>
<code>threshold_sigmoid</code>	<p>The minimum and maximum thresholds that define the extremes of the input for a sigmoid activation function, defined as: <code>(min_threshold, max_threshold)</code>. The default values are determined based on a vanishing gradient threshold of 0.0000001.</p>

Parameter Name	Description
	Optional Valid values: Float Default values: (-23, 16.99999)
threshold_inactivity	The percentage of inactivity below which the activation layer is considered to be saturated. The activation might be active in the beginning of a trial and then slowly become less active during the training process. Optional Valid values: Float Default values: 1.0
threshold_layer	Returns <code>True</code> if the number of saturated activations in a layer is greater than the <code>threshold_layer</code> percentage. Returns <code>False</code> if the number of saturated activations in a layer is less than the <code>threshold_layer</code> percentage. Optional Valid values: Float Default values: 50.0

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

VanishingGradient Rule

This rule detects if the gradients in a trial become extremely small or drop to a zero magnitude. If the mean of the absolute values of the gradients drops below a specified `threshold`, the rule returns `True`.

Parameters Descriptions for the VanishingGradient Rule

Parameter Name	Description
base_trial	The trial run using this rule. The rule inspects the tensors gathered from this trial. Required Valid values: String
threshold	The value at which the gradient is determined to be vanishing. Optional

Parameter Name	Description
	Valid values: Float Default value: 0 . 0000001.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

WeightUpdateRatio Rule

This rule keeps track of the ratio of updates to weights during training and detects if that ratio gets too large or too small. If the ratio of updates to weights is larger than the `large_threshold` value or if this ratio is smaller than `small_threshold`, the rule returns `True`.

Conditions for training are best when the updates are commensurate to the gradients. Excessively large updates can push the weights away from optimal values, and very small updates result in very slow convergence. This rule requires weights to be available for two consecutive steps, so `save_interval` needs to be set to 1.

Parameter Descriptions for the WeightUpdateRatio Rule

Parameter Name,	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>num_steps</code>	<p>The number of steps across which the rule checks to determine if the tensor has changed.</p> <p>The number of steps across which you want to compare the weight ratios. If you pass no value, the rule runs by default against the current step and the immediately previous saved step. If you override the default by passing a value for this parameter, the comparison is done between weights at step s and at a step that is $\geq s - num_steps$.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: None</p>
<code>large_threshold</code>	<p>The maximum value that the ratio of updates to weigh can take before the rule returns <code>True</code>.</p> <p>Optional</p> <p>Valid values: Float</p>

Parameter Name,	Description
	Default value: 10 . 0
small_threshold	<p>The minimum value that the ratio of updates to weigh can take, below which the rule returns True.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 0 . 00000001</p>
epsilon	<p>A small constant used to ensure that Debugger does not divide by zero when computing the ratio updates to weigh.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 0 . 00000001</p>

Note

If tensors have been saved with TRAIN mode on during training, the rule runs only on TRAIN mode steps. Otherwise, it runs by default on GLOBAL mode steps.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

This rule can't be applied to the XGBoost algorithm.

AllZero Rule

This rule detects if all or a specified percentage of the values in the tensors are zero.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the collection_names or tensor_regex parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameters Descriptions for the AllZero Rule

Parameter Name	Description
base_trial	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
collection_names	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p>

Parameter Name	Description
	<p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>threshold</code>	<p>Specifies the percentage of values in the tensor that needs to be zero for this rule to be invoked.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 100</p>

ClassImbalance Rule

This rule measures sampling imbalances between classes and throws errors if the imbalance exceeds a threshold or if too many mispredictions for underrepresented classes occur as a result of the imbalance.

Classification models require well-balanced classes in the training dataset or a proper weighting/sampling of classes during training. The rule performs two checks:

- It counts the occurrences per class. If the ratio of number of samples between smallest and largest class is larger than the `threshold_imbalance`, an error is thrown.
- It checks the prediction accuracy per class. If resampling or weighting has not been correctly applied, then the model can reach high accuracy for the class with many training samples, but low accuracy for the classes with few training samples. If a fraction of mispredictions for a certain class is above `threshold_misprediction`, an error is thrown.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the ClassImbalance Rule

Parameter Name	Description
<code>base_trial</code>	The trial run using this rule. The rule inspects the tensors gathered from this trial.

Parameter Name	Description
	Required Valid values: String
<code>threshold_imbalance</code>	The acceptable imbalance between the number of samples in the smallest class and in the largest class. Exceeding this threshold value throws an error. Optional Valid values: Float Default value: 10
<code>threshold_misprediction</code>	A limit on the fraction of mispredictions allowed for each class. Exceeding this threshold throws an error. The underrepresented classes are most at risk of crossing this threshold. Optional Valid values: Float Default value: 0 . 7
<code>samples</code>	The number of labels that have to be processed before an imbalance is evaluated. The rule might not be triggered until it has seen sufficient samples across several steps. The more classes that your dataset contains, the larger this sample number should be. Optional Valid values: Integer Default value: 500 (assuming a dataset like MNIST with 10 classes)
<code>argmax</code>	If <code>True</code> , <code>np.argmax</code> is applied to the prediction tensor. Required when you have a vector of probabilities for each class. It is used to determine which class has the highest probability. Conditional Valid values: Boolean Default value: <code>False</code>
<code>labels_regex</code>	The name of the tensor that contains the labels. Optional Valid values: String Default value: ". *labels"

Parameter Name	Description
<code>predictions_regex</code>	<p>The name of the tensor that contains the predictions.</p> <p>Optional</p> <p>Valid values: String</p> <p>Default value: ". *predictions"</p>

Confusion Rule

This rule evaluates the goodness of a confusion matrix for a classification problem.

It creates a matrix of size `category_no*category_no` and populates it with data coming from (`labels`, `predictions`) pairs. For each (`labels`, `predictions`) pair, the count in `confusion[labels][predictions]` is incremented by 1. When the matrix is fully populated, the ratio of data on-diagonal values and off-diagonal values are evaluated as follows:

- For elements on the diagonal: `confusion[i][i]/sum_j(confusion[j][j])>=min_diag`
- For elements off the diagonal: `confusion[j][i])/sum_j(confusion[j][i])<=max_off_diag`

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the Confusion Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>category_no</code>	<p>The number of categories.</p> <p>Optional</p> <p>Valid values: Integer ≥ 2</p> <p>Default value: The larger of the number of categories in <code>labels</code> and <code>predictions</code>.</p>
<code>labels</code>	<p>The tensor name for 1-d vector of true labels.</p> <p>Optional</p> <p>Valid values: String</p> <p>Default value: "labels"</p>

Parameter Name	Description
<code>predictions</code>	<p>The tensor name for 1-d vector of estimated labels.</p> <p>Optional</p> <p>Valid values: String</p> <p>Default value: "predictions"</p>
<code>labels_collection</code>	<p>The rule inspects the tensors in this collection for labels.</p> <p>Optional</p> <p>Valid values: String</p> <p>Default value: "labels"</p>
<code>predictions_collection</code>	<p>The rule inspects the tensors in this collection for predictions.</p> <p>Optional</p> <p>Valid values: String</p> <p>Default value: "predictions"</p>
<code>min_diag</code>	<p>The minimum value for the ratio of data on the diagonal.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 1$</p> <p>Default value: 0 . 9</p>
<code>max_off_diag</code>	<p>The maximum value for the ratio of data off the diagonal.</p> <p>Optional</p> <p>Valid values: $0 \leq \text{float} \leq 1$</p> <p>Default value: 0 . 1</p>

Note

This rule infers default values for the optional parameters if their values aren't specified.

LossNotDecreasing Rule

This rule detects when the loss is not decreasing in value at an adequate rate. These losses must be scalars.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the LossNotDecreasing Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>use_losses_collection</code>	<p>If set to <code>True</code>, looks for losses in the collection named "losses" when the collection is present.</p> <p>Optional</p> <p>Valid values: Boolean</p> <p>Default value: <code>True</code></p>
<code>num_steps</code>	<p>The minimum number of steps after which the rule checks if the loss has decreased. Rule evaluation happens every <code>num_steps</code>. The rule compares the loss for this step with the loss at a step which is at least <code>num_steps</code> behind the current step. For example, suppose that the loss is being saved every 3 steps, but <code>num_steps</code> is set to 10. At step 21, loss for step 21 is compared with the loss for step 9. The next step where loss is checked is 33, because 10 steps after 21 is 31, and at 31 and 32 loss is not being saved.</p>

Parameter Name	Description
	Optional Valid values: Integer Default value: 10
diff_percent	The minimum percentage difference by which the loss should decrease between num_steps. Optional Valid values: $0.0 < \text{float} < 100$ Default value: Checks if the loss is decreasing between num_steps.
mode	The name of the Debugger mode to query tensor values for rule checking. If this is not passed, the rule checks in order by default for the mode.EVAL, then mode.TRAIN, and then mode.GLOBAL. Optional Valid values: String (EVAL, TRAIN, OR GLOBAL) Default value: None

Overfit Rule

This rule detects if your model is being overfit to the training data by comparing the validation and training losses.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

A standard way to prevent overfitting is to regularize your model.

Parameter Descriptions for the Overfit Rule

Parameter Name	Description
base_trial	The trial run using this rule. The rule inspects the tensors gathered from this trial. Required Valid values: String
tensor_regex	A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered

Parameter Name	Description
	<p>in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>start_step</code>	<p>The step from which to start comparing the validation and training loss.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: 0</p>
<code>patience</code>	<p>The number of steps for which the <code>ratio_threshold</code> is allowed to exceed the value set before the model is considered to be overfit.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: 1</p>
<code>ratio_threshold</code>	<p>The maximum ratio of the difference between the mean validation loss and mean training loss to the mean training loss. If this threshold is exceeded for a patience number of steps, the model is being overfit and the rule returns <code>True</code>.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 0 . 1</p>

Overtraining Rule

This rule detects if the model is being overtrained.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Note

Overtraining can be avoided by early stopping. For information on early stopping, see [Stop Training Jobs Early \(p. 588\)](#). For an example that shows how to use spot training with Debugger, see [Enable Spot Training with Amazon SageMaker Debugger](#).

Parameter Descriptions for the Overtraining Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>patience_train</code>	<p>The number of steps to wait before the training loss is considered to not to be improving anymore.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: 5</p>
<code>patience_validation</code>	<p>The number of steps to wait before the validation loss is considered to not to be improving anymore.</p> <p>Optional</p> <p>Valid values: Integer</p> <p>Default value: 10</p>
<code>delta</code>	<p>The minimum threshold by how much the error should improve before it is considered as a new optimum.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: 0 . 1</p>

The following Python example shows how to implement this rule.

```
rules_specification = [
    {
        "RuleName": "Overtraining",
        "InstanceType": "ml.c5.4xlarge",
        "RuntimeConfigurations": {
            "patience_train" : "10",
            "patience_validation": "20"
        }
    }
]
```

SimilarAcrossRuns Rule

This rule compares tensors gathered from a base trial with tensors from another trial.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the SimilarAcrossRuns Rule

Parameter Name	Description
<code>base_trial</code>	The trial run using this rule. The rule inspects the tensors gathered from this trial. Required Valid values: String
<code>other_trial</code>	The trial whose tensors you want to compare to those tensors gathered from the <code>base_trial</code> . Required Valid values: String
<code>collection_names</code>	The list of collection names whose tensors the rule inspects. Optional Valid values: List of strings or a comma-separated string Default value: None
<code>tensor_regex</code>	A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched. Optional Valid values: List of strings or a comma-separated string Default value: None

TensorVariance Rule

This rule detects if you are having tensors with very high or low variances. Very high or low variances in a tensor could lead to neuron saturation, which reduces the learning ability of the neural network. Very high variance in tensors can also eventually lead to exploding tensors. Use this rule to detect such issues early.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the TensorVariance Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>max_threshold</code>	<p>The threshold for the upper bound of tensor variance.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: xxxx</p>
<code>min_threshold</code>	<p>The threshold for the lower bound of tensor variance.</p> <p>Optional</p> <p>Valid values: Float</p> <p>Default value: xxxx</p>

UnchangedTensor Rule

This rule detects whether a tensor is no longer changing across steps.

This rule runs the `numpy.allclose` method to check if the tensor isn't changing.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the UnchangedTensor Rule

Parameter Name	Description
<code>base_trial</code>	<p>The trial run using this rule. The rule inspects the tensors gathered from this trial.</p> <p>Required</p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p>Optional</p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>num_steps</code>	<p>The number of steps across which the rule checks to determine if the tensor has changed.</p> <p>This checks the last <code>num_steps</code> that are available. They don't need to be consecutive. If <code>num_steps</code> is 2, at step s it doesn't necessarily check for s-1 and s. If s-1 isn't available, it checks the last available step along with s. In that case, it checks the last available step with the current step.</p> <p>Optional</p>

Parameter Name	Description
	Valid values: Integer Default value: 3
rtol	The relative tolerance parameter to be passed to the numpy.allclose method. Optional Valid values: Float Default value: 1e-05
atol	The absolute tolerance parameter to be passed to the numpy.allclose method. Optional Valid values: Float Default value: 1e-08
equal_nan	Whether to compare NaNs as equal. If <code>True</code> , NaNs in input array <code>a</code> are considered equal to NaNs in input array <code>b</code> in the output array. This parameter is passed to the numpy.allclose method. Optional Valid values: Boolean Default value: <code>False</code>

CheckInputImages Rule

This rule checks if input images have been correctly normalized. Specifically, it detects if the mean of the sample data differs by more than a threshold value from zero. Many computer vision models require that input data has a zero mean and unit variance.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the CheckInputImages Rule

Parameter Name	Description
base_trial	The trial run using this rule. The rule inspects the tensors gathered from this trial. Required Valid values: String
threshold_mean	A threshold that defines by how much mean of the input data can differ from 0.

Parameter Name	Description
	Optional Valid values: Float Default value: 0 . 2
threshold_samples	The number of images that have to be sampled before an error can be thrown. If the value is too low, the estimation of the dataset mean will be inaccurate. Optional Valid values: Integer Default value: 500
regex	The name of the input data tensor. Optional Valid values: String Default value: ". *hybridsequential0_input_0" (the name of the input tensor for Apache MXNet models using HybridSequential)
channel	The position of the color channel in the input tensor. Optional Valid values: Integer Default value: 1 (for example, MXNet expects input data in the form of (batch_size, channel, height, width))

NLPSequenceRatio Rule

This rule calculates the ratio of specific tokens given the rest of the input sequence that is useful for optimizing performance. For example, you can calculate the percentage of padding end-of-sentence (EOS) tokens in your input sequence. If the number of EOS tokens is too high, an alternate bucketing strategy should be performed. You also can calculate the percentage of unknown tokens in your input sequence. If the number of unknown words is too high, an alternate vocabulary could be used.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the NLPSequenceRatio Rule

Parameter Name	Description
base_trial	The trial run using this rule. The rule inspects the tensors gathered from this trial.

Parameter Name	Description
	Required Valid values: String
<code>tensor_regex</code>	A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched. Optional Valid values: List of strings or a comma-separated string Default value: ". *embedding0_input_0" (assuming an embedding as the initial layer of the network)
<code>token_values</code>	A string of a list of the numerical values of the tokens. For example, "3, 0". Optional Valid values: Comma-separated string of numerical values Default value: 0
<code>token_thresholds_percent</code>	A string of a list of thresholds (in percentages) that correspond to each of the <code>token_values</code> . For example, "50.0, 50.0". Optional Valid values: Comma-separated string of floats. Default value: "50, 50"

TreeDepth Rule

This rule measures the depth of trees in an XGBoost model. XGBoost rejects splits if they don't improve loss. This regularizes the training. As a result, the tree might not grow as deep as defined in `max_depth`.

This rule is valid only for the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

Parameter Descriptions for the TreeDepth Rule

Parameter Name	Description
<code>base_trial</code>	The trial run using this rule. The rule inspects the tensors gathered from this trial.

Parameter Name	Description
	Required Valid values: String
depth	The depth of the tree. The depth of the tree is obtained by computing the base 2 logarithm of the largest node ID. Optional Valid values: Float Default value: 4

How to Use Built-in Rules for Model Analysis

Amazon SageMaker Debugger rules analyse tensors emitted during the training of a model. They monitor conditions that are critical for the success of a training job. For example, they can detect whether gradients are getting too large or too small or if a model is being overfit. Debugger comes pre-packaged with certain Python-coded, built-in rules.

You can deploy a built-in rule to monitor your training job either by using the [CreateTrainingJob](#) API or by using the open source [smdebug Python library](#) with the Amazon SageMaker Python SDK. The *smdebug* programming model provides the context for understanding this task. For information on the programming model, see [Analysis](#).

Note

You are not charged for the instances when running SageMaker built-in rules

Topics

- [Use the CreateTrainingJob API to Create a Built-in Rule \(p. 568\)](#)
- [Use smdebug Python library with the Amazon SageMaker Python SDK to Create a Built-in Rule \(p. 569\)](#)

Use the [CreateTrainingJob](#) API to Create a Built-in Rule

A built-in rule can be configured for a training job using the [DebugHookConfig](#) and [DebugRuleConfiguration](#) objects in the [CreateTrainingJob](#) API. These rules are run on one of our pre-built Docker images which are listed in the [Prebuilt Amazon SageMaker Docker Images for Rules \(p. 543\)](#) topic. You specify the URL registry address for the pre-built Docker image in the [RuleEvaluatorImage](#) parameter.

The following code sample shows how to configure a built-in [VanishingGradient](#) rule using this Amazon SageMaker API.

```
DebugRuleConfigurations: [
  {
    "RuleConfigurationName": "Amazon-VanishingGradient",
    "RuleEvaluatorImage": "503895931360.dkr.ecr.us-east-1.amazonaws.com/sagemaker-debugger-
rules:latest",
    "RuleParameters": {
      "rule_to_invoke": "VanishingGradient",
      "threshold": "20.0"
    }
  }
]
```

]

With a configuration like the one in this sample, Amazon SageMaker Debugger starts a rule evaluation job for your training job using the Amazon SageMaker VanishingGradient rule.

Use *smdebug* Python library with the Amazon SageMaker Python SDK to Create a Built-in Rule

For a sample notebook that shows you how to use a built-in rule when training job with a TensorFlow model in SM; see [Amazon SageMaker Debugger - Using built-in rule](#).

The following code sample shows how to run an Amazon SageMaker built-in Rule using the `Rule.sagemaker` method from the Amazon SageMaker Python SDK. The first argument to this method is the base configuration that is associated with the Rule. We configure the built-in rules with the `smdebug_ruleconfigs` that we populate for all of the built-in rules. For details, see [Amazon SageMaker Debugger RulesConfig](#). We provide default values for the parameters that are not required, but you have the option to modify these default values when using the built-in rules. You do not need to specify a pre-built Docker image when using the Amazon SageMaker Python SDK as the `Estimators` handle this task.

```
from sagemaker.debugger import Rule, CollectionConfig, rule_configs

exploding_tensor_rule = Rule.sagemaker(
    base_config=rule_configs.exploding_tensor(),
    rule_parameters={"collection_names": "weights,losses"},
    collections_to_save=[
        CollectionConfig("weights"),
        CollectionConfig("losses")
    ]
)

vanishing_gradient_rule = Rule.sagemaker(
    base_config=rule_configs.vanishing_gradient()
)

import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    rules=[exploding_tensor_rule, vanishing_gradient_rule],
)
sagemaker_estimator.fit()
```

You can use these rules or write your own rules using the Amazon SageMaker Debugger APIs. You can also analyze raw tensor data without using rules in, for example, an Amazon SageMaker notebook, using Debugger's full set of APIs.

Programming Model for Debugger

The programming model is organized around three main constructs that characterize Amazon SageMaker Debugger training jobs: trial, tensor, mode, and rules.

- **Trial:** A `smdebugtrial` is an object which lets you query for tensors for a given Debugger training job, specified by the path where `smdebug`'s artifacts are saved. Trial is capable of loading new tensors as

and when they become available at the given path, allowing you to do both offline as well as realtime analysis. For information on methods in the smdebug Trial API, see [Trial API](#).

- **Tensor:** Tensors define the state of the training job at any API particular instant in its lifecycle. An smdebug Tensor object can be retrieved through the `trial.tensor(name)` API. It is uniquely identified by the string representing its name. For information on methods in the smdebug Tensor API, see [Tensor API](#). For information on how to enable or disable a refresh of tensors when querying a trial with `smdebug.analysis.utils`, see [Utils](#).
- **Rule:** Debugger uses rules to execute a certain piece of code regularly on different steps of the jobs. A rule is assigned to a trial and can be invoked at each new step of the trial. You can evaluate a rule using tensors from the current step or any step before the current step. You must ensure that your logic respects these semantics, else you get a `TensorUnavailableForStep` exception as the data would not yet be available for future steps. You can use the build-in rules that Debugger provides or you can write your own custom rules. For information on how to write a custom rule, see [How to Use Custom Rules \(p. 570\)](#). For information on how to use built-in rules, see [How to Use Built-in Rules for Model Analysis \(p. 568\)](#).

For more information on the Amazon SageMaker Debugger programming model see [Programming Model for Analysis](#)

How to Use Custom Rules

You can deploy a custom rule to monitor your training job either by using the [CreateTrainingJob](#) API or by using the open source [smdebug Python library](#) with the Amazon SageMaker Python SDK. The `smdebug` programming model provides the context for understanding this task. For information on the programming model, see [Analysis](#).

To run a custom rule, you have to provide a few additional parameters for the interface. Key parameters are the Python file that has the implementation of your Rule class, the name of the Rule class, the type of instance to run the Rule job on, the size of the volume on that instance, and the docker image to use for running this job.

Topics

- [Use the CreateTrainingJob API to Create a Custom Rule \(p. 570\)](#)
- [Use smdebug Python library with the Amazon SageMaker Python SDK to Create a Custom Rule \(p. 571\)](#)

Use the `CreateTrainingJob` API to Create a Custom Rule

A custom rule can be configured for a training job using the `DebugHookConfig` and `DebugRuleConfiguration` objects in the `CreateTrainingJob` API. The following code sample shows how to configure a custom `ImproperActivation` rule written with the `smdebug` library using this Amazon SageMaker API. This example assumes that you've written the custom rule in `custom_rules.py` file and uploaded it to an S3 bucket. We provide pre-built Docker images that you can use to run your custom rules. These are listed at [Amazon SageMaker Custom Rule Evaluator Registry Ids \(p. 544\)](#). You specify the URL registry address for the pre-built Docker image in the `RuleEvaluatorImage` parameter.

```
DebugHookConfig: {
    "S3OutputPath": "s3://bucket/",
    "CollectionConfigurations": [
        {
            "CustomCollection": "",
            "CollectionParameters": {
                "include_regex": "relu",
                "save_interval": "500",
```

```

        "end_step": "5000"
    }
}
],
DebugRulesConfigurations: [
{
    "RuleConfigurationName": "improper_activation_job",
    "RuleEvaluatorImage": "552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-
debugger-rule-evaluator:latest",
    "InstanceType": "ml.c4.xlarge",
    "VolumeSizeInGB": 400,
    "RuleParameters": {
        "source_s3_uri": "s3://bucket/custom_rules.py",
        "rule_to_invoke": "ImproperActivation",
        "collection_names": "relu_activations"
    }
}
]

```

Use *smdebug* Python library with the Amazon SageMaker Python SDK to Create a Custom Rule

For a sample notebook that shows you how to use a custom rule to monitor your training job with a tf.keras ResNet example, see [Amazon SageMaker - Debugging with custom rules](#).

The following code sample shows how to configure a custom `ImproperActivation` rule using the open source [smdebug Python library](#) with the Amazon SageMaker Python SDK. This example assumes that the custom rule you've written has path `/rules/custom_rules.py`. You do not need to specify a pre-built Docker image when using the Amazon SageMaker Python SDK as the `Estimators` handle this task.

```

from sagemaker.debugger import Rule, CollectionConfig

custom_coll = CollectionConfig(
    name="relu_activations",
    parameters={
        "include_regex": "relu",
        "save_interval": "500",
        "end_step": "5000"
    })
improper_activation_rule = Rule.custom(
    name='improper_activation_job',
    image_uri='552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rule-
evaluator:latest',
    instance_type='ml.c4.xlarge',
    volume_size_in_gb=400,
    source='rules/custom_rules.py',
    rule_to_invoke='ImproperActivation',
    rule_parameters={"collection_names": "relu_activations"},
    collections_to_save=[custom_coll]
)

import sagemaker as sm
sagemaker_estimator = sm.tensorflow.TensorFlow(
    entry_point='src/mnist.py',
    role=sm.get_execution_role(),
    base_job_name='smdebug-demo-job',
    train_instance_count=1,
    train_instance_type="ml.m4.xlarge",
    framework_version="1.15",
    py_version="py3",
    # smdebug-specific arguments below
    rules=[improper_activation_rule],
)

```

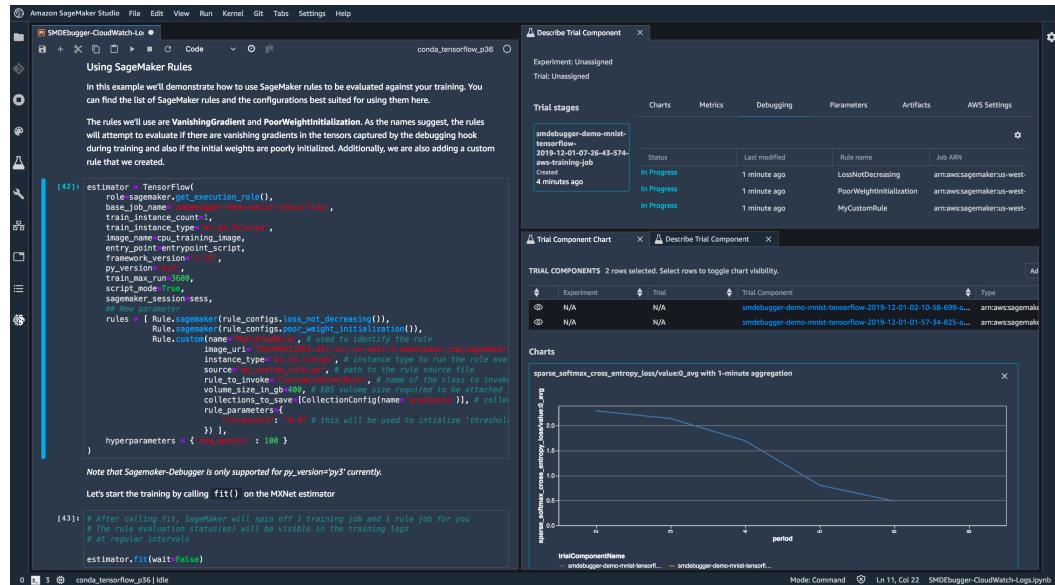
```
    }
sagemaker_estimator.fit()
```

Amazon SageMaker Studio Visualizations of Model Analysis Results

Amazon SageMaker Studio provides visualizations to interpret tensor outputs from model analysis.

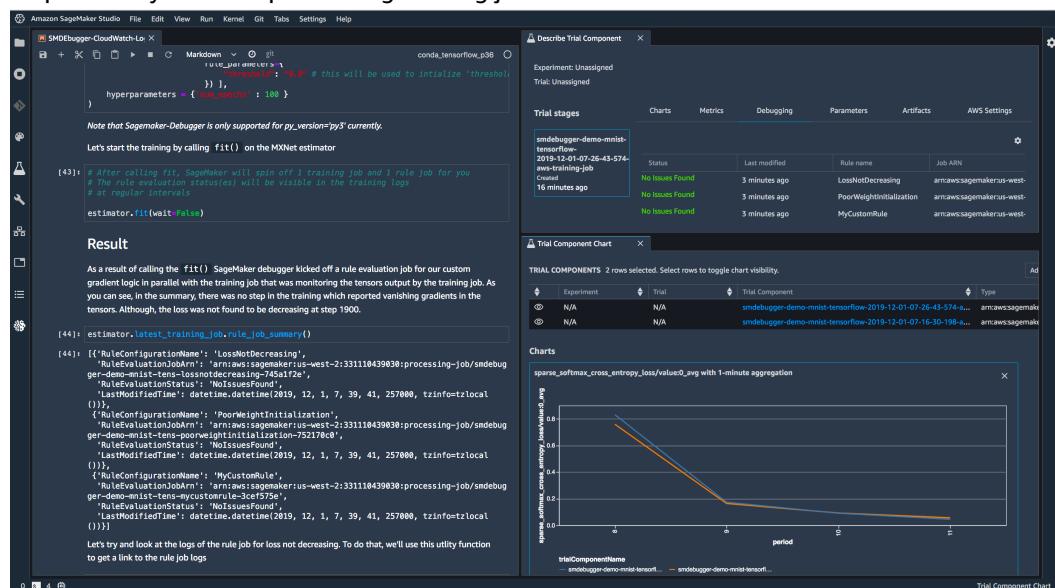
Loss curves while training is in progress

The following screenshot shows visualizations of loss curves for training. The training is in progress.



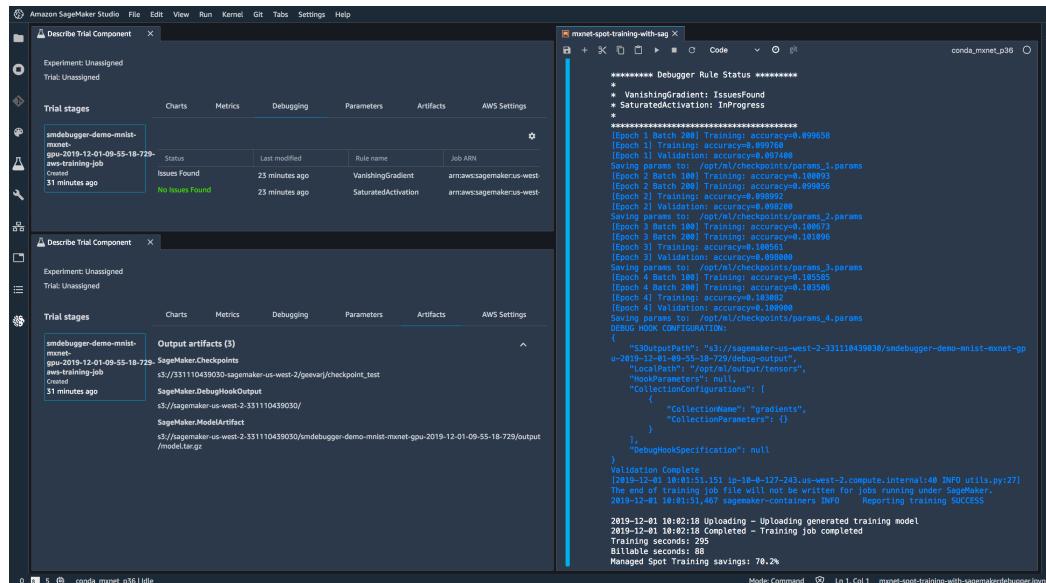
Analyzing training jobs - comparing loss curves across multiple jobs

Amazon SageMaker Studio allows simple comparison across multiple jobs - in this case the loss. This helps identify the best-performing training jobs.



Rules triggering + logs from Debugger Jobs

When rules are triggered for anomalous conditions, Amazon SageMaker Studio presents logs for the failing rule, allowing easy analysis of the causes of the condition.



Amazon SageMaker Debugger Reference Documentation

The following sections contain reference documentation on the APIs, exceptions and some known limitations for Debugger

Topics

- [Amazon SageMaker Debugger API \(p. 573\)](#)
- [Amazon SageMaker Debugger Exceptions \(p. 574\)](#)
- [Amazon SageMaker Debugger Best Practices \(p. 574\)](#)
- [Known Limitations with Amazon SageMaker Debugger \(p. 575\)](#)

Amazon SageMaker Debugger API

Amazon SageMaker Debugger has APIs in several locations that are used to implement its monitoring and analysis of model training.

Amazon SageMaker Debugger provides an open source *smdebug* Python Library at [awslabs/sagemaker-debugger](https://github.com/awslabs/sagemaker-debugger) that is used to configure built-in rules or to define custom rules used to analyze the tensor data from training jobs.

The [Amazon SageMaker Python SDK](#) is a high level SDK focused on machine learning experimentation. The SDK can be used to deploy built-in or custom rules defined with the *smdebug* Python Library to monitor and analyse these tensors using Amazon SageMaker estimators.

Debugger has added operations and types to the Amazon SageMaker API that enable the platform to use Debugger when training a model and to manage the configuration of inputs and outputs. Here is a list of the APIs to hook up Debugger.

- [CreateTrainingJob](#) and [DescribeTrainingJob](#) use the following types to configure tensor and rule configurations, hook up the *smdebug* library, and manage the storage of TensorBoard outputs:
 - [DebugRuleConfiguration](#)
 - [CollectionConfiguration](#)
 - [DebugHookConfig](#)
 - [TensorBoardOutputConfig](#)
- [DescribeTrainingJob](#) also provides an additional type to report on the status of rule evaluations:
 - [DebugRuleEvaluationStatus](#)
- [DescribeTrainingJob](#) also has parameters for accessing these Debugger configuration types as well as the time and billable time spent in model training.

Debugger also makes use of the Amazon SageMaker Processing functionality when analyzing model training. For more information on Processing, see [Process Data and Evaluate Models \(p. 240\)](#).

Amazon SageMaker Debugger Exceptions

Amazon SageMaker Debugger is designed to be aware that tensors required to execute a rule may not be available at every step. Hence it raises a few exceptions which allow us to control what happens when a tensor is missing. These are available in the [smdebug.exceptions module](#). You can import them as follows:

```
from smdebug.exceptions import *
```

Here are the exceptions and their meanings:

- [TensorUnavailableForStep](#): The tensor requested is not available for the step. This might mean that this step might not be saved at all by the hook, or that this step might have saved some tensors but the requested tensor is not part of them. Note that when you see this exception, it means that this tensor can never become available for this step in the future. If the tensor has reductions saved for the step, it notifies you they can be queried.
- [TensorUnavailable](#): This tensor is not being saved or has not been saved by the smdebug API. This means that this tensor will never be seen for any step in smdebug.
- [StepUnavailable](#): The step was not saved and Debugger has no data from the step.
- [StepNotYetAvailable](#): The step has not yet been seen by smdebug. It may be available in the future if the training is still going on. Debugger automatically loads new data as and when it becomes available.
- [NoMoreData](#): Raised when the training ends. Once you see this, you know that there are no more steps and no more tensors to be saved.
- [IndexReaderException](#): The index reader is not valid.
- [InvalidWorker](#): A worker was invoked that was not valid.
- [RuleEvaluationConditionMet](#): Evaluation of the rule at the step resulted in the condition being met"
- [InsufficientInformationForRuleInvocation](#): Insufficient information was provided to invoke the rule.

Amazon SageMaker Debugger Best Practices

Step Intervals: By default Python SDK uses step interval as 500 for emitting tensors. If you want to emit more frequently, choose specific tensors and run for fewer epochs to avoid stressing the CPU/disk.

Known Limitations with Amazon SageMaker Debugger

Here are the known limitations for Amazon SageMaker Debugger:

- **TensorFlow support:** It does not support TensorFlow 2.0.
- **Horovod support:** It does not support training on jobs that use distributed training with the TensorFlow Horovod container. (Distributed training with Horovod is supported for MXNet and PyTorch.)
- **Distributed training:** Parameter server-based distributed training is not supported for MXNet and TensorFlow.

Perform Automatic Model Tuning

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a [binary classification](#) problem on a marketing dataset. Your goal is to maximize the [area under the curve \(auc\)](#) metric of the algorithm by training an [XGBoost Algorithm \(p. 445\)](#) model. You don't know which values of the eta, alpha, min_child_weight, and max_depth hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that Amazon SageMaker hyperparameter tuning searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

You can use Amazon SageMaker automatic model tuning with built-in algorithms, custom algorithms, and Amazon SageMaker pre-built containers for machine learning frameworks.

Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

You should also prepare your dataset and algorithm so that they work in Amazon SageMaker and successfully run a training job at least once. For information about setting up and running a training job, see [Get Started with Amazon SageMaker \(p. 22\)](#).

Topics

- [How Hyperparameter Tuning Works \(p. 576\)](#)
- [Define Metrics \(p. 577\)](#)
- [Define Hyperparameter Ranges \(p. 578\)](#)
- [Example: Hyperparameter Tuning Job \(p. 580\)](#)
- [Stop Training Jobs Early \(p. 588\)](#)
- [Run a Warm Start Hyperparameter Tuning Job \(p. 590\)](#)
- [Automatic Model Tuning Resource Limits \(p. 593\)](#)
- [Best Practices for Hyperparameter Tuning \(p. 594\)](#)

How Hyperparameter Tuning Works

Random Search

In a random search, hyperparameter tuning chooses a random combination of values from within the ranges that you specify for hyperparameters for each training job it launches. Because the choice of hyperparameter values doesn't depend on the results of previous training jobs, you can run the maximum number of concurrent training jobs without affecting the performance of the search.

For an example notebook that uses random search, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.

Bayesian Search

Bayesian search treats hyperparameter tuning like a [\[regression\]](#) problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Hyperparameter tuning uses an Amazon SageMaker implementation of Bayesian optimization.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

For more information about Bayesian optimization, see the following:

Basic Topics on Bayesian Optimization

- [A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning](#)
- [Practical Bayesian Optimization of Machine Learning Algorithms](#)
- [Taking the Human Out of the Loop: A Review of Bayesian Optimization](#)

Speeding up Bayesian Optimization

- [Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization](#)
- [Google Vizier: A Service for Black-Box Optimization](#)
- [Learning Curve Prediction with Bayesian Neural Networks](#)
- [Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves](#)

Advanced Modeling and Transfer Learning

- [Scalable Hyperparameter Transfer Learning](#)
- [Bayesian Optimization with Tree-structured Dependencies](#)
- [Bayesian Optimization with Robust Bayesian Neural Networks](#)

- Scalable Bayesian Optimization Using Deep Neural Networks
- Input Warping for Bayesian Optimization of Non-stationary Functions

Note

Hyperparameter tuning might not improve your model. It is an advanced tool for building machine solutions, and, as such, should be considered part of the scientific development process.

When you build complex machine learning systems like deep learning neural networks, exploring all of the possible combinations is impractical. Hyperparameter tuning can accelerate your productivity by trying many variations of a model, focusing on the most promising combinations of hyperparameter values within the ranges that you specify. To get good results, you need to choose the right ranges to explore. Because the algorithm itself is stochastic, it's possible that the hyperparameter tuning model will fail to converge on the best answer, even if the best possible combination of values is within the ranges that you choose.

Define Metrics

Note

When you use one of the Amazon SageMaker built-in algorithms, you don't need to define metrics. Built-in algorithms automatically send metrics to hyperparameter tuning. You do need to choose one of the metrics that the built-in algorithm emits as the objective metric for the tuning job. For a list of metrics that a built-in algorithm emits, see the *Metrics* table for the algorithm in [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).

To optimize hyperparameters for a machine learning model, a tuning job evaluates the training jobs it launches by using a metric that the training algorithm writes to logs. Amazon SageMaker hyperparameter tuning parses your algorithm's `stdout` and `stderr` streams to find algorithm metrics, such as loss or validation-accuracy, that show how well the model is performing on the dataset.

Note

These are the same metrics that Amazon SageMaker sends to CloudWatch Logs. For more information, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

If you use your own algorithm for hyperparameter tuning, make sure that your algorithm emits at least one metric by writing evaluation data to `stderr` or `stdout`.

Note

Hyperparameter tuning sends an additional hyperparameter, `_tuning_objective_metric` to the training algorithm. This hyperparameter specifies the objective metric being used for the hyperparameter tuning job, so that your algorithm can use that information during training.

You can define up to 20 metrics for your tuning job to monitor. You choose one of those metrics to be the objective metric, which hyperparameter tuning uses to evaluate the training jobs. The hyperparameter tuning job returns the training job that returned the best value for the objective metric as the best training job.

You define metrics for a tuning job by specifying a name and a regular expression for each metric that your tuning job monitors. Design the regular expressions to capture the values of metrics that your algorithm emits. You pass these metrics to the `CreateHyperParameterTuningJob` operation in the `TrainingJobDefinition` parameter as the `MetricDefinitions` field of the `AlgorithmSpecification` field.

The following example defines 4 metrics:

```
=[
  {
    "Name": "loss",
    "Regex": "Loss = (.*)%;"}
```

```

},
{
    "Name": "ganloss",
    "Regex": "GAN_loss=(.*?);",
},
{
    "Name": "discloss",
    "Regex": "disc_train_loss=(.*?);",
},
{
    "Name": "disc-combined",
    "Regex": "disc-combined=(.*?);",
}
]

```

The following is an example of the log that the algorithm writes:

```

GAN_loss=0.138318; Scaled_reg=2.654134; disc:[-0.017371,0.102429] real 93.3% gen 0.0%
disc-combined=0.000000; disc_train_loss=1.374587; Loss = 16.020744; Iteration 0 took
0.704s; Elapsed=0s

```

Use the regular expression (regex) to match the algorithm's log output and capture the numeric values of metrics. For example, in the regex for the loss metric defined above, the first part of the regex finds the exact text "Loss = ", and the expression `(.*?);` captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, `.` means any character, `*` means zero or more, and `?` means capture only until the first instance of the `;` character.

Choose one of the metrics that you define as the objective metric for the tuning job. If you are using the API, specify the value of the `name` key in the `HyperParameterTuningJobObjective` field of the `HyperParameterTuningJobConfig` parameter that you send to the [CreateHyperParameterTuningJob](#) operation.

Define Hyperparameter Ranges

Hyperparameter tuning finds the best hyperparameter values for your model by searching over ranges of hyperparameters. You specify the hyperparameters and range of values over which to search by defining hyperparameter ranges for your tuning job. Choosing hyperparameters and ranges significantly affects the performance of your tuning job. For guidance on choosing hyperparameters and ranges, see [Best Practices for Hyperparameter Tuning \(p. 594\)](#).

To define hyperparameter ranges by using the low-level API, you specify the names of hyperparameters and ranges of values in the `ParameterRanges` field of the `HyperParameterTuningJobConfig` parameter that you pass to the [CreateHyperParameterTuningJob](#) operation. The `ParameterRanges` field has three subfields, one for each of the categorical, integer, and continuous hyperparameter ranges. You can define up to 20 hyperparameters to search over. Each value of a categorical hyperparameter range counts as a hyperparameter against the limit. Hyperparameter ranges have the following structure:

```

"ParameterRanges": {
    "CategoricalParameterRanges": [
        {
            "Name": "tree_method",
            "Values": ["auto", "exact", "approx", "hist"]
        }
    ],
    "ContinuousParameterRanges": [
        {
            "Name": "eta",

```

```
        "MaxValue" : "0.5",
        "MinValue": "0",
        "ScalingType": "Auto"
    }
],
"IntegerParameterRanges": [
{
    "Name": "max_depth",
    "MaxValue": "10",
    "MinValue": "1",
    "ScalingType": "Auto"
}
]
```

Hyperparameter Scaling

For integer and continuous hyperparameter ranges, you can choose the scale you want hyperparameter tuning to use to search the range of values by specifying a value for the `ScalingType` field of the hyperparameter range. You can choose from the following scaling types:

Auto

Amazon SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale.

Typically, you choose this if the range of all values from the lowest to the highest is relatively small (within one order of magnitude), because uniformly searching values from the range will give you a reasonable exploration of the entire range.

Logarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

Choose logarithmic scaling when you are searching a range that spans several orders of magnitude. For example, if you are tuning a [Tune a Linear Learner Model \(p. 365\)](#) model, and you specify a range of values between .0001 and 1.0 for the `learning_rate` hyperparameter, searching uniformly on a logarithmic scale gives you a better sample of the entire range than searching on a linear scale would, because searching on a linear scale would, on average, devote 90 percent of your training budget to only the values between .1 and 1.0, leaving only 10 percent of your training budget for the values between .0001 and .1.

ReverseLogarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale. reverse logarithmic scaling is supported only for continuous hyperparameter ranges. It is not supported for integer hyperparameter ranges.

Reverse logarithmic scaling works only for ranges that are entirely within the range $0 \leq x < 1.0$.

Choose reverse logarithmic scaling when you are searching a range that is highly sensitive to small changes that are very close to 1.

For an example notebook that uses hyperparameter scaling, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb.

Example: Hyperparameter Tuning Job

This example shows how to create a new notebook for configuring and launching a hyperparameter tuning job. The tuning job uses the [XGBoost Algorithm \(p. 445\)](#) to train a model to predict whether a customer will enroll for a term deposit at a bank after being contacted by phone.

You use the low-level AWS SDK for Python (Boto) to configure and launch the hyperparameter tuning job, and the AWS Management Console to monitor the status of hyperparameter training jobs. You can also use the Amazon SageMaker high-level Amazon SageMaker Python SDK to configure, run, monitor, and analyze hyperparameter tuning jobs. For more information, see <https://github.com/aws/sagemaker-python-sdk>.

Prerequisites

To run the code in this example, you need

- An AWS account and an administrator user (p. 15)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training (p. 40)
- A running Amazon SageMaker notebook instance (p. 41)

Topics

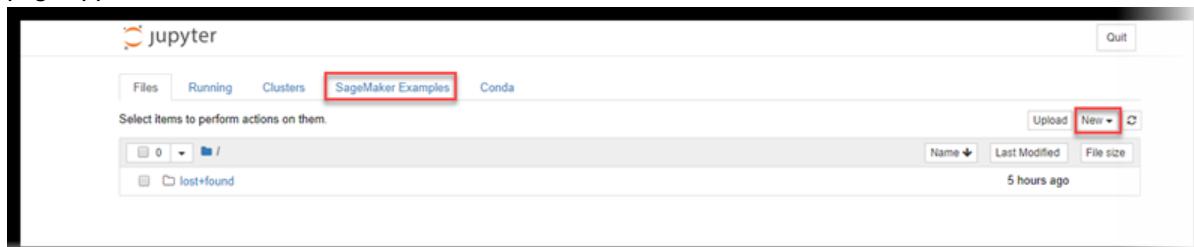
- [Create a Notebook \(p. 580\)](#)
- [Get the Amazon Sagemaker Boto 3 Client \(p. 581\)](#)
- [Get the Amazon SageMaker Execution Role \(p. 581\)](#)
- [Specify a Bucket and Data Output Location \(p. 581\)](#)
- [Download, Prepare, and Upload Training Data \(p. 582\)](#)
- [Configure and Launch a Hyperparameter Tuning Job \(p. 583\)](#)
- [Monitor the Progress of a Hyperparameter Tuning Job \(p. 586\)](#)
- [Clean up \(p. 588\)](#)

Create a Notebook

Create a Jupyter notebook that contains a preinstalled environment with the default Anaconda installation and Python3.

To create a Jupyter notebook

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Open a running notebook instance, by choosing **Open** next to its name. The Jupyter notebook server page appears:



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

4. Name the notebook.

Next Step

[Get the Amazon Sagemaker Boto 3 Client \(p. 581\)](#)

Get the Amazon Sagemaker Boto 3 Client

Import libraries and get a Boto3 client, which you use to call the hyperparameter tuning APIs.

In the new Jupyter notebook, type the following code:

```
import sagemaker
import boto3
from sagemaker.predictor import csv_serializer      # Converts strings for HTTP POST requests
                                                    on inference

import numpy as np                                  # For performing matrix operations and
                                                    numerical processing
import pandas as pd                                # For manipulating tabular data
from time import gmtime, strftime
import os

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

Next Step

[Get the Amazon SageMaker Execution Role \(p. 581\)](#)

Get the Amazon SageMaker Execution Role

Get the execution role for the notebook instance. This is the IAM role that you created when you created your notebook instance. You pass the role to the tuning job.

```
from sagemaker import get_execution_role

role = get_execution_role()
print(role)
```

Next Step

[Specify a Bucket and Data Output Location \(p. 581\)](#)

Specify a Bucket and Data Output Location

Specify the name of the Amazon S3 bucket where you want to store the output of the training jobs that the tuning job launches. The name of the bucket must contain **sagemaker**, and be globally unique. The bucket must be in the same AWS Region as the notebook instance that you use for this example. You can use the bucket that you created when you set up Amazon SageMaker, or you can create a new bucket. For information, see [Step 1: Create an Amazon S3 Bucket \(p. 40\)](#).

Note

The name of the bucket doesn't need to contain **sagemaker** if the role that you use to run the hyperparameter tuning job has a policy that gives the SageMaker service principle **S3FullAccess** permission.

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

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

Next Step

[Download, Prepare, and Upload Training Data \(p. 582\)](#)

Download, Prepare, and Upload Training Data

For this example, you use a training dataset of information about bank customers that includes the customer's job, marital status, and how they were contacted during the bank's direct marketing campaign. To use a dataset for a hyperparameter tuning job, you download it, transform the data, and then upload it to an Amazon S3 bucket.

For more information about the dataset and the data transformation that the example performs, see the `hpo_xgboost_direct_marketing_sagemaker_APIs` notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** tab in your notebook instance.

Download and Explore the Training Dataset

To download and explore the dataset, run the following code in your notebook:

```
!wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-
additional.zip
!unzip -o bank-additional.zip
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)      # Make sure we can see all of the columns
pd.set_option('display.max_rows', 5)            # Keep the output on one page
data
```

Prepare and Upload Data

Before creating the hyperparameter tuning job, prepare the data and upload it to an S3 bucket where the hyperparameter tuning job can access it.

Run the following code in your notebook:

```
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)
    # Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'unemployed']), 1, 0)
    # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)
    # Convert categorical variables to sets of indicators
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx',
    'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)

train_data, validation_data, test_data = np.split(model_data.sample(frac=1,
    random_state=1729), [int(0.7 * len(model_data)), int(0.9*len(model_data))])

pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)],
    axis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)],
    axis=1).to_csv('validation.csv', index=False, header=False)
```

```
pd.concat([test_data['y_yes'], test_data.drop(['y_no', 'y_yes'], axis=1)],  
          axis=1).to_csv('test.csv', index=False, header=False)  
  
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/  
train.csv')).upload_file('train.csv')  
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/  
validation.csv')).upload_file('validation.csv')
```

Next Step

[Configure and Launch a Hyperparameter Tuning Job \(p. 583\)](#)

Configure and Launch a Hyperparameter Tuning Job

To configure and launch a hyperparameter tuning job, complete the following steps.

Topics

- [Specify the Hyperparameter Tuning Job Settings \(p. 583\)](#)
- [Configure the Training Jobs \(p. 584\)](#)
- [Name and Launch the Hyperparameter Tuning Job \(p. 585\)](#)
- [Next Step \(p. 586\)](#)

Specify the Hyperparameter Tuning Job Settings

To specify settings for the hyperparameter tuning job, you define a JSON object. You pass the object as the value of the `HyperParameterTuningJobConfig` parameter to `CreateHyperParameterTuningJob` when you create the tuning job.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune. For more information, see [Define Hyperparameter Ranges \(p. 578\)](#)
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An *objective metric* is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

Note

To use your own algorithm for hyperparameter tuning, you need to define metrics for your algorithm. For information, see [Define Metrics \(p. 577\)](#).

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the [XGBoost Algorithm \(p. 445\)](#) built-in algorithm. The objective metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```
tuning_job_config = {  
    "ParameterRanges": {  
        "CategoricalParameterRanges": [],  
        "ContinuousParameterRanges": [  
            {  
                "MaxValue": "1",  
                "MinValue": "0",  
                "Name": "eta"  
            },  
            {  
                "MaxValue": "1",  
                "MinValue": "0",  
                "Name": "alpha"  
            }  
        ]  
    }  
}
```

```

        "MaxValue": "2",
        "MinValue": "0",
        "Name": "alpha"
    },
    {
        "MaxValue": "10",
        "MinValue": "1",
        "Name": "min_child_weight"
    }
],
"IntegerParameterRanges": [
    {
        "MaxValue": "10",
        "MinValue": "1",
        "Name": "max_depth"
    }
]
},
"ResourceLimits": {
    "MaxNumberOfTrainingJobs": 20,
    "MaxParallelTrainingJobs": 3
},
"Strategy": "Bayesian",
"HyperParameterTuningJobObjective": {
    "MetricName": "validation:auc",
    "Type": "Maximize"
}
}

```

Configure the Training Jobs

To configure the training jobs that the tuning job launches, define a JSON object that you pass as the value of the `TrainingJobDefinition` parameter of the [CreateHyperParameterTuningJob](#) call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.

Note

Define metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don't specify metrics. For information about defining metrics, see [Define Metrics \(p. 577\)](#).

- The container image that specifies the training algorithm.
- The input configuration for your training and test data.
- The storage location for the algorithm's output. Specify the S3 bucket where you want to store the output of the training jobs.
- The values of algorithm hyperparameters that are not tuned in the tuning job.
- The type of instance to use for the training jobs.
- The stopping condition for the training jobs. This is the maximum duration for each training job.

In this example, we set static values for the `eval_metric`, `num_round`, `objective`, `rate_drop`, and `tweedie_variance_power` parameters of the [XGBoost Algorithm \(p. 445\)](#) built-in algorithm.

```

from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(boto3.Session().region_name, 'xgboost')

s3_input_train = 's3://{}//{}//train'.format(bucket, prefix)
s3_input_validation ='s3://{}//{}//validation/'.format(bucket, prefix)

```

```

training_job_definition = {
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train
                }
            }
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation
                }
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://{}{}/output".format(bucket,prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.2xlarge",
        "VolumeSizeInGB": 10
    },
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 43200
    }
}

```

Name and Launch the Hyperparameter Tuning Job

Now you can provide a name for the hyperparameter tuning job and then launch it by calling the [CreateHyperParameterTuningJob](#) API. Pass `tuning_job_config`, and `training_job_definition` that you created in previous steps as the values of the parameters.

```

tuning_job_name = "MyTuningJob"
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = tuning_job_name,
                                            HyperParameterTuningJobConfig =
tuning_job_config,
                                            TrainingJobDefinition = training_job_definition)

```

Next Step

[Monitor the Progress of a Hyperparameter Tuning Job \(p. 586\)](#)

Monitor the Progress of a Hyperparameter Tuning Job

To monitor the progress of a hyperparameter tuning job and the training jobs that it launches, use the Amazon SageMaker console.

Topics

- [View the Status of the Hyperparameter Tuning Job \(p. 586\)](#)
- [View the Status of the Training Jobs \(p. 586\)](#)
- [View the Best Training Job \(p. 587\)](#)

View the Status of the Hyperparameter Tuning Job

To view the status of the hyperparameter tuning job

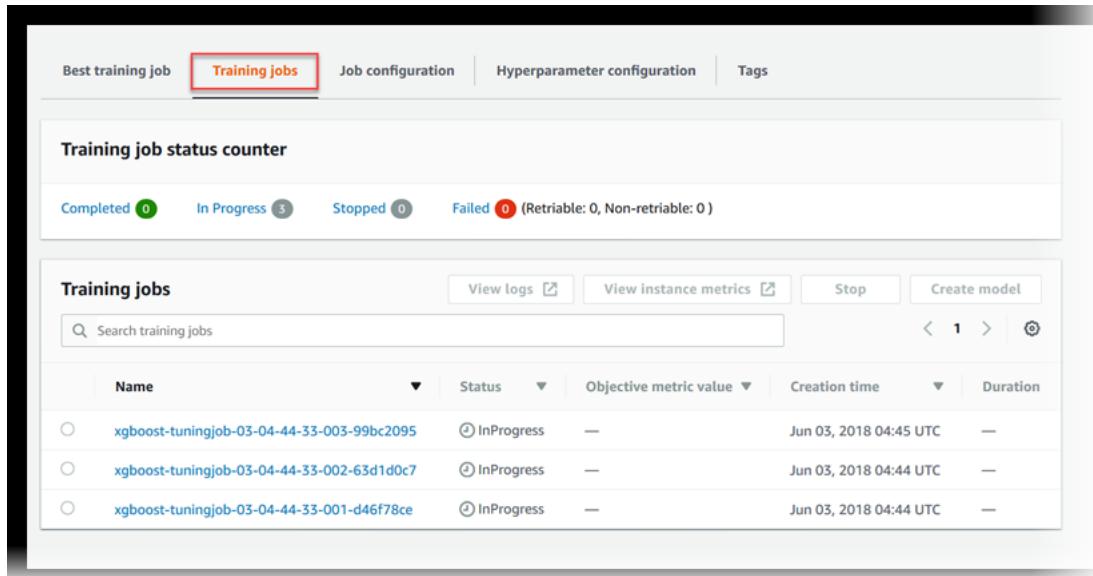
1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Hyperparameter tuning jobs**.

3. In the list of hyperparameter tuning jobs, check the status of the hyperparameter tuning job you launched. A tuning job can be:
 - **Completed**—The hyperparameter tuning job successfully completed.
 - **InProgress**—The hyperparameter tuning job is in progress. One or more training jobs are still running.
 - **Failed**—The hyperparameter tuning job failed.
 - **Stopped**—The hyperparameter tuning job was manually stopped before it completed. All training jobs that the hyperparameter tuning job launched are stopped.
 - **Stopping**—The hyperparameter tuning job is in the process of stopping.

View the Status of the Training Jobs

To view the status of the training jobs that the hyperparameter tuning job launched

1. In the list of hyperparameter tuning jobs, choose the job that you launched.
2. Choose **Training jobs**.



- View the status of each training job. To see more details about a job, choose it in the list of training jobs. To view a summary of the status of all of the training jobs that the hyperparameter tuning job launched, see **Training job status counter**.

A training job can be:

- Completed**—The training job successfully completed.
- InProgress**—The training job is in progress.
- Stopped**—The training job was manually stopped before it completed.
- Failed (Retriable)**—The training job failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
- Failed (Non-retriable)**—The training job failed and can't be retried. A failed training job can't be retried when a client error occurs.

View the Best Training Job

A hyperparameter tuning job uses the objective metric that each training job returns to evaluate training jobs. While the hyperparameter tuning job is in progress, the best training job is the one that has returned the best objective metric so far. After the hyperparameter tuning job is complete, the best training job is the one that returned the best objective metric.

To view the best training job, choose **Best training job**.

The screenshot shows the Amazon SageMaker console interface. At the top, there are tabs: 'Best training job' (highlighted in red), 'Training jobs', 'Job configuration', 'Hyperparameter configuration', and 'Tags'. Below the tabs, there's a section titled 'Best training job summary' containing a table with the following data:

Name xgboost-tuningjob-03-04-44-33-003-99bc2095	Status Completed	Objective metric validation:auc	Value 0.772566020488739
--	---------------------	------------------------------------	----------------------------

At the top right of this section is a button labeled 'Create model' with a red box drawn around it. Below this is another section titled 'Best training job hyperparameters' with a table:

Name	Type	Value
_tuning_objective_metric	Static	validation:auc
alpha	Continuous	1.9818243759579417
eta	Continuous	0.07404334782758304
eval_metric	Static	auc

To deploy the best training job as a model that you can host at an Amazon SageMaker endpoint, choose **Create model**.

Next Step

[Clean up \(p. 588\)](#)

Clean up

To avoid incurring unnecessary charges, when you are done with the example, use the AWS Management Console to delete the resources that you created for it.

Note

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

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/> and delete the notebook instance. Stop the instance before deleting it.
2. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/> and delete the bucket that you created to store model artifacts and the training dataset.
3. Open the IAM console at <https://console.aws.amazon.com/iam/> and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at <https://console.aws.amazon.com/cloudwatch/> and delete all of the log groups that have names starting with /aws/sagemaker/.

Stop Training Jobs Early

Stop the training jobs that a hyperparameter tuning job launches early when they are not improving significantly as measured by the objective metric. Stopping training jobs early can help reduce compute time and helps you avoid overfitting your model. To configure a hyperparameter tuning job to stop training jobs early, do one of the following:

- If you are using the AWS SDK for Python (Boto 3), set the `TrainingJobEarlyStoppingType` field of the `HyperParameterTuningJobConfig` object that you use to configure the tuning job to `AUTO`.

- If you are using the Amazon SageMaker Python SDK, set the `early_stopping_type` parameter of the [HyperParameterTuner](#) object to `Auto`.
- In the Amazon SageMaker console, in the **Create hyperparameter tuning job** workflow, under **Early stopping**, choose `Auto`.

For a sample notebook that demonstrates how to use early stopping, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_early_stopping/hpo_image_classification_early_stopping.ipynb or open the `hpo_image_classification_early_stopping.ipynb` notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** in a notebook instance. For information about using sample notebooks in a notebook instance, see [Use Example Notebooks \(p. 260\)](#).

How Early Stopping Works

When you enable early stopping for a hyperparameter tuning job, Amazon SageMaker evaluates each training job the hyperparameter tuning job launches as follows:

- After each epoch of training, get the value of the objective metric.
- Compute the running average of the objective metric for all previous training jobs up to the same epoch, and then compute the median of all of the running averages.
- If the value of the objective metric for the current training job is worse (higher when minimizing or lower when maximizing the objective metric) than the median value of running averages of the objective metric for previous training jobs up to the same epoch, Amazon SageMaker stops the current training job.

Algorithms That Support Early Stopping

To support early stopping, an algorithm must emit objective metrics for each epoch. The following built-in Amazon SageMaker algorithms support early stopping:

- [Linear Learner Algorithm \(p. 365\)](#)—Supported only if you use `objective_loss` as the objective metric.
- [XGBoost Algorithm \(p. 445\)](#)
- [Image Classification Algorithm \(p. 324\)](#)
- [Object Detection Algorithm \(p. 402\)](#)
- [Sequence-to-Sequence Algorithm \(p. 432\)](#)
- [IP Insights Algorithm \(p. 334\)](#)

Note

This list of built-in algorithms that support early stopping is current as of December 13, 2018. Other built-in algorithms might support early stopping in the future. If an algorithm emits a metric that can be used as an objective metric for a hyperparameter tuning job (preferably a validation metric), then it supports early stopping.

To use early stopping with your own algorithm, you must write your algorithms such that it emits the value of the objective metric after each epoch. The following list shows how you can do that in different frameworks:

TensorFlow

Use the `tf.contrib.learn.monitors.ValidationMonitor` class. For information, see https://www.tensorflow.org/api_docs/python/tf/contrib/learn/monitors.

MXNet

Use the `mxnet.callback.LogValidationMetricsCallback`. For information, see <https://mxnet.apache.org/api/python/callback/callback.html>.

Chainer

Extend chainer by using the `extensions.Evaluator` class. For information, see <https://docs.chainer.org/en/v1.24.0/reference/extensions.html#evaluator>.

PyTorch and Spark

There is no high-level support. You must explicitly write your training code so that it computes objective metrics and writes them to logs after each epoch.

Run a Warm Start Hyperparameter Tuning Job

Use warm start to start a hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job. Hyperparameter tuning uses either Bayesian or random search to choose combinations of hyperparameter values from ranges that you specify. For more information, see [How Hyperparameter Tuning Works \(p. 576\)](#). Using information from previous hyperparameter tuning jobs can help increase the performance of the new hyperparameter tuning job by making the search for the best combination of hyperparameters more efficient.

Note

Warm start tuning jobs typically take longer to start than standard hyperparameter tuning jobs, because the results from the parent jobs have to be loaded before the job can start. The increased time depends on the total number of training jobs launched by the parent jobs.

Reasons you might want to consider warm start include:

- You want to gradually increase the number of training jobs over several tuning jobs based on the results you see after each iteration.
- You get new data, and want to tune a model using the new data.
- You want to change the ranges of hyperparameters that you used in a previous tuning job, change static hyperparameters to tunable, or change tunable hyperparameters to static values.
- You stopped a previous hyperparameter job early or it stopped unexpectedly.

Topics

- [Types of Warm Start Tuning Jobs \(p. 590\)](#)
- [Warm Start Tuning Restrictions \(p. 591\)](#)
- [Warm Start Tuning Sample Notebook \(p. 592\)](#)
- [Create a Warm Start Tuning Job \(p. 592\)](#)

Types of Warm Start Tuning Jobs

There are two different types of warm start tuning jobs:

IDENTICAL_DATA_AND_ALGORITHM

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable

hyperparameters must remain the same as it is in all parent jobs. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed.

Use identical data and algorithm when you use the same training data as you used in a previous hyperparameter tuning job, but you want to increase the total number of training jobs or change ranges or values of hyperparameters.

When you run an warm start tuning job of type `IDENTICAL_DATA_AND_ALGORITHM`, there is an additional field in the response to [DescribeHyperParameterTuningJob](#) named `OverallBestTrainingJob`. The value of this field is the [TrainingJobSummary](#) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

TRANSFER LEARNING

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The training algorithm image can also be a different version from the version used in the parent hyperparameter tuning job. When you use transfer learning, changes in the dataset or the algorithm that significantly affect the value of the objective metric might reduce the usefulness of using warm start tuning.

Warm Start Tuning Restrictions

The following restrictions apply to all warm start tuning jobs:

- A tuning job can have a maximum of 5 parent jobs, and all parent jobs must be in a terminal state (`Completed`, `Stopped`, or `Failed`) before you start the new tuning job.
- The objective metric used in the new tuning job must be the same as the objective metric used in the parent jobs.
- The total number of static plus tunable hyperparameters must remain the same between parent jobs and the new tuning job. Because of this, if you think you might want to use a hyperparameter as tunable in a future warm start tuning job, you should add it as a static hyperparameter when you create a tuning job.
- The type of each hyperparameter (continuous, integer, categorical) must not change between parent jobs and the new tuning job.
- The number of total changes from tunable hyperparameters in the parent jobs to static hyperparameters in the new tuning job, plus the number of changes in the values of static hyperparameters cannot be more than 10. Each value in a categorical hyperparameter counts against this limit. For example, if the parent job has a tunable categorical hyperparameter with the possible values `red` and `blue`, you change that hyperparameter to static in the new tuning job, that counts as 2 changes against the allowed total of 10. If the same hyperparameter had a static value of `red` in the parent job, and you change the static value to `blue` in the new tuning job, it also counts as 2 changes.
- Warm start tuning is not recursive. For example, if you create `MyTuningJob3` as a warm start tuning job with `MyTuningJob2` as a parent job, and `MyTuningJob2` is itself an warm start tuning job with a parent job `MyTuningJob1`, the information that was learned when running `MyTuningJob1` is not used for `MyTuningJob3`. If you want to use the information from `MyTuningJob1`, you must explicitly add it as a parent for `MyTuningJob3`.
- The training jobs launched by every parent job in a warm start tuning job count against the 500 maximum training jobs for a tuning job.
- Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

Warm Start Tuning Sample Notebook

For a sample notebook that shows how to use warm start tuning, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_warmstart/hpo_image_classification_warmstart.ipynb. For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Example Notebooks \(p. 260\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The warm start tuning example notebook is located in the **Hyperparameter tuning** section, and is named `hpo_image_classification_warmstart.ipynb`. To open a notebook, click on its **Use** tab and select **Create copy**.

Create a Warm Start Tuning Job

You can use either the low-level AWS SDK for Python (Boto 3) or the high-level Amazon SageMaker Python SDK to create a warm start tuning job.

Topics

- [Create a Warm Start Tuning Job \(Low-level Amazon SageMaker API for Python \(Boto 3\)\) \(p. 592\)](#)
- [Create a Warm Start Tuning Job \(Amazon SageMaker Python SDK\) \(p. 592\)](#)

Create a Warm Start Tuning Job (Low-level Amazon SageMaker API for Python (Boto 3))

To use warm start tuning, you specify the values of a `HyperParameterTuningJobWarmStartConfig` object, and pass that as the `WarmStartConfig` field in a call to `CreateHyperParameterTuningJob`.

The following code shows how to create a `HyperParameterTuningJobWarmStartConfig` object and pass it to `CreateHyperParameterTuningJob` job by using the low-level Amazon SageMaker API for Python (Boto 3).

Create the `HyperParameterTuningJobWarmStartConfig` object:

```
warm_start_config = {
    "ParentHyperParameterTuningJobs" : [
        {"HyperParameterTuningJobName" : 'MyParentTuningJob'}
    ],
    "WarmStartType" : "IdenticalDataAndAlgorithm"
}
```

Create the warm start tuning job:

```
smclient = boto3.Session().client('sagemaker')
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName =
    'MyWarmStartTuningJob',
    HyperParameterTuningJobConfig = tuning_job_config, # See notebook for tuning
    configuration
    TrainingJobDefinition = training_job_definition, # See notebook for job definition
    WarmStartConfig = warm_start_config)
```

Create a Warm Start Tuning Job (Amazon SageMaker Python SDK)

To use the Amazon SageMaker Python SDK to run a warm start tuning job, you:

- Specify the parent jobs and the warm start type by using a `WarmStartConfig` object.
- Pass the `WarmStartConfig` object as the value of the `warm_start_config` argument of a `HyperparameterTuner` object.

- Call the `fit` method of the `HyperparameterTuner` object.

For more information about using the Amazon SageMaker Python SDK for hyperparameter tuning, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning>.

This example uses an estimator that uses the [Image Classification Algorithm \(p. 324\)](#) algorithm for training. The following code sets the hyperparameter ranges that the warm start tuning job searches within to find the best combination of values. For information about setting hyperparameter ranges, see [Define Hyperparameter Ranges \(p. 578\)](#).

```
hyperparameter_ranges = {'learning_rate': ContinuousParameter(0.0, 0.1),
                         'momentum': ContinuousParameter(0.0, 0.99)}
```

The following code configures the warm start tuning job by creating a `WarmStartConfig` object.

```
from sagemaker.tuner import WarmStartConfig,
                           WarmStartTypes

parent_tuning_job_name = "MyParentTuningJob"
warm_start_config = WarmStartConfig(type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
                                     parents={parent_tuning_job_name})
```

Now set the values for static hyperparameters, which are hyperparameters that keep the same value for every training job that the warm start tuning job launches. In the following code, `imageclassification` is an estimator that was created previously.

```
imageclassification.set_hyperparameters(num_layers=18,
                                         image_shape='3,224,224',
                                         num_classes=257,
                                         num_training_samples=15420,
                                         mini_batch_size=128,
                                         epochs=30,
                                         optimizer='sgd',
                                         top_k='2',
                                         precision_dtype='float32',
                                         augmentation_type='crop')
```

Now create the `HyperparameterTuner` object and pass the `WarmStartConfig` object that you previously created as the `warm_start_config` argument.

```
tuner_warm_start = HyperparameterTuner(imageclassification,
                                         'validation:accuracy',
                                         hyperparameter_ranges,
                                         objective_type='Maximize',
                                         max_jobs=10,
                                         max_parallel_jobs=2,
                                         base_tuning_job_name='warmstart',
                                         warm_start_config=warm_start_config)
```

Finally, call the `fit` method of the `HyperparameterTuner` object to launch the warm start tuning job.

```
tuner_warm_start.fit(
    {'train': s3_input_train, 'validation': s3_input_validation},
    include_cls_metadata=False)
```

Automatic Model Tuning Resource Limits

Amazon SageMaker sets default limits for the following resources:

- Number of concurrent hyperparameter tuning jobs - 100
- Number of hyperparameters that can be searched - 20

Note

Every possible value in a categorical hyperparameter counts against this limit.

- Number of metrics defined per hyperparameter tuning job - 20
- Number of concurrent training jobs per hyperparameter tuning job - 10
- Number of training jobs per hyperparameter tuning job - 500
- Maximum run time for a hyperparameter tuning job - 30 days

When you plan hyperparameter tuning jobs, you also have to take the limits on training resources into account. For information about the default resource limits for Amazon SageMaker training jobs, see [Amazon SageMaker Limits](#). Every concurrent training instance that all of your hyperparameter tuning jobs run on count against the total number of training instances allowed. For example, suppose you run 10 concurrent hyperparameter tuning jobs. Each of those hyperparameter tuning jobs runs 100 total training jobs, and runs 20 concurrent training jobs. Each of those training jobs runs on one **ml.m4.xlarge** instance. The following limits apply:

- Number of concurrent hyperparameter tuning jobs - You don't need to increase the limit, because 10 tuning jobs is below the limit of 100.
- Number of training jobs per hyperparameter tuning job - You don't need to increase the limit, because 100 training jobs is below the limit of 500.
- Number of concurrent training jobs per hyperparameter tuning job - You need to request a limit increase to 20, because the default limit is 10.
- Amazon SageMaker training **ml.m4.xlarge** instances - You need to request limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.
- Amazon SageMaker training total instance count - You need to request a limit increase to 200, because you have 10 hyperparameter tuning jobs, with each of them running 20 concurrent training jobs. The default limit is 20 instances.

For information about requesting limit increases for AWS resources, see [AWS Service Limits](#).

Best Practices for Hyperparameter Tuning

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

Topics

- [Choosing the Number of Hyperparameters \(p. 594\)](#)
- [Choosing Hyperparameter Ranges \(p. 595\)](#)
- [Using Logarithmic Scales for Hyperparameters \(p. 595\)](#)
- [Choosing the Best Number of Concurrent Training Jobs \(p. 595\)](#)
- [Running Training Jobs on Multiple Instances \(p. 595\)](#)

Choosing the Number of Hyperparameters

The difficulty of a hyperparameter tuning job depends primarily on the number of hyperparameters that Amazon SageMaker has to search. Although you can simultaneously use up to 20 variables in a hyperparameter tuning job, limiting your search to a much smaller number is likely to give better results.

Choosing Hyperparameter Ranges

The range of values for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. Although you might want to specify a very large range that covers every possible value for a hyperparameter, you will get better results by limiting your search to a small range of values. If you get the best metric values within a part of a range, consider limiting the range to that part.

Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, Amazon SageMaker attempts to figure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they should be log-scaled, it might take some time for Amazon SageMaker to discover that. If you know that a hyperparameter should be log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

Choosing the Best Number of Concurrent Training Jobs

Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.

Running Training Jobs on Multiple Instances

When a training job runs on multiple instances, hyperparameter tuning uses the last-reported objective metric value from all instances of that training job as the value of the objective metric for that training job. Design distributed training jobs so that the objective metric reported is the one that you want.

Tune Multiple Algorithms to Find the Best Model

When you create a new hyperparameter optimization (HPO) job with Amazon SageMaker, you have the option of using the console or the API. You provide one or more job specifications for the different algorithms you're testing. These are called training definitions. Each training definition has a name, an algorithm source, metrics selection, an objective metric, and a configuration for a set of hyperparameter values. It also has a data configuration for setting up the input data channels for the algorithm you choose, and a setting for the output data location. You select the resources you want to use for the training run.

Topics

- [Get Started \(p. 595\)](#)
- [Managing Hyperparameter Tuning Jobs \(p. 596\)](#)
- [Create a new single or multi-algorithm HPO tuning job \(p. 596\)](#)

Get Started

Using Multi-Algorithm HPO

To use multi-algorithm HPO you must add more than one training definition to your hyperparameter tuning job. Each training definition holds the configuration options for each algorithm you want to try.

In the console, you add training definitions when you create the HPO tuning job by choosing **Add training definition**, and then following the configuration steps for each algorithm that you want to use.

When you start the configuration steps, please note that the warm start and early stopping features are not available with multi-algorithm HPO. If you want to use these features, you can only tune a single algorithm at a time.

If you're using an API request, instead of the single `TrainingJobDefinition`, you must provide a list of training definitions using `TrainingJobDefinitions`. You must use one or the other, not both.

Managing Hyperparameter Tuning Jobs

You can clone a job, add or edit tags, or create a new hyperparameter tuning job from the console. You can also use the search feature to find jobs by their name, creation time, and status.

Creating a Hyperparameter Tuning Job

To create a new job, open the Amazon SageMaker console, choose **Training**, choose **Hyperparameter tuning jobs**, and then choose **Create hyperparameter tuning job**.

For instructions on using the API to create a tuning job, see [Example: Hyperparameter Tuning Job](#).

Cloning an existing training Job

You can save time by cloning a training job, which copies all of the job's settings, including data channels, S3 bucket locations, algorithms, and the hyperparameter options.

To clone a training job

- On the **Training jobs** page or on the **Hyperparameter tuning jobs** page, choose **Actions** and then choose **Clone**.

Editing Tags

You enter tags as key-value pairs. Values are not required. You can use just the key. To see the keys associated with a job, choose the **Tags** tab on the tuning job's details page.

Create a new single or multi-algorithm HPO tuning job

Defining job settings

Your tuning job settings are applied across all of the algorithms in the HPO tuning job. Warm start and early stopping are available only when tuning a single algorithm. After you define the job settings you will create individual training definitions for each algorithm or variation you want to tune.

Warm Start

If you cloned this job, you can choose to use the results from a previous tuning job to improve the performance of this tuning job. This is the warm start feature and it is only available when tuning a single algorithm. When you choose this option, you can choose up to five previous hyperparameter tuning jobs to use. Alternatively, you can use transfer learning to add additional data to the parent tuning job. When you select this option, you choose one previous tuning job as the parent.

Warm start is compatible with tuning jobs created after October 1, 2018. For more information, see [Run a warm start job](#).

Early Stopping

[Early stopping stops training jobs](#) when they are unlikely to improve the current best objective metric of the hyperparameter tuning job. Like warm start, this feature is only available when tuning a single algorithm. This is an automatic feature without configuration options, and it's disabled by default.

Tuning Strategy

Tuning strategy can be either random or bayesian. It specifies how the automatic tuning searches over specified hyperparameter ranges. You specify the ranges in a later step. For more information, see [How Hyperparameter Tuning Works](#).

Training Definitions

You must provide at least one training definition for each training job. Each training definition specifies the configuration for an algorithm. To create several definitions for your training job you can clone a definition.

Name

Provide a unique name for the training definition.

Permissions

Amazon SageMaker requires permissions to call other services on your behalf. Choose an IAM role or let AWS create a role that has the `AmazonSageMakerFullAccess` IAM policy attached.

Optional Security Settings

The network isolation setting prevents the container from making any outbound network calls. This is required for AWS Marketplace machine learning offerings.

You can also choose to use a private VPC.

Note

Inter-container encryption is only available when creating job definitions from the API.

Algorithm Options

You can choose one of the built-in algorithms, your own algorithm, your own container with an algorithm, or you can subscribe to an algorithm from AWS Marketplace.

If you choose a built-in algorithm, it has the ECR image information prepopulated. If you choose your own container, you must specify the ECR image information. You can select the input mode for the algorithm as file or pipe. If you plan to supply your data using a .CSV file from Amazon S3, you should select the file.

Metrics

When you choose a built-in algorithm, metrics are provided for you. If you choose your own algorithm, you need to define your metrics.

Objective Metric

To find the best training job, set an objective metric and tuning type. After the training job is complete, you can view the tuning job detail page for a summary of the best training job found using this objective metric.

Hyperparameter Configuration

When you choose a built-in algorithm, hyperparameters' default values are set for you, using ranges that are optimized for the particular algorithm. You can change these values as you see fit. Instead of a range, you can set a fixed value for a hyperparameter by setting the parameter's type to `static`. Each algorithm has different required and optional parameters. For more information, see [best practices](#) and [ranges](#).

Input Data Configuration

Input data is defined by channels, each with their own source location (Amazon S3 or Amazon Elastic File System), compression, and format options. You can define up to 20 channels of input sources. If the algorithm you chose supports multiple input channels, you can specify those too.

For example, when using the [XGBoost churn prediction notebook](#), you could add two channels: train and validation.

Checkpoint Configuration

Checkpoints are periodically generated during training. You must choose an Amazon S3 location for the checkpoints to be saved. Checkpoints are used in metrics reporting, and are also used to resume managed spot training jobs.

Output Data Configuration

You must define an Amazon S3 location for the artifacts of the training job to be stored. You have the option of adding encryption to the output using an AWS Key Management Service (AWS KMS) key.

Resource Limits and Configuration

Each training definition can have a different resource configuration. You choose the instance type and number of nodes.

Finalizing the Job Settings

You can run parallel jobs and limit the total number of jobs. The number of parallel jobs should not exceed the number of nodes you have requested across all of your training definitions. The total number of jobs can't exceed the number of jobs that your definitions are expected to run.

Use Reinforcement Learning with Amazon SageMaker

Reinforcement learning (RL) is a machine learning technique that attempts to learn a strategy, called a policy, that optimizes an objective for an agent acting in an environment. For example, the agent might be a robot, the environment might be a maze, and the goal might be to successfully navigate the maze in the smallest amount of time. In RL, the agent takes an action, observes the state of the environment, and gets a reward based on the value of the current state of the environment. The goal is to maximize the long-term reward that the agent receives as a result of its actions. RL is well-suited for solving problems where an agent can make autonomous decisions.

Topics

- [Why is Reinforcement Learning Important? \(p. 598\)](#)
- [Markov Decision Process \(MDP\) \(p. 599\)](#)
- [Key Features of Amazon SageMaker RL \(p. 599\)](#)
- [Sample RL Workflow Using Amazon SageMaker RL \(p. 601\)](#)
- [RL Environments in Amazon SageMaker \(p. 602\)](#)
- [Distributed Training with Amazon SageMaker RL \(p. 603\)](#)
- [Hyperparameter Tuning with Amazon SageMaker RL \(p. 604\)](#)

Why is Reinforcement Learning Important?

RL is well-suited for solving large, complex problems. For example, supply chain management, HVAC systems, industrial robotics, game artificial intelligence, dialog systems, and autonomous vehicles. Because RL models learn by a continuous process of receiving rewards and punishments for every action taken by the agent, it is possible to train systems to make decisions under uncertainty and in dynamic environments.

Markov Decision Process (MDP)

RL is based on models called Markov Decision Processes (MDPs). An MDP consists of a series of time steps. Each time step consists of the following:

Environment

Defines the space in which the RL model operates. This can be either a real-world environment or a simulator. For example, if you train a physical autonomous vehicle on a physical road, that would be a real-world environment. If you train a computer program that models an autonomous vehicle driving on a road, that would be a simulator.

State

Specifies all information about the environment and past steps that is relevant to the future. For example, in an RL model in which a robot can move in any direction at any time step, then the position of the robot at the current time step is the state, because if we know where the robot is, it isn't necessary to know the steps it took to get there.

Action

What the agent does. For example, the robot takes a step forward.

Reward

A number that represents the value of the state that resulted from the last action that the agent took. For example, if the goal is for a robot to find treasure, the reward for finding treasure might be 5, and the reward for not finding treasure might be 0. The RL model attempts to find a strategy that optimizes the cumulative reward over the long term. This strategy is called a *policy*.

Observation

Information about the state of the environment that is available to the agent at each step. This might be the entire state, or it might be just a part of the state. For example, the agent in a chess-playing model would be able to observe the entire state of the board at any step, but a robot in a maze might only be able to observe a small portion of the maze that it currently occupies.

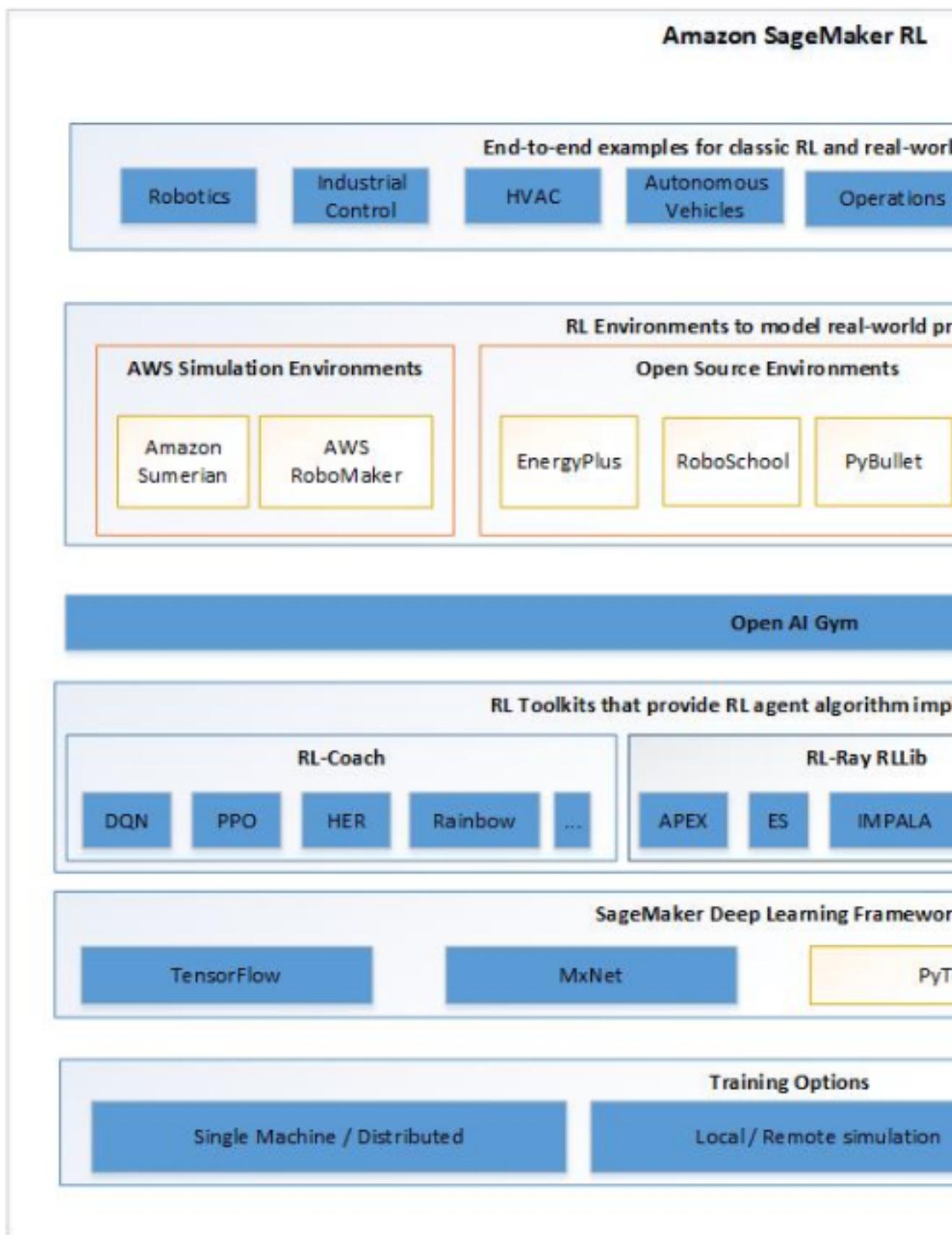
Typically, training in RL consists of many *episodes*. An episode consists of all of the time steps in an MDP from the initial state until the environment reaches the terminal state.

Key Features of Amazon SageMaker RL

To train RL models in Amazon SageMaker RL, use the following components:

- A deep learning (DL) framework. Currently, Amazon SageMaker supports RL in TensorFlow and Apache MXNet.
- An RL toolkit. An RL toolkit manages the interaction between the agent and the environment, and provides a wide selection of state of the art RL algorithms. Amazon SageMaker supports the Intel Coach and Ray RLLib toolkits. For information about Intel Coach, see <https://nervanasystems.github.io/coach/>. For information about Ray RLLib, see <https://ray.readthedocs.io/en/latest/rllib.html>.
- An RL environment. You can use custom environments, open-source environments, or commercial environments. For information, see [RL Environments in Amazon SageMaker \(p. 602\)](#).

The following diagram shows the RL components that are supported in Amazon SageMaker RL.



Sample RL Workflow Using Amazon SageMaker RL

The following example describes the steps for developing RL models using Amazon SageMaker RL.

For complete code examples, see the sample notebooks at <https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement-learning>.

1. **Formulate the RL problem**—First, formulate the business problem into an RL problem. For example, auto scaling enables services to dynamically increase or decrease capacity depending on conditions that you define. Currently, this requires setting up alarms, scaling policies, and thresholds, and other manual steps. To solve this with RL, we define the components of the Markov Decision Process:
 - a. **Objective**—Scale instance capacity so that it matches the desired load profile.
 - b. **Environment**—A custom environment that includes the load profile. It generates a simulated load with daily and weekly variations and occasional spikes. The simulated system has a delay between when new resources are requested and when they become available for serving requests.
 - c. **State**—The current load, number of failed jobs, and number of active machines
 - d. **Action**—Remove, add, or keep the same number of instances.
 - e. **Reward**—A positive reward for successful transactions, a high penalty for failing transactions beyond a specified threshold.
2. **Define the RL environment**—The RL environment can be the real world where the RL agent interacts or a simulation of the real world. You can connect open source and custom environments developed using Gym interfaces, and commercial simulation environments such as MATLAB and Simulink.
3. **Define the presets**—The presets configure the RL training jobs and define the hyperparameters for the RL algorithms.
4. **Write the training code**—Write training code as a Python script and pass the script to an Amazon SageMaker training job. In your training code, import the environment files and the preset files, and then define the `main()` function.
5. **Train the RL Model**—Use the Amazon SageMaker `RLEstimator` in the Amazon SageMaker Python SDK to start an RL training job. If you are using local mode, the training job runs on the notebook instance. When you use Amazon SageMaker for training, you can select GPU or CPU instances. Store the output from the training job in a local directory if you train in local mode, or on Amazon S3 if you use Amazon SageMaker training.

For information about using the Amazon SageMaker Python SDK for RL, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/rl/README.rst> .

The `RLEstimator` requires the following information as parameters.

- a. The source directory where the environment, presets, and training code are uploaded.
- b. The path to the training script.
- c. The RL toolkit and deep learning framework you want to use. This automatically resolves to the Amazon ECR path for the RL container.
- d. The training parameters, such as the instance count, job name, and S3 path for output.
- e. Metric definitions that you want to capture in your logs. These can also be visualized in CloudWatch and in Amazon SageMaker notebooks.
6. **Visualize training metrics and output**—After a training job that uses an RL model completes, you can view the metrics you defined in the training jobs in CloudWatch. You can also plot the metrics in a notebook by using the Amazon SageMaker Python SDK analytics library. Visualizing metrics helps you understand how the performance of the model as measured by the reward improves over time.

Note

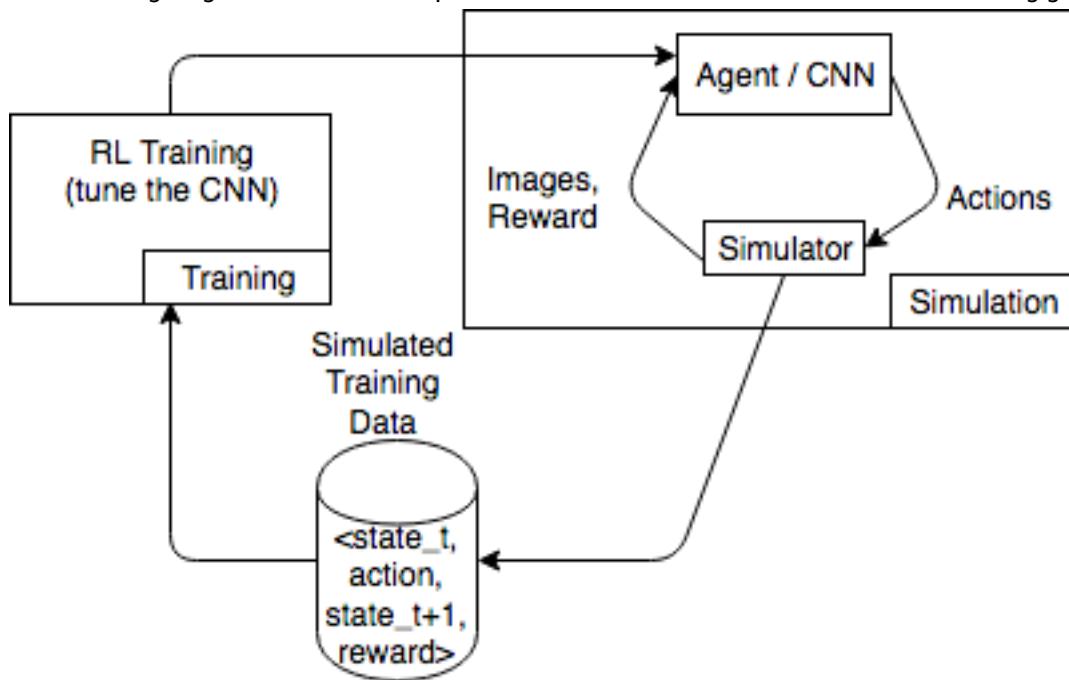
If you train in local mode, you can't visualize metrics in CloudWatch.

7. **Evaluate the model**—Checkpointed data from the previously trained models can be passed on for evaluation and inference in the checkpoint channel. In local mode, use the local directory. In Amazon SageMaker training mode, you need to upload the data to S3 first.
8. **Deploy RL models**—Finally, deploy the trained model on an endpoint hosted on Amazon SageMaker or on an Edge device by using AWS IoT Greengrass.

RL Environments in Amazon SageMaker

Amazon SageMaker RL uses environments to mimic real-world scenarios. Given the current state of the environment and an action taken by the agent or agents, the simulator processes the impact of the action, and returns the next state and a reward. Simulators are useful in cases where it is not safe to train an agent in the real world (for example, flying a drone) or if the RL algorithm takes a long time to converge (for example, when playing chess).

The following diagram shows an example of the interactions with a simulator for a car racing game.



The simulation environment consists of an agent and a simulator. Here, a convolutional neural network (CNN) consumes images from the simulator and generates actions to control the game controller. With multiple simulations, this environment generates training data of the form `state_t, action, state_t + 1, and reward_t+1`. Defining the reward is not trivial and impacts the RL model quality. We want to provide a few examples of reward functions, but would like to make it user-configurable.

Topics

- [Use OpenAI Gym Interface for Environments in Amazon SageMaker RL \(p. 603\)](#)
- [Use Open Source Environments \(p. 603\)](#)
- [Use Commercial Environments \(p. 603\)](#)

Use OpenAI Gym Interface for Environments in Amazon SageMaker RL

To use OpenAI Gym environments in Amazon SageMaker RL, use the following API elements. For more information about OpenAI Gym, see <https://gym.openai.com/docs/>.

- `env.action_space`—Defines the actions the agent can take, specifies whether each action is continuous or discrete, and specifies the minimum and maximum if the action is continuous.
- `env.observation_space`—Defines the observations the agent receives from the environment, as well as minimum and maximum for continuous observations.
- `env.reset()`—Initializes a training episode. The `reset()` function returns the initial state of the environment, and the agent uses the initial state to take its first action. The action is then sent to the `step()` repeatedly until the episode reaches a terminal state. When `step()` returns `done = True`, the episode ends. The RL toolkit re-initializes the environment by calling `reset()`.
- `step()`—Takes the agent action as input and outputs the next state of the environment, the reward, whether the episode has terminated, and an `info` dictionary to communicate debugging information. It is the responsibility of the environment to validate the inputs.
- `env.render()`—Used for environments that have visualization. The RL toolkit calls this function to capture visualizations of the environment after each call to the `step()` function.

Use Open Source Environments

You can use open source environments, such as EnergyPlus and RoboSchool, in Amazon SageMaker RL by building your own container. For more information about EnergyPlus, see <https://energyplus.net/>. For more information about RoboSchool, see <https://github.com/openai/roboschool>. The HVAC and RoboSchool examples in the samples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning show how to build a custom container to use with Amazon SageMaker RL:

Use Commercial Environments

You can use commercial environments, such as MATLAB and Simulink, in Amazon SageMaker RL by building your own container. You need to manage your own licenses.

Distributed Training with Amazon SageMaker RL

Amazon SageMaker RL supports multi-core and multi-instance distributed training. Depending on your use case, training and/or environment rollout can be distributed. For example, Amazon SageMaker RL works for the following distributed scenarios:

- Single training instance and multiple rollout instances of the same instance type. For an example, see the Neural Network Compression example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance and multiple rollout instances, where different instance types for training and rollouts. For an example, see the AWS DeepRacer / AWS RoboMaker example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.
- Single trainer instance that uses multiple cores for rollout. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning. This is useful if the simulation environment is light-weight and can run on a single thread.

- Multiple instances for training and rollouts. For an example, see the Roboschool example in the Amazon SageMaker examples repository at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning.

Hyperparameter Tuning with Amazon SageMaker RL

You can run a hyperparameter tuning job to optimize hyperparameters for Amazon SageMaker RL. The Roboschool example in the sample notebooks at https://github.com/awslabs/amazon-sagemaker-examples/tree/master/reinforcement_learning shows how you can do this with RL Coach. The launcher script shows how you can abstract parameters from the Coach preset file and optimize them.

Train a Deep Graph Network

In this overview, you learn how to get started with a deep graph network by using one of the DGL containers in Amazon Elastic Container Registry (Amazon ECR). You can also see links to practical examples for deep graph networks.

What Is a Deep Graph Network?

Deep graph networks refer to a type of neural network that is trained to solve graph problems. A deep graph network uses an underlying deep learning framework like PyTorch or MXNet. The potential for graph networks in practical AI applications are highlighted in the Amazon SageMaker tutorials for [Deep Graph Library](#) (DGL). Examples for training models on graph datasets include social networks, knowledge bases, biology, and chemistry.

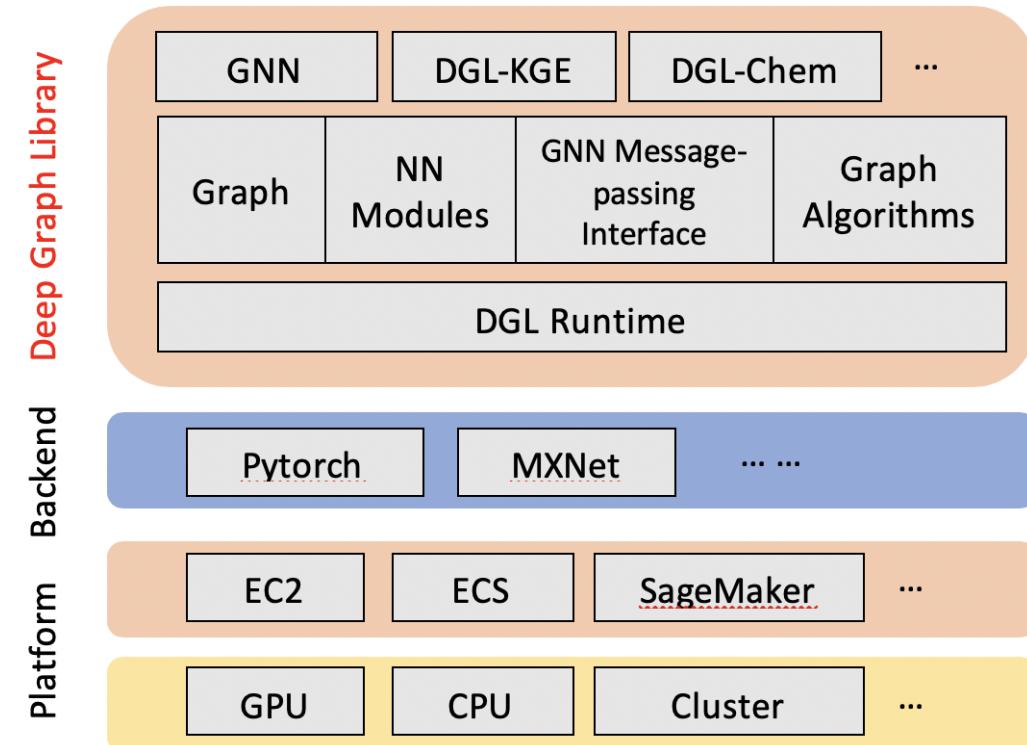


Figure 1. The DGL ecosystem

Several examples are provided using Amazon SageMaker's deep learning containers that are preconfigured with DGL. If you have special modules you want to use with DGL, you can also build your own container. The examples involve heterographs, which are graphs that have multiple types of nodes and edges, and draw on a variety of applications across disparate scientific fields, such as bioinformatics and social network analysis. DGL provides a wide array of [graph neural network implementations for different types models](#). Some of the highlights include:

- GCN - Graph convolutional network
- R-GCN - Relational graph convolutional network
- GAT - Graph attention network
- DGMG - Deep generative models of graphs
- JTNN - Junction tree neural network

Get Started

DGL is available as a deep learning container in Amazon ECR. You can select deep learning containers when you write your estimator function in an Amazon SageMaker notebook. You can also craft your own custom container with DGL by following the [Bring Your Own Container](#) guide. The easiest way to get started with a deep graph network uses one of the DGL containers in Amazon ECR.

Note

Backend framework support is limited to PyTorch and MXNet.

Setup

If you are using Amazon SageMaker Studio, you need to clone the examples repo first. If you are using a notebook instance, you can find the examples choosing the SageMaker icon at bottom of the left toolbar.

To clone the Amazon SageMaker SDK and notebook examples repository

1. From the Jupyter Lab view in Amazon SageMaker, go to the File Browser at the top of the left toolbar. From the file browser panel you can see a new navigation at the top of the panel.
2. Choose the icon on the far right to clone a git repository.
3. Add the repository URL: <https://github.com/awslabs/amazon-sagemaker-examples.git>
4. Browse the newly added folder and its contents. The DGL examples are stored in the sagemaker-python-sdk folder.

Run a Graph Network Training Example

To train a deep graph network

1. From the Jupyter Lab view in Amazon SageMaker, browse the [example notebooks](#) and look for dgl folders. Several files may be included to support an example. Examine the README for any prerequisites.
2. Run the .ipynb notebook example.
3. Find the estimator function, and note the line where it is using an Amazon ECR container for DGL and a specific instance type. You may want to update this to use a container in your preferred Region.
4. Run the function to launch the instance and use the DGL container for training a graph network. Charges are incurred for launching this instance. The instance self-terminates when the training is complete.

Examples

An example of knowledge graph embedding (KGE) is provided. It uses the Freebase dataset, a knowledge base of general facts. An example use case would be to graph the relationships of persons and predict their nationality.

An example implementation of a graph convolutional network (GCN) shows how you can train a graph network to predict toxicity. A physiology dataset, Tox21, provides toxicity measurements for how substances affect biological responses.

Another GCN example shows you how to train a graph network on a scientific publications bibliography dataset, known as Cora. You can use it to find relationships between authors, topics, and conferences.

The last example is a recommender system for movie reviews. It uses a graph convolutional matrix completion (GCMC) network trained on the MovieLens datasets. These datasets consist of movie titles, genres, and ratings by users.

Use a Deep Learning Container with DGL

The following examples use preconfigured deep learning containers. These are the easiest to try since they work out-of-the-box on Amazon SageMaker.

- [Semi-supervised classification of a knowledge base using a GCN](#)
- [Learning embeddings of large-scale knowledge graphs using a dataset of scientific publications](#)

Bring Your Own Container with DGL

The following examples enable you to bring your own container (BYOC). Read the [BYOC guide](#) and familiarize yourself with that process before trying these. Configuration is required.

- [Molecular property prediction of toxicity using a GCN](#)
- [Recommender system for movies using a GCMC implementation](#)

Monitor and Analyze Training Jobs Using Metrics

An Amazon SageMaker training job is an iterative process that teaches a model to make predictions by presenting examples from a training dataset. Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help diagnose whether the model is learning well and will generalize well for making predictions on unseen data. The training algorithm writes the values of these metrics to logs, which Amazon SageMaker monitors and sends to Amazon CloudWatch in real time. To analyze the performance of your training job, you can view graphs of these metrics in CloudWatch. When a training job has completed, you can also get a list of the metric values that it computes in its final iteration by calling the [DescribeTrainingJob](#) operation.

Topics

- [Training Metrics Sample Notebooks \(p. 607\)](#)
- [Defining Training Metrics \(p. 607\)](#)
- [Monitoring Training Job Metrics \(Console\) \(p. 609\)](#)
- [Monitoring Training Job Metrics \(Amazon SageMaker Console\) \(p. 609\)](#)
- [Example: Viewing a Training and Validation Curve \(p. 612\)](#)

Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics:

- An Introduction to the Amazon SageMaker ObjectToVec Model for Sequence-to-sequence Embedding (object2vec_sentence_similarity.ipynb)
- Regression with the Amazon SageMaker XGBoost Algorithm (xgboost_abalone.ipynb)

For instructions how to create and access Jupyter notebook instances that you can use to run the examples in Amazon SageMaker, see [Use Example Notebooks \(p. 260\)](#). To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. To access the example notebooks that show how to use training metrics, `object2vec_sentence_similarity.ipynb` and `xgboost_abalone.ipynb`, from the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Defining Training Metrics

Amazon SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you want Amazon SageMaker to parse logs from a custom algorithm and send metrics that the algorithm emits to CloudWatch, you have to specify the metrics that you want Amazon SageMaker to send to CloudWatch when you configure the training job. You specify the name of the metrics that you want to send and the regular expressions that Amazon SageMaker uses to parse the logs that your algorithm emits to find those metrics.

You can specify the metrics that you want to track with the Amazon SageMaker console, the Amazon SageMaker Python SDK (<https://github.com/aws/sagemaker-python-sdk>), or the low-level Amazon SageMaker API.

Topics

- [Defining Regular Expressions for Metrics \(p. 607\)](#)
- [Defining Training Metrics \(Low-level Amazon SageMaker API\) \(p. 608\)](#)
- [Defining Training Metrics \(Amazon SageMaker Python SDK\) \(p. 608\)](#)
- [Define Training Metrics \(Console\) \(p. 609\)](#)

Defining Regular Expressions for Metrics

To find a metric, Amazon SageMaker searches the logs that your algorithm emits and finds logs that match the regular expression that you specify for that metric. If you are using your own algorithm, do the following:

- Make sure that the algorithm writes the metrics that you want to capture to logs
- Define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

For example, suppose your algorithm emits metrics for training error and validation error by writing logs similar to the following to `stdout` or `stderr`:

```
Train_error=0.138318;  Valid_error = 0.324557;
```

If you want to monitor both of those metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```

"AlgorithmSpecification": {
    "TrainingImage": ContainerName,
    "TrainingInputMode": "File",
    "MetricDefinitions" : [
        {
            "Name": "train:error",
            "Regex": "Train_error=(.*?);"
        },
        {
            "Name": "validation:error",
            "Regex": "Valid_error=(.*?);"
        }
    ]
}

```

In the regex for the `train:error` metric defined above, the first part of the regex finds the exact text "Train_error=", and the expression `(.*?)`; captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, `.` means any character, `*` means zero or more, and `?` means capture only until the first instance of the `;` character.

Defining Training Metrics (Low-level Amazon SageMaker API)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions in the `MetricDefinitions` field of the `AlgorithmSpecification` input parameter that you pass to the `CreateTrainingJob` operation. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```

"AlgorithmSpecification": {
    "TrainingImage": ContainerName,
    "TrainingInputMode": "File",
    "MetricDefinitions" : [
        {
            "Name": "train:error",
            "Regex": "Train_error=(.*?);"
        },
        {
            "Name": "validation:error",
            "Regex": "Valid_error=(.*?);"
        }
    ]
}

```

For more information about defining and running a training job by using the low-level Amazon SageMaker API, see [Create and Run a Training Job \(AWS SDK for Python \(Boto 3\)\) \(p. 47\)](#).

Defining Training Metrics (Amazon SageMaker Python SDK)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions as the `metric_definitions` argument when you initialize an `Estimator` object. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `Estimator` initialization would look like the following:

```

estimator =
    Estimator(image_name=ImageName,
              role='SageMakerRole', train_instance_count=1,
              train_instance_type='ml.c4.xlarge',
              train_instance_type='ml.c4.xlarge',

```

```
k=10,  
sagemaker_session=sagemaker_session,  
metric_definitions=[  
    {'Name': 'train:error', 'Regex': 'Train_error=(.*?);'},  
    {'Name': 'validation:error', 'Regex': 'Valid_error=(.*?);'}  
]  
)
```

For more information about training by using Amazon SageMaker Python SDK estimators, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview>.

Define Training Metrics (Console)

You can define metrics for a custom algorithm in the console when you create a training job by providing the name and regular expression (regex) for **Metrics**.

For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your metric definitions would look like the following:

```
[  
    {  
        "Name": "train:error",  
        "Regex": "Train_error=(.*?);"  
    },  
    {  
        "Name": "validation:error",  
        "Regex": "Valid_error=(.*?);"  
    }  
]
```

Monitoring Training Job Metrics (Console)

You can monitor the metrics that a training job emits in real time in the CloudWatch console.

To monitor training job metrics (CloudWatch console)

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. Choose **Metrics**, then choose [/aws/sagemaker/TrainingJobs](#).
3. Choose **TrainingJobName**.
4. On the **All metrics** tab, choose the names of the training metrics that you want to monitor.
5. On the **Graphed metrics** tab, configure the graph options. For more information about using CloudWatch graphs, see [Graph Metrics](#) in the *Amazon CloudWatch User Guide*.

Monitoring Training Job Metrics (Amazon SageMaker Console)

You can monitor the metrics that a training job emits in real time by using the Amazon SageMaker console.

To monitor training job metrics (Amazon SageMaker console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.

2. Choose **Training jobs**, then choose the training job whose metrics you want to see.
3. Choose **TrainingJobName**.
4. In the **Monitor** section, you can review the graphs of instance utilization and algorithm metrics.

Amazon SageMaker Developer Guide
Monitoring Training Job Metrics
(Amazon SageMaker Console)



Example: Viewing a Training and Validation Curve

Typically, you split the data that you train your model on into training and validation datasets. You use the training set to train the model parameters that are used to make predictions on the training dataset. Then you test how well the model makes predictions by calculating predictions for the validation set. To analyze the performance of a training job, you commonly plot a training curve against a validation curve.

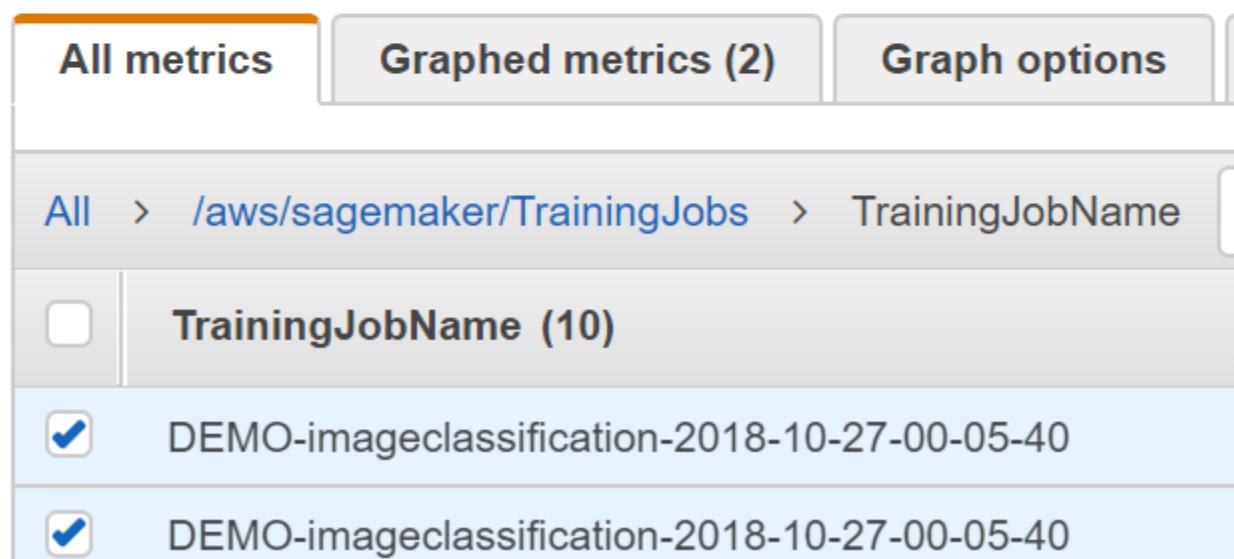
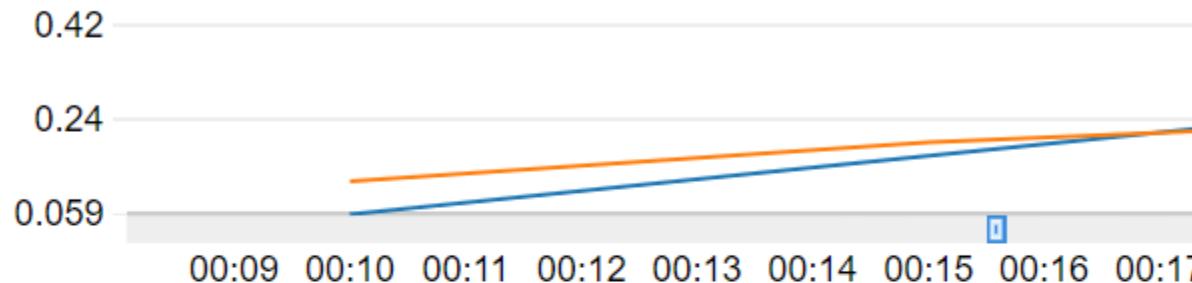
Viewing a graph that shows the accuracy for both the training and validation sets over time can help you to improve the performance of your model. For example, if training accuracy continues to increase over time, but, at some point, validation accuracy starts to decrease, you are likely overfitting your model. To address this, you can make adjustments to your model, such as increasing [regularization](#).

For this example, you can use the [Image-classification-full-training](#) example that is in the [Example notebooks](#) section of your Amazon SageMaker notebook instance. If you don't have an Amazon SageMaker notebook instance, create one by following the instructions at [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 41\)](#). If you prefer, you can follow along with the [End-to-End Multiclass Image Classification Example](#) in the example notebook on GitHub. You also need an Amazon S3 bucket to store the training data and for the model output. If you haven't created a bucket to use with Amazon SageMaker, create one by following the instructions at [Step 1: Create an Amazon S3 Bucket \(p. 40\)](#).

To view training and validation error curves

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Notebooks**, and then choose **Notebook instances**.
3. Choose the notebook instance that you want to use, and then choose **Open**.
4. On the dashboard for your notebook instance, choose **SageMaker Examples**.
5. Expand the **Introduction to Amazon Algorithms** section, and then choose **Use** next to **Image-classification-full-training.ipynb**.
6. Choose **Create copy**. Amazon SageMaker creates an editable copy of the **Image-classification-full-training.ipynb** notebook in your notebook instance.
7. In the first code cell of the notebook, replace `<>` with the name of your S3 bucket.
8. Run all of the cells in the notebook up to the **Deploy** section. You don't need to deploy an endpoint or get inference for this example.
9. After the training job starts, open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
10. Choose **Metrics**, then choose [/aws/sagemaker/TrainingJobs](#).
11. Choose **TrainingJobName**.
12. On the **All metrics** tab, choose the **train:accuracy** and **validation:accuracy** metrics for the training job that you created in the notebook.
13. On the graph, choose an area that the metric's values to zoom in. You should see something like the following:

Untitled graph



All metrics Graphed metrics (2) Graph options

All > /aws/sagemaker/TrainingJobs > TrainingJobName

<input type="checkbox"/>	TrainingJobName (10)
<input checked="" type="checkbox"/>	DEMO-imageclassification-2018-10-27-00-05-40
<input checked="" type="checkbox"/>	DEMO-imageclassification-2018-10-27-00-05-40

Incremental Training in Amazon SageMaker

Over time, you might find that a model generates inference that are not as good as they were in the past. With incremental training, you can use the artifacts from an existing model and use an expanded dataset to train a new model. Incremental training saves both time and resources.

Use incremental training to:

- Train a new model using an expanded dataset that contains an underlying pattern that was not accounted for in the previous training and which resulted in poor model performance.
- Use the model artifacts or a portion of the model artifacts from a popular publicly available model in a training job. You don't need to train a new model from scratch.

- Resume a training job that was stopped.
- Train several variants of a model, either with different hyperparameter settings or using different datasets.

For more information about training jobs, see [Train a Model with Amazon SageMaker \(p. 5\)](#).

You can train incrementally using the Amazon SageMaker console or the Amazon SageMaker Python SDK.

Important

Only three built-in algorithms currently support incremental training: [Object Detection Algorithm \(p. 402\)](#), [Image Classification Algorithm \(p. 324\)](#), and [Semantic Segmentation Algorithm \(p. 424\)](#).

Topics

- [Perform Incremental Training \(Console\) \(p. 614\)](#)
- [Perform Incremental Training \(API\) \(p. 616\)](#)

Perform Incremental Training (Console)

To complete this procedure, you need:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see [Common Parameters for Built-In Algorithms \(p. 274\)](#).
- The URL of the S3 bucket where you've stored the model artifacts that you want to use in incremental training. To find the URL for the model artifacts, see the details page of the training job used to create the model. To find the details page, in the Amazon SageMaker console, choose **Inference**, choose **Models**, and then choose the model.

To restart a stopped training job, use the URL to the model artifacts that are stored in the details page as you would with a model or a completed training job.

To perform incremental training (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. The training job name must have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).
6. (Optional) For **Resource configuration**, either leave the default values or increase the resource consumption to reduce computation time.
 - a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
 - b. For **Instance count**, use the default, 1.

- c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
 - a. For **Channel name**, either leave the default (**train**) or enter a more meaningful name for the training dataset, such as **expanded-training-dataset**.
 - b. For **InputMode**, choose **File**. For incremental training, you need to use file input mode.
 - c. For **S3 data distribution type**, choose **FullyReplicated**. This causes each ML compute instance to use a full replicate of the expanded dataset when training incrementally.
 - d. If the expanded dataset is uncompressed, set the **Compression type** to **None**. If the expanded dataset is compressed using Gzip, set it to **Gzip**.
 - e. (Optional) If you are using File input mode, leave **Content type** empty. For Pipe input mode, specify the appropriate MIME type. **Content type** is the multipurpose internet mail extension (MIME) type of the data.
 - f. For **Record wrapper**, if the dataset is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO formatted file, choose **None**.
 - g. For **S3 data type**, if the dataset is stored as a single file, choose **S3Prefix**. If the dataset is stored as several files in a folder, choose **Manifest**.
 - h. For **S3 location**, provide the URL to the path where you stored the expanded dataset.
 - i. Choose **Done**.
8. To use model artifacts in a training job, you need to add a new channel and provide the needed information about the model artifacts.
 - a. For **Input data configuration**, choose **Add channel**.
 - b. For **Channel name**, enter **model** to identify this channel as the source of the model artifacts.
 - c. For **InputMode**, choose **File**. Model artifacts are stored as files.
 - d. For **S3 data distribution type**, choose **FullyReplicated**. This indicates that each ML compute instance should use all of the model artifacts for training.
 - e. For **Compression type**, choose **None** because we are using a model for the channel.
 - f. Leave **Content type** empty. Content type is the multipurpose internet mail extension (MIME) type of the data. For model artifacts, we leave it empty.
 - g. Set **Record wrapper** to **None** because model artifacts are not stored in RecordIO format.
 - h. For **S3 data type**, if you are using a built-in algorithm or an algorithm that stores the model as a single file, choose **S3Prefix**. If you are using an algorithm that stores the model as several files, choose **Manifest**.
 - i. For **S3 location**, provide the URL to the path where you stored the model artifacts. Typically, the model is stored with the name **model.tar.gz**. To find the URL for the model artifacts, in the navigation pane, choose **Inference**, then choose **Models**. From the list of models, choose a model to display its details page. The URL for the model artifacts is listed under **Primary container**.
 - j. Choose **Done**.
9. For **Output data configuration**, provide the following information:
 - a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
 - b. (Optional) For **Encryption key**, you can add your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see [KMS-Managed Encryption Keys](#).
10. (Optional) For **Tags**, add one or more tags to the training job. A *tag* is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with

Project as a key and a value referring to a project that is related to the training job, such as [Home value forecasts](#).

11. Choose **Create training job**. Amazon SageMaker creates and runs training job.

After the training job has completed, the newly trained model artifacts are stored under the **S3 output path** that you provided in the **Output data configuration** field. To deploy the model to get predictions, see [Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#).

Perform Incremental Training (API)

This example shows how to use Amazon SageMaker APIs to train a model using the Amazon SageMaker image classification algorithm and the [Caltech 256 Image Dataset](#), then train a new model using the first one. It uses Amazon S3 for input and output sources. Please see the [incremental training sample notebook](#) for more details on using incremental training.

Note

In this example we used the original datasets in the incremental training, however you can use different datasets, such as ones that contain newly added samples. Upload the new datasets to S3 and make adjustments to the `data_channels` variable used to train the new model.

Get an AWS Identity and Access Management (IAM) role that grants required permissions and initialize environment variables:

```
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

sess = sagemaker.Session()

bucket=sess.default_bucket()
print(bucket)
prefix = 'ic-incr-training'
```

Get the training image for the image classification algorithm:

```
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sess.boto_region_name, 'image-classification',
    repo_version="latest")
#Display the training image
print (training_image)
```

Download the training and validation datasets, then upload them to Amazon Simple Storage Service (Amazon S3):

```
import os
import urllib.request
import boto3

# Define a download function
def download(url):
    filename = url.split("/")[-1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)
```

```
# Download the caltech-256 training and validation datasets
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')

# Create four channels: train, validation, train_lst, and validation_lst
s3train = 's3://{}//{}//train/'.format(bucket, prefix)
s3validation = 's3://{}//{}//validation/'.format(bucket, prefix)

# Upload the first files to the train and validation channels
!aws s3 cp caltech-256-60-train.rec $s3train --quiet
!aws s3 cp caltech-256-60-val.rec $s3validation --quiet
```

Define the training hyperparameters:

```
# Define hyperparameters for the estimator
hyperparams = { "num_layers": "18",
                "resize": "32",
                "num_training_samples": "50000",
                "num_classes": "10",
                "image_shape": "3,28,28",
                "mini_batch_size": "128",
                "epochs": "3",
                "learning_rate": "0.1",
                "lr_scheduler_step": "2,3",
                "lr_scheduler_factor": "0.1",
                "augmentation_type": "crop_color",
                "optimizer": "sgd",
                "momentum": "0.9",
                "weight_decay": "0.0001",
                "beta_1": "0.9",
                "beta_2": "0.999",
                "gamma": "0.9",
                "eps": "1e-8",
                "top_k": "5",
                "checkpoint_frequency": "1",
                "use_pretrained_model": "0",
                "model_prefix": "" }
```

Create an estimator object and train the first model using the training and validation datasets:

```
# Fit the base estimator
s3_output_location = 's3://{}//{}//output'.format(bucket, prefix)
ic = sagemaker.estimator.Estimator(training_image,
                                     role,
                                     train_instance_count=1,
                                     train_instance_type='ml.p2.xlarge',
                                     train_volume_size=50,
                                     train_max_run=360000,
                                     input_mode='File',
                                     output_path=s3_output_location,
                                     sagemaker_session=sess,
                                     hyperparameters=hyperparams)

train_data = sagemaker.session.s3_input(s3train, distribution='FullyReplicated',
                                         content_type='application/x-recordio',
                                         s3_data_type='S3Prefix')
validation_data = sagemaker.session.s3_input(s3validation, distribution='FullyReplicated',
                                             content_type='application/x-recordio',
                                             s3_data_type='S3Prefix')

data_channels = {'train': train_data, 'validation': validation_data}

ic.fit(inputs=data_channels, logs=True)
```

To use the model to incrementally train another model, create a new estimator object and use the model artifacts (`ic.model_data`, in this example) for the `model_uri` input argument:

```
# Given the base estimator, create a new one for incremental training
incr_ic = sagemaker.estimator.Estimator(training_image,
                                         role,
                                         train_instance_count=1,
                                         train_instance_type='ml.p2.xlarge',
                                         train_volume_size=50,
                                         train_max_run=360000,
                                         input_mode='File',
                                         output_path=s3_output_location,
                                         sagemaker_session=sess,
                                         hyperparameters=hyperparams,
                                         model_uri=ic.model_data) # This parameter will
                                         ingest the previous job's model as a new channel
incr_ic.fit(inputs=data_channels, logs=True)
```

After the training job has completed, the newly trained model artifacts are stored under the S3 `output_path` that you provided in `Output_path`. To deploy the model to get predictions, see [Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#).

Managed Spot Training in Amazon SageMaker

Amazon SageMaker makes it easy to train machine learning models using managed Amazon EC2 Spot instances. Managed spot training can optimize the cost of training models up to 90% over on-demand instances. Amazon SageMaker manages the Spot interruptions on your behalf.

Managed Spot Training uses Amazon EC2 Spot instance to run training jobs instead of on-demand instances. You can specify which training jobs use spot instances and a stopping condition that specifies how long Amazon SageMaker waits for a job to run using Amazon EC2 Spot instances. Metrics and logs generated during training runs are available in CloudWatch.

Spot instances can be interrupted, causing jobs to take longer to start or finish. You can configure your managed spot training job to use checkpoints. Amazon SageMaker copies checkpoint data from a local path to Amazon S3. When the job is restarted, Amazon SageMaker copies the data from Amazon S3 back into the local path. The training can then resume from the last checkpoint instead of restarting. For more information about checkpointing, see [Use Checkpoints in Amazon SageMaker \(p. 619\)](#).

Note

Unless your training job will complete quickly, we recommend you use checkpointing with managed spot training. SageMaker built-in algorithms and marketplace algorithms that do not checkpoint are currently limited to a `MaxWaitTimeInSeconds` of 3600 seconds (60 minutes).

Topics

- [Using Managed Spot Training \(p. 618\)](#)
- [Managed Spot Training Lifecycle \(p. 619\)](#)

Using Managed Spot Training

To use managed spot training, create a training job. Set `EnableManagedSpotTraining` to `True` and specify the `MaxWaitTimeInSeconds`. `MaxWaitTimeInSeconds` must be larger than `MaxRuntimeInSeconds`. For more information about creating a training job, see [DescribeTrainingJob](#).

You can calculate the savings from using managed spot training using the formula $(1 - \text{BillableTimeInSeconds} / \text{TrainingTimeInSeconds}) * 100$. For example, if `BillableTimeInSeconds` is 100 and `TrainingTimeInSeconds` is 500, the savings is 80%.

Managed Spot Training Lifecycle

You can monitor a training job using `TrainingJobStatus` and `SecondaryStatus` returned by [DescribeTrainingJob](#). The list below shows how `TrainingJobStatus` and `SecondaryStatus` values change depending on the training scenario:

- **Spot instances acquired with no interruption during training**
 1. InProgress: Starting → Downloading → Training → Uploading
- **Spot instances interrupted once. Later, enough spot instances were acquired to finish the training job.**
 1. InProgress: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Uploading
- **Spot instances interrupted twice and `MaxWaitTimeInSeconds` exceeded.**
 1. InProgress: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Interrupted → Downloading → Training
 2. Stopping: Stopping
 3. Stopped: MaxWaitTimeExceeded
- **Spot instances were never launched.**
 1. InProgress: Starting
 2. Stopping: Stopping
 3. Stopped: MaxWaitTimeExceeded

Use Checkpoints in Amazon SageMaker

A checkpoint is a snapshot of the state of the model. They can be used with Managed Spot Training. If a training job is interrupted, a snapshot can be used to resume from a previously saved point. This can save training time.

Snapshots are saved to an Amazon S3 location you specify. You can configure the local path to use for snapshots or use the default. When a training job is interrupted, Amazon SageMaker copies the training data to Amazon S3. When the training job is restarted, the checkpoint data is copied to the local path. It can be used to resume at the checkpoint.

To enable checkpoints, provide an Amazon S3 location. You can optionally provide a local path and choose to use a shared folder. The default local path is `/opt/ml/checkpoints/`. For more information, see [CreateTrainingJob](#).

Provide Dataset Metadata to Training Jobs with an Augmented Manifest File

To classify data into different groupings, you train a model by using a dataset and metadata that act as labels. To include metadata with your dataset in a training job, use an augmented manifest file. When using an augmented manifest file, your dataset must be stored in Amazon Simple Storage Service (Amazon S3) and you must configure your training job to use dataset stored there. You specify the location and format of this dataset for one or more [Channel](#). Augmented manifests can only support Pipe input mode. See the section, [InputMode](#) in [Channel](#) to learn more about pipe input mode.

When specifying a channel's parameters, you specify a path to the file, called a `s3Uri`. Amazon SageMaker interprets this URI based on the specified `S3DataType` in [S3DataSource](#). The

The `AugmentedManifestFile` option defines a manifest format that includes metadata with the input data. Using an augmented manifest file is an alternative to preprocessing when you have labeled data. For training jobs using labeled data, you typically need to preprocess the dataset to combine input data with metadata before training. If your training dataset is large, preprocessing can be time consuming and expensive.

Augmented Manifest File Format

An augmented manifest file must be formatted in [JSON Lines](#) format. In JSON Lines format, each line in the file is a complete JSON object followed by a newline separator.

During training, Amazon SageMaker parses each JSON line and sends some or all of its attributes on to the training algorithm. You specify which attribute contents to pass and the order in which to pass them with the `AttributeNames` parameter of the [CreateTrainingJob](#) API. The `AttributeNames` parameter is an ordered list of attribute names that Amazon SageMaker looks for in the JSON object to use as training input.

For example, if you list `["line", "book"]` for `AttributeNames`, the input data must include the attribute names of `line` and `book` in the specified order. For this example, the following augmented manifest file content is valid:

```
{"author": "Herman Melville", "line": "Call me Ishmael", "book": "Moby Dick"}  
{"line": "It was love at first sight.", "author": "Joseph Heller", "book": "Catch-22"}
```

Amazon SageMaker ignores unlisted attribute names even if they precede, follow, or are in between listed attributes.

When using augmented manifest files, observe the following guidelines:

- The order of the attributes listed in the `AttributeNames` parameter determines the order of the attributes passed to the algorithm in the training job.
- The listed `AttributeNames` can be a subset of all of the attributes in the JSON line. Amazon SageMaker ignores unlisted attributes in the file.
- You can specify any type of data allowed by the JSON format in `AttributeNames`, including text, numerical, data arrays, or objects.
- To include an S3 URI as an attribute name, add the suffix `-ref` to it.

If an attribute name contains the suffix `-ref`, the attribute's value must be an S3 URI to a data file that is accessible to the training job. For example, if `AttributeNames` contains `["image-ref", "is-a-cat"]`, a valid augmented manifest file might contain these lines:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "is-a-cat": 1}  
{"image-ref": "s3://mybucket/sample02/image2.jpg", "is-a-cat": 0}
```

For the first line of this manifest, Amazon SageMaker retrieves the contents of the S3 object `s3://mybucket/foo/image1.jpg` and streams it to the algorithm for training. The second line is the string representation of the `is-a-cat` attribute `"1"`, which is followed by the contents of the second line.

To create an augmented manifest file, use Amazon SageMaker Ground Truth to create a labeling job. For more information, see [Output Data \(p. 103\)](#).

Stream Augmented Manifest File Data

Augmented manifest files are supported only for channels using Pipe input mode. For each channel, the data is extracted from its augmented manifest file and streamed (in order) to the algorithm through the

channel's named pipe. Pipe mode uses the first in first out (FIFO) method, so records are processed in the order in which they are queued. For information about Pipe input mode, see [Input Mode](#).

Attribute names with a "-ref" suffix point to preformatted binary data. In some cases, the algorithm knows how to parse the data. In other cases, you might need to wrap the data so that records are delimited for the algorithm. If the algorithm is compatible with [RecordIO-formatted data](#), specifying RecordIO for RecordWrapperType solves this issue. If the algorithm is not compatible with RecordIO format, specify None for RecordWrapperType and make sure that your data is parsed correctly for your algorithm. Using the ["image-ref", "is-a-cat"] example, if you use RecordIO wrapping, the following stream of data is sent to the queue:

```
recordio_formatted(s3://mybucket/foo/  
image1.jpg)recordio_formatted("1")recordio_formatted(s3://mybucket/bar/  
image2.jpg)recordio_formatted("0")
```

Images that aren't wrapped with RecordIO format, are streamed with the corresponding `is-a-cat` attribute value as one record. This can cause a problem because the algorithm might not delimit the images and attributes correctly.

With augmented manifest files and Pipe mode in general, size limits of the EBS volume do not apply. This includes settings that otherwise must be within the EBS volume size limit such as `S3DataDistributionType`. For more information about Pipe mode and how to use it, see [Using Your Own Training Algorithms - Input Data Configuration](#).

Use an Augmented Manifest File (Console)

To complete this procedure, you need:

- The URL of the S3 bucket where you've stored the augmented manifest file.
- To store the data that is listed in the augmented manifest file in an S3 bucket.
- The URL of the S3 bucket where you want to store the output of the job.

To use an augmented manifest file in a training job (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an AWS Region in an AWS account. It can have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ _ % - (hyphen).
5. Choose the algorithm that you want to use. For information about supported built-in algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#). If you want to use a custom algorithm, make sure that it is compatible with Pipe mode.
6. (Optional) For **Resource configuration**, either accept the default values or, to reduce computation time, increase the resource consumption.
 - a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
 - b. For **Instance count**, use the default, 1.
 - c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
 - a. For **Channel name**, either accept the default (**train**) or enter a more meaningful name, such as **training-augmented-manifest-file**.

- b. For **InputMode**, choose **Pipe**.
 - c. For **S3 data distribution type**, choose **FullyReplicated**. When training incrementally, fully replicating causes each ML compute instance to use a complete copy of the expanded dataset. For neural-based algorithms, such as [Neural Topic Model \(NTM\) Algorithm \(p. 380\)](#), choose **ShardedByS3Key**.
 - d. If the data specified in the augmented manifest file is uncompressed, set the **Compression type** to **None**. If the data is compressed using gzip, set it to **Gzip**.
 - e. (Optional) For **Content type**, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.
 - f. For **Record wrapper**, if the dataset specified in the augmented manifest file is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO-formatted file, choose **None**.
 - g. For **S3 data type**, choose **AugmentedManifestFile**.
 - h. For **S3 location**, provide the path to the bucket where you stored the augmented manifest file.
 - i. For **AugmentedManifestFile attribute names**, specify the name of an attribute that you want to use. The attribute name must be present within the augmented manifest file, and is case-sensitive.
 - j. (Optional) To add more attribute names, choose **Add row** and specify another attribute name for each attribute.
 - k. (Optional) To adjust the order of attribute names, choose the up or down buttons next to the names. When using an augmented manifest file, the order of the specified attribute names is important.
 - l. Choose **Done**.
8. For **Output data configuration**, provide the following information:
 - a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
 - b. (Optional) You can use your AWS Key Management Service (AWS KMS) encryption key to encrypt the output data at rest. For **Encryption key**, provide the key ID or its Amazon Resource Number (ARN). For more information, see [KMS-Managed Encryption Keys](#).
 9. (Optional) For **Tags**, add one or more tags to the training job. A *tag* is metadata that you can define and assign to AWS resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with **Project** as a key and a value that refers to a project that is related to the training job, such as **Home value forecasts**.
 10. Choose **Create training job**. Amazon SageMaker creates and runs the training job.

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see [Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#).

Use an Augmented Manifest File (API)

The following shows how to train a model with an augmented manifest file using the Amazon SageMaker high-level Python library:

```
# Create a model object set to using "Pipe" mode.
model = sagemaker.estimator.Estimator(training_image,
                                         role,
                                         train_instance_count=1,
                                         train_instance_type='ml.p3.2xlarge',
                                         train_volume_size = 50,
                                         train_max_run = 360000,
                                         input_mode = 'Pipe',
```

```
        output_path=s3_output_location,
        sagemaker_session=session)

# Create a train data channel with S3_data_type as 'AugmentedManifestFile' and attribute
names.
train_data = sagemaker.session.s3_input(your_augmented_manifest_file,
                                         distribution='FullyReplicated',
                                         content_type='image/jpeg',
                                         s3_data_type='AugmentedManifestFile',
                                         attribute_names=[ 'source-ref', 'annotations'])
data_channels = {'train': train_data}

# Train a model.
model.fit(inputs=data_channels, logs=True)
```

After the training job has finished, Amazon SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see [Step 6: Deploy the Model to Amazon SageMaker \(p. 49\)](#).

Deploy Models

After you build and train your models, you can deploy them to get predictions in one of two ways:

- To set up a persistent endpoint to get predictions from your models, use Amazon SageMaker hosting services. For an overview on deploying a single model or multiple models with Amazon SageMaker hosting services, see [Deploy a Model on Amazon SageMaker Hosting Services \(p. 8\)](#).
- To get predictions for an entire dataset, use Amazon SageMaker batch transform. For an overview on deploying a model with Amazon SageMaker batch transform, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#).

Prerequisites

These topics assume that you have built and trained one or more machine learning models and are ready to deploy them. If you are new to Amazon SageMaker and have not completed these prerequisite tasks, work through the steps in the [Get Started with Amazon SageMaker \(p. 22\)](#) tutorial to familiarize yourself with an example of how Amazon SageMaker manages the data science process and how it handles model deployment. For more information about building a model, see [Build Models \(p. 247\)](#). For information about training a model, see [Train Models \(p. 524\)](#).

What do you want to do?

Amazon SageMaker provides features to manage resources and optimize inference performance when deploying machine learning models. For guidance on using inference pipelines, compiling and deploying models with Neo, Elastic Inference, and automatic model scaling, see the following topics.

- To manage data processing and real-time predictions or to process batch transforms in a pipeline, see [Deploy an Inference Pipeline \(p. 661\)](#).
- To train TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models once and optimize them to deploy on ARM, Intel, and Nvidia processors, see [Compile and Deploy Models with Amazon SageMaker Neo \(p. 682\)](#).
- To preprocess entire datasets quickly or to get inferences from a trained model for large datasets when you don't need a persistent endpoint, see [Use Batch Transform \(p. 675\)](#).
- To speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as Amazon SageMaker hosted models using a GPU instance for your endpoint, see [Use Amazon SageMaker Elastic Inference \(EI\) \(p. 702\)](#).
- To dynamically adjust the number of instances provisioned in response to changes in your workload, see [Automatically Scale Amazon SageMaker Models \(p. 713\)](#).
- To create an endpoint that can host multiple models using a shared serving container, see [Host Multiple Models with Multi-Model Endpoints \(p. 625\)](#).

Manage Model Deployments

For guidance on managing model deployments, including monitoring, troubleshooting, and best practices, and for information on storage associated with inference hosting instances:

- For tools that can be used to monitor model deployments, see [Monitor Amazon SageMaker \(p. 731\)](#).
- For troubleshooting model deployments, see [Troubleshoot Amazon SageMaker Model Deployments \(p. 727\)](#).
- For model deployment best practices, see [Deployment Best Practices \(p. 728\)](#).
- For information about the size of storage volumes provided for different sizes of hosting instances, see [Host Instance Storage Volumes \(p. 729\)](#).

Deploy Your Own Inference Code

For developers that need more advanced guidance on how to run your own inference code:

- To run your own inference code hosting services, see [Use Your Own Inference Code with Hosting Services \(p. 501\)](#).
- To run your own inference code for batch transforms, see [Use Your Own Inference Code with Batch Transform \(p. 503\)](#).

Guide to Amazon SageMaker

[What Is Amazon SageMaker? \(p. 1\)](#)

Host Multiple Models with Multi-Model Endpoints

To create an endpoint that can host multiple models, use multi-model endpoints. Multi-model endpoints provide a scalable and cost-effective solution to deploying large numbers of models. They use a shared serving container that is enabled to host multiple models. This reduces hosting costs by improving endpoint utilization compared with using single-model endpoints. It also reduces deployment overhead because Amazon SageMaker manages loading models in memory and scaling them based on the traffic patterns to them.

Multi-model endpoints also enable time-sharing of memory resources across your models. This works best when the models are fairly similar in size and invocation latency. When this is the case, multi-model endpoints can effectively use instances across all models. If you have models that have significantly higher transactions per second (TPS) or latency requirements, we recommend hosting them on dedicated endpoints. Multi-model endpoints are also well suited to cases that can tolerate occasional cold-start-related latency penalties that occur when invoking infrequently used models.

Multi-model endpoints support A/B testing. They work with Auto Scaling and AWS PrivateLink. However, you can't use multi-model-enabled containers with serial inference pipelines or with Amazon Elastic Inference (EIA).

You can use the AWS SDK for Python (Boto) or the Amazon SageMaker console to create a multi-model endpoint. You can use multi-model endpoints with custom-built containers by integrating the [Multi Model Server library](#).

Topics

- [Sample Notebooks for Multi-Model Endpoints \(p. 626\)](#)
- [How Multi-Model Endpoints Work \(p. 626\)](#)
- [Multi-Model Endpoint Security \(p. 627\)](#)

- [Instance Recommendations for Multi-Model Endpoint Deployments \(p. 627\)](#)
- [CloudWatch Metrics for Multi-Model Endpoint Deployments \(p. 628\)](#)
- [Create a Multi-Model Endpoint \(p. 628\)](#)
- [Build Your Own Container with Multi Model Server \(p. 631\)](#)
- [Invoke a Multi-Model Endpoint \(p. 634\)](#)
- [Add or Remove Models \(p. 634\)](#)

Sample Notebooks for Multi-Model Endpoints

For a sample notebook that uses Amazon SageMaker to deploy multiple XGBoost models to an endpoint, see the [Multi-Model Endpoint XGBoost Sample Notebook](#). For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in Amazon SageMaker, see the [Multi-Model Endpoint BYOC Sample Notebook](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). After you've created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. The Multi-Model Endpoint notebook is located in the **ADVANCED FUNCTIONALITY** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

How Multi-Model Endpoints Work

Amazon SageMaker manages the lifecycle of models hosted on multi-model endpoints in the container's memory. Instead of downloading all of the models from an Amazon S3 bucket to the container when you create the endpoint, Amazon SageMaker dynamically loads them when you invoke them. When Amazon SageMaker receives an invocation request for a particular model, it does the following:

1. Routes the request to a single Amazon EC2 instance behind the endpoint.
2. Downloads the model from the S3 bucket to that instance's storage volume.
3. Loads the model to the container's memory on that instance. If the model is already loaded in the container's memory, invocation is faster because Amazon SageMaker doesn't need to download and load it.

Amazon SageMaker continues to route requests for a model to the instance where the model is already loaded. However, if the model receives many invocation requests, and there are additional instances for the multi-model endpoint, Amazon SageMaker routes some requests to another instance to accommodate the traffic. If the model isn't already loaded on the second instance, the model is downloaded to that instance's storage volume and loaded into the container's memory.

When an instance's memory utilization is high and Amazon SageMaker needs to load another model into memory, it unloads unused models from that instance's container to ensure that there is enough memory to load the model. Models that are unloaded remain on the instance's storage volume and can be loaded into the container's memory later without being downloaded again from the S3 bucket. If the instance's storage volume reaches its capacity, Amazon SageMaker deletes any unused models from the storage volume.

Adding models to, and deleting them from, a multi-model endpoint doesn't require updating the endpoint itself. To add a model, you upload it to the S3 bucket and invoke it. To delete a model, stop sending requests and delete it from the S3 bucket. Amazon SageMaker provides multi-model endpoint capability in a serving container. You don't need code changes to use it.

When you update a multi-model endpoint, invocation requests on the endpoint might experience higher latencies as traffic is directed to the instances in the updated endpoint.

Multi-Model Endpoint Security

Models and data in a multi-model endpoint are co-located on instance storage volume and in container memory. All instances for Amazon SageMaker endpoints run on a single tenant container that you own. Only your models can run on your multi-model endpoint. It's your responsibility to manage the mapping of requests to models and to provide access for users to the correct target models. Amazon SageMaker uses [IAM roles](#) to provide IAM identity-based policies that you use to specify allowed or denied actions and resources and the conditions under which actions are allowed or denied.

An IAM principal with [InvokeEndpoint](#) permissions on a multi-model endpoint can invoke any model at the address of the S3 prefix defined in the [CreateModel](#) operation, provided that the IAM Execution Role defined in operation has permissions to download the model. If you need to restrict [InvokeEndpoint](#) access to a limited set of models in S3, you can create multi-model endpoints with more restrictive S3 prefixes. For more information about how Amazon SageMaker uses roles to manage access to endpoints and perform operations on your behalf, see [Amazon SageMaker Roles \(p. 776\)](#). Your customers might also have certain data isolation requirements dictated by their own compliance requirements that can be satisfied using IAM identities.

Instance Recommendations for Multi-Model Endpoint Deployments

There are several items to consider when selecting a SageMaker ML instance type for a multi-model endpoint. Provision sufficient [Amazon Elastic Block Store \(Amazon EBS\)](#) capacity for all of the models that need to be served. Balance performance (minimize cold starts) and cost (don't over-provision instance capacity). For information about the size of the storage volume that Amazon SageMaker attaches for each instance type for an endpoint and for a multi-model endpoint, see [Host Instance Storage Volumes \(p. 729\)](#). For a container configured to run in `MultiModel` mode, the storage volume provisioned for its instances has more memory. This allows more models to be cached on the instance storage volume.

When choosing an Amazon SageMaker ML instance type, consider the following:

- The traffic distribution (access patterns) to the models that you want to host behind the multi-model endpoint, along with the model size (how many models could be loaded in memory on the instance).:
 - Think of the amount of memory on an instance as the cache space for models to be loaded. Think of the number of vCPUs as the concurrency limit to perform inference on the loaded models (assuming that invoking a model is bound to CPU).
 - A higher amount of instance memory allows you to have more models loaded and ready to serve inference requests. You don't need to waste time loading the model.
 - A higher amount of vCPUs allows you to invoke more unique models concurrently (again assuming that inference is bound to CPU).
 - Have some "slack" memory available so that unused models can be unloaded, and especially for multi-model endpoints with multiple instances. If an instance or an Availability Zone fails, the models on those instances will be rerouted to other instances behind the endpoint.
- Tolerance to loading/downloading times:
 - d instance type families (for example, m5d, c5d, or r5d) come with an NVMe (non-volatile memory express) SSD, which offers high I/O performance and might reduce the time it takes to download models to the storage volume and for the container to load the model from the storage volume.
 - Because d instance types come with an NVMe SSD storage, Amazon SageMaker does not attach an Amazon EBS storage volume to these ML compute instances that hosts the multi-model endpoint..

In some cases, you might opt to reduce costs by choosing an instance type that can't hold all of the targeted models in memory at once. Amazon SageMaker dynamically unloads models when it runs out

of memory to make room for a newly targeted model. For infrequently requested models, you are going to pay a price with the dynamic load latency. In cases with more stringent latency needs, you might opt for larger instance types or more instances. Investing time up front for proper performance testing and analysis will pay great dividends in successful production deployments.

You can use the `Average` statistic of the `ModelCacheHit` metric to monitor the ratio of requests where the model is already loaded. You can use the `SampleCount` statistic for the `ModelUnloadingTime` metric to monitor the number of unload requests sent to the container during a time period. If models are unloaded too frequently (an indicator of thrashing, where models are being unloaded and loaded again because there is insufficient cache space for the working set of models), consider using a larger instance type with more memory or increasing the number of instances behind the multi-model endpoint. For multi-model endpoints with multiple instances, be aware that a model might be loaded on more than 1 instance.

Amazon SageMaker multi-model endpoints fully supports Auto Scaling, which manages replicas of models to ensure models scale based on traffic patterns. We recommend that you configure your multi-model endpoint and the size of your instances by considering all of the above and also set up auto scaling for your endpoint. The invocation rates used to trigger an auto-scale event is based on the aggregate set of predictions across the full set of models served by the endpoint.

CloudWatch Metrics for Multi-Model Endpoint Deployments

Amazon SageMaker provides metrics for endpoints so you can monitor the cache hit rate, the number of models loaded, and the model wait times for loading, downloading, and uploading at a multi-model endpoint. For information, see **Multi-Model Endpoint Model Loading Metrics** and **Multi-Model Endpoint Model Instance Metrics** in [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#). Per-model metrics aren't supported.

Create a Multi-Model Endpoint

You can use the AWS SDK for Python (Boto) or the Amazon SageMaker to create a multi-model endpoint.

Topics

- [Create a Multi-Model Endpoint \(AWS SDK for Python \(Boto\)\) \(p. 628\)](#)
- [Create a Multi-Model Endpoint \(Console\) \(p. 629\)](#)

Create a Multi-Model Endpoint (AWS SDK for Python (Boto))

You create a multi-model endpoint using the Amazon SageMaker `CreateModel`, `CreateEndpointConfig`, and `CreateEndpoint` APIs just as you would create a single model endpoint, but with two changes. When defining the container, you need to pass a new `Mode` parameter value, `MultiModel`. You also need to pass the `ModelDataUrl` field that specifies the prefix in Amazon S3 where the model artifacts are located, instead of the path to a single model artifact, as you would when deploying a single model.

For a sample notebook that uses Amazon SageMaker to deploy multiple XGBoost models to an endpoint, see [Multi-Model Endpoint XGBoost Sample Notebook](#).

The following procedure outlines the key steps used in that sample to create a multi-model endpoint.

To deploy the model (AWS SDK for Python (Boto 3))

1. Get a container whose image supports deploying models.

```
container = { 'Image': '123456789012.dkr.ecr.us-east-1.amazonaws.com/  
myimage:mytag',  
    'ModelDataURL': 's3://my-bucket/path/to/artifacts/',  
    'Mode': 'MultiModel'  
}
```

2. Create the model that uses this container.

```
response = sm_client.create_model(  
    ModelName = 'my-multi-model-name',  
    ExecutionRoleArn = role,  
    Containers = [container])
```

3. Configure the multi-model endpoint for the model. We recommend configuring your endpoints with at least two instances. This allows Amazon SageMaker to provide a highly available set of predictions across multiple Availability Zones for the models.

```
response = sm_client.create_endpoint_config(  
    EndpointConfigName = 'my-epc',  
    ProductionVariants=[{  
        'InstanceType': 'ml.m4.xlarge',  
        'InitialInstanceCount': 2,  
        'InitialVariantWeight': 1,  
        'ModelName': 'my-multi-model-name',  
        'VariantName': 'AllTraffic'}])
```

4. Create the multi-model endpoint using the `EndpointName` and `EndpointConfigName` parameters.

```
response = sm_client.create_endpoint(  
    EndpointName = 'my-endpoint',  
    EndpointConfigName = 'my-epc')
```

Create a Multi-Model Endpoint (Console)

To create a multi-model endpoint (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Model**, and then from the **Inference** group, choose **Create model**.
3. For **Model name**, enter a name.
4. For **IAM role**, choose or create an IAM role that has the `AmazonSageMakerFullAccess` IAM policy attached.
5. In the **Container definition** section, for **Provide model artifacts and inference image options** choose **Use multiple models**.

The screenshot shows the 'Create model' wizard in the Amazon SageMaker console. The top navigation bar includes the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, and a bell icon. The breadcrumb path is 'Amazon SageMaker > Models > Create model'. The main title is 'Create model'. A note below says: 'To deploy a model to Amazon SageMaker, first create the model by providing the location of your inference code. See [Deploying a Model on Amazon SageMaker Hosting Services](#) [Learn more]'. The 'Model settings' section has a 'Model name' input field containing 'mml-test-model'. Below it is a note: 'Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique across all models in your account in an AWS Region.' The 'IAM role' section shows a selected role: 'AmazonSageMaker-ExecutionRole- XXXXXXXXXXXXXXXXXXXX'. The 'Container definition 1' section has a 'Container input options' section with two choices: 'Use a single model' (unchecked) and 'Use multiple models' (checked). A red oval highlights the 'Use multiple models' option. Below it is a 'Location of inference code image' input field containing '123456789012.dkr.ecr.us-east-1.amazonaws.com/myimage:mytag'. The 'Location of model artifacts' input field at the bottom contains 's3://my-bucket/path/to/artifacts/'. The page footer shows the number '630'.

Create model

To deploy a model to Amazon SageMaker, first create the model by providing the location of your inference code. See [Deploying a Model on Amazon SageMaker Hosting Services](#) [Learn more]

Model settings

Model name

mml-test-model

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique across all models in your account in an AWS Region.

IAM role

Amazon SageMaker requires permissions to call other services on your behalf. Choose a role or let Amazon SageMaker create a role that has the [AmazonSageMakerFullAccess](#) IAM policy attached.

AmazonSageMaker-ExecutionRole- XXXXXXXXXXXXXXXXXXXX

Container definition 1

► Container input options

Provide model artifacts and inference image location

▼ Provide model artifacts and inference image options

Use a single model
Use this to host a single model in this container.

Use multiple models
Use this to host multiple models in this container.

Location of inference code image

Type the registry path where the inference code image is stored in Amazon ECR.

123456789012.dkr.ecr.us-east-1.amazonaws.com/myimage:mytag

630

Location of model artifacts

Type the URL where model artifacts are stored in S3.

s3://my-bucket/path/to/artifacts/

6. Choose **Create model**.
7. Deploy your multi-model endpoint as you would a single model endpoint.

Build Your Own Container with Multi Model Server

Custom Elastic Container Registry (ECR) images deployed in Amazon SageMaker are expected to adhere to the basic contract described in [Use Your Own Inference Code with Hosting Services \(p. 501\)](#) that govern how Amazon SageMaker interacts with a Docker container that runs your own inference code. For a container to be capable of loading and serving multiple models concurrently, there are additional APIs and behaviors that must be followed. This additional contract includes new APIs to load, list, get, and unload models, and a different API to invoke models. There are also different behaviors for error scenarios that the APIs need to abide by. To indicate that the container complies with the additional requirements, you can add the following command to your Docker file:

```
LABEL com.amazonaws.sagemaker.capabilities.multi-models=true
```

Amazon SageMaker also injects an environment variable into the container

```
SAGEMAKER_MULTI_MODEL=true
```

To help you implement these requirements for a custom container, two libraries are available:

- [Multi Model Server](#) is an open source framework for serving machine learning models that can be installed in containers to provide the front end that fulfills the requirements for the new multi-model endpoint container APIs. It provides the HTTP front end and model management capabilities required by multi-model endpoints to host multiple models within a single container, load models into and unload models out of the container dynamically, and performs inference on a specified loaded model. It also provides a pluggable backend that supports a pluggable custom backend handler where you can implement your own algorithm.
- [Amazon SageMaker Inference Toolkit](#) is a library that bootstraps Multi Model Server with a configuration and settings that make it compatible with Amazon SageMaker multi-model endpoints. It also allows you to tweak important performance parameters, such as the number of workers per model, depending on the needs of your scenario.

For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in Amazon SageMaker, see the [Multi-Model Endpoint BYOC Sample Notebook](#).

Contract for Custom Containers to Serve Multiple Model

To handle multiple models, your container must support a set of APIs that enable the Amazon SageMaker platform to communicate with the container for loading, listing, getting, and unloading models as required. The `model_name` is used in the new set of APIs as the key input parameter. The customer container is expected to keep track of the loaded models using `model_name` as the mapping key. Also, the `model_name` is an opaque identifier and is not necessarily the value of the `TargetModel` parameter passed into the `InvokeEndpoint` API. The original `TargetModel` value in the `InvokeEndpoint` request is passed to container in the APIs as a `X-Amzn-Target-Model` header that can be used for logging purposes.

Topics

- [LOAD MODEL API \(p. 632\)](#)
- [LIST MODEL API \(p. 632\)](#)
- [GET MODEL API \(p. 633\)](#)

- [UNLOAD MODEL API \(p. 633\)](#)
- [INVOKE MODEL API \(p. 633\)](#)

LOAD MODEL API

Instructs the container to load a particular model present in the `url` field of the body into the memory of the customer container and to keep track of it with the assigned `model_name`. After a model is loaded, the container should be ready to serve inference requests using this `model_name`.

```
POST /models HTTP/1.1
Content-Type: application/json
Accept: application/json

{
    "model_name" : "{model_name}",
    "url" : "/opt/ml/models/{model_name}/model",
}
```

Note

If `model_name` is already loaded, this API should return 409. Any time a model cannot be loaded due to lack of memory or to any other resource, this API should return a 507 HTTP status code to Amazon SageMaker, which then initiates unloading unused models to reclaim.

LIST MODEL API

Returns the list of models loaded into the memory of the customer container.

```
GET /models HTTP/1.1
Accept: application/json

Response =
{
    "models": [
        {
            "modelName" : "{model_name}",
            "modelUrl" : "/opt/ml/models/{model_name}/model",
        },
        {
            "modelName" : "{model_name}",
            "modelUrl" : "/opt/ml/models/{model_name}/model",
        },
        ...
    ]
}
```

This API also supports pagination.

```
GET /models HTTP/1.1
Accept: application/json

Response =
{
    "models": [
        {
            "modelName" : "{model_name}",
            "modelUrl" : "/opt/ml/models/{model_name}/model",
        },
        {

```

```
        "modelName" : "{model_name}",
        "modelUrl" : "/opt/ml/models/{model_name}/model",
    },
    ...
}
```

Amazon SageMaker can initially call the List Models API without providing a value for next_page_token. If a nextPageToken field is returned as part of the response, it will be provided as the value for next_page_token in a subsequent List Models call. If a nextPageToken is not returned, it means that there are no more models to return.

GET MODEL API

This is a simple read API on the model_name entity.

```
GET /models/{model_name} HTTP/1.1
Accept: application/json

{
    "modelName" : "{model_name}",
    "modelUrl" : "/opt/ml/models/{model_name}/model",
}
```

Note

If model_name is not loaded, this API should return 404.

UNLOAD MODEL API

Instructs the Amazon SageMaker platform to instruct the customer container to unload a model from memory. This initiates the eviction of a candidate model as determined by the platform when starting the process of loading a new model. The resources provisioned to model_name should be reclaimed by the container when this API returns a response.

```
DELETE /models/{model_name}
```

Note

If model_name is not loaded, this API should return 404.

INVOKE MODEL API

Makes a prediction request from the particular model_name supplied. The Amazon SageMaker Runtime InvokeEndpoint request supports X-Amzn-Target-Model as a new header that takes the relative path of the model specified for invocation. The Amazon SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the CreateModel API call with the relative path of the model.

```
POST /models/{model_name}/invoke HTTP/1.1
Content-Type: ContentType
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes
X-Amzn-SageMaker-Target-Model: [relativePath]/{artifactName}.tar.gz
```

Note

If model_name is not loaded, this API should return 404.

Invoke a Multi-Model Endpoint

To invoke a multi-model endpoint, use the [runtime_InvokeEndpoint](#) from the Amazon SageMaker Runtime just as you would invoke a single model endpoint, with one change. Pass a new `TargetModel` parameter that specifies which of the models at the endpoint to target. The Amazon SageMaker Runtime `InvokeEndpoint` request supports `X-Amzn-Target-Model` as a new header that takes the relative path of the model specified for invocation. The Amazon SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the `CreateModel` API call with the relative path of the model.

The following example prediction request uses the [AWS SDK for Python \(Boto 3\)](#) in the sample notebook.

```
response = runtime_sm_client.invoke_endpoint(  
    EndpointName = 'my-endpoint',  
    ContentType = 'text/csv',  
    TargetModel = 'Houston_TX.tar.gz',  
    Body = body)
```

The multi-model endpoint dynamically loads target models as needed. You can observe this when running the [MME Sample Notebook](#) as it iterates through random invocations against multiple target models hosted behind a single endpoint. The first request against a given model takes longer because the model has to be downloaded from Amazon Simple Storage Service (Amazon S3) and loaded into memory. (This is called a cold start.) Subsequent calls finish faster because there's no additional overhead after the model has loaded.

Note

Invoking multi-model endpoints using the Amazon SageMaker Python SDK isn't supported.

Add or Remove Models

You can deploy additional models to a multi-model endpoint and invoke them through that endpoint immediately. When adding a new model, you don't need to update or bring down the endpoint, so you avoid the cost of creating and running a separate endpoint for each new model.

Amazon SageMaker unloads unused models from the container when the instance is reaching memory capacity and more models need to be downloaded into the container. Amazon SageMaker also deletes unused model artifacts from the instance storage volume when the volume is reaching capacity and new models need to be downloaded. The first invocation to a newly added model takes longer because the endpoint takes time to download the model from S3 to the container's memory in instance hosting the endpoint.

With the endpoint already running, copy a new set of model artifacts to the Amazon S3 location where you store your models.

```
# Add an AdditionalModel to the endpoint and exercise it  
aws s3 cp AdditionalModel.tar.gz s3://my-bucket/path/to/artifacts/
```

Important

To update a model, proceed as you would when adding a new model. Use a new and unique name. Don't overwrite model artifacts in Amazon S3 because the old version of the model might still be loaded in the containers or on the storage volume of the instances on the endpoint. Invocations to the new model could then invoke the old version of the model.

Client applications can request predictions from the additional target model as soon as it is stored in S3.

```
response = runtime_sm_client.invoke_endpoint(  
    EndpointName = 'my-endpoint',  
    ContentType = 'text/csv',  
    TargetModel = 'AdditionalModel.tar.gz',  
    Body = body)
```

```
EndpointName='endpoint_name',
ContentType='text/csv',
TargetModel='AdditionalModel.tar.gz',
Body=body)
```

To delete a model from a multi-model endpoint, stop invoking the model from the clients and remove it from the S3 location where model artifacts are stored.

Amazon SageMaker Model Monitor

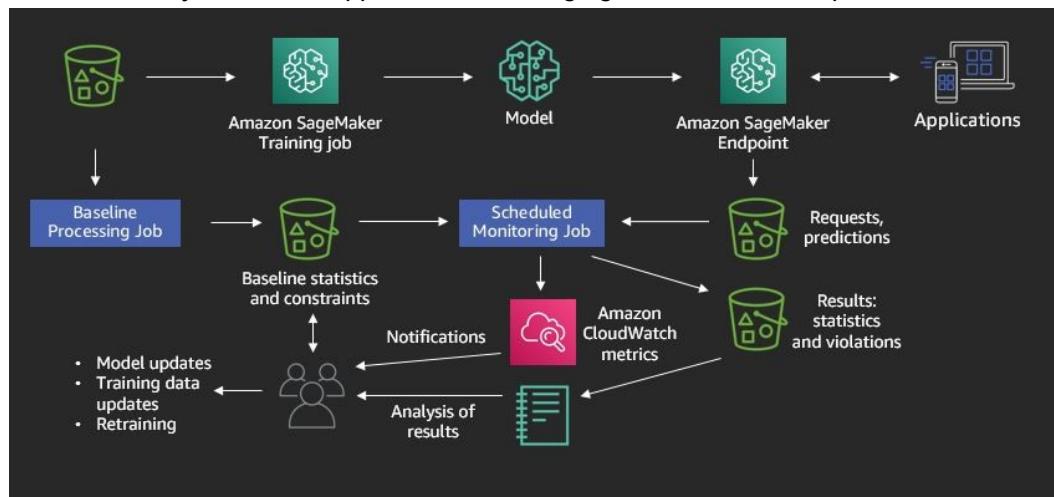
Amazon SageMaker Model Monitor continuously monitors the quality of Amazon SageMaker machine learning models in production. It enables developers to set alerts for when there are deviations in the model quality, such as data drift. Early and pro-active detection of these deviations enables you to take corrective actions, such as retraining models, auditing upstream systems, or fixing data quality issues without having to monitor models manually or build additional tooling. You can use Model Monitor pre-built monitoring capabilities that do not require coding. You also have the flexibility to monitor models by coding to provide custom analysis.

Topics

- [How Model Monitor Works \(p. 635\)](#)
- [Capture Data \(p. 636\)](#)
- [Create a Baseline \(p. 638\)](#)
- [Schedule Monitoring Jobs \(p. 639\)](#)
- [Interpret Results \(p. 642\)](#)
- [Advanced Topics \(p. 648\)](#)

How Model Monitor Works

Amazon SageMaker Model Monitor automatically monitors machine learning (ML) models in production and notifies you when data quality issues arise. ML models in production have to make predictions on real-life data that is not carefully curated like most training datasets. If the statistical nature of the data that your model receives while in production drifts away from the nature of the baseline data it was trained on, the model begins to lose accuracy in its predictions. Model Monitor uses rules to detect data drift and alerts you when it happens. The following figure shows how this process works.



To enable model monitoring, you take the following steps, which follow the path of the data through the various data collection, monitoring, and analysis processes.

- **Capture Data (p. 636)**: Enable the endpoint to capture data from incoming requests to a trained ML model and the resulting model predictions.
- **Create a Baseline (p. 638)**: Create a baseline from the dataset that was used to train the model. Compute baseline schema constraints and statistics for each feature using [Deequ](#), an open source library built on Apache Spark, which is used to measure data quality in large datasets.
- **Schedule Monitoring Jobs (p. 639)**: Create a monitoring schedule specifying what data to collect, how often to collect it, how to analyze it, and which reports to produce.
- **Interpret Results (p. 642)**: Inspect the reports, which compare the latest data with the baseline, and watch for any violations reported and for metrics and notifications from Amazon CloudWatch.

Note

Amazon SageMaker Model Monitor currently supports only endpoints that host a single model and does not support monitoring multi-model endpoints. For information on using multi-model endpoints, see [Host Multiple Models with Multi-Model Endpoints \(p. 625\)](#).

Model Monitor Sample Notebooks

For a sample notebook that takes you through the full end-to-end workflow for Model Monitor, see the [Introduction to Amazon SageMaker Model Monitor](#).

For a sample notebook that enables the model monitoring experience for an existing endpoint, see the [Enable Model Monitoring](#).

For a sample notebook that visualizes the statistics.json file for a selected execution in a monitoring schedule, see the [Model Monitor Visualization](#).

For instructions how to create and access Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). Once you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker samples. To open a notebook, choose its **Use** tab and choose **Create copy**.

Capture Data

After you have created an endpoint, configure the permissions and paths to Amazon S3 locations for storing data, report, and processing code.

```
import boto3
import re
import json
from sagemaker import get_execution_role, session

region= boto3.Session().region_name

role = get_execution_role()
print("RoleArn: {}".format(role))

# You can use a different bucket, but make sure the role you chose for this notebook
# has s3:PutObject permissions. This is the bucket into which the data is captured
bucket = session.Session(boto3.Session()).default_bucket()
print("Demo Bucket: {}".format(bucket))
prefix = 'sagemaker/DEMO-ModelMonitor'

data_capture_prefix = '{}/datacapture'.format(prefix)
s3_capture_upload_path = 's3://{}{}'.format(bucket, data_capture_prefix)
```

```

reports_prefix = '{}/reports'.format(prefix)
s3_report_path = 's3://{}{}'.format(bucket, reports_prefix)
code_prefix = '{}/code'.format(prefix)
s3_code_preprocessor_uri = 's3://{}{}'.format(bucket, code_prefix, 'preprocessor.py')
s3_code_postprocessor_uri = 's3://{}{}'.format(bucket, code_prefix, 'postprocessor.py')

print("Capture path: {}".format(s3_capture_upload_path))
print("Report path: {}".format(s3_report_path))
print("Preproc Code path: {}".format(s3_code_preprocessor_uri))
print("Postproc Code path: {}".format(s3_code_postprocessor_uri))

```

Upload the pre-trained model to Amazon S3:

```

model_file = open("model/your-prediction-model.tar.gz", 'rb')
s3_key = os.path.join(prefix, 'your-prediction-model.tar.gz')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(model_file)

```

Enable data capture: You specify the capture option called [DataCaptureConfig](#). You can capture the request payload, the response payload, or both with this configuration. The capture configuration applies to all variants.

```

from sagemaker.model_monitor import DataCaptureConfig

endpoint_name = 'your-pred-model-monitor-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config=DataCaptureConfig(
    enable_capture = True,
    sampling_percentage=100,
    destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                         instance_type='ml.m4.xlarge',
                         endpoint_name=endpoint_name
                         data_capture_config=data_capture_config)

```

Invoke the deployed model: You can now send data to this endpoint to get inferences in real time. Because you enabled the data capture in the previous steps, the request and response payload, along with some additional metadata, is saved in the Amazon S3 location that you specified in [DataCaptureConfig](#).

```

from sagemaker.predictor import RealTimePredictor
import time

predictor = RealTimePredictor(endpoint=endpoint_name,content_type='text/csv')

# get a subset of test data for a quick test
!head -120 test_data/test-dataset-input-cols.csv > test_data/test_sample.csv
print("Sending test traffic to the endpoint {}. \nPlease wait...".format(endpoint_name))

with open('test_data/test_sample.csv', 'r') as f:
    for row in f:
        payload = row.rstrip('\n')
        response = predictor.predict(data=payload)
        time.sleep(0.5)

print("Done!")

```

View captured data: List the data capture files stored in Amazon S3. Expect to see different files from different time periods, organized based on the hour when the invocation occurred.

```
s3_client = boto3.Session().client('s3')
current_endpoint_capture_prefix = '{}/{}'.format(data_capture_prefix, endpoint_name)
result = s3_client.list_objects(Bucket=bucket, Prefix=current_endpoint_capture_prefix)
capture_files = [capture_file.get("Key") for capture_file in result.get('Contents')]
print("Found Capture Files:")
print("\n ".join(capture_files))
```

The format of the Amazon S3 path is:

```
s3://{{destination-bucket-prefix}}/{{endpoint-name}}/{{variant-name}}/yyyy/mm/dd/hh/
filename.jsonl
```

Create a Baseline

The baseline calculations of statistics and constraints are needed as a standard against which data drift and other data quality issues can be detected. Amazon SageMaker Model Monitor provides a built-in container that provides the ability to suggest the constraints automatically for CSV and flat JSON input. This *sagemaker-model-monitor-analyzer* container also provides you with a range of model monitoring capabilities, including constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with [Deequ](#).

The training dataset that you used to trained the model is usually a good baseline dataset. The training dataset data schema and the inference dataset schema should exactly match (the number and order of the features). Note that the prediction/output column(s) are assumed to be the 1st column(s) in the training dataset. From the training dataset, you can ask Amazon SageMaker to suggest a set of baseline constraints and generate descriptive statistics to explore the data. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already have it in Amazon S3, you can point to it directly.

Create a baseline from a training dataset: When you have your training data ready and stored in Amazon S3, start a baseline processing job with `DefaultModelMonitor.suggest_baseline(..)` using the Amazon SageMaker Python SDK. This uses an [Amazon SageMaker Model Monitor Pre-built Container \(p. 641\)](#) that generates baseline statistics and suggests baseline constraints for the dataset and writes them to the `output_s3_uri` location that you specify.

```
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri+'/training-dataset-with-header.csv',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

Note

If you provide the feature/column names in the training dataset as the 1st row and set the `header=True` option as in the code sample above, Amazon SageMaker uses the feature name in the constraints and statistics file.

The baseline statistics for the dataset are contained in the `statistics.json` file and the suggested baseline constraints are contained in the `constraints.json` file in the location you specify with `output_s3_uri`.

Table: Output Files for Tabular Dataset Statistics and Constraints

File Name	Description
<code>statistics.json</code>	This file is expected to have columnar statistics for each feature in the dataset that is analyzed. See the schema for this file in the Schema for Statistics (statistics.json file) (p. 653) section.
<code>constraints.json</code>	This file is expected to have the constraints on the features observed. See the schema for this file in the Schema for Constraints (constraints.json file) (p. 655) section.

The Amazon SageMaker Python SDK provides convenience functions described to generate the baseline statistics and constraints. But if you want to call processing job directly for this purpose instead, you need to set the `Environment` map as in the following example.

```
"Environment": {
    "dataset_format": "{\"csv\": { \"header\": true}}",
    "dataset_source": "/opt/ml/processing/sm_input",
    "output_path": "/opt/ml/processing/sm_output",
    "publish_cloudwatch_metrics": "Disabled",
}
```

Schedule Monitoring Jobs

Amazon SageMaker Model Monitor provides you the ability to continuously monitor the data collected from the endpoints on a schedule. You can create a monitoring schedule with the [CreateMonitoringSchedule](#) API with a predefined periodic interval. For example, every x hours (x can range from 1 to 23).

With a Monitoring Schedule, Amazon SageMaker can kick off processing jobs at a specified frequency to analyze the data collected during a given period. Amazon SageMaker provides a pre-built container for performing analysis on tabular datasets. In the processing job, Amazon SageMaker compares the dataset for the current analysis with the baseline statistics, constraints provided and generate a violations report. In addition, CloudWatch metrics are emitted for each feature under analysis. Alternatively, you could choose to bring your own container as outlined in the [Bring Your Own Containers \(p. 650\)](#) topic.

You can create a model monitoring schedule for the endpoint created earlier. Use the baseline resources (constraints and statistics) to compare against the real-time traffic. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already have it in Amazon S3, you can point to it directly.

```
# copy over the training dataset to Amazon S3 (if you already have it in Amazon S3, you
could reuse it)
baseline_prefix = prefix + '/baselining'
baseline_data_prefix = baseline_prefix + '/data'
baseline_results_prefix = baseline_prefix + '/results'

baseline_data_uri = 's3://{}{}'.format(bucket,baseline_data_prefix)
baseline_results_uri = 's3://{}{}'.format(bucket, baseline_results_prefix)
print('Baseline data uri: {}'.format(baseline_data_uri))
print('Baseline results uri: {}'.format(baseline_results_uri))
```

```
training_data_file = open("test_data/training-dataset-with-header.csv", 'rb')
s3_key = os.path.join(baseline_prefix, 'data', 'training-dataset-with-header.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(training_data_file)
```

Create a model monitoring schedule for the endpoint using the baseline constraints and statistics to compare against real-time traffic.

```
from sagemaker.model_monitor import CronExpressionGenerator
from time import gmtime, strftime

mon_schedule_name = 'DEMO-xgb-churn-pred-model-monitor-schedule-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
my_default_monitor.create_monitoring_schedule(
    schedule_name=mon_schedule_name,
    endpoint_input=predictor.endpoint,
    post_analytics_processor_script=s3_code_postprocessor_uri,
    output_s3_uri=s3_report_path,
    statistics=my_default_monitor.baseline_statistics(),
    constraints=my_default_monitor.suggested_constraints(),
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
)
```

Describe and inspect the schedule: After you describe it, observe that the `MonitoringScheduleStatus` in `MonitoringScheduleSummary` returned by the `ListMonitoringSchedules` API changes to `Scheduled`.

```
desc_schedule_result = my_default_monitor.describe_schedule()
print('Schedule status: {}'.format(desc_schedule_result['MonitoringScheduleStatus']))
```

The cron Expression for Monitoring Schedule

To provide details for the monitoring schedule, use `ScheduleConfig`, which is a cron expression that describes details about the monitoring schedule.

Amazon SageMaker Model Monitor supports the following cron expressions:

- To set the job to start every hour:

Hourly: `cron(0 * ? * * *)`

- To run the job daily:

`cron(0 [00-23] ? * * *)`

For example, the following are valid cron expressions:

- Daily at 12 PM UTC: `cron(0 12 ? * * *)`
- Daily at 12 AM UTC: `cron(0 0 ? * * *)`

To support running every 6, 12 hours, Model Monitoring supports the following expression:

`cron(0 [00-23]/[01-24] ? * * *)`

For example, the following are valid cron expressions:

- Every 12 hours, starting at 5 PM UTC: `cron(0 17/12 ? * * *)`

- Every two hours, starting at 12 AM UTC: cron(0 0/2 ? * * *)

Note

- Although the cron expression is set to start at 5 PM UTC, note that there could be a delay of 0-20 minutes from the actual requested time to run the execution.
- If you want to run on a daily schedule, don't provide this parameter. Amazon SageMaker picks a time to run every day
- Currently, Amazon SageMaker only supports hourly integer rates between 1 hour and 24 hours.

Amazon SageMaker Model Monitor Pre-built Container

Amazon SageMaker provides a built-in container sagemaker-model-monitor-analyzer that provides you with a range of model monitoring capabilities, including constraint suggestion, statistics generation, constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with [Deequ](#). The prebuilt container for SageMaker Model Monitor can be accessed at:

`<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-model-monitor-analyzer`

For example: `159807026194.dkr.ecr.us-west-1.amazonaws.com/sagemaker-model-monitor-analyzer`

The following table lists the supported values for account IDs and corresponding AWS Region names.

ACCOUNT_ID	REGION_NAME
895015795356	eu-north-1
607024016150	me-south-1
126357580389	ap-south-1
680080141114	us-east-2
777275614652	us-east-2
468650794304	eu-west-1
048819808253	eu-central-1
539772159869	sa-east-1
001633400207	ap-east-1
156813124566	us-east-1
709848358524	ap-northeast-2
749857270468	eu-west-2
574779866223	ap-northeast-1
159807026194	us-west-2
890145073186	us-west-1

ACCOUNT_ID	REGION_NAME
245545462676	ap-southeast-1
563025443158	ap-southeast-2
536280801234	ca-central-1
453000072557	cn-north-1
453252182341	cn-northwest-1

To write your own analysis container, see the container contract described in [Customize Monitoring \(p. 648\)](#).

Interpret Results

Once you have run a baseline processing job and obtained statistics and constraint for your dataset, you can execute monitoring jobs that calculate statistics and list any violations encountered relative to the baseline constraints. Amazon CloudWatch metrics are also reported in your account by default. For information on viewing the results of monitoring in Amazon SageMaker Studio, see [Visualize Results in Amazon SageMaker Studio \(p. 647\)](#).

List executions: The schedule starts monitoring jobs at the specified intervals. The following code lists the latest five executions. If you are running this code after creating the hourly schedule, the executions might be empty, and you might have to wait until you cross the hour boundary (in UTC) to see the executions start. The following code includes the logic for waiting.

```
mon_executions = my_default_monitor.list_executions()
print("We created a hourly schedule above and it will kick off executions ON the hour (plus
0 - 20 min buffer.\nWe will have to wait till we hit the hour...")

while len(mon_executions) == 0:
    print("Waiting for the 1st execution to happen...")
    time.sleep(60)
    mon_executions = my_default_monitor.list_executions()
```

Inspect a specific execution: In the previous cell, you picked up the latest completed or failed scheduled execution. You can explore what went right or wrong. The terminal states are:

- **Completed:** The monitoring execution completed and no issues were found in the violations report.
- **CompletedWithViolations:** The execution completed, but constraint violations were detected.
- **Failed:** The monitoring execution failed, possibly due to client error (for example, a role issues) or infrastructure issues. To identify the cause, see the `FailureReason` and `ExitMessage`.

```
latest_execution = mon_executions[-1] # latest execution's index is -1, previous is -2 and
so on..
time.sleep(60)
latest_execution.wait(logs=False)

print("Latest execution status: {}".format(latest_execution.describe()['ProcessingJobStatus']))
print("Latest execution result: {}".format(latest_execution.describe()['ExitMessage']))

latest_job = latest_execution.describe()
if (latest_job['ProcessingJobStatus'] != 'Completed'):
```

```
print("====STOP==== \n No completed executions to inspect further. Please wait till an execution completes or investigate previously reported failures.")
```

```
report_uri=latest_execution.output.destination
print('Report Uri: {}'.format(report_uri))
```

List the generated reports:

```
from urllib.parse import urlparse
s3uri = urlparse(report_uri)
report_bucket = s3uri.netloc
report_key = s3uri.path.lstrip('/')
print('Report bucket: {}'.format(report_bucket))
print('Report key: {}'.format(report_key))

s3_client = boto3.Session().client('s3')
result = s3_client.list_objects(Bucket=report_bucket, Prefix=report_key)
report_files = [report_file.get("Key") for report_file in result.get('Contents')]
print("Found Report Files:")
print("\n ".join(report_files))
```

Violations report: If there are violations compared to the baseline, they are generated in the violations report. List the violations.

```
violations = my_default_monitor.latest_monitoring_constraint_violations()
pd.set_option('display.max_colwidth', -1)
constraints_df = pd.io.json.json_normalize(violations.body_dict["violations"])
constraints_df.head(10)
```

This applies only to datasets that contain tabular data. The following schema files specify the statistics calculated and the violations monitored for.

Table: Output Files for Tabular Datasets

File Name	Description
statistics.json	Contains columnar statistics for each feature in the dataset that is analyzed. See the schema of this file in the next topic.
constraints_violations.json	Contains a list of violations found in this current set of data as compared to the baseline statistics and constraints file specified in the <code>baseline_constraints</code> and <code>baseline_statistics</code> paths.

The [Amazon SageMaker Model Monitor Pre-built Container \(p. 641\)](#) saves a set of Amazon CloudWatch metrics for each feature by default.

The container code can emit CloudWatch metrics in this location: `/opt/ml/output/metrics/cloudwatch`. The schema for these files is outlined in the following topics.

Topics

- [Schema for Statistics \(statistics.json file\) \(p. 644\)](#)
- [Schema for Violations \(constraint_violations.json file\) \(p. 645\)](#)
- [CloudWatch Metrics \(p. 646\)](#)
- [Visualize Results in Amazon SageMaker Studio \(p. 647\)](#)

Schema for Statistics (statistics.json file)

Amazon SageMaker Model Monitor pre-built container computes per column/feature statistics. The statistics are calculated for the baseline dataset and also for the current dataset that is being analyzed.

```
{
    "version": 0,
    # dataset level stats
    "dataset": {
        "item_count": number
    },
    # feature level stats
    "features": [
        {
            "name": "feature-name",
            "inferred_type": "Fractional" | "Integral",
            "numerical_statistics": {
                "common": {
                    "num_present": number,
                    "num_missing": number
                },
                "mean": number,
                "sum": number,
                "std_dev": number,
                "min": number,
                "max": number,
                "distribution": {
                    "kll": {
                        "buckets": [
                            {
                                "lower_bound": number,
                                "upper_bound": number,
                                "count": number
                            }
                        ],
                        "sketch": {
                            "parameters": {
                                "c": number,
                                "k": number
                            },
                            "data": [
                                [
                                    num,
                                    num,
                                    num,
                                    num
                                ],
                                [
                                    num,
                                    num
                                ],
                                [
                                    num,
                                    num
                                ]
                            ]
                        }
                    }#sketch
                }#KLL
            }#distribution
        }#num_stats
    },
    {
        "name": "feature-name",
        "inferred_type": "String",
        "string_statistics": {

```

```

        "common": {
            "num_present": number,
            "num_missing": number
        },
        "distinct_count": number,
        "distribution": {
            "categorical": {
                "buckets": [
                    {
                        "value": "string",
                        "count": number
                    }
                ]
            }
        },
        #provision for custom stats
    }
}
]
}
}

```

Note the following:

- The prebuilt containers compute [KLL sketch](#) which is a compact quantiles sketch.
- By default, we materialize the distribution in 10 buckets. This is not currently configurable.

Schema for Violations (constraint_violations.json file)

The violations file is generated as the output of a `MonitoringExecution`, which lists the results of evaluating the constraints (specified in the `constraints.json` file) against the current dataset that was analyzed. The Amazon SageMaker Model Monitor pre-built container provides the following violation checks.

```

{
    "violations": [
        {
            "feature_name" : "string",
            "constraint_check_type" :
                "data_type_check",
                | "completeness_check",
                | "baseline_drift_check",
                | "missing_column_check",
                | "extra_column_check",
                | "categorical_values_check"
            "description" : "string"
        }
    ]
}

```

Table: Types of Violations Monitored

Violation Check Type	Description
<code>data_type_check</code>	If the data types in the current execution are not the same as in the baseline dataset, this violation is flagged. During the baseline step, the generated constraints suggest the inferred data type for each column. The <code>monitoring_config.datatype_check_threshold</code>

Violation Check Type	Description
	parameter can be tuned to adjust the threshold on when it is flagged as a violation.
completeness_check	If the completeness (% of non-null items) observed in the current execution exceeds the threshold specified in completeness threshold specified per feature, this violation is flagged. During the baseline step, the generated constraints suggest a completeness value.
baseline_drift_check	If the calculated distribution distance between the current and the baseline datasets is more than the threshold specified in monitoring_config.comparison_threshold, this violation is flagged.
missing_column_check	If the number of columns in the current dataset is less than the number in the baseline dataset, this violation is flagged.
extra_column_check	If the number of columns in the current dataset is more than the number in the baseline, this violation is flagged.
categorical_values_check	If there are more unknown values in the current dataset than in the baseline dataset, this violation is flagged. This value is dictated by the threshold in monitoring_config.domain_content_threshold.

CloudWatch Metrics

Built-in Amazon SageMaker Model Monitor container for CloudWatch metrics: When the emit_metrics option is Enabled in the baseline constraints file, Amazon SageMaker emits these metrics for each feature/column observed in the dataset in the /aws/sagemaker/Endpoints/data-metric namespace with EndpointName and ScheduleName dimensions.

For numerical fields:

- Metric : Max → query for MetricName: feature_data_{feature_name}, Stat: Max
- Metric : Min → query for MetricName: feature_data_{feature_name}, Stat: Min
- Metric : Sum → query for MetricName: feature_data_{feature_name}, Stat: Sum
- Metric : SampleCount → query for MetricName: feature_data_{feature_name}, Stat: SampleCount
- Metric : Average → query for MetricName: feature_data_{feature_name}, Stat: Average

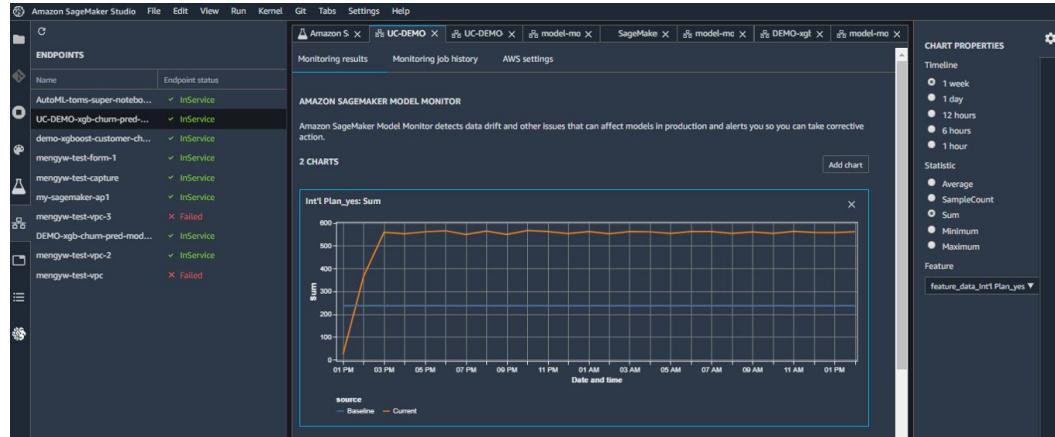
For both numerical and string fields:

- Metric: Completeness → query for MetricName: feature_non_null_{feature_name}, Stat: Sum
- Metric: Baseline Drift → query for MetricName: feature_baseline_drift_{feature_name}, Stat: Sum

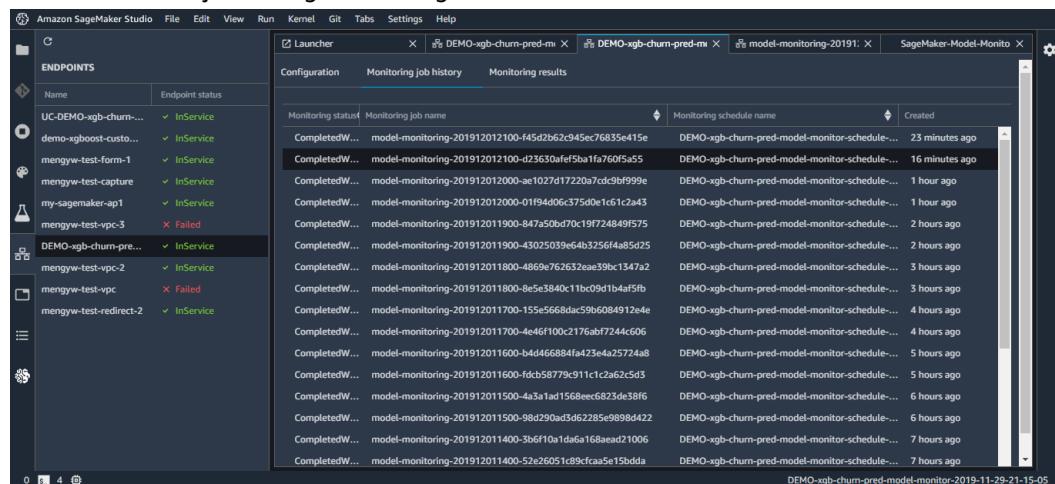
Visualize Results in Amazon SageMaker Studio

You can also visualize the results of monitoring in Amazon SageMaker Studio. For information about the onboarding process for using Studio, see [Onboard to Amazon SageMaker Studio \(p. 16\)](#).

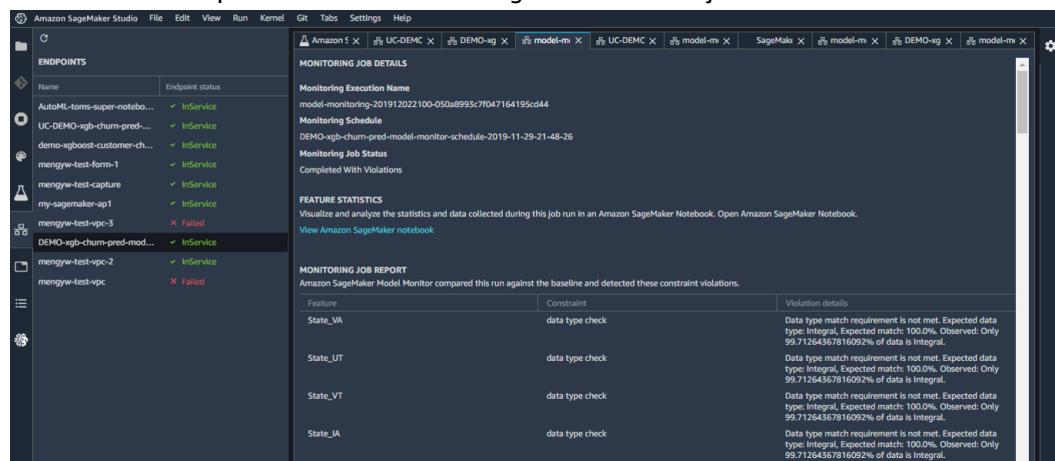
You can view monitoring results at your endpoints.



You can view the jobs being monitored.



You can take a deep dive into each monitoring results for each job.



Advanced Topics

The following sections contain more advanced tasks that explain how to customize monitoring using preprocessing and postprocessing scripts, how to build your own container, and how to use CloudFormation to create a monitoring schedule.

Topics

- [Customize Monitoring \(p. 648\)](#)
- [Create a Monitoring Schedule with an AWS CloudFormation Custom Resource \(p. 657\)](#)

Customize Monitoring

In addition to using the built-in monitoring mechanisms, you can create your own custom monitoring schedules and procedures using preprocessing and postprocessing scripts or by using or building your own container.

Topics

- [Preprocessing and Postprocessing \(p. 648\)](#)
- [Bring Your Own Containers \(p. 650\)](#)

Preprocessing and Postprocessing

In addition to using the built-in mechanisms, you can extend the code with the preprocessing and postprocessing scripts.

Topics

- [Postprocessing Script \(p. 648\)](#)
- [Preprocessing Script \(p. 648\)](#)

Postprocessing Script

You can extend the code with the post processing script by following this contract.

```
def postprocess_handler():
    print("Hello from post-proc script!")
```

Specify it as a path in Amazon Simple Storage Service (Amazon S3) in the CreateMonitoringSchedule request.

```
.MonitoringAppSpecification.PostAnalyticsProcessorSourceUri.
```

Preprocessing Script

The Amazon SageMaker Model Monitor container works only with tabular or flattened json structures. We provide a per-record preprocessor for some small changes required to transform the dataset. For example, if your output is an array [1.0, 2.1], you need to convert this into a flattened JSON, like {"prediction0": 1.0, "prediction1": 2.1}. A sample implementation might look like the following.

```
def preprocess_handler(inference_record):
    event_data = inference_record.event_data
    input_data = {}
    output_data = {}
```

```

    input_data['feature0'] = random.randint(1, 3)
    input_data['feature1'] = random.uniform(0, 1.6)
    input_data['feature2'] = random.uniform(0, 1.6)

    output_data['prediction0'] = random.uniform(1, 30)

    return {**input_data, **output_data}

```

Specify it as a path in Amazon S3 in the `CreateMonitoringSchedule` request:

```
.MonitoringAppSpecification.RecordPreprocessorSourceUri.
```

The structure of the `inference_record` is defined as follows.

```

KEY_EVENT_METADATA = "eventMetadata"
KEY_EVENT_METADATA_EVENT_ID = "eventId"
KEY_EVENT_METADATA_EVENT_TIME = "inferenceTime"
KEY_EVENT_METADATA_CUSTOM_ATTR = "customAttributes"

KEY_EVENTDATA = "captureData"
KEY_EVENTDATA_INPUT = "endpointInput"
KEY_EVENTDATA_OUTPUT = "endpointOutput"
KEY_EVENTDATA_ENCODING = "encoding"
KEY_EVENTDATA_DATA = "data"
KEY_EVENTDATA_OBSERVED_CONTENT_TYPE = "observedContentType"
KEY_EVENTDATA_MODE = "mode"

KEY_EVENT_VERSION = "eventVersion"

"""
{
    "captureData": {
        "endpointInput": {
            "observedContentType": "text/csv",
            "mode": "INPUT",
            "data": "132,25,113.2,96,269.9,107,,0,0,0,0,0,0,1,0,1,0,0,1",
            "encoding": "CSV"
        },
        "endpointOutput": {
            "observedContentType": "text/csv; charset=utf-8",
            "mode": "OUTPUT",
            "data": "0.01076381653547287",
            "encoding": "CSV"
        }
    },
    "eventMetadata": {
        "eventId": "fecab1ab1-8025-47e3-8f6a-99e3fdd7b8d9",
        "inferenceTime": "2019-11-20T23:33:12Z"
    },
    "eventVersion": "0"
}
"""

class EventConfig:
    def __init__(self, endpoint, variant, start_time, end_time):
        self.endpoint = endpoint
        self.variant = variant
        self.start_time = start_time
        self.end_time = end_time

class EventMetadata:
    def __init__(self, event_metadata_dict):
        self.event_id = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_ID, None)

```

```

        self.event_time = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_TIME, None)
        self.custom_attribute =
event_metadata_dict.get(KEY_EVENTDATA_OBSERVED_CONTENT_TYPE, None)

class EventData:
    def __init__(self, data_dict):
        self.encoding = data_dict.get(KEY_EVENTDATA_ENCODING, None)
        self.data = data_dict.get(KEY_EVENTDATA_DATA, None)
        self.observedContentType = data_dict.get(KEY_EVENTDATA_OBSERVED_CONTENT_TYPE, None)
        self.mode = data_dict.get(KEY_EVENTDATA_MODE, None)

    def as_dict(self):
        ret = {
            KEY_EVENTDATA_ENCODING: self.encoding,
            KEY_EVENTDATA_DATA: self.data,
            KEY_EVENTDATA_OBSERVED_CONTENT_TYPE: self.observedContentType,
        }
        return ret

class CapturedData:
    def __init__(self, event_dict):
        self.event_metadata = None
        self.endpoint_input = None
        self.endpoint_output = None
        self.event_version = None
        self.event_dict = event_dict
        self._event_dict_postprocessed = False
        if KEY_EVENT_METADATA in event_dict:
            self.event_metadata = EventMetadata(event_dict[KEY_EVENT_METADATA])
        if KEY_EVENTDATA in event_dict:
            if KEY_EVENTDATA_INPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_input = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT])
            if KEY_EVENTDATA_OUTPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_output = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_OUTPUT])
        if KEY_EVENT_VERSION in event_dict:
            self.event_version = event_dict[KEY_EVENT_VERSION]

    def as_dict(self):
        if self._event_dict_postprocessed is True:
            return self.event_dict
        if KEY_EVENTDATA in self.event_dict:
            if KEY_EVENTDATA_INPUT in self.event_dict[KEY_EVENTDATA]:
                self.event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT] =
self.endpoint_input.as_dict()
            if KEY_EVENTDATA_OUTPUT in self.event_dict[KEY_EVENTDATA]:
                self.event_dict[KEY_EVENTDATA][KEY_EVENTDATA_OUTPUT] =
self.endpoint_output.as_dict()
        self._event_dict_postprocessed = True
        return self.event_dict

```

Bring Your Own Containers

Amazon SageMaker Model Monitor provides a prebuilt container with ability to analyze the data captured from Endpoints for tabular datasets. If you would like to bring your own container, Model Monitor provides extension points which you can leverage.

Under the hood, when you create a `MonitoringSchedule`, Amazon SageMaker Model Monitor ultimately kicks off processing jobs. Hence the container needs to be aware of the processing job contract documented in the [Build Your Own Processing Container \(p. 242\)](#) topic. Note that Amazon

SageMaker Model Monitor kicks off the processing job on your behalf per the schedule. While invoking Model Monitor sets up additional environment variables for you so that your container has enough context to process the data for that particular execution of the scheduled monitoring. For additional information on container inputs, see the [Container Contract Inputs \(p. 651\)](#).

In the container, using the above environment variables/context, you can now analyze the dataset for the current period in your custom code. Once this analysis is completed, you can choose to emit your reports to be uploaded to S3. The reports that the pre-built container generates are documented in [Container Contract Outputs \(p. 652\)](#). If you would like the visualization of the reports to work in Amazon SageMaker Studio, you should follow the same format. You can also choose to emit completely custom reports.

You also have an option to emit CloudWatch metrics from the container by following the instructions in <https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-byoc-cloudwatch.html>.

Topics

- [Container Contract Inputs \(p. 651\)](#)
- [Container Contract Outputs \(p. 652\)](#)
- [CloudWatch Metrics \(p. 657\)](#)

Container Contract Inputs

The Amazon SageMaker Model Monitor platform invokes your container code according to a specified schedule. If you chose to write your own container code, the following environment variables are available for your container code. In this context, you can analyze the current dataset or evaluate the constraints if you chose to and emit metrics, if applicable.

```
"Environment": {  
    "dataset_format": "{\"sagemakerCaptureJson\": {\"captureIndexNames\": [\"endpointInput\", \"endpointOutput\"]}}",  
    "dataset_source": "/opt/ml/processing/endpointdata",  
    "end_time": "2019-12-01T16: 20: 00Z",  
    "output_path": "/opt/ml/processing/resultdata",  
    "publish_cloudwatch_metrics": "Disabled",  
    "sagemaker_endpoint_name": "endpoint-name",  
    "sagemaker_monitoring_schedule_name": "schedule-name",  
    "start_time": "2019-12-01T15: 20: 00Z"  
}
```

Table: Parameters

Parameter Name	Description
dataset_format	For a job started from a MonitoringSchedule backed by an Endpoint, this is sageMakerCaptureJson with the capture indices endpointInput,or endpointOutput, or both.
dataset_source	The local path in which the data corresponding to the monitoring period, as specified by start_time and end_time, are available. At this path, the data is available in /{endpoint-name}/{variant-name}/yyyy/mm/dd/hh. We sometimes download more than what is specified by the start and end times. It is up to the container code to parse the data as required.

Parameter Name	Description
<code>output_path</code>	The local path to write output reports and other files. You specify this parameter in the <code>CreateMonitoringSchedule</code> request as <code>MonitoringOutputConfig.MonitoringOutput[0].LocalPath</code> . It is uploaded to the S3Uri path specified in <code>MonitoringOutputConfig.MonitoringOutput[0].S3Uri</code> .
<code>publish_cloudwatch_metrics</code>	For a job launched by <code>CreateMonitoringSchedule</code> , this parameter is set to <code>Enabled</code> . The container can choose to write the Amazon CloudWatch output file at <code>[filepath]</code> .
<code>sagemaker_endpoint_name</code>	The name of the <code>Endpoint</code> that this scheduled job was launched for.
<code>sagemaker_monitoring_schedule_name</code>	The name of the <code>MonitoringSchedule</code> that launched this job.
<code>*sagemaker_endpoint_datacapture_prefix*</code>	The prefix specified in the <code>DataCaptureConfig</code> parameter of the <code>Endpoint</code> . The container can use this if it needs to directly access more data than already downloaded by Amazon SageMaker at the <code>dataset_source</code> path.
<code>start_time, end_time</code>	The time window for this analysis run. For example, for a job scheduled to run at 05:00 UTC and a job that runs on 20/02/2020, <code>start_time:</code> is <code>2020-02-19T06:00:00Z</code> and <code>end_time:</code> is <code>2020-02-20T05:00:00Z</code>
<code>baseline_constraints:</code>	The local path of the baseline constraint file specified in <code>BaselineConfig.ConstraintResource.S3Uri</code> . This is available only if this parameter was specified in the <code>CreateMonitoringSchedule</code> request.
<code>baseline_statistics</code>	The local path to the baseline statistics file specified in <code>BaselineConfig.StatisticsResource.S3Uri</code> . This is available only if this parameter was specified in the <code>CreateMonitoringSchedule</code> request.

Container Contract Outputs

The container can analyze the data available in the `*dataset_source*` path and write reports to the path in `*output_path*`. The container code can write any reports that suit your needs.

If you use the following structure and contract, certain output files are treated specially by Amazon SageMaker in the visualization and API affordances. This applies only to tabular datasets.

Table: Output Files for Tabular Datasets

File Name	Description
statistics.json	This file is expected to have columnar statistics for each feature in the dataset that is analyzed. See the schema for this file in the next section.
constraints.json	This file is expected to have the constraints on the features observed. See the schema of this file below.
constraints_violations.json	This file is expected to have the list of violations found in this current set of data as compared to the baseline statistics and constraints file specified in the <code>baseline_constraints</code> and <code>baseline_statistics</code> path.

In addition, if the `publish_cloudwatch_metrics` value is "Enabled" container code can emit Amazon CloudWatch metrics in this location: `/opt/ml/output/metrics/cloudwatch`. The schema for these files is described in the following sections.

Topics

- [Schema for Statistics \(statistics.json file\) \(p. 653\)](#)
- [Schema for Constraints \(constraints.json file\) \(p. 655\)](#)

Schema for Statistics (statistics.json file)

The schema defined in the `statistics.json` file specifies the statistical parameters to be calculated for the baseline and data that is captured. It also configures the bucket to be used by [KLL](#), a very compact quantiles sketch with lazy compaction scheme.

```
{
    "version": 0,
    # dataset level stats
    "dataset": {
        "item_count": number
    },
    # feature level stats
    "features": [
        {
            "name": "feature-name",
            "inferred_type": "Fractional" | "Integral",
            "numerical_statistics": {
                "common": {
                    "num_present": number,
                    "num_missing": number
                },
                "mean": number,
                "sum": number,
                "std_dev": number,
                "min": number,
                "max": number,
                "distribution": {
                    "kll": {
                        "buckets": [
                            {
                                "lower_bound": number,
                                "upper_bound": number,
                                "count": number
                            }
                        ]
                    }
                }
            }
        }
    ]
}
```

```

                "count": number
            }
        ],
        "sketch": {
            "parameters": {
                "c": number,
                "k": number
            },
            "data": [
                [
                    num,
                    num,
                    num,
                    num
                ],
                [
                    num,
                    num
                ][
                    num,
                    num
                ]
            ]
        }#sketch
    }#KLL
}#distribution
}#num_stats
},
{
    "name": "feature-name",
    "inferred_type": "String",
    "string_statistics": {
        "common": {
            "num_present": number,
            "num_missing": number
        },
        "distinct_count": number,
        "distribution": {
            "categorical": {
                "buckets": [
                    {
                        "value": "string",
                        "count": number
                    }
                ]
            }
        },
        "#provision for custom stats
    }
}
]
}

```

Note the following:

- The specified metrics are recognized by Amazon SageMaker in later visualization changes. The container can emit more metrics if required.
- **KLL sketch** is the recognized sketch. Custom containers can write their own representation, but it won't be recognized by Amazon SageMaker in visualizations.
- By default, the distribution is materialized in 10 buckets. You can't change this.

Schema for Constraints (constraints.json file)

A constraints.json file is used to express the constraints that a dataset must satisfy. Amazon SageMaker Model Monitor containers can use the constraints.json file to evaluate datasets against. Pre-built containers provide the ability to generate the constraints.json file automatically for a baseline dataset. If you bring your own container, you can provide it with similar abilities or you can create the constraints.json file in some other way. Here is the schema for the constraint file that the prebuilt container uses. Bring our own containers can adopt the same format or enhance it as required.

```
{
    "version" : 0,

    "features": [
        {
            "name": "string",
            "inferred_type": "Integral" | "Fractional" |
                | "String" | "Unknown",
            "completeness": number, # denotes observed non-null value percentage
            "num_constraints" : {
                "is_non_negative": boolean,
            },
            "string_constraints" : {
                "domains": [
                    "list of",
                    "observed values",
                    "for small cardinality"
                ],
            },
            "monitoringConfigOverrides" : {

                }#monitoringConfigOverrides
            }#feature
        ]#features

        # options to control monitoring for this feature with monitoring jobs
        # See the following table for notes on what each constraint is doing.
    "monitoring_config": {
        "evaluate_constraints": "Enabled",
        "emit_metrics": "Enabled",
        "datatype_check_threshold": 1.0,
        "domain_content_threshold": 1.0,
        "distribution_constraints": {
            "perform_comparison": "Enabled",
            "comparison_threshold": 0.1,
            "comparison_method": "Simple" | "Robust"
        }
    }#schema
}
```

Table: Monitoring Constraints

Constraint	Description
<code>evaluate_constraints</code>	When <code>Enabled</code> , evaluates whether the current dataset being analysed satisfies the constraints specified in the constraints.json file taken as a baseline. Valid values: <code>Enabled</code> or <code>Disabled</code> Default: <code>Enabled</code>
<code>emit_metrics</code>	When <code>Enabled</code> , emits CloudWatch metrics for the data contained in the file.

Constraint	Description
	<p>Valid values: <code>Enabled</code> or <code>Disabled</code></p> <p>Default: <code>Enabled</code></p>
<code>datatype_check_threshold</code>	<p>If the threshold is above the value of the specified <code>datatype_check_threshold</code>, this causes a failure that is treated as a violation in the violation report. If the data types in the current execution are not the same as in the baseline dataset, this threshold is used to evaluate if it needs to be flagged as a violation.</p> <p>During the baseline step, the generated constraints suggest the inferred data type for each column. The <code>datatype_check_threshold</code> parameter can be tuned to adjust the threshold on when it is flagged as a violation.</p> <p>Valid values: <code>float</code></p> <p>Default: <code>1.0</code></p>
<code>domain_content_threshold</code>	<p>If there are more unknown values for a String field in the current dataset than in the baseline dataset, this threshold can be used to dictate if it needs to be flagged as a violation.</p> <p>Valid values: <code>float</code></p> <p>Default: <code>1.0</code></p>
<code>distribution_constraints</code>	<p><code>perform_comparison</code></p> <p>When <code>Enabled</code>, this flag instructs the code to perform a distribution comparison between the baseline distribution and the distribution observed for the current dataset.</p> <p>Valid values: <code>Enabled</code> or <code>Disabled</code></p> <p>Default: <code>Enabled</code></p> <p><code>comparison_threshold</code></p> <p>If the threshold is above the value set for the <code>comparison_threshold</code>, this causes a failure that is treated as a violation in the violation report. The distance is calculated by getting the maximum absolute difference between the cumulative distribution functions of two distributions.</p> <p>Valid values: <code>float</code></p> <p>Default: <code>1.0</code>.</p>

Constraint	Description
	<p>comparison_method</p> <p>Whether to calculate <code>linf_simple</code> or <code>linf_robust</code>. The <code>linf_simple</code> is based on the maximum absolute difference between the cumulative distribution functions of two distributions. Calculating <code>linf_robust</code> is based on <code>linf_simple</code>, but is used when there are not enough samples. The <code>linf_robust</code> formula is based on the Two-sample Kolmogorov-Smirnov test.</p> <p>Valid values: <code>linf_simple</code> or <code>linf_robust</code>.</p>

CloudWatch Metrics

If the `publish_cloudwatch_metrics` value is `Enabled` in the `Environment` map in the `/opt/ml/processing/processingjobconfig.json` file, the container code emits Amazon CloudWatch metrics in this location: `/opt/ml/output/metrics/cloudwatch`.

The schema for this file is closely based on the CloudWatch PutMetrics API. The namespace is not specified here. It defaults to `/aws/sagemaker/Endpoint/data-metrics`. However, you can specify dimensions. We recommend that you add the `Endpoint` and `MonitoringSchedule` dimensions at a minimum.

```
{
    "MetricName": "", # Required
    "Timestamp": "2019-11-26T03:00:00Z", # Required
    "Dimensions" : [{"Name":"Endpoint","Value":"endpoint_0"}, {"Name":"MonitoringSchedule","Value":"schedule_0"}]
    "Value": Float,
    # Either the Value or the StatisticValues field can be populated and not both.
    "StatisticValues": {
        "SampleCount": Float,
        "Sum": Float,
        "Minimum": Float,
        "Maximum": Float
    },
    "Unit": "Count", # Optional
}
```

Create a Monitoring Schedule with an AWS CloudFormation Custom Resource

To use AWS CloudFormation to create a monitoring schedule, use an AWS CloudFormation custom resource. The custom resource is in Python. To deploy it, see [Python Lambda deployment](#).

Custom Resource

Start by adding a custom resource to your AWS CloudFormation template. This will point to a AWS Lambda function that you create next.

This resource allows you to customize the parameters for the monitoring schedule. You can add or remove more parameters by modifying the AWS CloudFormation resource and the Lambda function in the following example resource.

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Resources": {
        "MonitoringSchedule": {
            "Type": "Custom::MonitoringSchedule",
            "Version": "1.0",
            "Properties": {
                "ServiceToken": "arn:aws:lambda:us-west-2:111111111111:function:lambda-
name",
                "ScheduleName": "YourScheduleName",
                "EndpointName": "YourEndpointName",
                "BaselineConstraintsUri": "s3://your-baseline-constraints/
constraints.json",
                "BaselineStatisticsUri": "s3://your-baseline-stats/statistics.json",
                "PostAnalyticsProcessorSourceUri": "s3://your-post-processor/
postprocessor.py",
                "RecordPreprocessorSourceUri": "s3://your-preprocessor/preprocessor.py",
                "InputLocalPath": "/opt/ml/processing/endpointdata",
                "OutputLocalPath": "/opt/ml/processing/localpath",
                "OutputS3URI": "s3://your-output-uri",
                "ImageURI": "111111111111.dkr.ecr.us-west-2.amazonaws.com/your-image",
                "ScheduleExpression": "cron(0 * ? * * *)",
                "PassRoleArn": "arn:aws:iam::111111111111:role/AmazonSageMaker-
ExecutionRole"
            }
        }
    }
}
```

Lambda Custom Resource Code

This AWS CloudFormation custom resource uses the [Custom Resource Helper](#) AWS library, which you can install with pip using `pip install crhelper`.

This Lambda function is invoked by AWS CloudFormation during the creation and deletion of the stack. This Lambda function is responsible for creating and deleting the monitoring schedule and using the parameters defined in the custom resource described in the preceding section.

```
import boto3
import botocore
import logging

from crhelper import CfnResource
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
sm = boto3.client('sagemaker')

# cfnhelper makes it easier to implement a CloudFormation custom resource
helper = CfnResource()

# CFN Handlers

def handler(event, context):
    helper(event, context)

@helper.create
def create_handler(event, context):
    """
    Called when CloudFormation custom resource sends the create event
    """
    pass
```

```

create_monitoring_schedule(event)

@helper.delete
def delete_handler(event, context):
    """
    Called when CloudFormation custom resource sends the delete event
    """
    schedule_name = get_schedule_name(event)
    delete_monitoring_schedule(schedule_name)

@helper.poll_create
def poll_create(event, context):
    """
    Return true if the resource has been created and false otherwise so
    CloudFormation polls again.
    """
    schedule_name = get_schedule_name(event)
    logger.info('Polling for creation of schedule: %s', schedule_name)
    return is_schedule_ready(schedule_name)

@helper.update
def noop():
    """
    Not currently implemented but crhelper will throw an error if it isn't added
    """
    pass

# Helper Functions

def get_schedule_name(event):
    return event['ResourceProperties']['ScheduleName']

def create_monitoring_schedule(event):
    schedule_name = get_schedule_name(event)
    monitoring_schedule_config = create_monitoring_schedule_config(event)

    logger.info('Creating monitoring schedule with name: %s', schedule_name)

    sm.create_monitoring_schedule(
        MonitoringScheduleName=schedule_name,
        MonitoringScheduleConfig=monitoring_schedule_config)

def is_schedule_ready(schedule_name):
    is_ready = False

    schedule = sm.describe_monitoring_schedule(MonitoringScheduleName=schedule_name)
    status = schedule['MonitoringScheduleStatus']

    if status == 'Scheduled':
        logger.info('Monitoring schedule (%s) is ready', schedule_name)
        is_ready = True
    elif status == 'Pending':
        logger.info('Monitoring schedule (%s) still creating, waiting and polling
again...', schedule_name)
    else:
        raise Exception('Monitoring schedule ({}) has unexpected status:
{}'.format(schedule_name, status))

    return is_ready

def create_monitoring_schedule_config(event):
    props = event['ResourceProperties']

    return {

```

```

    "ScheduleConfig": {
        "ScheduleExpression": props["ScheduleExpression"],
    },
    "MonitoringJobDefinition": {
        "BaselineConfig": {
            "ConstraintsResource": {
                "S3Uri": props['BaselineConstraintsUri'],
            },
            "StatisticsResource": {
                "S3Uri": props['BaselineStatisticsUri'],
            }
        },
        "MonitoringInputs": [
            {
                "EndpointInput": {
                    "EndpointName": props["EndpointName"],
                    "LocalPath": props["InputLocalPath"],
                }
            }
        ],
        "MonitoringOutputConfig": {
            "MonitoringOutputs": [
                {
                    "S3Output": {
                        "S3Uri": props["OutputS3URI"],
                        "LocalPath": props["OutputLocalPath"],
                    }
                }
            ],
        },
        "MonitoringResources": {
            "ClusterConfig": {
                "InstanceCount": 1,
                "InstanceType": "ml.t3.medium",
                "VolumeSizeInGB": 50,
            }
        },
        "MonitoringAppSpecification": {
            "ImageUri": props["ImageURI"],
            "RecordPreprocessorSourceUri": props['PostAnalyticsProcessorSourceUri'],
            "PostAnalyticsProcessorSourceUri": props['PostAnalyticsProcessorSourceUri'],
            "ImageUri": props["ImageURI"],
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 300
        },
        "RoleArn": props["PassRoleArn"],
    }
}

def delete_monitoring_schedule(schedule_name):
    logger.info('Deleting schedule: %s', schedule_name)
    try:
        sm.delete_monitoring_schedule(MonitoringScheduleName=schedule_name)
    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFound':
            logger.info('Resource not found, nothing to delete')
        else:
            logger.error('Unexpected error while trying to delete monitoring schedule')
            raise e

```

Deploy an Inference Pipeline

An *inference pipeline* is an Amazon SageMaker model that is composed of a linear sequence of two to five containers that process requests for inferences on data. You use an inference pipeline to define and deploy any combination of pretrained Amazon SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. You can use an inference pipeline to combine preprocessing, predictions, and post-processing data science tasks. Inference pipelines are fully managed.

You can add Amazon SageMaker Spark ML Serving and scikit-learn containers that reuse the data transformers developed for training models. The entire assembled inference pipeline can be considered as an Amazon SageMaker model that you can use to make either real-time predictions or to process batch transforms directly without any external preprocessing.

Within an inference pipeline model, Amazon SageMaker handles invocations as a sequence of HTTP requests. The first container in the pipeline handles the initial request, then the intermediate response is sent as a request to the second container, and so on, for each container in the pipeline. Amazon SageMaker returns the final response to the client.

When you deploy the pipeline model, Amazon SageMaker installs and runs all of the containers on each Amazon Elastic Compute Cloud (Amazon EC2) instance in the endpoint or transform job. Feature processing and inferences run with low latency because the containers are co-located on the same EC2 instances. You define the containers for a pipeline model using the [CreateModel](#) operation or from the console. Instead of setting one `PrimaryContainer`, you use the `Containers` parameter. To set the containers that make up the pipeline, you also specify the order in which the containers are executed.

A pipeline model is immutable, but you can update an inference pipeline by deploying a new one using the [UpdateEndpoint](#) operation. This modularity supports greater flexibility during experimentation.

There are no additional costs for using this feature. You pay only for the instances running on an endpoint.

Topics

- [Sample Notebooks for Inference Pipelines \(p. 661\)](#)
- [Feature Processing with Spark ML and Scikit-learn \(p. 662\)](#)
- [Create a Pipeline Model \(p. 662\)](#)
- [Run Real-time Predictions with an Inference Pipeline \(p. 665\)](#)
- [Run Batch Transforms with Inference Pipelines \(p. 667\)](#)
- [Inference Pipeline Logs and Metrics \(p. 668\)](#)
- [Troubleshoot Inference Pipelines \(p. 673\)](#)

Sample Notebooks for Inference Pipelines

For a sample notebook that uploads and processes a dataset, trains a model, and builds a pipeline model, see the [Inference Pipelines with Spark ML and XGBoost on Abalone](#) notebook. This notebook shows how you can build your machine learning pipeline by using Spark feature Transformers and the Amazon SageMaker XGBoost algorithm. After training the model, the sample shows how to deploy the pipeline (feature Transformer and XGBoost) for real-time predictions and also performs a batch transform job using the same pipeline.

For more examples that show how to create and deploy inference pipelines, see the [Inference Pipelines with SparkML and BlazingText on DBpedia](#) and [Training using SparkML on EMR and hosting on SageMaker](#) sample notebooks. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#).

To see a list of all the Amazon SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. There are three inference pipeline notebooks. The first two inference pipeline notebooks just described are located in the `advanced_functionality` folder and the third notebook is in the `sagemaker-python-sdk` folder. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Feature Processing with Spark ML and Scikit-learn

Before training a model with either Amazon SageMaker built-in algorithms or custom algorithms, you can use Spark and scikit-learn preprocessors to transform your data and engineer features.

Feature Processing with Spark ML

You can run Spark ML jobs with [AWS Glue](#), a serverless ETL (extract, transform, load) service, from your Amazon SageMaker notebook. You can also connect to existing EMR clusters to run Spark ML jobs with [Amazon EMR](#). To do this, you need an AWS Identity and Access Management (IAM) role that grants permission for making calls from your Amazon SageMaker notebook to AWS Glue.

Note

To see which Python and Spark versions AWS Glue supports, refer to [AWS Glue Release Notes](#).

After engineering features, you package and serialize Spark ML jobs with MLeap into MLeap containers that you can add to an inference pipeline. You don't need to use externally managed Spark clusters. With this approach, you can seamlessly scale from a sample of rows to terabytes of data. The same transformers work for both training and inference, so you don't need to duplicate preprocessing and feature engineering logic or develop a one-time solution to make the models persist. With inference pipelines, you don't need to maintain outside infrastructure, and you can make predictions directly from data inputs.

When you run a Spark ML job on AWS Glue, a Spark ML pipeline is serialized into [MLeap](#) format. Then, you can use the job with the [SparkML Model Serving Container](#) in an Amazon SageMaker Inference Pipeline. *MLeap* is a serialization format and execution engine for machine learning pipelines. It supports Spark, Scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

Feature Processing with Sci-kit Learn

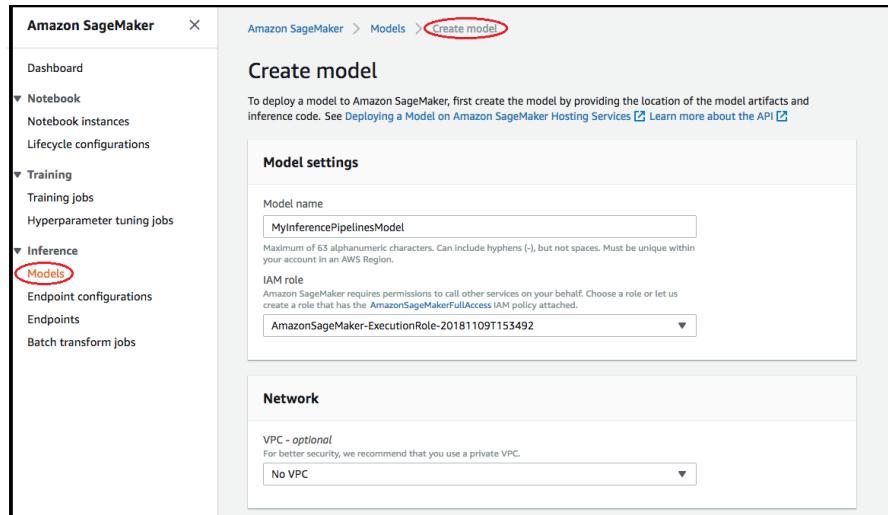
You can run and package scikit-learn jobs into containers directly in Amazon SageMaker. For an example of Python code for building a scikit-learn featurizer model that trains on [Fisher's Iris flower data set](#) and predicts the species of Iris based on morphological measurements, see [IRIS Training and Prediction with Sagemaker Scikit-learn](#).

Create a Pipeline Model

To create a pipeline model that can be deployed to an endpoint or used for a batch transform job, use the Amazon SageMaker console or the `CreateModel` operation.

To create an inference pipeline (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. Choose **Models**, and then choose **Create models** from the **Inference** group.
3. On the **Create model** page, provide a model name, choose an IAM role, and, if you want to use a private VPC, specify VPC values.



4. To add information about the containers in the inference pipeline, choose **Add container**, then choose **Next**.
5. Complete the fields for each container in the order that you want to execute them, up to the maximum of five. Complete the **Container input options**, **Location of inference code image**, and, optionally, **Location of model artifacts**, **Container host name**, and **Environmental variables** fields.

Container definition 1

Container input options

Provide model artifacts and inference image.

Provide model artifacts and inference image

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

Location of model artifacts - *optional*
The URL for the S3 location where model artifacts are stored.

The path must point to a single gzip compressed tar archive (.tar.gz suffix).

Container host name - *optional*
The DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Environment variables - *optional*

Key	Value	Remove
key1	value1	<input type="button" value="Remove"/>
key2	value2	<input type="button" value="Remove"/>

[Add environment variable](#)

[Remove](#)

Container definition 2 - *optional*

Container input options

Provide model artifacts and inference image.

Provide model artifacts and inference image

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

Location of model artifacts - *optional*
The URL for the S3 location where model artifacts are stored.

The path must point to a single gzip compressed tar archive (.tar.gz suffix).

Container host name - *optional*
The DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Environment variables - *optional*

Key	Value	Remove
		<input type="button" value="Remove"/>

[Add environment variable](#)

[Remove container](#)

[Remove](#)

Container definition 3 - *optional*

Container input options

Provide model artifacts and inference image.

Provide model artifacts and inference image

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

The **MyInferencePipelineModel** page summarizes the settings for the containers that provide input for the model. If you provided the environment variables in a corresponding container definition, Amazon SageMaker shows them in the **Environment variables** field.

The screenshot shows the Amazon SageMaker interface for managing inference pipelines. On the left, there's a sidebar with options like Dashboard, Notebook, Training, Inference, and Models. The main area is titled 'MyInferencePipelinesModel' and contains sections for Model settings, Container 1, Container 2, Container 3, Container 4, Container 5, Network, and Tags. Each container section includes fields for Container Name, Image, Model data URL, and Scanning status. Under Container 1, there is a table showing environment variables with keys 'key1' and 'key2' and values 'value1' and 'value2'. The Network section indicates 'No custom VPC settings applied.' The Tags section has an 'Edit' button.

Run Real-time Predictions with an Inference Pipeline

You can use trained models in an inference pipeline to make real-time predictions directly without performing external preprocessing. When you configure the pipeline, you can choose to use the built-in feature transformers already available in Amazon SageMaker. Or, you can implement your own transformation logic using just a few lines of scikit-learn or Spark code.

[MLeap](#), a serialization format and execution engine for machine learning pipelines, supports Spark, scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

The containers in a pipeline listen on the port specified in the `SAGEMAKER_BIND_TO_PORT` environment variable (instead of 8080). When running in an inference pipeline, Amazon SageMaker automatically provides this environment variable to containers. If this environment variable isn't present, containers

default to using port 8080. To indicate that your container complies with this requirement, use the following command to add a label to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

If your container needs to listen on a second port, choose a port in the range specified by the `SAGEMAKER_SAFE_PORT_RANGE` environment variable. Specify the value as an inclusive range in the format "`XXXX-YYYY`", where `XXXX` and `YYYY` are multi-digit integers. Amazon SageMaker provides this value automatically when you run the container in a multicontainer pipeline.

Note

To use custom Docker images in a pipeline that includes [Amazon SageMaker built-in algorithms](#), you need an [Amazon Elastic Container Registry \(Amazon ECR\) policy](#). Your Amazon ECR repository must grant Amazon SageMaker permission to pull the image. For more information, see [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 673\)](#).

Create and Deploy an Inference Pipeline Endpoint

The following code creates and deploys a real-time inference pipeline model with SparkML and XGBoost models in series using the Amazon SageMaker SDK.

```
from sagemaker.model import Model
from sagemaker.pipeline_model import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = 's3://{{}}/{{}}/{{}}'.format(s3_model_bucket, s3_model_key_prefix, 'model.tar.gz')
sparkml_model = SparkMLModel(model_data=sparkml_data)
xgb_model = Model(model_data=xgb_model.model_data, image=training_image)

model_name = 'serial-inference-' + timestamp_prefix
endpoint_name = 'serial-inference-ep-' + timestamp_prefix
sm_model = PipelineModel(name=model_name, role=role, models=[sparkml_model, xgb_model])
sm_model.deploy(initial_instance_count=1, instance_type='ml.c4.xlarge',
    endpoint_name=endpoint_name)
```

Request Real-Time Inference from an Inference Pipeline Endpoint

The following example shows how to make real-time predictions by calling an inference endpoint and passing a request payload in JSON format:

```
from sagemaker.predictor import json_serializer, json_deserializer, RealTimePredictor
from sagemaker.content_types import CONTENT_TYPE_CSV, CONTENT_TYPE_JSON

payload = {
    "input": [
        {
            "name": "Pclass",
            "type": "float",
            "val": "1.0"
        },
        {
            "name": "Embarked",
            "type": "string",
            "val": "Q"
        },
        {
            "name": "Age",
            "type": "double",
            "val": "30.0"
        }
    ]
}
```

```

        "val": "48.0"
    },
    {
        "name": "Fare",
        "type": "double",
        "val": "100.67"
    },
    {
        "name": "SibSp",
        "type": "double",
        "val": "1.0"
    },
    {
        "name": "Sex",
        "type": "string",
        "val": "male"
    }
],
"output": {
    "name": "features",
    "type": "double",
    "struct": "vector"
}
}

predictor = RealTimePredictor(endpoint=endpoint_name, sagemaker_session=sess,
    serializer=json_serializer,
                           content_type=CONTENT_TYPE_JSON, accept=CONTENT_TYPE_CSV)

print(predictor.predict(payload))

```

Run Batch Transforms with Inference Pipelines

To get inferences on an entire dataset you run a batch transform on a trained model. To run inferences on a full dataset, you can use the same inference pipeline model created and deployed to an endpoint for real-time processing in a batch transform job. To run a batch transform job in a pipeline, you download the input data from Amazon S3 and send it in one or more HTTP requests to the inference pipeline model. For an example that shows how to prepare data for a batch transform, see the ["Preparing Data for Batch Transform" section of the ML Pipeline with SparkML and XGBoost - Training and Inference sample notebook](#). For information about Amazon SageMaker batch transforms, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#).

Note

To use custom Docker images in a pipeline that includes [Amazon SageMaker built-in algorithms](#), you need an [Amazon Elastic Container Registry \(Amazon ECR\) policy](#). Your Amazon ECR repository must grant Amazon SageMaker permission to pull the image. For more information, see [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 673\)](#).

The following example shows how to run a transform job using the Amazon SageMaker Python SDK. In this example, `model_name` is the inference pipeline that combines SparkML and XGBoost models (created in previous examples). The Amazon S3 location specified by `input_data_path` contains the input data, in CSV format, to be downloaded and sent to the Spark ML model. After the transform job has finished, the Amazon S3 location specified by `output_data_path` contains the output data returned by the XGBoost model in CVS format.

```

input_data_path = 's3://{}//{}//{}'.format(default_bucket, 'key', 'file_name')
output_data_path = 's3://{}//{}'.format(default_bucket, 'key')
transform_job = sagemaker.transformer.Transformer(
    model_name = model_name,
    instance_count = 1,
    instance_type = 'ml.m4.xlarge',

```

```

strategy = 'SingleRecord',
assemble_with = 'Line',
output_path = output_data_path,
base_transform_job_name='inference-pipelines-batch',
sagemaker_session=sess,
accept = CONTENT_TYPE_CSV)
transform_job.transform(data = input_data_path,
                      content_type = CONTENT_TYPE_CSV,
                      split_type = 'Line')

```

Inference Pipeline Logs and Metrics

Monitoring is important for maintaining the reliability, availability, and performance of Amazon SageMaker resources. To monitor and troubleshoot inference pipeline performance, use Amazon CloudWatch logs and error messages. For information about the monitoring tools that Amazon SageMaker provides, see [Monitor Amazon SageMaker \(p. 731\)](#).

Use Metrics to Monitor Multi-container Models

To monitor the multi-container models in Inference Pipelines, use Amazon CloudWatch. CloudWatch collects raw data and processes it into readable, near real-time metrics. Amazon SageMaker training jobs and endpoints write CloudWatch metrics and logs in the `AWS/SageMaker` namespace.

The following tables list the metrics and dimensions for the following:

- Endpoint invocations
- Training jobs, batch transform jobs, and endpoint instances

A *dimension* is a name/value pair that uniquely identifies a metric. You can assign up to 10 dimensions to a metric. For more information on monitoring with CloudWatch, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#).

Endpoint Invocation Metrics

The `AWS/SageMaker` namespace includes the following request metrics from calls to `InvokeEndpoint`.

Metrics are reported at a 1-minute intervals.

Metric	Description
<code>Invocation4XXErrors</code>	<p>The number of <code>InvokeEndpoint</code> requests that the model returned a <code>4xx</code> HTTP response code for. For each <code>4xx</code> response, Amazon SageMaker sends a 1.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum</p>
<code>Invocation5XXErrors</code>	<p>The number of <code>InvokeEndpoint</code> requests that the model returned a <code>5xx</code> HTTP response code for. For each <code>5xx</code> response, Amazon SageMaker sends a 1.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum</p>

Metric	Description
Invocations	<p>The number of <code>InvokeEndpoint</code> requests sent to a model endpoint.</p> <p>To get the total number of requests sent to a model endpoint, use the <code>Sum</code> statistic.</p> <p>Units: None</p> <p>Valid statistics: <code>Sum</code>, <code>Sample Count</code></p>
InvocationsPerInstance	<p>The number of endpoint invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code>. Amazon SageMaker sends $1/\text{numberOfInstances}$ as the value for each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> at the endpoint at the time of the request.</p> <p>Units: None</p> <p>Valid statistics: <code>Sum</code></p>
ModelLatency	<p>The time the model or models took to respond. This includes the time it took to send the request, to fetch the response from the model container, and to complete the inference in the container. <code>ModelLatency</code> is the total time taken by all containers in an inference pipeline.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>
OverheadLatency	<p>The time added to the time taken to respond to a client request by Amazon SageMaker for overhead. <code>OverheadLatency</code> is measured from the time that Amazon SageMaker receives the request until it returns a response to the client, minus the <code>ModelLatency</code>. Overhead latency can vary depending on request and response payload sizes, request frequency, and authentication or authorization of the request, among other factors.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>
ContainerLatency	<p>The time it took for an Inference Pipelines container to respond as viewed from Amazon SageMaker. <code>ContainerLatency</code> includes the time it took to send the request, to fetch the response from the model's container, and to complete inference in the container.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>

Dimensions for Endpoint Invocation Metrics

Dimension	Description
<code>EndpointName</code> , <code>VariantName</code> , <code>ContainerName</code>	Filters endpoint invocation metrics for a <code>ProductionVariant</code> at the specified endpoint and for the specified variant.

For an inference pipeline endpoint, CloudWatch lists per-container latency metrics in your account as **Endpoint Container Metrics** and **Endpoint Variant Metrics** in the **SageMaker** namespace, as follows. The **ContainerLatency** metric appears only for inferences pipelines.

The screenshot shows the AWS CloudWatch Metrics console. At the top, there are tabs: All metrics (selected), Graphed metrics, Graph options, and Source. Below the tabs, the URL is shown as All > SageMaker. A search bar is present with the placeholder "Search for any metric, dimension or resource id". The main area displays "86 Metrics". Under "Endpoint Container Metrics", there are 20 Metrics. Under "Endpoint Variant Metrics", there are 66 Metrics. The interface is clean with a light gray background and white cards for each metric category.

For each endpoint and each container, latency metrics display names for the container, endpoint, variant, and metric.

The screenshot shows the AWS CloudWatch Metrics console with the path All > SageMaker > Endpoint Container Metrics > MyInferencePipelinesEndpoint. The table has four columns: ContainerName (5), EndpointName, VariantName, and Metric Name. The data shows five rows of container names mapped to the same endpoint name, with variant names like MyInferencePipelinesVariant and ContainerLatency.

ContainerName (5)	EndpointName	VariantName	Metric Name
MyContainerName1	MyInferencePipelinesEndpoint	MyInferencePipelinesVariant	ContainerLatency
MyContainerName2	MyInferencePipelinesEndpoint	MyInferencePipelinesVariant	ContainerLatency
MyContainerName3	MyInferencePipelinesEndpoint	MyInferencePipelinesVariant	ContainerLatency
MyContainerName4	MyInferencePipelinesEndpoint	MyInferencePipelinesVariant	ContainerLatency
MyContainerName5	MyInferencePipelinesEndpoint	MyInferencePipelinesVariant	ContainerLatency

Training Job, Batch Transform Job, and Endpoint Instance Metrics

The namespaces `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs`, and `/aws/sagemaker/Endpoints` include the following metrics for training jobs and endpoint instances.

Metrics are reported at a 1-minute intervals.

Metric	Description
CPUUtilization	<p>The percentage of CPU units that are used by the containers running on an instance. The value ranges from 0% to 100%, and is multiplied by the number of CPUs. For example, if there are four CPUs, CPUUtilization can range from 0% to 400%.</p> <p>For training jobs, CPUUtilization is the CPU utilization of the algorithm container running on the instance.</p> <p>For batch transform jobs, CPUUtilization is the CPU utilization of the transform container running on the instance.</p> <p>For multi-container models, CPUUtilization is the sum of CPU utilization by all containers running on the instance.</p> <p>For endpoint variants, CPUUtilization is the sum of CPU utilization by all of the containers running on the instance.</p> <p>Units: Percent</p>
MemoryUtilization	<p>The percentage of memory that is used by the containers running on an instance. This value ranges from 0% to 100%.</p> <p>For training jobs, MemoryUtilization is the memory used by the algorithm container running on the instance.</p> <p>For batch transform jobs, MemoryUtilization is the memory used by the transform container running on the instance.</p>

Metric	Description
	<p>For multi-container models, <code>MemoryUtilization</code> is the sum of memory used by all containers running on the instance.</p> <p>For endpoint variants, <code>MemoryUtilization</code> is the sum of memory used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>GPUUtilization</code>	<p>The percentage of GPU units that are used by the containers running on an instance. <code>GPUUtilization</code> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <code>GPUUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, <code>GPUUtilization</code> is the GPU used by the algorithm container running on the instance.</p> <p>For batch transform jobs, <code>GPUUtilization</code> is the GPU used by the transform container running on the instance.</p> <p>For multi-container models, <code>GPUUtilization</code> is the sum of GPU used by all containers running on the instance.</p> <p>For endpoint variants, <code>GPUUtilization</code> is the sum of GPU used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>GPUMemoryUtilization</code>	<p>The percentage of GPU memory used by the containers running on an instance. <code>GPUMemoryUtilization</code> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <code>GPUMemoryUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, <code>GPUMemoryUtilization</code> is the GPU memory used by the algorithm container running on the instance.</p> <p>For batch transform jobs, <code>GPUMemoryUtilization</code> is the GPU memory used by the transform container running on the instance.</p> <p>For multi-container models, <code>GPUMemoryUtilization</code> is sum of GPU used by all containers running on the instance.</p> <p>For endpoint variants, <code>GPUMemoryUtilization</code> is the sum of the GPU memory used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>DiskUtilization</code>	<p>The percentage of disk space used by the containers running on an instance. <code>DiskUtilization</code> ranges from 0% to 100%. This metric is not supported for batch transform jobs.</p> <p>For training jobs, <code>DiskUtilization</code> is the disk space used by the algorithm container running on the instance.</p> <p>For endpoint variants, <code>DiskUtilization</code> is the sum of the disk space used by all of the provided containers running on the instance.</p> <p>Units: Percent</p>

Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

Dimension	Description
Host	<p>For training jobs, Host has the format [training-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the /aws/sagemaker/TrainingJobs namespace.</p> <p>For batch transform jobs, Host has the format [transform-job-name]/[instance-id]. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the /aws/sagemaker/TransformJobs namespace.</p> <p>For endpoints, Host has the format [endpoint-name]/[production-variant-name]/[instance-id]. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the /aws/sagemaker/Endpoints namespace.</p>

To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, Amazon SageMaker also sends anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` to Amazon CloudWatch Logs. You can use this information for debugging and to analyze progress.

Use Logs to Monitor an Inference Pipeline

The following table lists the log groups and log streams Amazon SageMaker sends to Amazon CloudWatch

A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

Logs

Log Group Name	Log Stream Name
/aws/sagemaker/TrainingJobs	[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]
/aws/sagemaker/Endpoints/[EndpointName]	[production-variant-name]/[instance-id] [production-variant-name]/[instance-id] [production-variant-name]/[instance-id]/[container-name provided in the Amazon SageMaker model] (For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.
/aws/sagemaker/NotebookInstances	[notebook-instance-name]/[LifecycleConfigHook]
/aws/sagemaker/TransformJobs	[transform-job-name]/[instance-id]-[epoch_timestamp] [transform-job-name]/[instance-id]-[epoch_timestamp]/data-log

Log Group Name	Log Stream Name
	[transform-job-name]/[instance-id]-[epoch_timestamp]/[container-name provided in the Amazon SageMaker model] (For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.

Note

Amazon SageMaker creates the `/aws/sagemaker/NotebookInstances` log group when you create a notebook instance with a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).

For more information about Amazon SageMaker logging, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

Troubleshoot Inference Pipelines

To troubleshoot inference pipeline issues, use CloudWatch logs and error messages. If you are using custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you might also encounter permissions problems. To grant the required permissions, create an Amazon Elastic Container Registry (Amazon ECR) policy.

Topics

- [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 673\)](#)
- [Use CloudWatch Logs to Troubleshoot Amazon SageMaker Inference Pipelines \(p. 674\)](#)
- [Use Error Messages to Troubleshoot Inference Pipelines \(p. 674\)](#)

Troubleshoot Amazon ECR Permissions for Inference Pipelines

When you use custom Docker images in a pipeline that includes [Amazon SageMaker built-in algorithms](#), you need an [Amazon ECR policy](#). The policy allows your Amazon ECR repository to grant permission for Amazon SageMaker to pull the image. The policy must add the following permissions:

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "allowSageMakerToPull",
            "Effect": "Allow",
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        }
    ]
}
```

Use CloudWatch Logs to Troubleshoot Amazon SageMaker Inference Pipelines

Amazon SageMaker publishes the container logs for endpoints that deploy an inference pipeline to Amazon CloudWatch at the following path for each container.

```
/aws/sagemaker/Endpoints/{EndpointName}/{Variant}/{InstanceId}/{ContainerHostname}
```

For example, logs for this endpoint are published to the following log groups and streams:

```
EndpointName: MyInferencePipelinesEndpoint
Variant: MyInferencePipelinesVariant
InstanceId: i-0179208609ff7e488
ContainerHostname: MyContainerName1 and MyContainerName2
```

```
logGroup: /aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName1
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName2
```

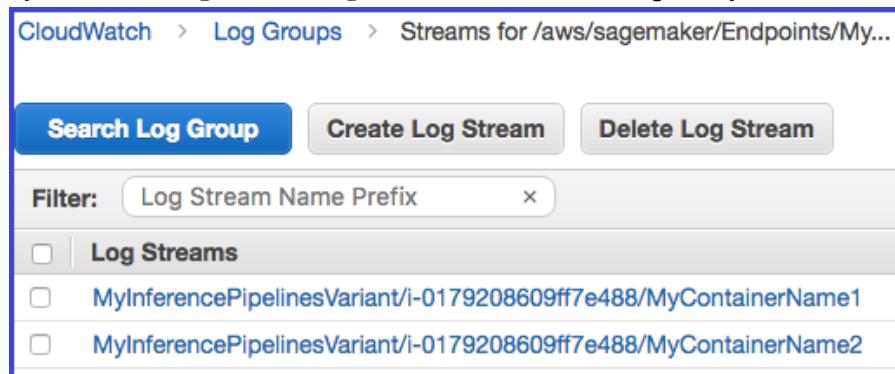
A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

To see the log groups and streams

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation page, choose **Logs**.
3. In **Log Groups**, filter on **MyInferencePipelinesEndpoint**:



4. To see the log streams, on the CloudWatch **Log Groups** page, choose **MyInferencePipelinesEndpoint**, and then **Search Log Group**.



For a list of the logs that Amazon SageMaker publishes, see [Inference Pipeline Logs and Metrics \(p. 668\)](#).

Use Error Messages to Troubleshoot Inference Pipelines

The inference pipeline error messages indicate which containers failed.

If an error occurs while Amazon SageMaker is invoking an endpoint, the service returns a `ModelError` (error code 424), which indicates which container failed. If the request payload (the response from the previous container) exceeds the limit of 5 MB, Amazon SageMaker provides a detailed error message, such as:

Received response from MyContainerName1 with status code 200. However, the request payload from MyContainerName1 to MyContainerName2 is 6000000 bytes, which has exceeded the maximum limit of 5 MB. See <https://us-west-2.console.aws.amazon.com/cloudwatch/home?region=us-west-2#logEventViewer:group=/aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint> in account 123456789012 for more information.

If a container fails the ping health check while Amazon SageMaker is creating an endpoint, it returns a `ClientError` and indicates all of the containers that failed the ping check in the last health check.

Use Batch Transform

Use batch transform when you need to do the following:

- Preprocess datasets to remove noise or bias that interferes with training or inference from your dataset.
- Get inferences from large datasets.
- Run inference when you don't need a persistent endpoint.
- Associate input records with inferences to assist the interpretation of results.

To filter input data before performing inferences or to associate input records with inferences about those records, see [Associate Prediction Results with Input Records \(p. 678\)](#). For example, you can filter input data to provide context for creating and interpreting reports about the output data.

For more information about batch transforms, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 11\)](#).

Topics

- [Use Batch Transform to Get Inferences from Large Datasets \(p. 675\)](#)
- [Speed up a Batch Transform Job \(p. 677\)](#)
- [Use Batch Transform to Test Production Variants \(p. 677\)](#)
- [Batch Transform Errors \(p. 677\)](#)
- [Batch Transform Sample Notebooks \(p. 677\)](#)
- [Associate Prediction Results with Input Records \(p. 678\)](#)

Use Batch Transform to Get Inferences from Large Datasets

Batch transform automatically manages the processing of large datasets within the limits of specified parameters. For example, suppose that you have a dataset file, `input1.csv`, stored in an S3 bucket. The content of the input file might look like the following.:

```
Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-AttributeM
```

```
Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-AttributeM  
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM  
...  
RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-AttributeM
```

When a batch transform job starts, Amazon SageMaker initializes compute instances and distributes the inference or preprocessing workload between them. When you have multiple files, one instance might process `input1.csv`, and another instance might process the file named `input2.csv`.

To keep large payloads below the `MaxPayloadInMB` limit, you can split an input file into several mini-batches. For example, you might create a mini-batch from `input1.csv` by including only two of the records.

```
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM  
Record4-Attribute1, Record4-Attribute2, Record4-Attribute3, ..., Record4-AttributeM
```

Note

Amazon SageMaker processes each input file separately. It doesn't combine mini-batches from different input files to comply with the `MaxPayloadInMB` limit.

To split input files into mini-batches, when you create a batch transform job, set the `SplitType` parameter value to `Line`. If `SplitType` is set to `None` or if an input file can't be split into mini-batches, Amazon SageMaker uses the entire input file in a single request.

If the batch transform job successfully processes all of the records in an input file, it creates an output file with the same name and the `.out` file extension. For multiple input files, such as `input1.csv` and `input2.csv`, the output files are named `input1.csv.out` and `input2.csv.out`. The batch transform job stores the output files in the specified location in Amazon S3, such as `s3://awsexamplebucket/output/`.

The predictions in an output file are listed in the same order as the corresponding records in the input file. The output file `input1.csv.out`, based on the input file shown earlier, would look like the following.

```
Inference1-Attribute1, Inference1-Attribute2, Inference1-Attribute3, ..., Inference1-  
AttributeM  
Inference2-Attribute1, Inference2-Attribute2, Inference2-Attribute3, ..., Inference2-  
AttributeM  
Inference3-Attribute1, Inference3-Attribute2, Inference3-Attribute3, ..., Inference3-  
AttributeM  
...  
InferenceN-Attribute1, Inference3-Attribute2, Inference3-Attribute3, ..., InferenceN-  
AttributeM
```

To combine the results of multiple output files into a single output file, set the `AssembleWith` parameter to `Line`.

When the input data is very large and is transmitted using HTTP chunked encoding, to stream the data to the algorithm, set `MaxPayloadInMB` to 0. Amazon SageMaker built-in algorithms don't support this feature.

For information about using the API to create a batch transform job, see the [CreateTransformJob](#) API. For more information about the correlation between batch transform input and output objects, see [OutputDataConfig](#). For an example of how to use batch transform, see [Step 6.2: Deploy the Model with Batch Transform \(p. 52\)](#).

Speed up a Batch Transform Job

If you are using the [CreateTransformJob](#) API, you can reduce the time it takes to complete batch transform jobs by using optimal values for parameters such as [MaxPayloadInMB](#), [MaxConcurrentTransforms](#), or [BatchStrategy](#). If you are using the Amazon SageMaker console, you can specify these optimal parameter values in the **Additional configuration** section of the **Batch transform job configuration** page. Amazon SageMaker automatically finds the optimal parameter settings for built-in algorithms. For custom algorithms, provide these values through an [execution-parameters](#) endpoint.

Use Batch Transform to Test Production Variants

To test different models or various hyperparameter settings, create a separate transform job for each new model variant and use a validation dataset. For each transform job, specify a unique model name and location in Amazon S3 for the output file. To analyze the results, use [Inference Pipeline Logs and Metrics](#) (p. 668).

Batch Transform Errors

Amazon SageMaker uses the Amazon S3 [Multipart Upload API](#) to upload results from a batch transform job to Amazon S3. If an error occurs, the uploaded results are removed from Amazon S3. In some cases, such as when a network outage occurs, an incomplete multipart upload might remain in Amazon S3. To avoid incurring storage charges, we recommend that you add the [S3 bucket policy](#) to the S3 bucket lifecycle rules. This policy deletes incomplete multipart uploads that might be stored in the S3 bucket. For more information, see [Object Lifecycle Management](#).

If a batch transform job fails to process an input file because of a problem with the dataset, Amazon SageMaker marks the job as `Failed`. If an input file contains a bad record, the transform job doesn't create an output file for that input file because doing so prevents it from maintaining the same order in the transformed data as in the input file. When your dataset has multiple input files, a transform job continues to process input files even if it fails to process one. The processed files still generate useable results.

Exceeding the [MaxPayloadInMB](#) limit causes an error. This might happen with a large dataset if it can't be split, the [SplitType](#) parameter is set to none, or individual records within the dataset exceed the limit.

If you are using your own algorithms, you can use placeholder text, such as `ERROR`, when the algorithm finds a bad record in an input file. For example, if the last record in a dataset is bad, the algorithm places the placeholder text for that record in the output file.

Batch Transform Sample Notebooks

For a sample notebook that uses batch transform with a principal component analysis (PCA) model to reduce data in a user-item review matrix, followed by the application of a density-based spatial clustering of applications with noise (DBSCAN) algorithm to cluster movies, see [Batch Transform with PCA and DBSCAN Movie Clusters](#). For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances](#) (p. 252). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all the Amazon SageMaker examples. The topic modeling example notebooks that use the NTM algorithms are located in the **Advanced functionality** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Associate Prediction Results with Input Records

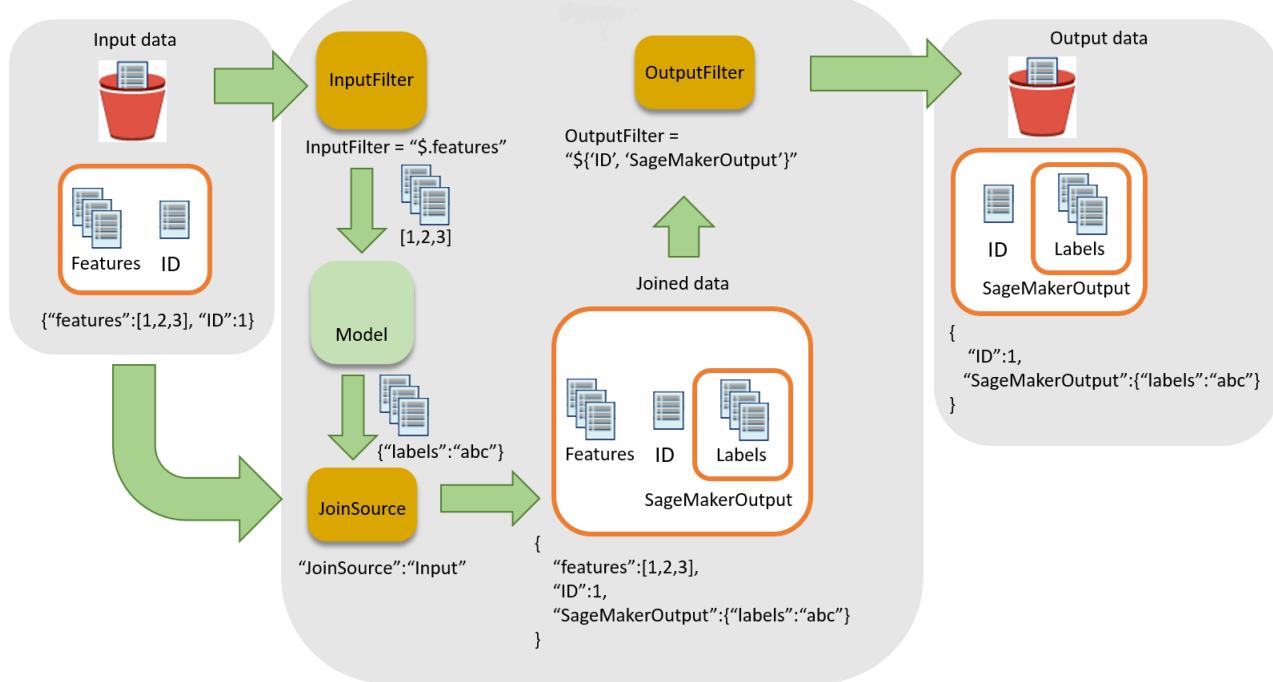
When making predictions on a large dataset, you can exclude attributes that aren't needed for prediction. After the predictions have been made, you can associate some of the excluded attributes with those predictions or with other input data in your report. By using batch transform to perform these data processing steps, you can often eliminate additional preprocessing or postprocessing. You can use input files in JSON and CSV format only.

Topics

- [Workflow for Associating Inferences with Input Records \(p. 678\)](#)
- [Use Data Processing in Batch Transform Jobs \(p. 679\)](#)
- [Supported JSONPath Operators \(p. 679\)](#)
- [Batch Transform Examples \(p. 680\)](#)

Workflow for Associating Inferences with Input Records

The following diagram shows the workflow for associating inferences with input records.



To associate inferences with input data, there are three main steps:

1. Filter the input data that is not needed for inference before passing the input data to the batch transform job. Use the `InputFilter` parameter to determine which attributes to use as input for the model.
2. Associate the input data with the inference results. Use the `JoinSource` parameter to combine the input data with the inference.
3. Filter the joined data to retain the inputs that are needed to provide context for interpreting the predictions in the reports. Use `OutputFilter` to store the specified portion of the joined dataset in the output file.

Use Data Processing in Batch Transform Jobs

When creating a batch transform job with [CreateTransformJob](#) to process data:

1. Specify the portion of the input to pass to the model with the `InputFilter` parameter in the `DataProcessing` data structure.
2. Join the raw input data with the transformed data with the `JoinSource` parameter.
3. Specify which portion of the joined input and transformed data from the batch transform job to include in the output file with the `OutputFilter` parameter.
4. Choose either JSON- or CSV-formatted files for input:
 - For JSON- or JSON Lines-formatted input files, Amazon SageMaker either adds the `SageMakerOutput` attribute to the input file or creates a new JSON output file with the `SageMakerInput` and `SageMakerOutput` attributes. For more information, see [DataProcessing](#).
 - For CSV-formatted input files, the joined input data is followed by the transformed data and the output is a CSV file.

If you use an algorithm with the `DataProcessing` structure, it must support your chosen format for *both* input and output files. For example, with the `TransformOutput` field of the `CreateTransformJob` API, you must set both the `Content Type` and `Accept` parameters to one of the following values: `text/csv`, `application/json`, or `application/jsonlines`. The syntax for specifying columns in a CSV file and specifying attributes in a JSON file are different. Using the wrong syntax causes an error. For more information, see [Batch Transform Examples \(p. 680\)](#). For more information about input and output file formats for built-in algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 272\)](#).

The record delimiters for the input and output must also be consistent with your chosen file input. The `SplitType` parameter indicates how to split the records in the input dataset. The `AssembleWith` parameter indicates how to reassemble the records for the output. If you set input and output formats to `text/csv`, you must also set the `SplitType` and `AssemblyType` parameters to `line`. If you set the input and output formats to `application/jsonlines`, you can set both `SplitType` and `AssemblyType` to `line`.

For JSON files, the attribute name `SageMakerOutput` is reserved for output. The JSON input file can't have an attribute with this name. If it does, the data in the input file might be overwritten.

Supported JSONPath Operators

To filter and join the input data and inference, use a JSONPath subexpression. Amazon SageMaker supports only a subset of the defined JSONPath operators. The following table lists the supported JSONPath operators. For CSV data, each row is taken as a JSON array, so only index based JSONPaths can be applied, e.g. `[$[0], $[1 :]]`. CSV data should also follow [RFC format](#).

JSONPath Operator	Description	Example
<code>\$</code>	The root element to a query. This operator is required at the beginning of all path expressions.	<code>\$</code>
<code>. <name></code>	A dot-notated child element.	<code>\$.id</code>
<code>*</code>	A wildcard. Use in place of an attribute name or numeric value.	<code>\$.id.*</code>

JSONPath Operator	Description	Example
[' <i><name></i> ' (, ' <i><name></i> ')]	A bracket-notated element or multiple child elements.	\$['id', 'SageMakerOutput']
[<i><number></i> (, <i><number></i>)]	An index or array of indexes. Negative index values are also supported. A -1 index refers to the last element in an array.	\$[1] , \$[1,3,5]
[<i><start></i> : <i><end></i>]	An array slice operator. The array slice() method extracts a section of an array and returns a new array. If you omit <i><start></i> , Amazon SageMaker uses the first element of the array. If you omit <i><end></i> , Amazon SageMaker uses the last element of the array.	\$[2:5], \$[:5], \$[2:]

When using the bracket-notation to specify multiple child elements of a given field, additional nesting of children within brackets is not supported. For example, `$.field1.['child1', 'child2']` is supported while `$.field1.['child1', 'child2.grandchild']` is not.

For more information about JSONPath operators, see [JsonPath](#) on GitHub.

Batch Transform Examples

The following examples show some common ways to join input data with prediction results.

Topics

- [Example: Output Only Inferences \(p. 680\)](#)
- [Example: Output Input Data and Inferences \(p. 681\)](#)
- [Example: Output an ID Column with Results and Exclude the ID Column from the Input \(CSV\) \(p. 681\)](#)
- [Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input \(JSON\) \(p. 682\)](#)

Example: Output Only Inferences

By default, the `DataProcessing` parameter doesn't join inference results with input. It outputs only the inference results.

If you want to explicitly specify to not join results with input, use the Amazon SageMaker Python SDK and specify the following settings in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(...)
sm_transformer.transform(..., input_filter="$", join_source= "None", output_filter="$")
```

To output inferences using the AWS SDK for Python, add the following code to your `CreateTransformJob` request. The following code mimics the default behavior.

```
{
    "DataProcessing": {
        "InputFilter": "$",
        "JoinSource": "None",
        "OutputFilter": "$"
    }
}
```

```
}
```

Example: Output Input Data and Inferences

If you're using the Amazon SageMaker Python SDK, to combine the input data with the inferences in the output file, specify "Input" for the `JoinSource` parameter in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(...)  
sm_transformer.transform(..., join_source= "Input")
```

If you're using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your `CreateTransformJob` request.

```
{  
    "DataProcessing":  
    {  
        "JoinSource": "Input"  
    }  
}
```

For JSON or JSON Lines input files, the results are in the `SageMakerOutput` key in the input JSON file. For example, if the input is a JSON file that contains the key-value pair `{"key":1}`, the data transform result might be `{"label":1}`.

Amazon SageMaker stores both in the input file in the `SageMakerInput` key.

```
{  
    "key":1,  
    "SageMakerOutput":{"label":1}  
}
```

Note

The joined result for JSON must be a key-value pair object. If the input isn't a key-value pair object, Amazon SageMaker creates a new JSON file. In the new JSON file, the input data is stored in the `SageMakerInput` key and the results are stored as the `SageMakerOutput` value.

For a CSV file, for example, if the record is `[1, 2, 3]`, and the label result is `[1]`, then the output file would contain `[1, 2, 3, 1]`.

Example: Output an ID Column with Results and Exclude the ID Column from the Input (CSV)

If you are using the Amazon SageMaker Python SDK, to include results or an ID column in the output, specify indexes of the joined dataset in a transformer call. For example, if your data includes five columns and the first one is the ID column, use the following transformer request.

```
sm_transformer = sagemaker.transformer.Transformer(...)  
sm_transformer.transform(..., input_filter ="$[1:]", join_source= "Input", output_filter="$")
```

If you are using the AWS SDK for Python (Boto 3), add the following code to your `CreateTransformJob` request.

```
{  
    "DataProcessing": {  
        "InputFilter": "$[1:]",  
    }  
}
```

```

        "JoinSource": "Input",
        "OutputFilter": "$"
    }
}

```

To specify columns in Amazon SageMaker, use the index of the array elements. The first column is index 0, the second column is index 1, and the sixth column is index 5.

To exclude the first column from the input, set `InputFilter` to `"$[1:]"`. The colon (`:`) tells Amazon SageMaker to include all of the elements between two values, inclusive. For example, `$[1:4]` specifies the second through fifth columns.

If you omit the number after the colon, for example, `[5:]`, the subset includes all columns from the 6th column through the last column. If you omit the number before the colon, for example, `[:5]`, the subset includes all columns from the first column (index 0) through the sixth column.

Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input (JSON)

If you are using the Amazon SageMaker Python SDK, include results of an ID attribute in the output by specifying it in a transformer call. For example, if you store data in the `features` attribute and the record ID in the `ID` attribute, you would use the following transformer request.

```

sm_transformer = sagemaker.transformer.Transformer(...)
sm_transformer.transform(..., input_filter=".features", join_source= "Input",
                      output_filter="['id','SageMakerOutput']")

```

If you are using the AWS SDK for Python (Boto 3), join all input data with the inference by adding the following code to your `CreateTransformJob` request.

```

{
    "DataProcessing": {
        "InputFilter": ".features",
        "JoinSource": "Input",
        "OutputFilter": "['id','SageMakerOutput']"
    }
}

```

Warning

If you are using a JSON-formatted input file, the file can't contain the attribute name `SageMakerOutput`. This attribute name is reserved for the output file. If your JSON-formatted input file contains an attribute with this name, values in the input file might be overwritten with the inference.

Compile and Deploy Models with Amazon SageMaker Neo

Neo is a new capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

Ordinarily, optimizing machine learning models for inference on multiple platforms is extremely difficult because you need to hand-tune models for the specific hardware and software configuration of each platform. If you want to get optimal performance for a given workload, you need to know the hardware architecture, instruction set, memory access patterns, and input data shapes among other factors. For

traditional software development, tools such as compilers and profilers simplify the process. For machine learning, most tools are specific to the framework or to the hardware. This forces you into a manual trial-and-error process that is unreliable and unproductive.

Neo eliminates the time and effort required to do this by automatically optimizing TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models for deployment on ARM, Intel, and Nvidia processors. Neo consists of a compiler and a runtime. First, the Neo compilation API reads models exported from various frameworks. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimizations. Then it generates binary code for the optimized operations, writes them to a shared object library, and saves the model definition and parameters into separate files. Neo also provides a runtime for each target platform that loads and executes the compiled model.

You can create a Neo compilation job from either the Amazon SageMaker console, AWS Command Line Interface (AWS CLI), Python notebook, or the Amazon SageMaker SDK. With a few CLI commands, an API invocation, or a few clicks, you can convert a model for your chosen platform. You can deploy the model to an Amazon SageMaker endpoint or on an AWS IoT Greengrass device quickly. Amazon SageMaker provides Neo container images for Amazon SageMaker XGBoost and Image Classification models, and supports Amazon SageMaker-compatible containers for your own compiled models.

Note

Neo currently supports image classification models exported as frozen graphs from TensorFlow, MXNet, or PyTorch, and XGBoost models. Neo is available in the following AWS [Regions](#) where Amazon SageMaker is supported:

- **Asia Pacific** (Hong Kong, Mumbai, Seoul, Singapore, Sydney, Tokyo)
- **Canada** (Central)
- **EU** (Frankfurt, Ireland, London, Paris, Stockholm)
- **North America** (N. Virginia, Ohio, Oregon, N. California)
- **South America** (Sao Paulo)

Topics

- [Amazon SageMaker Neo Sample Notebooks \(p. 683\)](#)
- [Use Neo to Compile a Model \(p. 684\)](#)
- [Deploy a Model \(p. 688\)](#)
- [Request Inferences from a Deployed Service \(p. 696\)](#)
- [Troubleshooting Neo Compilation Errors \(p. 696\)](#)

Amazon SageMaker Neo Sample Notebooks

For sample notebooks that uses Amazon SageMaker Neo to train, compile, optimize, and deploy machine learning models to make inferences, see:

- [MNIST Training, Compilation and Deployment with MXNet Module](#)
- [MNIST Training, Compilation and Deployment with Tensorflow Module](#)
- [Deploying pre-trained PyTorch vision models with Amazon SageMaker Neo](#)
- [Model Optimization with an Image Classification Example](#)
- [Model Optimization with XGBoost Example](#)

For instructions on how to run these example notebooks in Amazon SageMaker, see [Use Example Notebooks \(p. 260\)](#). If you need intructions on how to create a notebook instance to run these examples,

see Amazon SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 252\)](#). To navigate to the relevant example in your notebook instance, choose the **Amazon SageMaker Examples** tab to see a list of all of the Amazon SageMaker samples. To open a notebook, choose its **Use** tab, then choose **Create copy**.

Use Neo to Compile a Model

This section shows how to create, describe, stop, and list compilation jobs. There are three options available in Neo for managing the compilation jobs for machine learning models: Using the Neo CLI, the Amazon SageMaker console, or the Amazon SageMaker SDK.

Topics

- [Compile a Model \(API\) \(p. 684\)](#)
- [Compile a Model \(Console\) \(p. 685\)](#)
- [Compile a Model \(Amazon SageMaker SDK\) \(p. 688\)](#)

Compile a Model (API)

This section shows how to manage compilation jobs for machine learning models. You can create, describe, stop, and list compilation jobs.

Create a Compilation Job

As shown in the following JSON file, you specify the data input format, the S3 bucket where you stored your model, the S3 bucket where you want to write the compiled model, and the target hardware:

```
job.json
{
    "CompilationJobName": "job002",
    "RoleArn": "arn:aws:iam::<your-account>:role/service-role/AmazonSageMaker-
ExecutionRole-20180829T140091",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/train",
        "DataInputConfig": "{\"data\": [1,3,1024,1024]}",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/DEMO-breast-cancer-prediction/
compile",
        "TargetDevice": "ml_c5"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}

aws sagemaker create-compilation-job \
--cli-input-json file://job.json \
--region us-west-2

# You should get CompilationJobArn
```

Describe a Compilation Job

```
aws sagemaker describe-compilation-job \
--compilation-job-name $JOB_NM \
```

```
--region us-west-2
```

Stop a Compilation Job

```
aws sagemaker stop-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2

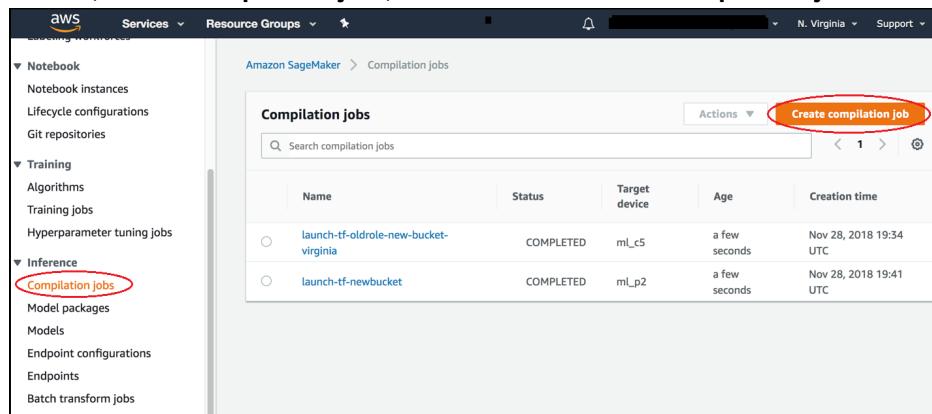
# There is no output for compilation-job operation
```

List a Compilation Job

```
aws sagemaker list-compilation-jobs \
--region us-west-2
```

Compile a Model (Console)

You can create a Neo compilation job in the Amazon SageMaker console. In the **Amazon SageMaker** console, choose **Compilation jobs**, and then choose **Create compilation job**.



On the **Create compilation job** page, for **Job name**, enter a name. Then select an **IAM role**.

Job settings
The settings define the job and the credentials for accessing Amazon S3, and set constraints on the cost of running the job.

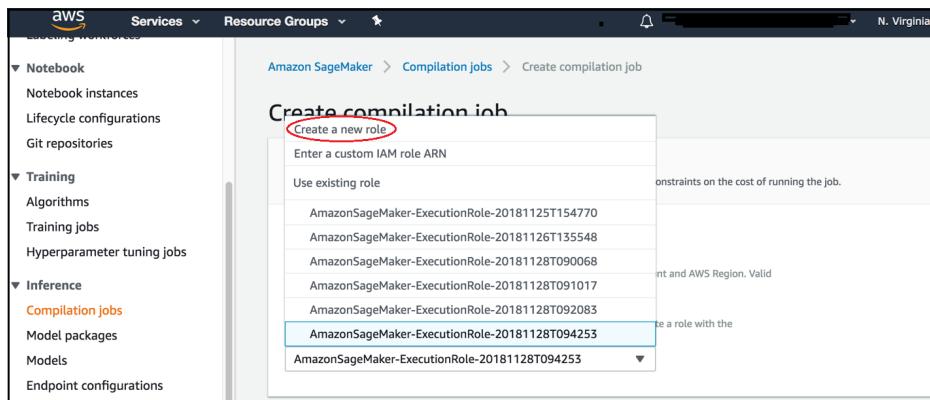
Job name

The name must be from 1 to 63 characters and must be unique in your AWS account and AWS Region. Valid characters are a-z, A-Z, 0-9, and hyphen (-)

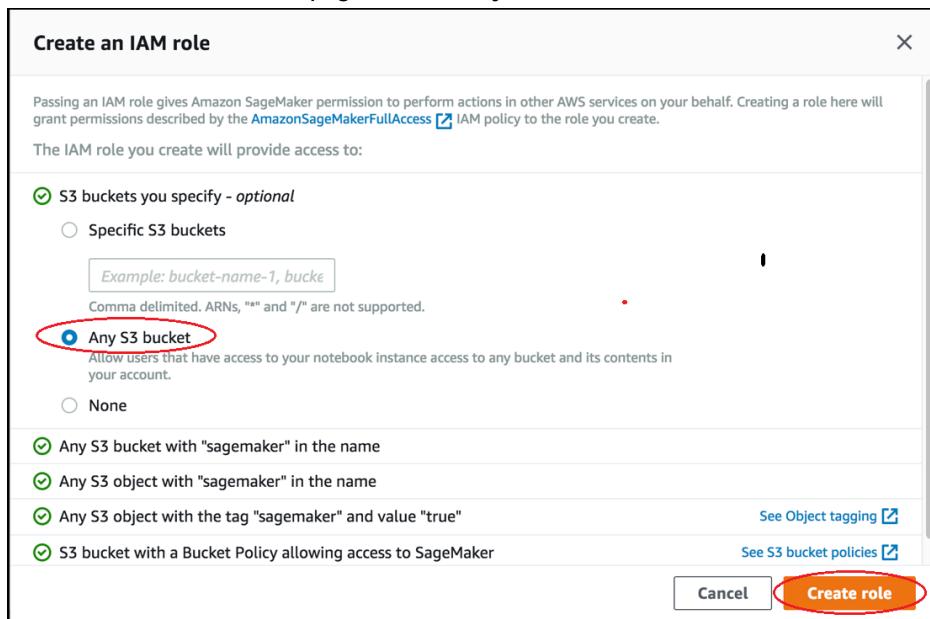
IAM role
Compiling jobs require permissions to call Amazon S3. Choose a role or let us create a role with the [AmazonSageMakerFullAccess](#) IAM policy attached.

AmazonSageMaker-ExecutionRole-20181128T122699

If you don't have an IAM role, choose **Create a new role**.



On the **Create an IAM role** page, choose **Any S3 bucket**, and choose **Create role**.



In the **Input configuration** section, for **Location of model artifacts**, enter the path of the S3 bucket that contains your model artifacts. For **Data input configuration**, enter the JSON string that specifies how many data matrix inputs you and the shape of each data matrices. For **Machine learning framework**, choose the framework.

Input configuration

Amazon SageMaker needs to know where model artifacts are stored, what the shape of the data matrix is, and which machine learning framework to use.

Location of model artifacts
Amazon SageMaker needs the path to the model artifacts in Amazon S3. To find the path, look in your Amazon S3 directories.

s3://chenchou-neo-virginia/TF_uncompiled/mobilenet_v1_1.0_224

To find a path, [go to Amazon S3](#)

Data input configuration
Amazon SageMaker needs to know what the shape of the data matrix is.

{"data": [1, 224, 224, 3]}

Machine learning framework
Choose the machine learning framework that your model was trained in.

TensorFlow

In the **Output configuration** section, for **S3 Output location**, enter the path to the S3 bucket or folder where you want to store the model. For **Target device**, choose which device you want to deploy your model to, and choose **Create job**.

Output configuration

Amazon SageMaker needs to know where to store the modules compiled with this job.

S3 Output location
Amazon SageMaker needs the path to the S3 bucket or folder where you want to store the compiled module. To find a path, go to [Amazon S3](#)

s3://chenchou-neo-virginia/TF_result

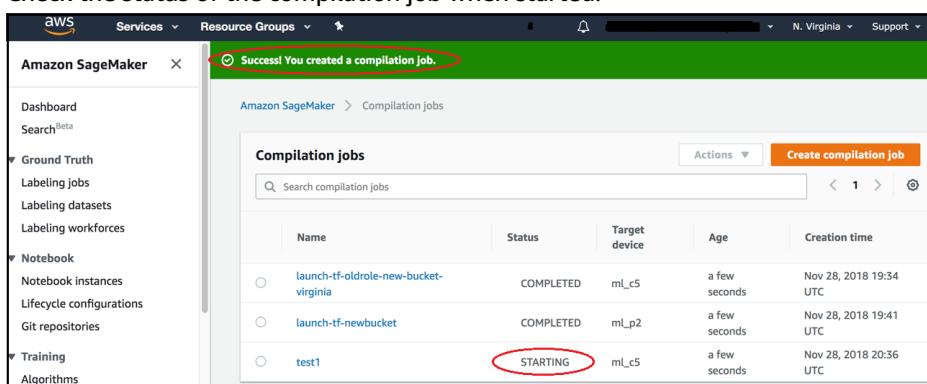
Target device
Amazon SageMaker needs to know where you intend to deploy your model: to an Amazon SageMaker ML instance or to an AWS IoT Greengrass device.

ml_c5

Cancel

Create job

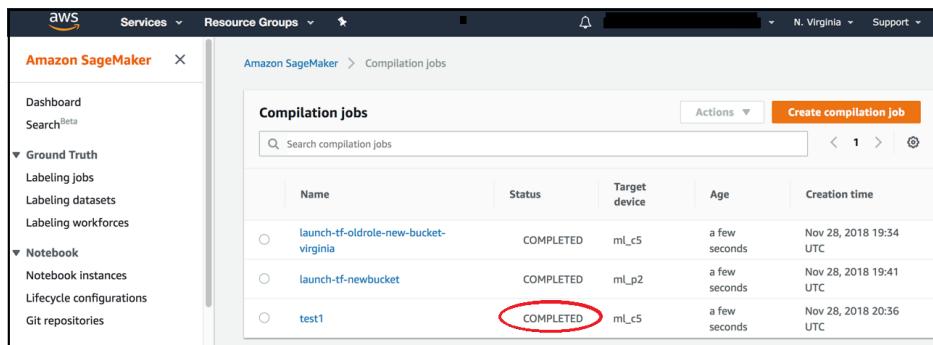
Check the status of the compilation job when started.



The screenshot shows the Amazon SageMaker console with the 'Compilation jobs' page. A success message 'Success! You created a compilation job.' is displayed at the top. The table lists three compilation jobs:

Name	Status	Target device	Age	Creation time
launch-tf-oldrole-new-bucket-virginia	COMPLETED	ml_c5	a few seconds	Nov 28, 2018 19:34 UTC
launch-tf-newbucket	COMPLETED	ml_p2	a few seconds	Nov 28, 2018 19:41 UTC
test1	STARTING	ml_c5	a few seconds	Nov 28, 2018 20:36 UTC

Check the status of the compilation job when completed.



Compile a Model (Amazon SageMaker SDK)

Follow the steps described in the [Running the Training Job](#) section of the [MNIST Training, Compilation and Deployment with MXNet Module](#) sample to produce a machine learning model train using Amazon SageMaker. Then you can use Neo to further optimize the model with the following code:

```
output_path = '/'.join( mnist_estimator.output_path.split('/')[-1:-1])
compiled_model = mnist_estimator.compile_model(target_instance_family='ml_c5',
                                                input_shape={'data':[1, 784]},
                                                role=role,
                                                output_path=output_path)
```

This code compiles the model and saves the optimized model in `output_path`. Sample notebooks of using SDK are provided in the [Amazon SageMaker Neo Sample Notebooks \(p. 683\)](#) section.

Deploy a Model

You can deploy the compact module to performance-critical cloud services with Amazon SageMaker Hosting Services or to resource-constrained edge devices with AWS IoT Greengrass.

Topics

- [Deploy a Model Compiled with Neo with Hosting Services \(p. 688\)](#)
- [Deploy a Model Compiled with Neo \(AWS IoT Greengrass\) \(p. 695\)](#)

Deploy a Model Compiled with Neo with Hosting Services

To deploy a Neo-compiled model to an HTTPS endpoint, you must configure and create the endpoint for the model using Amazon SageMaker hosting services. Currently developers can use Amazon SageMaker APIs to deploy modules on to ml.c5, ml.c4, ml.m5, ml.m4, ml.p3, and ml.p2 instances.

When you deploy a compiled model, you need to use the same instance for the target that you used for compilation. This creates an Amazon SageMaker endpoint that you can use to perform inferences. There are three options available for deploying Neo-compiled models:

Topics

- [Deploy a Model Compiled with Neo \(AWS CLI\) \(p. 688\)](#)
- [Deploy a Model Compiled with Neo \(Console\) \(p. 690\)](#)
- [Deploy a Model Compiled with Neo \(Amazon SageMaker SDK\) \(p. 695\)](#)

Deploy a Model Compiled with Neo (AWS CLI)

The deployment of a Neo-compiled model with the CLI has three steps.

Topics

- [Create a Model That Was Compiled with Neo \(AWS CLI\) \(p. 689\)](#)
- [Create the Endpoint Configuration \(AWS CLI\) \(p. 690\)](#)
- [Create an Endpoint \(AWS CLI\) \(p. 690\)](#)

Create a Model That Was Compiled with Neo (AWS CLI)

For the full syntax of the `CreateModel` API, see [CreateModel](#).

For Neo-compiled models, use one of the following values for `PrimaryContainer/ContainerHostname`, depending on your region and applications:

- **Amazon SageMaker Image Classification**
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest
- **Amazon SageMaker XGBoost**
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest
- **TensorFlow**: The TensorFlow version used must be in [TensorFlow SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
- **MXNet** The MXNet version used must be in [MXNet SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
- **Pytorch** The Pytorch version used must be in [Pytorch SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3

Also, if you are using **TensorFlow**, **Pytorch**, or **MXNet**, add the following key-value pair to **PrimaryContainer/Environment**:

```
"Environment": {
    "SAGEMAKER_SUBMIT_DIRECTORY" : "[Full S3 path for *.tar.gz file containing the training script]"
}
```

The script must be packaged as a `*.tar.gz` file. The `*.tar.gz` file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

- `neo_preprocess(payload, content_type)`: Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
- `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the [Amazon SageMaker Neo Sample Notebooks \(p. 683\)](#) for examples using these functions.

Create the Endpoint Configuration (AWS CLI)

For the full syntax of the `CreateEndpointConfig` API, see [CreateEndpointConfig](#). You must specify the correct instance type in `ProductionVariants/InstanceType`. It is imperative that this value matches the instance type specified in your compilation job.

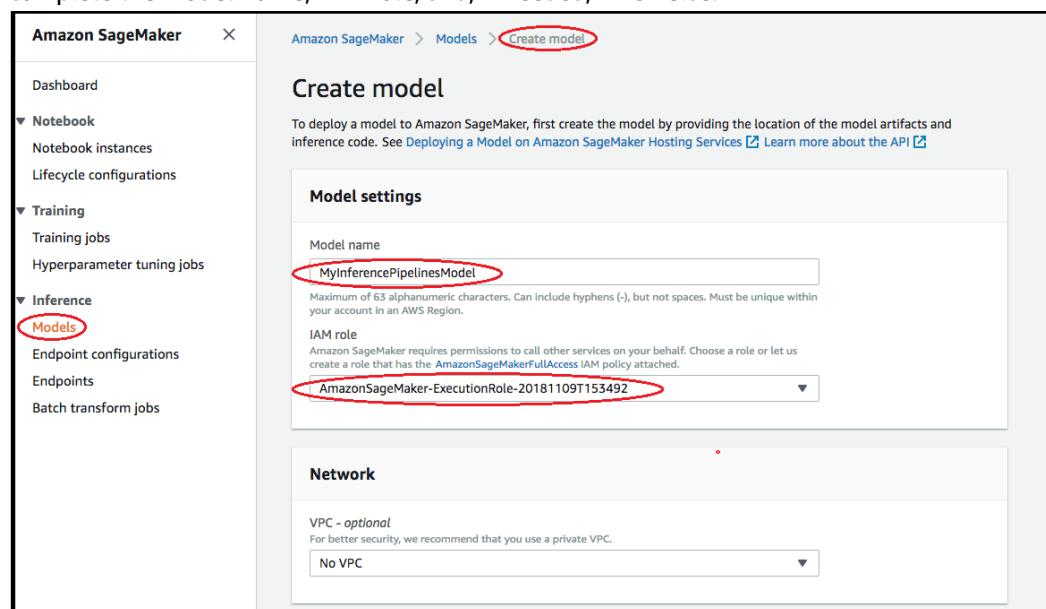
Create an Endpoint (AWS CLI)

For the full syntax of the `CreateEndpoint` API, see [CreateEndpoint](#).

Deploy a Model Compiled with Neo (Console)

You can create a Neo endpoint in the Amazon SageMaker console. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.

Choose **Models**, and then choose **Create models** from the **Inference** group. On the **Create model** page, complete the **Model name**, **IAM role**, and, if needed, **VPC** fields.



To add information about the container used to deploy your model, choose **Add container**, then choose **Next**. Complete the **Container input options**, **Location of inference code image**, and **Location of model artifacts**, and optionally, **Container host name**, and **Environmental variables** fields.

Container definition 1

▼ Container input options

Provide model artifacts and inference image.

▼ Provide model artifacts and inference image

Location of inference code image
The registry path where the inference code image is stored in Amazon ECR.

Location of model artifacts - *optional*
The URL for the S3 location where model artifacts are stored.

Container host name - *optional*
The DNS host name for the container.

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

▼ Environment variables - *optional*

Key	Value	
key1	value1	Remove
key2	value2	Remove

[Add environment variable](#)

To deploy Neo-compiled models, choose the following:

- **Container input options:** Provide model artifacts and inference image
- **Location of inference code image:** Choose one of the following images, depending the region and kind of application:
 - **Amazon SageMaker Image Classification**
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:latest
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/image-classification-neo:latest
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:latest
 - **Amazon SageMaker XGBoost**
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/xgboost-neo:latest
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest

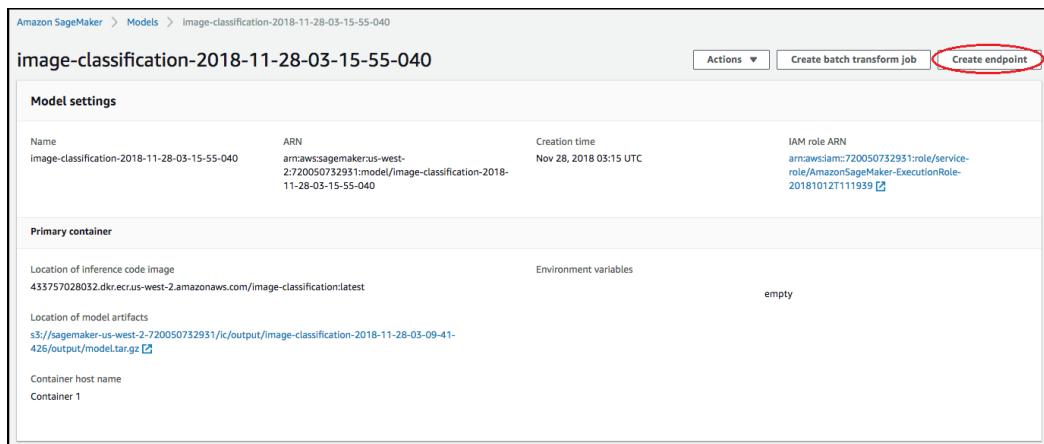
- 007439368137.dkr.ecr.us-east-2.amazonaws.com/xgboost-neo:latest
- 802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:latest
- **TensorFlow** : The TensorFlow version used must be in [TensorFlow SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-tensorflow:[tensorflow-version]-[cpu/gpu]-py3
- **MXNet** The MXNet version used must be in [MXNet SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:[mxnet-version]-[cpu/gpu]-py3
- **Pytorch** The Pytorch version used must be in [Pytorch SageMaker Estimators](#) list.
 - 301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
 - 802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:[pytorch-version]-[cpu/gpu]-py3
- **Location of model artifact:** the full S3 path of the compiled model artifact generated by the Neo compilation API.
- **Environmental variables:**
 - Omit this field for [SageMaker Image Classification](#) and [SageMaker XGBoost](#).
 - For [TensorFlow](#), [Pytorch](#), and [MXNet](#), specify the environment variable **SAGEMAKER_SUBMIT_DIRECTORY** as the full S3 path that contains the training script.

The script must be packaged as a `*.tar.gz` file. The `*.tar.gz` file must contain the training script at the root level. The script must contain two additional functions for Neo serving containers:

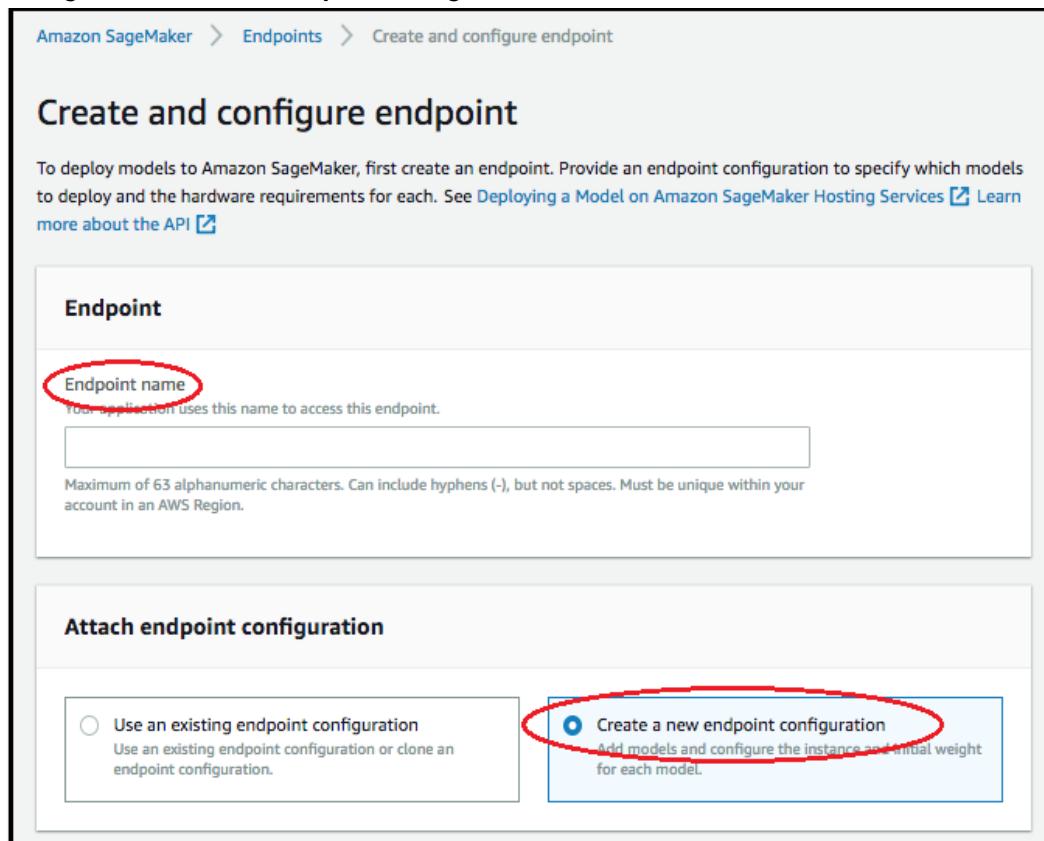
- `neo_preprocess(payload, content_type)`: Function that takes in the payload and Content-Type of each incoming request and returns a NumPy array.
- `neo_postprocess(result)`: Function that takes the prediction results produced by Deep Learning Runtime and returns the response body.

Neither of these two functions use any functionalities of MXNet, Pytorch, or Tensorflow. See the [Amazon SageMaker Neo Sample Notebooks \(p. 683\)](#) for examples using these functions.

Confirm that the information for the containers is accurate, and then choose **Create model**. This takes you to the create model landing page. Select the **Create endpoint** button there.



In **Create and configure endpoint** diagram, specify the **Endpoint name**. Choose **Create a new endpoint configuration** in **Attach endpoint configuration**.



In **New endpoint configuration** page, specify the **Endpoint configuration name**.

New endpoint configuration

To deploy models to Amazon SageMaker, first create an endpoint configuration. In the configuration, specify which models to deploy, and the relative traffic weighting and hardware requirements for each.

Endpoint configuration name

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in an AWS Region.

Encryption key - optional

Encrypt your data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption

Production variants

Model name	Variant name	Instance type	Initial instance count	Initial weight	Actions
image-classification-2018-11-28-03-15-55-040	default-variant-name	ml.m4.xlarge	1	1	Edit Remove

Add model

Create endpoint configuration

Then press **Edit** next to the name of the model and specify the correct **Instance type** on the **Edit Production Variant** page. It is imperative that the **Instance type** value match the one specified in your compilation job.

Edit Production Variant

Model name
image-classification-2018-11-28-03-15-55-040

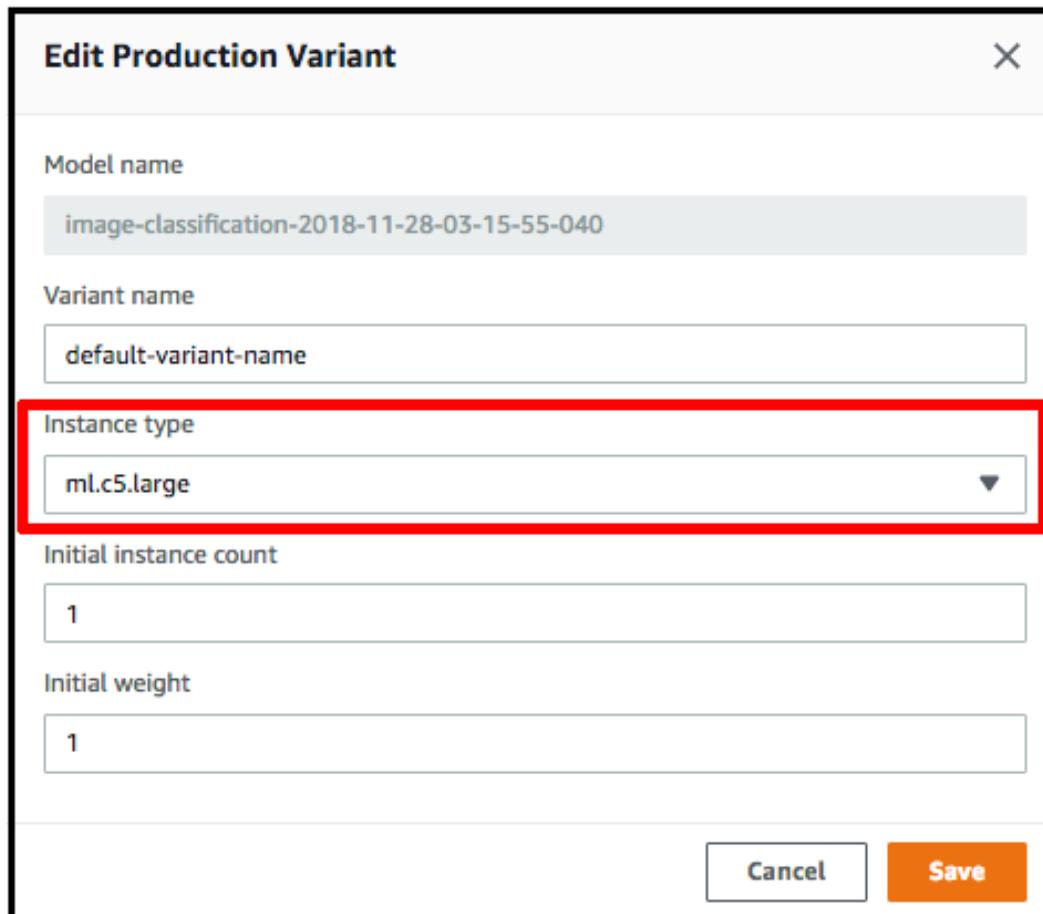
Variant name
default-variant-name

Instance type
ml.c5.large

Initial instance count
1

Initial weight
1

Cancel **Save**



When you're done click **Save**, then click **Create endpoint configuration** on the **New endpoint configuration** page, and then click **Create endpoint**.

Deploy a Model Compiled with Neo (Amazon SageMaker SDK)

The object handle for the compiled model supplies the `deploy` function, which allows you to create an endpoint to serve inference requests. The function lets you set the number and type of instances that are used for the endpoint. You must choose an instance for which you have compiled your model. For example, in the job compiled in [Compile a Model \(Amazon SageMaker SDK\) \(p. 688\)](#) section, this is `ml_c5`. The Neo API uses a special runtime, the *Neo runtime*, to run Neo-optimized models.

```
predictor = compiled_model.deploy(initial_instance_count = 1, instance_type =  
'ml.c5.4xlarge')
```

After the command is done, the name of the newly created endpoint is printed in the jupyter notebook.

Deploy a Model Compiled with Neo (AWS IoT Greengrass)

[AWS IoT Greengrass](#) extends cloud capabilities to local devices. It enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. With AWS IoT Greengrass, you can perform machine learning inference at the edge on locally generated data using cloud-trained models. Currently, you can deploy models on to all AWS IoT Greengrass devices based on ARM® Cortex-A™, Intel® Atom™, and Nvidia® Jetson™ series processors. For more information on deploying a Lambda inference application to perform machine learning inferences with AWS IoT Greengrass, see [Perform Machine Learning Inference](#).

Request Inferences from a Deployed Service

If you have followed instructions in [Deploy a Model Compiled with Neo with Hosting Services \(p. 688\)](#), you should have an Amazon SageMaker endpoint set up and running. You can now submit inference requests using Boto3 client. Here is an example of sending an image for inference:

```
import boto3
import json

endpoint = '<insert name of your endpoint here>'

runtime = boto3.Session().client('sagemaker-runtime')

# Read image into memory
with open(image, 'rb') as f:
    payload = f.read()
# Send image via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='application/x-
image', Body=payload)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

For XGBoost application, you should submit a CSV text instead:

```
import boto3
import json

endpoint = '<insert your endpoint here>'

runtime = boto3.Session().client('sagemaker-runtime')

csv_text = '1,-1.0,1.0,1.5,2.6'
# Send CSV text via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv',
    Body=csv_text)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

Note that BYOM allows for a custom content type. For more information, see [runtime_InvokeEndpoint](#).

Troubleshooting Neo Compilation Errors

This section contains information about how to understand and prevent common errors, the error messages they generate, and guidance on how to resolve these errors. It also contains lists of the frameworks and the operations in each of those frameworks that Neo supports.

Topics

- [Prevent Neo Input Errors \(p. 696\)](#)
- [Neo Error Messages \(p. 701\)](#)
- [Resolve Neo Errors \(p. 702\)](#)

Prevent Neo Input Errors

Some of the most common errors are due to invalid inputs. This section contains information arranged in question and answer form to help you avoid these errors.

Which frameworks does Neo support?

- [TensorFlow](#)
- [Keras](#)
- [PyTorch](#)
- [Apache MXNET](#)
- [XGBoost](#)
- [ONNX](#)

Which operators does Amazon SageMaker Neo support for these frameworks?

The following table lists the supported operations for each framework.

MXNet	TensorFlow	PyTorch/ONNX
'_add_scalar'	'Add'	'Abs'
'_add_symbol'	'ArgMax'	'Add'
'_contrib_MultiBoxDetection'	'ArgMin'	'ArgMax'
'_contrib_MultiBoxPrior'	'AvgPool'	'ArgMin'
'_copy'	'BatchNormWithGlobalNormalization'	'AveragePool'
'_div_scalar'	'BiasAdd'	'BatchNormalization'
'_div_symbol'	'Cast'	'Cast'
'_minus_scalar'	'Ceil'	'Ceil'
'_minus_scalar'	'CheckNumerics'	'Clip'
'_mul_symbol'	'Concat'	'Concat'
'_Plus'	'ConcatV2'	'Constant'
'_plus_scalar'	'Conv2D'	'ConstantFill'
'_pow_scalar'	'DecodeJpeg'	'Conv'
'_rdiv_scalar'	'DepthwiseConv2dNative'	'ConvTranspose'
'_rminus_scalar'	'Elu'	'Div'
'_rpow_scalar'	'Equal'	'Dropout'
'_rsub_scalar'	'ExpandDims'	'Elu'
'_sub_scalar'	'Fill'	'Exp'
'_sub_symbol'	'Floor'	'FC'
'Activation'	'FusedBatchNorm'	'Flatten'
'add_n'	'FusedBatchNormV2'	'Floor'
'argmax'	'GatherV2'	'Gather'
'BatchNorm'	'Greater'	'Gemm'
'BatchNorm_v1'	'GreaterEqual'	'GlobalAveragePool'

MXNet	TensorFlow	PyTorch/ONNX
'broadcast_add'	'Identity'	'GlobalMaxPool'
'broadcast_div'	'LeakyRelu'	'HardSigmoid'
'broadcast_mul'	'Less'	'Identity'
'broadcast_sub'	'LessEqual'	'ImageScaler'
'broadcast_to'	'LRN'	'LeakyRelu'
'cast'	'MatMul'	'Log'
'Cast'	'Maximum'	'LogSoftmax'
'clip'	'MaxPool'	'LRN'
'Concat'	'Mean'	'MatMul'
'concat'	'Minimum'	'Max'
'Convolution'	'Mul'	'MaxPool'
'Convolution_v1'	'NotEqual'	'Mean'
'Crop'	'Pack'	'Min'
'Deconvolution'	'Pad'	'Mul'
'Dropout'	'PadV2'	'Neg'
'elemwise_add'	'Range'	'Pad'
'elemwise_div'	'Rank'	'ParametricSoftplus'
'elemwise_mul'	'Relu'	'Pow'
'elemwise_sub'	'Relu6'	'PReLU'
'exp'	'Reshape'	'Reciprocal'
'expand_dims'	'ResizeBilinear'	'ReduceMax'
'flatten'	'Rsqrt'	'ReduceMean'
'Flatten'	'Selu'	'ReduceMin'
'FullyConnected'	'Shape'	'ReduceProd'
'LeakyReLU'	'Sigmoid'	'ReduceSum'
'LinearRegressionOutput'	'Softmax'	'Relu'
'log'	'Square'	'Reshape'
'log_softmax'	'Squeeze'	'Scale'
'LRN'	'StridedSlice'	'ScaledTanh'
'max'	'Sub'	'Selu'
'max_axis'	'Sum'	'Shape'

MXNet	TensorFlow	PyTorch/ONNX
'min'	'Tanh'	'Sigmoid'
'min_axis'	'Transpose'	'Slice'
'negative'		'Softmax'
'Pooling'		'SoftPlus'
'Pooling_v1'		'Softsign'
'relu'		'SpatialBN'
'Reshape'		'Split'
'reshape'		'Sqrt'
'reshape_like'		'Squeeze'
'sigmoid'		'Sub'
'slice_like'		'Sum'
'SliceChannel'		'Tanh'
'softmax'		'ThresholdedRelu'
'Softmax'		'Transpose'
'SoftmaxActivation'		'Unsqueeze'
'SoftmaxOutput'		'Upsample'
'split'		
'sum'		
'sum_axis'		
'tanh'		
'transpose'		
'UpSampling'		

Which model architectures does Neo support?

Neo supports image classification models.

Which model format files does Neo read in?

The file needs to be formatted as a tar.gz file that includes additional files that depend on the type of framework.

- **TensorFlow:** Neo supports saved models and frozen models.

For *saved models*, Neo expects one .pb or one .pbtxt file and a variables directory that contains variables.

For *frozen models*, Neo expect only one .pb or .pbtxt file.

- **Keras:** Neo expects one .h5 file containing the model definition.

- **PyTorch:** Neo expects one .pth file containing the model definition.
- **MXNET:** Neo expects one symbol file (.json) and one parameter file (.params).
- **XGBoost:** Neo expects one XGBoost model file (.model) where the number of nodes in a tree can't exceed 2^{31} .
- **ONNX:** Neo expects one .onnx file.

What input data shapes does Neo expect?

Neo expects the name and shape of the expected data inputs for your trained model with a JSON dictionary form or list form. The data inputs are framework specific.

- **TensorFlow:** You must specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
 - Examples for one input:
 - If using the console, `{"input": [1, 1024, 1024, 3]}`
 - If using the CLI, `{\"input\": [1, 1024, 1024, 3]}`
 - Examples for two inputs:
 - If using the console, `{"data1": [1, 28, 28, 1], "data2": [1, 28, 28, 1]}`
 - If using the CLI, `{\"data1\": [1, 28, 28, 1], \"data2\": [1, 28, 28, 1]}`
- **KERAS:** You must specify the name and shape (NCHW format) of expected data inputs using a dictionary format for your trained model. Note that while Keras model artifacts should be uploaded in NHWC (channel-last) format, DataInputConfig should be specified in NCHW (channel-first) format. The dictionary formats required for the console and CLI are different.
 - Examples for one input:
 - If using the console, `{"input_1": [1, 3, 224, 224]}`
 - If using the CLI, `{\"input_1\": [1, 3, 224, 224]}`
 - Examples for two inputs:
 - If using the console, `{"input_1": [1, 3, 224, 224], "input_2": [1, 3, 224, 224]}`
 - If using the CLI, `{\"input_1\": [1, 3, 224, 224], \"input_2\": [1, 3, 224, 224]}`
- **MXNET/ONNX:** You must specify the name and shape (NCHW format) of the expected data inputs in order using a dictionary format for your trained model. The dictionary formats required for the console and CLI are different.
 - Examples for one input:
 - If using the console, `{"data": [1, 3, 1024, 1024]}`
 - If using the CLI, `{\"data\": [1, 3, 1024, 1024]}`
 - Examples for two inputs:
 - If using the console, `{"var1": [1, 1, 28, 28], "var2": [1, 1, 28, 28]}`
 - If using the CLI, `{\"var1\": [1, 1, 28, 28], \"var2\": [1, 1, 28, 28]}`
- **PyTorch:** You can either specify the name and shape (NCHW format) of expected data inputs in order using a dictionary format for your trained model or you can specify the shape only using a list format. The dictionary formats required for the console and CLI are different. The list formats for the console and CLI are the same.
 - Examples for one input in dictionary format:
 - If using the console, `{"input0": [1, 3, 224, 224]}`
 - If using the CLI, `{\"input0\": [1, 3, 224, 224]}`
 - Example for one input in list format: `[[1, 3, 224, 224]]`
 - Examples for two inputs in dictionary format:
 - If using the console, `{"input0": [1, 3, 224, 224], "input1": [1, 3, 224, 224]}`

- If using the CLI, `{"input0": [1, 3, 224, 224], "input1": [1, 3, 224, 224]}`
- Example for two inputs in list format: `[[1, 3, 224, 224], [1, 3, 224, 224]]`
- XGBOOST: input data name and shape are not needed.

Neo Error Messages

This section lists and classifies Neo errors and error messages.

Neo Error Messages

This list catalogs the user and system error messages you can receive from Neo deployments.

- **User error messages**

- **Client permission error:** Neo passes the errors for these straight through from the dependent service.

Access Denied when calling sts:AssumeRole

Any *400 error* when calling S3 to download or upload a client model.

PassRole error

- **Load error:** Keywords in error messages, 'InputConfiguration','ModelErrorTooBig'.

Load Error: InputConfiguration: Exactly one {xxx} file is allowed for {yyy} model.

Load Error: ModelSizeTooBig: number of nodes in a tree can't exceed 2^31

- **Compilation error:** Keywords in error messages, 'OperatorNotImplemented','OperatorAttributeNotImplemented', 'OperatorAttributeRequired', 'OperatorAttributeValueNotValid'.

OperatorNotImplemented: {xxx} is not supported.

OperatorAttributeNotImplemented: {xxx} is not supported in {yyy}.

OperatorAttributeRequired: Required attribute {xxx} not found in {yyy}.

OperatorAttributeValueNotValid: The value of attribute {xxx} in operator {yyy} cannot be negative.

- **Any Malformed Input Errors**

- **System error messages**

- For system errors, Neo shows only one error message similar to the following: There was an unexpected error during compilation, check your inputs and try again in a few minutes.
- This covers all unexpected errors and errors that are not user errors.

Neo Error Classifications

This list classifies the *user errors* you can receive from Neo. These include access and permission errors and load errors for each of the supported frameworks. All other errors are *system errors*.

- **Client permission error:** Neo passes the errors for these straight through from the dependent service.

Access Denied when calling sts:AssumeRole

Any *400 error* when calling Amazon S3 to download or upload a client model.

PassRole error

- **Load error:** Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, check whether the tarball contains the necessary files for compilation. The checking criteria is framework-specific:
 - **TensorFlow:** Expects only protobuf file (*.pb or *.pbtxt). For *saved models*, expects one variables folder.
 - **Pytorch:** Expect only one pytorch file (*.pth).
 - **MXNET:** Expect only one symbol file (*.json) and one parameter file (*.params).
 - **XGBoost:** Expect only one XGBoost model file (*.model). The input model has size limitation.
- **Compilation error:** Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, and that the tarball contains necessary files for compilation. The checking criteria is:
 - **OperatorNotImplemented:** An operator has not been implemented.
 - **OperatorAttributeNotImplemented:** The attribute in the specified operator has not been implemented.
 - **OperatorAttributeRequired:** An attribute is required for an internal symbol graph, but it is not listed in the user input model graph.
 - **OperatorAttributeValueNotValid:** The value of the attribute in the specific operator is not valid.

Resolve Neo Errors

This section provides guidance on troubleshooting common issues with Neo. These include permission, load, compilation, and system errors and errors involving invalid inputs and unsupported operations.

- **Catalog of Known Issues:**
 - If you see **Client Permission Error**, review the set up documentation and make sure that you have correctly granted the permissions that are failing.
 - If you see **Load Error**, check the model format files that Neo expects for different frameworks.
 - If you see **Compilation Error**, check and address the details error message in your input model graph.
 - If you see **System Error**, try again in a few minutes. If that fails, file a ticket.
- **Lack of Roles and Permissions:** Review the set up documentation and make sure that you have correctly granted the permissions that are failing.
- **Invalid API and Console Inputs:** Fix your input as described in the validation error.
- **Unsupported Operators:**
 - Check the failure reason where Neo has listed all unsupported operators with the keyword 'OperatorNotImplemented'.
 - For example: Compilation Error: OperatorNotImplemented: The following operators are not implemented: {'_sample_multinomial', 'RNN'}
 - Remove the unsupported operators from your input model graph and test it again.

Use Amazon SageMaker Elastic Inference (EI)

By using Amazon Elastic Inference (EI), you can speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as [Amazon SageMaker hosted models](#), but at a fraction of the cost of using a GPU instance for your endpoint. EI allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance. Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type, and then add that model as a production variant to an endpoint configuration that you use to deploy a hosted endpoint. You can also add an EI accelerator to a Amazon SageMaker [notebook instance](#) so that you can test and evaluate inference performance when you are building your models.

Elastic Inference is supported in EI-enabled versions of TensorFlow and MXNet. To use any other deep learning framework, export your model by using ONNX, and then import your model into MXNet. You can then use your model with EI as an MXNet model. For information about importing an ONNX model into MXNet, see https://mxnet.apache.org/api/python/docs/tutorials/packages/onnx/super_resolution.html.

Topics

- [How EI Works \(p. 703\)](#)
- [Choose an EI Accelerator Type \(p. 703\)](#)
- [Use EI in a Amazon SageMaker Notebook Instance \(p. 704\)](#)
- [Use EI on a Hosted Endpoint \(p. 704\)](#)
- [Frameworks that Support EI \(p. 704\)](#)
- [Use EI with Amazon SageMaker Built-in Algorithms \(p. 705\)](#)
- [EI Sample Notebooks \(p. 705\)](#)
- [Set Up to Use EI \(p. 705\)](#)
- [Attach EI to a Notebook Instance \(p. 708\)](#)
- [Use EI on Amazon SageMaker Hosted Endpoints \(p. 710\)](#)

How EI Works

Amazon Elastic Inference accelerators are network attached devices that work along with Amazon SageMaker instances in your endpoint to accelerate your inference calls. Elastic Inference accelerates inference by allowing you to attach fractional GPUs to any Amazon SageMaker instance. You can select the client instance to run your application and attach an Elastic Inference accelerator to use the right amount of GPU acceleration for your inference needs. Elastic Inference helps you lower your cost when not fully utilizing your GPU instance for inference. We recommend trying Elastic Inference with your model using different CPU instances and accelerator sizes.

The following EI accelerator types are available. You can configure your endpoints or notebook instances with any EI accelerator type.

In the table, the throughput in teraflops (TFLOPS) is listed for both single-precision floating-point (F32) and half-precision floating-point (F16) operations. The memory in GB is also listed.

Accelerator Type	F32 Throughput in TFLOPS	F16 Throughput in TFLOPS	Memory in GB
ml.eia2.medium	1	8	2
ml.eia2.large	2	16	4
ml.eia2.xlarge	4	32	8
ml.eia1.medium	1	8	1
ml.eia1.large	2	16	2
ml.eia1.xlarge	4	32	4

Choose an EI Accelerator Type

Consider the following factors when choosing an accelerator type for a hosted model:

- Models, input tensors and batch sizes influence the amount of accelerator memory you need. Start with an accelerator type that provides at least as much memory as the file size of your trained model.
- Demands on CPU compute resources, GPU-based acceleration, and CPU memory vary significantly between different kinds of deep learning models. The latency and throughput requirements of the application also determine the amount of compute and acceleration you need. Thoroughly test different configurations of instance types and EI accelerator sizes to make sure you choose the configuration that best fits the performance needs of your application.

For more information on selecting an EI accelerator, see:

- [Amazon Elastic Inference Overview](#)
- [Choosing an Instance and Accelerator Type for Your Model](#)
- [Optimizing costs in Amazon Elastic Inference with TensorFlow](#)

Use EI in a Amazon SageMaker Notebook Instance

Typically, you build and test machine learning models in a Amazon SageMaker notebook before you deploy them for production. You can attach EI to your notebook instance when you create the notebook instance. You can set up an endpoint that is hosted locally on the notebook instance by using the local mode supported by TensorFlow and MXNet estimators and models in the Amazon SageMaker Python SDK to test inference performance. For instructions on how to attach EI to a notebook instance and set up a local endpoint for inference, see [Attach EI to a Notebook Instance \(p. 708\)](#).

Use EI on a Hosted Endpoint

When you are ready to deploy your model for production to provide inferences, you create a Amazon SageMaker hosted endpoint. You can attach EI to the instance where your endpoint is hosted to increase its performance at providing inferences. For instructions on how to attach EI to a hosted endpoint instance, see [Use EI on Amazon SageMaker Hosted Endpoints \(p. 710\)](#).

Frameworks that Support EI

EI is designed to be used with AWS enhanced versions of TensorFlow or Apache MXNet machine learning frameworks. These enhanced versions of the frameworks are automatically built into containers when you use the Amazon SageMaker Python SDK, or you can download them as binary files and import them in your own Docker containers. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at <https://s3.console.aws.amazon.com/s3/buckets/amazonei-tensorflow>. For information about building a container that uses the EI-enabled version of TensorFlow, see <https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container>. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at <https://s3.console.aws.amazon.com/s3/buckets/amazonei-apachemxnet>. For information about building a container that uses the EI-enabled version of MXNet, see <https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container>.

To use EI in a hosted endpoint, you can use any of the following, depending on your needs.

- SageMaker Python SDK TensorFlow - if you want to use TensorFlow and you don't need to build a custom container.
- SageMaker Python SDK MXNet - if you want to use MXNet and you don't need to build a custom container.
- The low-level AWS Amazon SageMaker SDK for Python (Boto 3) - if you need to build a custom container.

Typically, you don't need to create a custom container unless your model is very complex and requires extensions to a framework that the Amazon SageMaker pre-built containers do not support.

Use EI with Amazon SageMaker Built-in Algorithms

Currently, the [Image Classification Algorithm \(p. 324\)](#) and [Object Detection Algorithm \(p. 402\)](#) built-in algorithms support EI. For an example that uses the Image Classification algorithm with EI, see https://github.com/awslabs/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/imageclassification_caltech/Image-classification-fulltraining.ipynb.

EI Sample Notebooks

The following Sample notebooks provide examples of using EI in Amazon SageMaker:

- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb
- https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_serving_using_elastic_inference_with_your_own_model/tensorflow_serving_pretrained_model_elastic_inference.ipynb

Set Up to Use EI

Use the instructions in this topic only if one of the following applies to you:

- You want to use a customized role or permission policy.
- You want to use a VPC for your hosted model or notebook instance.

Note

If you already have an execution role that has the `AmazonSageMakerFullAccess` managed policy attached (this is true for any IAM role that you create when you create a notebook instance, training job, or model in the console) and you are not connecting to an EI model or notebook instance in a VPC, you do not need to make any of these changes to use EI in Amazon SageMaker.

Topics

- [Set Up Required Permissions \(p. 705\)](#)
- [Use a Custom VPC to Connect to EI \(p. 708\)](#)

Set Up Required Permissions

To use EI in Amazon SageMaker, the role that you use to open a notebook instance or create a deployable model must have a policy with the required permissions attached. You can attach the `AmazonSageMakerFullAccess` managed policy, which contains the required permissions, to the role, or you can add a custom policy that has the required permissions. For information about creating an IAM

role, see [Creating a Role for an AWS Service \(Console\)](#) in the *AWS Identity and Access Management User Guide*. For information about attaching a policy to a role, see [Adding and Removing IAM Policies](#).

Add these permissions specifically for connecting EI in an IAM policy:

```
{
    "Effect": "Allow",
    "Action": [
        "elastic-inference:Connect",
        "ec2:DescribeVpcEndpoints"
    ],
    "Resource": "*"
}
```

The following IAM policy is the complete list of required permissions to use EI in Amazon SageMaker:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "elastic-inference:Connect",
                "ec2:DescribeVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:/*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "cloudwatch:PutMetricData",
                "cloudwatch:PutMetricAlarm",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:DeleteAlarms",
                "ec2>CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2>DeleteNetworkInterface",
                "ec2>DeleteNetworkInterfacePermission",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "application-autoscaling>DeleteScalingPolicy",
                "application-autoscaling>DeleteScheduledAction",
                "application-autoscaling>DeregisterScalableTarget",
                "application-autoscaling>DescribeScalableTargets",
                "application-autoscaling>DescribeScalingActivities",
                "application-autoscaling>DescribeScalingPolicies",
                "application-autoscaling>DescribeScheduledActions",
                "application-autoscaling>PutScalingPolicy",
                "application-autoscaling>PutScheduledAction",
                "application-autoscaling>PutScalingPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

```

        "application-autoscaling:RegisterScalableTarget",
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs>DescribeLogStreams",
        "logs>GetLogEvents",
        "logs>PutLogEvents"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3>DeleteObject"
    ],
    "Resource": [
        "arn:aws:s3::::*SageMaker*",
        "arn:aws:s3::::*Sagemaker*",
        "arn:aws:s3::::sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3>CreateBucket",
        "s3>GetBucketLocation",
        "s3>ListBucket",
        "s3>ListAllMyBuckets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3>GetObject"
    ],
    "Resource": "*",
    "Condition": {
        "StringEqualsIgnoreCase": {
            "s3:ExistingObjectTag/SageMaker": "true"
        }
    }
},
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::role/aws-service-role/sagemaker.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam>PassRole"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "sagemaker.amazonaws.com"
        }
    }
}

```

```
        ]  
    }
```

Use a Custom VPC to Connect to EI

To use EI with Amazon SageMaker in a VPC, you need to create and configure two security groups, and set up a PrivateLink VPC interface endpoint. EI uses VPC interface endpoint to communicate with Amazon SageMaker endpoints in your VPC. The security groups you create are used to connect to the VPC interface endpoint.

Set up Security Groups to Connect to EI

To use EI within a VPC, you need to create two security groups:

- A security group to control access to the VPC interface endpoint that you will set up for EI.
- A security group that allows Amazon SageMaker to call into the first security group.

Complete the following steps to configure the two security groups:

1. Create a security group with no outbound connections. You will attach this to the VPC endpoint interface you create in the next section.
2. Create a second security group with no inbound connections, but with an outbound connection to the first security group.
3. Edit the first security group to allow inbound connections only to the second security group and all outbound connections.

For more information about VPC security groups, see [Security Groups for Your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

Set up a VPC Interface Endpoint to Connect to EI

To use EI with Amazon SageMaker in a custom VPC, you need to set up a VPC interface endpoint (PrivateLink) for the EI service.

- Set up a VPC interface endpoint (PrivateLink) for the EI. Follow the instructions at [Creating an Interface Endpoint](#). In the list of services, choose **com.amazonaws.<region>.elastic-inference.runtime**. For **Security group**, make sure you select the first security group you created in the previous section to the endpoint.
- When you set up the interface endpoint, choose all of the Availability Zones where EI is available. EI fails if you do not set up at least two Availability Zones. For information about VPC subnets, see [VPCs and Subnets](#).

Attach EI to a Notebook Instance

To test and evaluate inference performance using EI, you can attach EI to a notebook instance when you create or update a notebook instance. You can then use EI in local mode to host a model at an endpoint hosted on the notebook instance. You should test various sizes of notebook instances and EI accelerators to evaluate the configuration that works best for your use case.

Set Up to Use EI

To use EI locally in a notebook instance, create a notebook instance with an EI instance. To do this:

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>
2. In the navigation pane, choose **Notebook instances**.
3. Choose **Create notebook instance**.
4. For **Notebook instance name**, provide a unique name for your notebook instance.
5. For **notebook instance type**, choose a CPU instance such as **ml.t2.medium**.
6. For **Elastic Inference (EI)**, choose an instance from the list, such as **ml.eia2.medium**.
7. For **IAM role**, choose an IAM role that has the required permissions to use Amazon SageMaker and EI.
8. (Optional) For **VPC - Optional**, if you want the notebook instance to use a VPC, choose one from the available list, otherwise leave it as **No VPC**. If you use a VPC follow the instructions at [Use a Custom VPC to Connect to EI \(p. 708\)](#).
9. (Optional) For **Lifecycle configuration - optional**, either leave it as **No configuration** or choose a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).
10. (Optional) For **Encryption key - optional**, if you want Amazon SageMaker to use an AWS Key Management Service key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.
11. (Optional) For **Volume Size In GB - optional**, leave the default value of 5.
12. (Optional) For **Tags**, add tags to the notebook instance. A tag is a label you assign to help manage your notebook instances. A tag consists of a key and a value both of which you define.
13. Choose **Create Notebook Instance**.

After you create your notebook instance with EI attached, you can create a Jupyter notebook and set up an EI endpoint that is hosted locally on the notebook instance.

Topics

- [Use EI in Local Mode in Amazon SageMaker \(p. 709\)](#)

Use EI in Local Mode in Amazon SageMaker

To use EI locally in an endpoint hosted on a notebook instance, use local mode with the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet estimators or models. For more information about local mode support in the Amazon SageMaker Python SDK, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview>.

Topics

- [Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models \(p. 709\)](#)
- [Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models \(p. 710\)](#)

Use EI in Local Mode with Amazon SageMaker TensorFlow Estimators and Models

To use EI with TensorFlow in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about Amazon SageMaker Python SDK TensorFlow estimators and models, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst>.

The following code shows how to use local mode with an estimator object. To call the `deploy` method, you must have previously either:

- Trained the model by calling the `fit` method of an estimator.
- Pass a model artifact when you initialize the model object.

```
# Deploys the model to a local endpoint
tf_predictor = tf_model.deploy(initial_instance_count=1,
                               instance_type='local',
                               accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with TensorFlow, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/tensorflow_iris_dnn_classifier_using_estimators/tensorflow_iris_dnn_classifier_using_estimators_elastic_inference_local.ipynb

Use EI in Local Mode with Amazon SageMaker Apache MXNet Estimators and Models

To use EI with MXNet in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about Amazon SageMaker Python SDK MXNet estimators and models, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst>.

The following code shows how to use local mode with an estimator object. You must have previously called the `fit` method of the estimator to train the model.

```
# Deploys the model to a local endpoint
mxnet_predictor = mxnet_estimator.deploy(initial_instance_count=1,
                                         instance_type='local',
                                         accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with MXNet, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.ipynb.

Use EI on Amazon SageMaker Hosted Endpoints

To use Elastic Inference (EI) in Amazon SageMaker with a hosted endpoint for real-time inference, specify an EI accelerator when you create the deployable model to be hosted at that endpoint. You can do this in one of the following ways:

- Use the Amazon SageMaker Python SDK versions of either the TensorFlow or MXNet and the Amazon SageMaker pre-built containers for TensorFlow and MXNet
- Build your own container, and use the low-level Amazon SageMaker API (Boto 3). You will need to import the EI-enabled version of either TensorFlow or MXNet from the provided Amazon S3 locations into your container, and use one of those versions to write your training script.
- Use either the [Image Classification Algorithm \(p. 324\)](#) or [Object Detection Algorithm \(p. 402\)](#) build-in algorithms, and use Boto 3 to run your training job and create your deployable model and hosted endpoint.

Topics

- [Use EI with an Amazon SageMaker TensorFlow Container \(p. 711\)](#)
- [Use EI with an Amazon SageMaker MXNet Container \(p. 711\)](#)
- [Use EI with Your Own Container \(p. 712\)](#)

Use EI with an Amazon SageMaker TensorFlow Container

To use TensorFlow with EI in Amazon SageMaker, you need to call the `deploy` method of either the [Estimator](#) or [Model](#) objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using TensorFlow in the Amazon SageMaker Python SDK, see: <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/tensorflow/README.rst>.

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

Use an Estimator Object

To use an estimator object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                             instance_type='ml.m4.xlarge',
                             accelerator_type='ml.eia2.medium')
```

Use a Model Object

To use a model object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                        instance_type='ml.m4.xlarge',
                        accelerator_type='ml.eia2.medium')
```

Use EI with an Amazon SageMaker MXNet Container

To use MXNet with EI in Amazon SageMaker, you need to call the `deploy` method of either the [Estimator](#) or [Model](#) objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using MXNet in the Amazon SageMaker Python SDK, see <https://github.com/aws/sagemaker-python-sdk/blob/master/src/sagemaker/mxnet/README.rst>

Amazon SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

Use an Estimator Object

To use an estimator object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                             instance_type='ml.m4.xlarge',
                             accelerator_type='ml.eia2.medium')
```

Use a Model Object

To use a model object with EI, include the `accelerator_type` input argument when you use the `deploy` method. The estimator returns a predictor object which we call its `deploy` method as shown in the example code:

```
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                         instance_type='ml.m4.xlarge',
                         accelerator_type='ml.eia2.medium')
```

For a complete example of using EI with MXNet in Amazon SageMaker, see the sample notebook at https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb

Use EI with Your Own Container

To use EI with a model in a custom container that you build, use the low-level Amazon SageMaker SDK for Python (Boto 3). download and import the AWS EI-enabled versions of TensorFlow or Apache MXNet machine learning frameworks, and write your training script using those frameworks.

Import the EI Version of TensorFlow or MXNet into Your Docker Container

To use EI with your own container, you need to import either the Amazon EI TensorFlow Serving library or the Amazon EI Apache MXNet library into your container. The EI-enabled versions of TensorFlow and MXNet are currently available as binary files stored in Amazon S3 locations. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at <https://s3.console.aws.amazon.com/s3/buckets/amazonei-tensorflow>. For information about building a container that uses the EI-enabled version of TensorFlow, see <https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container>. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at <https://s3.console.aws.amazon.com/s3/buckets/amazonei-apachemxnet>. For information about building a container that uses the EI-enabled version of MXNet, see <https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container>.

Create an EI Endpoint with Boto 3

To create an endpoint by using Boto 3, you first create an endpoint configuration. The endpoint configuration specifies one or more models (called production variants) that you want to host at the endpoint. To attach EI to one or more of the production variants hosted at the endpoint, you specify one of the EI instance types as the AcceleratorType field for that ProductionVariant. You then pass that endpoint configuration when you create the endpoint.

Create an Endpoint Configuration

To use EI, you need to specify an accelerator type in the endpoint configuration:

```
# Create Endpoint Configuration
from time import gmtime, strftime

endpoint_config_name = 'ImageClassificationEndpointConfig-' + 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',
        'AcceleratorType':'ml.eia2.medium'}])

print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

Create an Endpoint

After you create an endpoint configuration with an accelerator type, you can proceed to create an endpoint.

```
endpoint_name = 'ImageClassificationEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
endpoint_response = sagemaker.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
```

After the endpoint is created you can invoke it using the `invoke_endpoint` method in a `boto3` runtime object as you would any other endpoint.

Automatically Scale Amazon SageMaker Models

Amazon SageMaker supports automatic scaling for production variants. *Automatic scaling* dynamically adjusts the number of instances provisioned for a production variant in response to changes in your workload. When the workload increases, automatic scaling brings more instances online. When the workload decreases, automatic scaling removes unnecessary instances so that you don't pay for provisioned variant instances that you aren't using.

To use automatic scaling for a production variant, you define and apply a scaling policy that uses Amazon CloudWatch metrics and target values that you assign. Automatic scaling uses the policy to increase or decrease the number of instances in response to actual workloads.

You can use the AWS Management Console to apply a scaling policy based on a predefined metric. A *predefined metric* is defined in an enumeration so that you can specify it by name in code or use it in the AWS Management Console. Alternatively, you can use either the AWS Command Line Interface (AWS CLI) or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric. We strongly recommend that you load test your automatic scaling configuration to ensure that it works correctly before using it to manage production traffic.

For information about deploying trained models as endpoints, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#).

Topics

- [Automatic Scaling Components \(p. 713\)](#)
- [Before You Begin \(p. 716\)](#)
- [Related Topics \(p. 716\)](#)
- [Configure Automatic Scaling for a Production Variant \(p. 716\)](#)
- [Edit a Scaling Policy \(p. 722\)](#)
- [Delete a Scaling Policy \(p. 723\)](#)
- [Update Endpoints that Use Automatic Scaling \(p. 724\)](#)
- [Load Testing for Production Variant Automatic Scaling \(p. 725\)](#)
- [Best Practices for Configuring Automatic Scaling \(p. 726\)](#)

Automatic Scaling Components

To adjust the number of instances hosting a production variant, Amazon SageMaker automatic scaling uses a scaling policy. Automatic scaling has the following components:

- Required permissions—Permissions that are required to perform automatic scaling actions.
- A service-linked role—An AWS Identity and Access Management (IAM) role that is linked to a specific AWS service. A service-linked role includes all of the permissions that the service requires to call other AWS services on your behalf. Amazon SageMaker automatic scaling automatically generates this role, `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint`, for you.
- A target metric—The Amazon CloudWatch metric that Amazon SageMaker automatic scaling uses to determine when and how much to scale.
- Minimum and maximum capacity—The minimum and maximum number of instances to use for scaling the variant.
- A cool down period—The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.

Required Permissions for Automatic Scaling

The `SagemakerFullAccessPolicy` IAM policy has all of the permissions required to perform automatic scaling actions. For more information about Amazon SageMaker IAM roles, see [Amazon SageMaker Roles \(p. 776\)](#).

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

```
{
    "Effect": "Allow",
    "Action": [
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:UpdateEndpointWeightsAndCapacities"
    ],
    "Resource": "*"
}
{
    "Action": [
        "application-autoscaling:)"
    ],
    "Effect": "Allow",
    "Resource": "*"
}

{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": { "iam:AWSPropertyName": "sagemaker.application-autoscaling.amazonaws.com" }
    }
}

{
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:DeleteAlarms"
    ],
    "Resource": "*"
}
```

Service-Linked Role for Automatic Scaling

A service-linked role is a unique type of IAM role that is linked directly to an AWS service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other AWS services on your behalf. Automatic scaling uses the `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint` service-linked role. For more information, see [Service-Linked Roles for Application Auto Scaling](#) in the *Application Auto Scaling User Guide*.

Target Metric for Automatic Scaling

Amazon SageMaker automatic scaling uses target-tracking scaling policies. You configure the *target-tracking scaling policy* by specifying a predefined or custom metric and a target value for the metric. For more information, see [Target Tracking Scaling Policies](#).

Amazon CloudWatch alarms trigger the scaling policy, which calculate how to adjust scaling based on the metric and target value that you set. The scaling policy adds or removes endpoint instances as required to keep the metric at, or close to, the specified target value. In addition, a target-tracking scaling policy also adjusts to fluctuations in the metric when a workload changes. The policy minimizes rapid fluctuations in the number of available instances for your variant.

For example, a scaling policy that uses the predefined `InvocationsPerInstance` metric with a target value of 70 can keep `InvocationsPerInstance` at, or close to 70.

Minimum and Maximum Capacity for Automatic Scaling

You can specify the maximum number of endpoint instances that Application Auto Scaling manages for the variant. The maximum value must be equal to or greater than the value specified for the minimum number of endpoint instances. Amazon SageMaker automatic scaling does not enforce a limit for this value.

You can also specify the minimum number of instances that Application Auto Scaling manages for the variant. This value must be at least 1, and equal to or less than the value specified for the maximum number of variant instances.

To determine the minimum and maximum number of instances that you need for typical traffic, test your automatic scaling configuration with the expected rate of traffic to your variant.

Cooldown Period for Automatic Scaling

Tune the responsiveness of a target-tracking scaling policy by adding a cooldown period. A *cooldown period* controls when your variant is scaled in and out by blocking subsequent scale-in or scale-out requests until the period expires. This slows the deletion of variant instances for scale-in requests, and the creation of variant instances for scale-out requests. A cooldown period helps to ensure that it doesn't launch or terminate additional instances before the previous scaling activity takes effect. After automatic scaling dynamically scales using a scaling policy, it waits for the cooldown period to complete before resuming scaling activities.

You configure the cooldown period in your automatic scaling policy. You can specify the following cooldown periods:

- A scale-in activity reduces the number of variant instances. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.
- A scale-out activity increases the number of variant instances. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

If you don't specify a scale-in or a scale-out cooldown period automatic scaling use the default, which is 300 seconds for each.

If instances are being added or removed too quickly when you test your automatic scaling configuration, consider increasing this value. You can see this behavior if the traffic to your variant has a lot of spikes, or if you have multiple automatic scaling policies defined for a variant.

If instances are not being added quickly enough to address increased traffic, consider decreasing this value.

Before You Begin

Before you can use automatically scaled model deployment, create an Amazon SageMaker model deployment. For more information about deploying a model endpoint, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#).

When automatic scaling adds a new variant instance, it is the same instance class as the one used by the primary instance.

Related Topics

- [What Is Application Auto Scaling?](#)

Configure Automatic Scaling for a Production Variant

You can configure automatic scaling for a variant with the AWS Management Console, the AWS CLI, or the Application Auto Scaling API.

Topics

- [Configure Automatic Scaling for a Production Variant \(Console\) \(p. 716\)](#)
- [Configure Automatic Scaling for a Production Variant \(AWS CLI or the Application Auto Scaling API\) \(p. 717\)](#)

Configure Automatic Scaling for a Production Variant (Console)

To configure automatic scaling for a production variant (console)

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint that you want to configure.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. For **Endpoint runtime settings**, choose **Configure auto scaling**.

The **Configure variant automatic scaling** page appears.

6. For **Minimum capacity**, type the minimum number of instances that you want the scaling policy to maintain. At least 1 instance is required.
7. For **Maximum capacity**, type the maximum number of instances that you want the scaling policy to maintain.
8. For the target value, type the average number of invocations per instance per minute for the variant. To determine this value, follow the guidelines in [Load Testing \(p. 725\)](#).

Application Auto Scaling adds or removes instances to keep the metric close to the value that you specify.

9. For **Scale-in cool down (seconds)** and **Scale-out cool down (seconds)**, type the number seconds for each cool down period. Assuming that the order in the list is based on either most important to less important of first applied to last applied.
10. Select **Disable scale in** to prevent the scaling policy from deleting variant instances if you want to ensure that your variant scales out to address increased traffic, but are not concerned with removing instances to reduce costs when traffic decreases, disable scale-in activities.

Scale-out activities are always enabled so that the scaling policy can create endpoint instances as needed.
11. Choose **Save**.

This procedure registers a variant as a scalable target with Application Auto Scaling. When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a burstable performance instance such as T2

Note

Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because they already allow for increased capacity under increased workloads. For information about burstable performance instances, see [Amazon EC2 Instance Types](#).

Configure Automatic Scaling for a Production Variant (AWS CLI or the Application Auto Scaling API)

With the AWS CLI or the Application Auto Scaling API, you can configure automatic scaling based on either a predefined or a custom metric.

Register a Production Variant

To define the scaling limits for the variant, register your variant with Application Auto Scaling. Application Auto Scaling dynamically scales the number of variant instances.

To register your variant, you can use either the AWS CLI or the Application Auto Scaling API.

When you register a variant, Application Auto Scaling performs validation checks to ensure the following:

- The variant resource exists
- The permissions are sufficient
- You aren't registering a variant with an instance that is a Burstable Performance Instance such as T2

Note

Amazon SageMaker automatic scaling doesn't support automatic scaling for burstable instances such as T2, because burstable instances already allow for increased capacity under increased workloads. For information about Burstable Performance Instances, see [Amazon EC2 Instance Types](#).

Register a Production Variant (AWS CLI)

To register your endpoint, use the `register-scalable-target` AWS CLI command with the following parameters:

- **--service-namespace**—Set this value to `sagemaker`.
- **--resource-id**—The resource identifier for the production variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/MyEndPoint/variant/MyVariant`.
- **--scalable-dimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- **--min-capacity**—The minimum number of instances that Application Auto Scaling must manage for this endpoint. Set `min-capacity` to at least 1. It must be equal to or less than the value specified for `max-capacity`.
- **--max-capacity**—The maximum number of instances that Application Auto Scaling should manage. Set `max-capacity` to a minimum of 1. It must be equal to or greater than the value specified for `min-capacity`.

Example

The following example shows how to register an endpoint variant named `MyVariant` that is dynamically scaled to have one to eight instances:

```
aws application-autoscaling register-scalable-target \
--service-namespace sagemaker \
--resource-id endpoint/MyEndPoint/variant/MyVariant \
--scalable-dimension sagemaker:variant:DesiredInstanceCount \
--min-capacity 1 \
--max-capacity 8
```

Register a Production Variant (Application Auto Scaling API)

To register your endpoint variant with Application Auto Scaling, use the [RegisterScalableTarget](#) Application Auto Scaling API action with the following parameters:

- **ServiceNamespace**—Set this value to `sagemaker`.
- **ResourceID**—The resource identifier for the production variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant, for example `endpoint/MyEndPoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- **MinCapacity**—The minimum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or less than the value specified for `MaxCapacity`.
- **MaxCapacity**—The maximum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or greater than the value specified for `MinCapacity`.

Example

The following example shows how to register an Amazon SageMaker production variant that is dynamically scaled to use one to eight instances:

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
```

```
"ServiceNamespace": "sagemaker",
"ResourceId": "endpoint/MyEndPoint/variant/MyVariant",
"ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
"MinCapacity": 1,
"MaxCapacity": 8
}
```

Define a Target-Tracking Scaling Policy

To specify the metrics and target values for a scaling policy, you configure a target-tracking scaling policy. You can use either a predefined metric or a custom metric.

Scaling policy configuration is represented by a JSON block. You save your scaling policy configuration as a JSON block in a text file. You use that text file when invoking the AWS CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration.

Topics

- [Use a Predefined Metric \(p. 719\)](#)
- [Use a Custom Metric \(p. 719\)](#)
- [Add a Cooldown Period \(p. 720\)](#)
- [Disable Scale-in Activity \(p. 720\)](#)

Use a Predefined Metric

To quickly define a target-tracking scaling policy for a variant, use the `SageMakerVariantInvocationsPerInstance` predefined metric.

`SageMakerVariantInvocationsPerInstance` is the average number of times per minute that each instance for a variant is invoked. We strongly recommend using this metric.

To use a predefined metric in a scaling policy, create a target tracking configuration for your policy. In the target tracking configuration, include a `PredefinedMetricSpecification` for the predefined metric and a `TargetValue` for the target value of that metric.

Example

The following example is a typical policy configuration for target-tracking scaling for a variant. In this configuration, we use the `SageMakerVariantInvocationsPerInstance` predefined metric to adjust the number of variant instances so that each instance has a `InvocationsPerInstance` metric of 70.

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

Use a Custom Metric

If you need to define a target-tracking scaling policy that meets your custom requirements, define a custom metric. You can define a custom metric based on any production variant metric that changes in proportion to scaling.

Not all Amazon SageMaker metrics work for target tracking. The metric must be a valid utilization metric, and it must describe how busy an instance is. The value of the metric must increase or decrease

in inverse proportion to the number of variant instances. That is, the value of the metric should decrease when the number of instances increases.

Important

Before deploying automatic scaling in production, you must test automatic scaling with your custom metric.

Example

The following example is a target-tracking configuration for a scaling policy. In this configuration, for a variant named `my-variant`, a custom metric adjusts the variant based on an average CPU utilization of 50 percent across all instances.

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

[Add a Cooldown Period](#)

To add a cooldown period for scaling out your variant, specify a value, in seconds, for `ScaleOutCooldown`. Similarly, to add a cooldown period for scaling in your variant, add a value, in seconds, for `ScaleInCooldown`. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

Example

The following is an example of a target-tracking policy configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric is used to adjust a variant based on an average of 70 across all instances of that variant. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

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

[Disable Scale-in Activity](#)

You can prevent the target-tracking scaling policy configuration from scaling in your variant by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting instances, while still allowing it to create them as needed.

To enable or disable scale-in activity for your variant, specify a Boolean value for `DisableScaleIn`. For more information about `DisableScaleIn`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

Example

The following is an example of a target-tracking configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric adjusts a variant based on an average of 70 across all instances of that variant. The configuration disables scale-in activity for the scaling policy.

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

Apply a Scaling Policy to a Production Variant

After registering your variant and defining a scaling policy, apply the scaling policy to the registered variant. To apply a scaling policy to a variant, you can use the AWS CLI or the Application Auto Scaling API.

Apply a Scaling Policy to a Production Variant (AWS CLI)

To apply a scaling policy to your variant, use the `put-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--policy-type`—Set this value to `TargetTrackingScaling`.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--target-tracking-scaling-policy-configuration`—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses with Application Auto Scaling to apply a target-tracking scaling policy named `myscalablepolicy` to a variant named `myscalablevariant`. The policy configuration is saved in a file named `config.json`.

```
aws application-autoscaling put-scaling-policy \  
    --policy-name myscalablepolicy \  
    --policy-type TargetTrackingScaling \  
    --resource-id endpoint/MyEndpoint/variant/MyVariant \  
    --service-namespace sagemaker \  
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \  
    --target-tracking-scaling-policy-configuration file://config.json
```

Apply a Scaling Policy to a Production Variant (Application Auto Scaling API)

To apply a scaling policy to a variant with the Application Auto Scaling API, use the `PutScalingPolicy` Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to `sagemaker`.
- **ResourceID**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- **PolicyType**—Set this value to `TargetTrackingScaling`.
- **TargetTrackingScalingPolicyConfiguration**—The target-tracking scaling policy configuration to use for the variant.

Example

The following example uses Application Auto Scaling to apply a target-tracking scaling policy named `myscalablepolicy` to a variant named `myscalablevariant`. It uses a policy configuration based on the `SageMakerVariantInvocationsPerInstance` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 70.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
        }
    }
}
```

Edit a Scaling Policy

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

Edit a Scaling Policy (Console)

To edit a scaling policy with the AWS Management Console, use the same procedure that you used to [Configure Automatic Scaling for a Production Variant \(Console\) \(p. 716\)](#).

Edit a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- With the AWS CLI, specify the name of the policy that you want to edit in the `--policy-name` parameter. Specify new values for the parameters that you want to change.
- With the Application Auto Scaling API, specify the name of the policy that you want to edit in the `PolicyName` parameter. Specify new values for the parameters that you want to change.

For more information, see [Apply a Scaling Policy to a Production Variant \(p. 721\)](#).

Delete a Scaling Policy

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

Delete a Scaling Policy (Console)

To delete an automatic scaling policy for a variant (console)

- Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
- In the navigation pane, choose **Endpoints**.
- Choose the endpoint for which you want to delete automatic scaling.
- For **Endpoint runtime settings**, choose the variant that you want to configure.
- Choose **Configure auto scaling**.
- Choose **Deregister auto scaling**.

Delete a Scaling Policy (AWS CLI or Application Auto Scaling API)

You can use the AWS CLI or the Application Auto Scaling API to delete a scaling policy from a variant.

Delete a Scaling Policy (AWS CLI)

To delete a scaling policy from a variant, use the `delete-scaling-policy` AWS CLI command with the following parameters:

- `--policy-name`—The name of the scaling policy.
- `--resource-id`—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- `--service-namespace`—Set this value to `sagemaker`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.

Example

The following example deletes a target-tracking scaling policy named `mscalablepolicy` from a variant named `mscalablevariant`.

```
aws application-autoscaling delete-scaling-policy \
--policy-name mscalablepolicy \
--resource-id endpoint/MyEndpoint/variant/MyVariant \
--service-namespace sagemaker \
```

```
--scalable-dimension sagemaker:variant:DesiredInstanceCount
```

Delete a Scaling Policy (Application Auto Scaling API)

To delete a scaling policy from your variant, use the [DeleteScalingPolicy](#) Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to `sagemaker`.
- **ResourceId**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant,. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.

Example

The following example uses the Application Auto Scaling API to delete a target-tracking scaling policy named `mscalablepolicy` from a variant named `mscalablevariant`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "mscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```

Update Endpoints that Use Automatic Scaling

When you update Amazon SageMaker endpoints that have automatic scaling applied, complete the following steps:

To update an endpoint that has automatic scaling applied

1. Deregister the endpoint as a scalable target by calling [DeregisterScalableTarget](#).
2. Because you turn off automatic scaling before you update the endpoint, you might want to take the additional precaution of increasing the number of instances for your endpoint during the update. To do this, update the instance counts for the production variants hosted at the endpoint by calling [UpdateEndpointWeightsAndCapacities](#).
3. Call [DescribeEndpoint](#) repeatedly until the value of the `EndpointStatus` field of the response is `InService`.
4. Call [DescribeEndpointConfig](#) to get the values of the current endpoint config.
5. Create a new endpoint config by calling [CreateEndpointConfig](#). For the `InitialInstanceCount` field of each production variant, specify the corresponding value

- of `DesiredInstanceCount` from the response to the previous call to [DescribeEndpoint](#). For all other values, use the values that you got as the response when you called [DescribeEndpointConfig](#) in the previous step.
6. Update the endpoint by calling [UpdateEndpoint](#). Specify the endpoint config you created in the previous step as the `EndpointConfig` field.
 7. Re-enable automatic scaling by calling [RegisterScalableTarget](#).

Load Testing for Production Variant Automatic Scaling

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

For an example of load testing to optimize automatic scaling for a Amazon SageMaker endpoint, see [Load test and optimize an Amazon SageMaker endpoint using automatic scaling](#).

The following guidelines for load testing assume you are using an automatic scaling policy that uses the predefined target metric `SageMakerVariantInvocationsPerInstance`.

Topics

- [Determine the Performance Characteristics of a Production Variant \(p. 725\)](#)
- [Calculate the Target SageMakerVariantInvocationsPerInstance \(p. 725\)](#)

Determine the Performance Characteristics of a Production Variant

Perform load testing to find the peak `InvocationsPerInstance` that your variant instance can handle, and the latency of requests, as concurrency increases.

This value depends on the instance type chosen, payloads that clients of your variant typically send, and the performance of any external dependencies your variant has.

To find the peak requests-per-second (RPS) your variant can handle and latency of requests

1. Set up an endpoint with your variant using a single instance. For information about how to set up an endpoint, see [Step 6.1: Deploy the Model to Amazon SageMaker Hosting Services \(p. 50\)](#).
2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the output of the load testing tool.

Note

You can also monitor requests-per-minute instead of RPS. In that case don't multiply by 60 in the equation to calculate `SageMakerVariantInvocationsPerInstance` shown below.

When the model latency increases or the proportion of successful transactions decreases, this is the peak RPS that your variant can handle.

Calculate the Target SageMakerVariantInvocationsPerInstance

After you find the performance characteristics of the variant, you can determine the maximum RPS we should allow to be sent to an instance. The threshold used for scaling must be less than this maximum value. Use the following equation in combination with load testing to determine the correct value for the `SageMakerVariantInvocationsPerInstance` target metric in your automatic scaling configuration.

```
SageMakerVariantInvocationsPerInstance = (MAX_RPS * SAFETY_FACTOR) * 60
```

Where `MAX_RPS` is the maximum RPS that you determined previously, and `SAFETY_FACTOR` is the safety factor that you chose to ensure that your clients don't exceed the maximum RPS. Multiply by 60 to convert from RPS to invocations-per-minute to match the per-minute CloudWatch metric that Amazon SageMaker uses to implement automatic scaling (you don't need to do this if you measured requests-per-minute instead of requests-per-second).

Note

Amazon SageMaker recommends that you start testing with a `SAFETY_FACTOR` of 0.5. Test your automatic scaling configuration to ensure it operates in the way you expect with your model for both increasing and decreasing customer traffic on your endpoint.

Best Practices for Configuring Automatic Scaling

When configuring automatic scaling, consider the following general guidelines.

Test Your Automatic Scaling Configuration

It is important that you test your automatic scaling configuration to confirm that it works with your model the way you expect it to.

Update Endpoints Configured for Automatic Scaling

When you update an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If the update would change the instance type for any variant that is a target for automatic scaling, the update fails.

In the AWS Management Console, you see a warning that you must deregister the variant from automatic scaling before you can update it. If you are trying to update the endpoint by calling the [UpdateEndpoint](#) API, the call fails. Before you update the endpoint, delete any scaling policies configured for it by calling the [DeleteScalingPolicy](#) Application Auto Scaling API action, then call [DeregisterScalableTarget](#) to deregister the variant as a scalable target. After you update the endpoint, you can register the variant as a scalable target and attach an automatic scaling policy to the updated variant.

There is one exception. If you change the model for a variant that is configured for automatic scaling, Amazon SageMaker automatic scaling allows the update. This is because changing the model doesn't typically affect performance enough to change automatic scaling behavior. If you do update a model for a variant configured for automatic scaling, ensure that the change to the model doesn't significantly affect performance and automatic scaling behavior.

For instructions on how to update an endpoint that uses automatic scaling, see [Update Endpoints that Use Automatic Scaling \(p. 724\)](#).

Delete Endpoints Configured for Automatic Scaling

If you delete an endpoint, Application Auto Scaling checks to see whether any of the variants on that endpoint are targets for automatic scaling. If any are and you have permission to deregister the variant, Application Auto Scaling deregisters those variants as scalable targets without notifying you. If you use a custom permission policy that doesn't provide permission for the [DeleteScalingPolicy](#) and [DeregisterScalableTarget](#) actions, you must delete automatic scaling policies and deregister scalable targets and before deleting the endpoint.

Note

You, as an IAM user, might not have sufficient permission to delete an endpoint if another IAM user configured automatic scaling for a variant on that endpoint.

Using Step Scaling Policies

Although Amazon SageMaker automatic scaling supports using Application Auto Scaling step scaling policies, we recommend using target tracking policies, instead. For information about using Application Auto Scaling step scaling policies, see [Step Scaling Policies](#).

Scaling In When There Is No Traffic

If a variant's traffic becomes zero, Amazon SageMaker automatic scaling doesn't scale in. This is because Amazon SageMaker doesn't emit metrics with a value of zero.

As a workaround, do either of the following:

- Send requests to the variant until automatic scaling scales in to the minimum capacity
- Change the policy to reduce the maximum provisioned capacity to match the minimum provisioned capacity

Troubleshoot Amazon SageMaker Model Deployments

If you encounter an issue when deploying machine learning models in Amazon SageMaker, see the following guidance.

Topics

- [Detection Errors in the Active CPU Count \(p. 727\)](#)

Detection Errors in the Active CPU Count

If you deploy an Amazon SageMaker model with a Linux Java Virtual Machine (JVM), you might encounter detection errors that prevent using available CPU resources. This issue affects some JVMs that support Java 8 and Java 9, and most that support Java 10 and Java 11. These JVMs implement a mechanism that detects and handles the CPU count and the maximum memory available when running a model in a Docker container, and, more generally, within Linux taskset commands or control groups (cgroups). Amazon SageMaker deployments take advantage of some of the settings that the JVM uses for managing these resources. Currently, this causes the container to incorrectly detect the number of available CPUs.

Amazon SageMaker doesn't limit access to CPUs on an instance. However, the JVM might detect the CPU count as 1 when more CPUs are available for the container. As a result, the JVM adjusts all of its internal settings to run as if only 1 CPU core is available. These settings affect garbage collection, locks, compiler threads, and other JVM internals that negatively affect the concurrency, throughput, and latency of the container.

For an example of the misdetection, in a container configured for Amazon SageMaker that is deployed with a JVM that is based on Java8_191 and that has four available CPUs on the instance, run the following command to start your JVM:

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
```

```
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Many of the JVMs affected by this issue have an option to disable this behavior and reestablish full access to all of the CPUs on the instance. Disable the unwanted behaviour and establish full access to all instance CPUs by including the `-XX:-UseContainerSupport` parameter when starting Java applications. For example, run the `java` command to start your JVM as follows:

```
java -XX:-UseContainerSupport -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Check whether the JVM used in your container supports the `-XX:-UseContainerSupport` parameter. If it does, always pass the parameter when you start your JVM. This provides access to all of the CPUs in your instances.

You might also encounter this issue when indirectly using a JVM in Amazon SageMaker containers. For example, when using a JVM to support SparkML Scala. The `-XX:-UseContainerSupport` parameter also affects the output returned by the `Java Runtime.getRuntime().availableProcessors()` API .

Deployment Best Practices

This topic provides guidance on best practices for deploying machine learning models in Amazon SageMaker.

Topics

- [Deploy Multiple Instances Across Availability Zones \(p. 728\)](#)

Deploy Multiple Instances Across Availability Zones

Create robust endpoints when hosting your model. Amazon SageMaker endpoints can help protect your application from [Availability Zone](#) outages and instance failures. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones. For this reason, we strongly recommended that you deploy multiple instances for each production endpoint.

If you are using an [Amazon Virtual Private Cloud \(VPC\)](#), configure the VPC with at least two [Subnets](#), each in a different Availability Zone. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones.

In general, to achieve more reliable performance, use more small [Instance Types](#) in different Availability Zones to host your endpoints.

Host Instance Storage Volumes

When you create an endpoint, Amazon SageMaker attaches an Amazon Elastic Block Store (Amazon EBS) storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type.

The following table shows the size of the storage volume that Amazon SageMaker attaches for each instance type for a single endpoint and for a multi-model endpoint. For a MultiModel-enabled container, the storage volume provisioned for its instances has more memory. This allows more models to be cached on the instance storage volume. MultiModel containers with GPU instance types (for example, P2, P3, and G4 instance families) aren't supported on multi-model endpoints.

Note

Because d instance types come with an NVMe SSD storage, Amazon SageMaker doesn't attach an Amazon EBS storage volume to these ML compute instances that host the multi-model endpoint.

I	R	Storage Volume for Multi- Model Endpoint in GB
ml29	medium	128
ml42	large	216
ml82	xlarge	522
ml162	2xlarge	1042
ml324	3xlarge	1534
ml644	4xlarge	3064
ml1284	8xlarge	6128
ml32010	10xlarge	122010
ml3216	16xlarge	243216
ml416	large	216
ml835	xlarge	522
ml1652	2xlarge	10452
ml3284	4xlarge	20848
ml328412	12xlarge	41684
ml328324	32xlarge	83284

Storage Type	Storage Volume for Multi-Model Endpoint in GB
ml474large	474
ml415xlarge	415
ml8302xlarge	830
ml1604xlarge	160
ml3028xlarge	3028
ml285.large	285
ml415xlarge	415
ml8322xlarge	832
ml1624xlarge	1624
ml3049xlarge	3049
ml3088xlarge	3088
ml3011xlarge supported	3011
ml3008xlarge supported	3008
ml3016xlarge supported	3016
ml3022xlarge supported	3022
ml3018xlarge supported	3018
ml3016xlarge supported	3016
ml832large	832
ml164xlarge	164
ml3028xlarge	3028
ml3054xlarge	3054
ml30522xlarge	30522
ml30524xlarge	30524

Monitor Amazon SageMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon SageMaker and your other AWS solutions. AWS provides the following monitoring tools to watch Amazon SageMaker, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from EC2 instances, AWS CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [AWS CloudTrail User Guide](#).
- *CloudWatch Events* delivers a near real-time stream of system events that describe changes in AWS resources. Create CloudWatch Events rules react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job

Topics

- [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#)
- [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#)
- [Log Amazon SageMaker API Calls with AWS CloudTrail \(p. 739\)](#)
- [React to Amazon SageMaker Job Status Changes with CloudWatch Events \(p. 742\)](#)

Monitor Amazon SageMaker with Amazon CloudWatch

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. However, the Amazon CloudWatch console limits the search to metrics that were updated in the last 2 weeks. This limitation ensures that the most current jobs are shown in your namespace. To graph metrics without using a search, specify its exact name in the source view. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

Amazon SageMaker model training jobs and endpoints write CloudWatch metrics and logs. The following tables list the metrics and dimensions for Amazon SageMaker.

Endpoint Invocation Metrics

The `aws/sagemaker` namespace includes the following request metrics from calls to [InvokeEndpoint](#).

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the *Amazon CloudWatch API Reference*.

Metric	Description
<code>Invocation4XXErrors</code>	The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum
<code>Invocation5XXErrors</code>	The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent. Units: None Valid statistics: Average, Sum
<code>Invocations</code>	The number of <code>InvokeEndpoint</code> requests sent to a model endpoint. To get the total number of requests sent to a model endpoint, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
<code>InvocationsPerInstance</code>	The number of invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code> . $1/\text{numberOfInstances}$ is sent as the value on each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> behind the endpoint at the time of the request. Units: None Valid statistics: Sum
<code>ModelLatency</code>	The interval of time taken by a model to respond as viewed from Amazon SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
<code>OverheadLatency</code>	The interval of time added to the time taken to respond to a client request by Amazon SageMaker overheads. This interval is measured from the time Amazon SageMaker receives the request until it returns a response to the client, minus the <code>ModelLatency</code> . Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request. Units: Microseconds

Metric	Description
	Valid statistics: Average, Sum, Min, Max, Sample Count

Dimensions for Endpoint Invocation Metrics

Dimension	Description
EndpointName , VariantName	Filters endpoint invocation metrics for a <code>ProductionVariant</code> of the specified endpoint and variant.

Multi-Model Endpoint Model Loading Metrics

The `aws/SageMaker` namespace includes the following model loading metrics from calls to `InvokeEndpoint`.

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the *Amazon CloudWatch API Reference*.

Metric	Description
<code>ModelLoadingWaitTime</code>	The interval of time that an invocation request has waited for the target model to be downloaded, or loaded, or both in order to perform inference. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
<code>ModelUnloadingTime</code>	The interval of time that it took to unload the model through the container's <code>UnloadModel</code> API call. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
<code>ModelDownloadingTime</code>	The interval of time that it took to download the model from Amazon Simple Storage Service (Amazon S3). Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
<code>ModelLoadingTime</code>	The interval of time that it took to load the model through the container's <code>LoadModel</code> API call. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count
<code>ModelCacheHit</code>	The number of <code>InvokeEndpoint</code> requests sent to the multi-model endpoint for which the model was already loaded. The Average statistic shows the ratio of requests for which the model was already loaded. Units: None

Metric	Description
	Valid statistics: Average, Sum, Sample Count

Dimensions for Multi-Model Endpoint Model Loading Metrics

Dimension	Description
EndpointName , VariantName	Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.

Multi-Model Endpoint Model Instance Metrics

The `/aws/sagemaker/Endpoints` namespaces include the following instance metrics from calls to [InvokeEndpoint](#).

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the *Amazon CloudWatch API Reference*.

Metric	Description
LoadedModelCount	<p>The number of models loaded in the containers of the multi-model endpoint. This metric is emitted per instance.</p> <p>The Average statistic with a period of 1 minute tells you the average number of models loaded per instance.</p> <p>The Sum statistic tells you the total number of models loaded across all instances in the endpoint.</p> <p>The models that this metric tracks are not necessarily unique because a model might be loaded in multiple containers at the endpoint.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>

Dimensions for Multi-Model Endpoint Model Loading Metrics

Dimension	Description
EndpointName , VariantName	Filters endpoint invocation metrics for a ProductionVariant of the specified endpoint and variant.

Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics

The `/aws/sagemaker/ProcessingJobs`, `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs` and `/aws/sagemaker/Endpoints` namespaces include the following metrics for the training jobs and endpoint instances.

Metrics are available at a 1-minute frequency.

Metric	Description
CPUUtilization	<p>The percentage of CPU units that are used by the containers on an instance. The value can range between 0 and 100, and is multiplied by the number of CPUs. For example, if there are four CPUs, <code>CPUUtilization</code> can range from 0% to 400%.</p> <p>For processing jobs, the value is the CPU utilization of the processing container on the instance.</p> <p>For training jobs, the value is the CPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the CPU utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance.</p> <p>Note For multi-instance, each instance reports CPU utilization metrics. However, the default view in CloudWatch shows the average CPU utilization across all instances.</p> <p>Units: Percent</p>
MemoryUtilization	<p>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%.</p> <p>For processing jobs, the value is the memory utilization of the processing container on the instance.</p> <p>For training jobs, the value is the memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the memory utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p> <p>Note For multi-instance, each instance reports memory utilization metrics. However, the default view in CloudWatch shows the average memory utilization across all instances.</p>
GPUUtilization	<p>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, <code>GPUUtilization</code> can range from 0% to 400%.</p> <p>For processing jobs, the value is the GPU utilization of the processing container on the instance.</p> <p>For training jobs, the value is the GPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU utilization of the transform container on the instance.</p>

Metric	Description
	<p>For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance.</p> <p>Note For multi-instance, each instance reports GPU utilization metrics. However, the default view in CloudWatch shows the average GPU utilization across all instances.</p> <p>Units: Percent</p>
GPUMemoryUtilization	<p>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%.</p> <p>For processing jobs, the value is the GPU memory utilization of the processing container on the instance.</p> <p>For training jobs, the value is the GPU memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance.</p> <p>Note For multi-instance, each instance reports GPU memory utilization metrics. However, the default view in CloudWatch shows the average GPU memory utilization across all instances.</p> <p>Units: Percent</p>
DiskUtilization	<p>The percentage of disk space used by the containers on an instance uses. This value can range between 0% and 100%. This metric is not supported for batch transform jobs.</p> <p>For processing jobs, the value is the disk space utilization of the processing container on the instance.</p> <p>For training jobs, the value is the disk space utilization of the algorithm container on the instance.</p> <p>For endpoint variants, the value is the sum of the disk space utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p> <p>Note For multi-instance, each instance reports disk utilization metrics. However, the default view in CloudWatch shows the average disk utilization across all instances.</p>

Dimensions for Processing Job, Training Job and Batch Transform Job Instance Metrics

Dimension	Description
Host	<p>For processing jobs, the value for this dimension has the format [processing-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified processing job and instance. This dimension format is present only in the /aws/sagemaker/ProcessingJobs namespace.</p> <p>For training jobs, the value for this dimension has the format [training-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the /aws/sagemaker/TrainingJobs namespace.</p> <p>For batch transform jobs, the value for this dimension has the format [transform-job-name]/[instance-id]. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the /aws/sagemaker/TransformJobs namespace.</p>

Amazon SageMaker Ground Truth Metrics

Metric	Description
ActiveWorkers	<p>The number of workers on a private work team performing a labeling job.</p> <p>Units: None</p> <p>Valid statistics: Max</p>
DatasetObjectsAutoAnnotated	<p>The number of dataset objects auto-annotated in a labeling job. This metric is only emitted when automated labeling is enabled. To view the labeling job progress, use the Max metric.</p> <p>Units: None</p> <p>Valid statistics: Max</p>
DatasetObjectsHumanAnnotated	<p>The number of dataset objects annotated by a human in a labeling job. To view the labeling job progress, use the Max metric.</p> <p>Units: None</p> <p>Valid statistics: Max</p>
DatasetObjectsLabelingFailed	<p>The number of dataset objects that failed labeling in a labeling job. To view the labeling job progress, use the Max metric.</p> <p>Units: None</p> <p>Valid statistics: Max</p>
JobsFailed	<p>The number of labeling jobs that failed. To get the total number of labeling jobs that failed, use the Sum statistic.</p> <p>Units: None</p> <p>Valid statistics: Sum, Sample Count</p>

Metric	Description
JobsSucceeded	The number of labeling jobs that succeeded. To get the total number of labeling jobs that succeeded, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
JobsStopped	The number of labeling jobs that were stopped. To get the total number of labeling jobs that were stopped, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
TasksSubmitted	The number of tasks submitted/completed by a private work team. Units: None Valid statistics: Max
TimeSpent	Time spent on a task completed by a private work team. Units: Seconds Valid statistics: Max
TotalDatasetObjects	The number of dataset objects labeled successfully in a labeling job. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max

Dimensions for Dataset Object Metrics

Dimension	Description
LabelingJobName	Filters dataset object count metrics for a labeling job.

Log Amazon SageMaker Events with Amazon CloudWatch

To help you debug your processing jobs, training jobs, endpoints, transform jobs, notebook instances, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.

Logs

The following table lists all of the logs provided by Amazon SageMaker.

Logs

Log Group Name	Log Stream Name
/aws/sagemaker/ ProcessingJobs	[processing-job-name]/[hostname]-[epoch_timestamp]
/aws/sagemaker/ TrainingJobs	[training-job-name]/algo-[instance-number-in-cluster]- [epoch_timestamp]
/aws/sagemaker/ Endpoints/ [EndpointName]	[production-variant-name]/[instance-id] [production-variant-name]/[instance-id]/[container-name provided in SageMaker model] (For Inference Pipelines)
/aws/sagemaker/ NotebookInstances	[notebook-instance-name]/[LifecycleConfigHook] [notebook-instance-name]/jupyter.log
/aws/sagemaker/ TransformJobs	[transform-job-name]/[instance-id]-[epoch_timestamp] [transform-job-name]/[instance-id]-[epoch_timestamp]/data- log [transform-job-name]/[instance-id]-[epoch_timestamp]/ [container-name provided in SageMaker model] (For Inference Pipelines)
/aws/sagemaker/ LabelingJobs	[labeling-job-name]
/aws/sagemaker/ groundtruth/ WorkerActivity	aws/sagemaker/groundtruth/worker-activity/[requester-AWS- Id]-[region]/[timestamp]

Note

1. The /aws/sagemaker/NotebookInstances/[LifecycleConfigHook] log stream is created when you create a notebook instance with a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 256\)](#).
2. For Inference Pipelines, if you don't provide container names, the platform uses **container-1, container-2**, and so on, corresponding to the order provided in the Amazon SageMaker model.

For more information about logging events with CloudWatch logging, see [What is Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch User Guide*.

Log Amazon SageMaker API Calls with AWS CloudTrail

Amazon SageMaker is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. CloudTrail captures all API calls for Amazon SageMaker, with the exception of [InvokeEndpoint](#), as events. The calls captured include calls from the Amazon SageMaker console and code calls to the Amazon SageMaker API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon SageMaker. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in [Event history](#). Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

By default, log data is stored in CloudWatch Logs indefinitely. However, you can configure how long to store log data in a log group. For information, see [Change Log Data Retention in CloudWatch Logs](#) in the [Amazon CloudWatch Logs User Guide](#).

Amazon SageMaker Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon SageMaker, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for Amazon SageMaker, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All Amazon SageMaker actions, with the exception of [InvokeEndpoint](#), are logged by CloudTrail and are documented in the [Operations](#). For example, calls to the `CreateTrainingJob`, `CreateEndpoint` and `CreateNotebookInstance` actions generate entries in the CloudTrail log files.

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

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Operations Performed by Automatic Model Tuning

Amazon SageMaker supports logging non-API service events to your CloudTrail log files, for automatic model tuning jobs. These events are related to your tuning jobs but, are not the direct result of a customer request to the public AWS API. For example, when you create a hyperparameter tuning job by calling [CreateHyperParameterTuningJob](#), Amazon SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call [StopHyperParameterTuningJob](#) to stop a hyperparameter tuning job, Amazon SageMaker might stop any of the associated running training jobs. Non-API events for your tuning jobs are logged to CloudTrail to help you improve governance, compliance, and operational and risk auditing of your AWS account.

Log entries that result from non-API service events have an `eventType` of `AwsServiceEvent` instead of `AwsApiCall`.

Understanding Amazon SageMaker Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following examples a log entry for the `CreateEndpoint` action, which creates an endpoint to deploy a trained model.

```
{  
    "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",  
    "sourceIPAddress": "127.0.0.1",  
    "userAgent": "USER_AGENT",  
    "requestParameters": {  
        "endpointName": "ExampleEndpoint",  
        "endpointConfigName": "ExampleEndpointConfig"  
    },  
    "responseElements": {  
        "endpointArn": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/exampleendpoint"  
    },  
    "requestID": "6b1b42b9-EXAMPLE",  
    "eventID": "a6f85b21-EXAMPLE",  
    "eventType": "AwsApiCall",  
    "recipientAccountId": "444455556666"  
}
```

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

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",  
        "arn": "arn:aws:iam::123456789012:user/intern",  
        "accountId": "123456789012",  
        "accessKeyId": "ASXIAGXEXAMPLEQULKNXV",  
        "userName": "intern"  
    },  
    "eventTime": "2018-01-02T15:23:46Z",  
    "eventSource": "sagemaker.amazonaws.com",  
    "eventName": "CreateModel",  
    "awsRegion": "us-west-2",  
    "sourceIPAddress": "127.0.0.1",  
    "userAgent": "USER_AGENT",  
    "requestParameters": {  
        "modelName": "ExampleModel",  
        "primaryContainer": {  
            "image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/examplemodel:  
                latest",  
            "containerDefinition": {  
                "cpu": 100,  
                "memory": 512,  
                "essential": true,  
                "portMappings": [  
                    {  
                        "hostPort": 8080,  
                        "containerPort": 8080  
                    }  
                ]  
            }  
        }  
    }  
}
```

```
        "image":"174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest"
    },
    "executionRoleArn":"arn:aws:iam::123456789012:role/EXAMPLEARN"
},
"responseElements": {
    "modelArn":"arn:aws:sagemaker:us-west-2:123456789012:model/
barkinghappy2018-01-02t15-23-32-275z-ivrdog"
},
"requestID":"417b8dab-EXAMPLE",
"eventID":"0f2b3e81-EXAMPLE",
"eventType":"AwsApiCall",
"recipientAccountId":"444455556666"
}
```

React to Amazon SageMaker Job Status Changes with CloudWatch Events

To react to a status change in a Amazon SageMaker training, hyperparameter tuning, or batch transform job, create a rule in CloudWatch Events that use the **SageMaker Training Job State Change**, **SageMaker Hyperparameter Tuning Job State Change**, or **SageMaker Transform Job State Change** event type as the event source for the rule.

Every time the status of a Amazon SageMaker job changes, it triggers an event that CloudWatch Events monitors, and you can create a rule that calls a AWS Lambda function when the status changes. For information about the status values and meanings for Amazon SageMaker jobs, see the following:

- [TrainingJobStatus](#)
- [HyperParameterTuningJobStatus](#)
- [TransformJobStatus](#)

For information about creating CloudWatch Events rules, see [Creating a CloudWatch Events Rule That Triggers on an Event](#) in the *CloudWatch Events User Guide*. For detailed information about the format of the Amazon SageMaker events that CloudWatch Events monitors, see [Amazon SageMaker Events](#).

Security in Amazon SageMaker

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon SageMaker, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon SageMaker. The following topics show you how to configure Amazon SageMaker to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon SageMaker resources.

Topics

- [Data Protection in Amazon SageMaker \(p. 743\)](#)
- [Identity and Access Management for Amazon SageMaker \(p. 747\)](#)
- [Logging and Monitoring \(p. 797\)](#)
- [Compliance Validation for Amazon SageMaker \(p. 797\)](#)
- [Resilience in Amazon SageMaker \(p. 798\)](#)
- [Infrastructure Security in Amazon SageMaker \(p. 798\)](#)

Data Protection in Amazon SageMaker

Amazon SageMaker conforms to the AWS [shared responsibility model](#), which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon SageMaker or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon SageMaker or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the [AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog](#).

Topics

- [Protect Data at Rest Using Encryption \(p. 744\)](#)
- [Protecting Data in Transit with Encryption \(p. 745\)](#)
- [Key Management \(p. 747\)](#)
- [Internetwork Traffic Privacy \(p. 747\)](#)

Protect Data at Rest Using Encryption

To protect your model-building data and model artifacts, you can use encrypted Amazon Simple Storage Service (Amazon S3) buckets. To encrypt the machine learning (ML) storage volume that is attached to notebooks, processing jobs, training jobs, hyperparameter tuning jobs, batch transform jobs, and endpoints, you can pass a AWS Key Management Service (AWS KMS) key to Amazon SageMaker . If you don't specify a KMS key, Amazon SageMaker encrypts storage volumes with a transient key and discards it immediately after encrypting the storage volume.

You can use an AWS managed KMS key to encrypt all instance OS volumes. You can encrypt all ML data volumes for all Amazon SageMaker instances with a KMS key that you specify. ML storage volumes are mounted as follows:

- Notebooks - /home/ec2-user/SageMaker
- Processing - /opt/ml/processing and /tmp/
- Training - /opt/ml/ and /tmp/
- Batch - /opt/ml/ and /tmp/
- Endpoints - /opt/ml/ and /tmp/

Processing, batch transform, and training job containers and their storage are ephemeral in nature. When the job completes, output is uploaded to Amazon S3 using AWS KMS encryption (with an optional KMS key you specify) and the instance is torn down.

Important

Sensitive data that needs to be encrypted with a KMS key for compliance reasons should be stored in the ML storage volume or in Amazon S3, both of which can be encrypted using a KMS key you specify.

When you open a notebook instance, Amazon SageMaker saves it and any files associated with it in the Amazon SageMaker folder in the ML storage volume by default. When you stop a notebook instance, Amazon SageMaker creates a snapshot of the ML storage volume. Any customizations to the operating system of the stopped instance, such as installed custom libraries or operating system level settings, are lost. Consider using a lifecycle configuration to automate customizations of the default notebook instance. When you terminate an instance, the snapshot and the ML storage volume are deleted. Any

data that you need to persist beyond the lifespan of the notebook instance should be transferred to an Amazon S3 bucket.

Note

Certain Nitro-based Amazon SageMaker instances include local storage, depending on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't use a KMS key on an instance type with local storage. For a list of instance types that support local instance storage, see [Instance Store Volumes](#). For more information about storage volumes on Nitro-based instances, see [Amazon EBS and NVMe on Linux Instances](#). For more information about local instance storage encryption, see [SSD Instance Store Volumes](#).

Protecting Data in Transit with Encryption

All inter-network data in transit supports TLS 1.2 encryption.

Amazon SageMaker ensures that machine learning (ML) model artifacts and other system artifacts are encrypted in transit and at rest. Requests to the Amazon SageMaker API and console are made over a secure (SSL) connection. You pass AWS Identity and Access Management roles to Amazon SageMaker to provide permissions to access resources on your behalf for training and deployment. You can use encrypted Amazon S3 buckets for model artifacts and data, as well as pass a AWS KMS key to Amazon SageMaker instances to encrypt the attached ML storage volumes.

Some intra-network data in-transit (inside the service platform) is unencrypted. This includes:

- Command and control communications between the service control plane and training job instances (not customer data).
- Communications between nodes in distributed processing jobs (intra-network).
- Communications between nodes in distributed training jobs (intra-network).

There are no inter-node communications for batch processing.

You can choose to encrypt internode training communications. Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. For affected algorithms, adding this additional level of security also increases cost. The training time for most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

FIPS validated endpoints are available for the Amazon SageMaker API and request router for hosted models (runtime). For information about FIPS compliant endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

Protect Communications Between ML Compute Instances in a Distributed Training Job

By default, Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud (Amazon VPC) to help keep your data secure. You can add another level of security to protect your training containers and data by configuring a *private* VPC. Distributed ML frameworks and algorithms usually transmit information that is directly related to the model such as weights, not the training dataset. When performing distributed training, you can further protect data that is transmitted between instances. This can help you to comply with regulatory requirements. To do this, use inter-container traffic encryption.

Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. Enabling inter-container traffic encryption doesn't affect training jobs with a single compute instance. However, for training jobs with several compute instances, the effect on training time depends on the amount of communication between compute instances. For affected algorithms, adding this additional level of security also increases cost. The training time for

most Amazon SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

You can enable inter-container traffic encryption for training jobs or hyperparameter tuning jobs. You can use Amazon SageMaker APIs or console to enable inter-container traffic encryption.

For information about running training jobs in a private VPC, see [Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 808\)](#).

Enable Inter-Container Traffic Encryption (API)

Before enabling inter-container traffic encryption on training or hyperparameter tuning jobs with APIs, you need to add inbound and outbound rules to your private VPC's security group.

To enable inter-container traffic encryption (API)

1. Add the following inbound and outbound rules in the security group for your private VPC:

Protocol	Port Range	Source
UDP	500	<i>Self Security Group ID</i>
50	N/A	<i>Self Security Group ID</i>

2. When you send a request to the [CreateTrainingJob](#) or [CreateHyperParameterTuningJob](#) API, specify `True` for the `EnableInterContainerTrafficEncryption` parameter.

Note

The AWS Security Group Console might show display ports range as "All", however EC2 ignores the specified port range because it is not applicable for the ESP 50 IP protocol.

Enable Inter-Container Traffic Encryption (Console)

Enable Inter-container Traffic Encryption in a Training Job

To enable inter-container traffic encryption in a training job

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you have created.
5. Choose **Enable inter-container traffic encryption**.

After you enable inter-container traffic encryption, finish creating the training job. For more information, see [Step 5: Train a Model \(p. 45\)](#).

Enable Inter-container Traffic Encryption in a Hyperparameter Tuning Job

To enable inter-container traffic encryption in a hyperparameter tuning job

1. Open the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker>
2. In the navigation pane, choose **Training**, then choose **Hyperparameter tuning jobs**.
3. Choose **Create hyperparameter tuning job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you created.
5. Choose **Enable inter-container traffic encryption**.

After enabling inter-container traffic encryption, finish creating the hyperparameter tuning job. For more information, see [Configure and Launch a Hyperparameter Tuning Job \(p. 583\)](#).

Key Management

Customers can specify AWS KMS keys, including bring your own keys (BYOK), to use for envelope encryption with Amazon S3 input/output buckets and machine learning (ML) Amazon EBS volumes. ML volumes for notebook instances and for processing, training, and hosted model Docker containers can be optionally encrypted by using AWS KMS customer-owned keys. All instance OS volumes are encrypted with an AWS-managed AWS KMS key.

Note

Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a `VolumeKmsKeyId` when using an instance type with local storage.

For a list of instance types that support local instance storage, see [Instance Store Volumes](#).

For more information about local instance storage encryption, see [SSD Instance Store Volumes](#).

For more information about storage volumes on nitro-based instances, see [Amazon EBS and NVMe on Linux Instances](#).

For information about AWS KMS keys see [What is AWS Key Management Service?](#) in the [AWS Key Management Service Developer Guide](#).

Internet Traffic Privacy

This topic describes how Amazon SageMaker secures connections from the service to other locations.

Internet communications support TLS 1.2 encryption between all components and clients.

Instances can be connected to Customer VPC, providing access to S3 VPC endpoints or customer repositories. Internet egress can be managed through this interface by the customer if service platform internet egress is disabled for notebooks. For training and hosting, egress through the service platform is not available when connected to the customer's VPC.

By default, API calls made to published endpoints traverse the public network to the request router. Amazon SageMaker supports Amazon Virtual Private Cloud interface endpoints powered by AWS PrivateLink for private connectivity between the customer's VPC and the request router to access hosted model endpoints. For information about Amazon VPC, see [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 800\)](#)

Identity and Access Management for Amazon SageMaker

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon SageMaker resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience \(p. 748\)](#)
- [Authenticating with Identities \(p. 748\)](#)
- [Managing Access Using Policies \(p. 750\)](#)
- [How Amazon SageMaker Works with IAM \(p. 751\)](#)
- [Amazon SageMaker Identity-Based Policy Examples \(p. 754\)](#)

- [Amazon SageMaker Roles \(p. 776\)](#)
- [AWS Managed \(Predefined\) Policies for Amazon SageMaker \(p. 790\)](#)
- [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 790\)](#)
- [Troubleshooting Amazon SageMaker Identity and Access \(p. 795\)](#)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in Amazon SageMaker.

Service user – If you use the Amazon SageMaker service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon SageMaker features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in Amazon SageMaker, see [Troubleshooting Amazon SageMaker Identity and Access \(p. 795\)](#).

Service administrator – If you're in charge of Amazon SageMaker resources at your company, you probably have full access to Amazon SageMaker. It's your job to determine which Amazon SageMaker features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with Amazon SageMaker, see [How Amazon SageMaker Works with IAM \(p. 751\)](#).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon SageMaker. To view example Amazon SageMaker identity-based policies that you can use in IAM, see [Amazon SageMaker Identity-Based Policy Examples \(p. 754\)](#).

Authenticating with Identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see [The IAM Console and Sign-in Page](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [AWS Management Console](#), use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using *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*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.

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 Users and Groups

An [IAM user](#) is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing Access Keys for IAM Users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An [IAM group](#) is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to Create an IAM User \(Instead of a Role\)](#) in the *IAM User Guide*.

IAM Roles

An [IAM role](#) is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by [switching roles](#). You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM Roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing 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*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see [Creating a Role to Delegate Permissions to an AWS Service](#) in the *IAM User Guide*.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests.

This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles, see [When to Create an IAM Role \(Instead of a User\)](#) in the *IAM User Guide*.

Managing Access Using Policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON Policies](#) in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM Policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing Between Managed Policies and Inline Policies](#) in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

Access Control Lists (ACLs)

Access control lists (ACLs) are a type of policy that controls which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format. Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access Control List \(ACL\) Overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions Boundaries for IAM Entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see [How SCPs Work](#) in the *AWS Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session Policies](#) in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy Evaluation Logic](#) in the *IAM User Guide*.

How Amazon SageMaker Works with IAM

Before you use IAM to manage access to Amazon SageMaker, you should understand what IAM features are available to use with Amazon SageMaker. To get a high-level view of how Amazon SageMaker and other AWS services work with IAM, see [AWS Services That Work with IAM](#) in the *IAM User Guide*.

Topics

- [Amazon SageMaker Identity-Based Policies \(p. 751\)](#)

Amazon SageMaker Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. Amazon SageMaker supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

Actions

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in Amazon SageMaker use the following prefix before the action: `sagemaker:`. For example, to grant someone permission to run an Amazon SageMaker training job with the Amazon

SageMaker `CreateTrainingJob` API operation, you include the `sagemaker:CreateTrainingJob` action in their policy. Policy statements must include either an `Action` or `NotAction` element. Amazon SageMaker defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "sagemaker:action1",  
    "sagemaker:action2"]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "sagemaker:Describe*"
```

To see a list of Amazon SageMaker actions, see [Actions Defined by Amazon SageMaker](#) in the *IAM User Guide*.

Resources

Amazon SageMaker does not support specifying resource ARNs in a policy.

Condition Keys

The `Condition` element (or `Condition block`) lets you specify conditions in which a statement is in effect. The `Condition` element is optional. You can build conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple `Condition` elements in a statement, or multiple keys in a single `Condition` element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM Policy Elements: Variables and Tags](#) in the *IAM User Guide*.

Amazon SageMaker defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *IAM User Guide*.

Amazon SageMaker supports a number of service-specific condition keys that you can use for fine-grained access control for the following operations:

- [CreateProcessingJob](#)
- [CreateTrainingJob](#)
- [CreateModel](#)
- [CreateEndpointConfig](#)
- [CreateTransformJob](#)
- [CreateHyperParameterTuningJob](#)
- [CreateLabelingJob](#)
- [CreateNotebookInstance](#)
- [UpdateNotebookInstance](#)

To see a list of Amazon SageMaker condition keys, see [Condition Keys for Amazon SageMaker](#) in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see [Actions Defined by Amazon SageMaker](#).

For examples of using Amazon SageMaker condition keys, see the following: [Control Creation of Amazon SageMaker Resources with Condition Keys \(p. 764\)](#).

Examples

To view examples of Amazon SageMaker identity-based policies, see [Amazon SageMaker Identity-Based Policy Examples \(p. 754\)](#).

Amazon SageMaker Resource-Based Policies

Amazon SageMaker does not support resource-based policies.

Authorization Based on Amazon SageMaker Tags

You can attach tags to Amazon SageMaker resources or pass tags in a request to Amazon SageMaker. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `sagemaker:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging Amazon SageMaker resources, see [Control Access to Amazon SageMaker Resources by Using Tags \(p. 773\)](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Control Access to Amazon SageMaker Resources by Using Tags \(p. 773\)](#).

Amazon SageMaker IAM Roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Using Temporary Credentials with Amazon SageMaker

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as [AssumeRole](#) or [GetFederationToken](#).

Amazon SageMaker supports using temporary credentials.

Service-Linked Roles

Amazon SageMaker doesn't support service-linked roles.

Service Roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

Amazon SageMaker supports service roles.

Choosing an IAM Role in Amazon SageMaker

When you create a notebook instance, processing job, training job, hosted endpoint, or batch transform job resource in Amazon SageMaker, you must choose a role to allow Amazon SageMaker to access Amazon SageMaker on your behalf. If you have previously created a service role or service-linked role, then Amazon SageMaker provides you with a list of roles to choose from. It's important to choose a role that allows access to the AWS operations and resources you need. For more information, see [Amazon SageMaker Roles \(p. 776\)](#).

Amazon SageMaker Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify Amazon SageMaker resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions. To learn how to attach policies to an IAM user or group, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

Topics

- [Policy Best Practices \(p. 754\)](#)
- [Using the Amazon SageMaker Console \(p. 754\)](#)
- [Allow Users to View Their Own Permissions \(p. 763\)](#)
- [Control Creation of Amazon SageMaker Resources with Condition Keys \(p. 764\)](#)
- [Control Access to the Amazon SageMaker API by Using Identity-based Policies \(p. 771\)](#)
- [Limit Access to Amazon SageMaker API and Runtime Calls by IP Address \(p. 772\)](#)
- [Limit Access to a Notebook Instance by IP Address \(p. 773\)](#)
- [Control Access to Amazon SageMaker Resources by Using Tags \(p. 773\)](#)
- [Require the Presence or Absence of Tags for API Calls \(p. 775\)](#)
- [Use Tags with Hyperparameter Tuning Jobs \(p. 776\)](#)

Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete Amazon SageMaker resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using Amazon SageMaker quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get Started Using Permissions With AWS Managed Policies](#) in the *IAM User Guide*.
- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant Least Privilege](#) in the *IAM User Guide*.
- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using Multi-Factor Authentication \(MFA\) in AWS](#) in the *IAM User Guide*.
- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

Using the Amazon SageMaker Console

To access the Amazon SageMaker console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon SageMaker resources in your

AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the Amazon SageMaker console, also attach the following AWS managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*:

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

Topics

- [Permissions Required to Use the Amazon SageMaker Console \(p. 755\)](#)
- [Permissions Required to Use the Amazon SageMaker Ground Truth Console \(p. 756\)](#)
- [Permissions Required to Use the Amazon Augmented AI \(Preview\) Console \(p. 757\)](#)

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 \(p. 790\)](#).

To use the Amazon SageMaker console, you need to grant permissions for additional actions. Specifically, the console needs permissions that allow the ec2 actions to display subnets, VPCs, and security groups. Optionally, the console needs permission to create *execution roles* for tasks such as `CreateNotebook`, `CreateTrainingJob`, and `CreateModel`. Grant these permissions with the following permissions policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "SageMakerApis",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "VpcConfigurationForCreateForms",  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeVpcs",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeSecurityGroups"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "KmsKeysForCreateForms",  
            "Effect": "Allow",  
            "Action": [  
                "kms:DescribeKey",  
                "kms>ListAliases"  
            ],  
            "Resource": "*"  
        },  
        {  
    ]  
}
```

```

    "Sid": "AccessAwsMarketplaceSubscriptions",
    "Effect": "Allow",
    "Action": [
        "aws-marketplace:ViewSubscriptions"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit:BatchGetRepositories",
        "codecommit>CreateRepository",
        "codecommit:GetRepository",
        "codecommit>ListRepositories",
        "codecommit>ListBranches",
        "secretsmanager>CreateSecret",
        "secretsmanager:DescribeSecret",
        "secretsmanager>ListSecrets"
    ],
    "Resource": "*"
},
{
    "Sid": "ListAndCreateExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam>ListRoles",
        "iam>CreateRole",
        "iam>CreateRole",
        "iam>CreatePolicy",
        "iam>AttachRolePolicy"
    ],
    "Resource": "*"
},
{
    "Sid": "DescribeECRMetaData",
    "Effect": "Allow",
    "Action": [
        "ecr:Describe*"
    ],
    "Resource": "*"
},
{
    "Sid": "PassRoleForExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "sagemaker.amazonaws.com"
        }
    }
}
]
}

```

Permissions Required to Use the Amazon SageMaker Ground Truth Console

To use the Amazon SageMaker Ground Truth console, you need to grant permissions for additional resources. Specifically, the console needs permissions for the AWS Marketplace to view subscriptions, Amazon Cognito operations to manage your private workforce, Amazon S3 actions for access to your input and output files, and AWS Lambda actions to list and invoke functions. Grant these permissions with the following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GroundTruthConsole",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:DescribeListings",
                "aws-marketplace:ViewSubscriptions",

                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminDisableUser",
                "cognito-idp:AdminEnableUser",
                "cognito-idp:AdminRemoveUserFromGroup",
                "cognito-idp>CreateGroup",
                "cognito-idp:CreateUserPool",
                "cognito-idp:CreateUserPoolClient",
                "cognito-idp:CreateUserPoolDomain",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp>ListGroups",
                "cognito-idp>ListIdentityProviders",
                "cognito-idp>ListUsers",
                "cognito-idp>ListUsersInGroup",
                "cognito-idp>ListUserPoolClients",
                "cognito-idp>ListUserPools",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:UpdateUserPoolClient",

                "groundtruthlabeling:DescribeConsoleJob",
                "groundtruthlabeling>ListDatasetObjects",
                "groundtruthlabeling:RunFilterOrSampleManifestJob",
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",

                "lambda:InvokeFunction",
                "lambda>ListFunctions",

                "s3:GetObject",
                "s3:PutObject",
                "s3>SelectObjectContent"
            ],
            "Resource": "*"
        }
    ]
}
```

Permissions Required to Use the Amazon Augmented AI (Preview) Console

To use the Augmented AI console, you need to grant permissions for additional resources. Grant these permissions with the following permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*Algorithm",
                "sagemaker:*Algorithms",
                "sagemaker:*App",
                "sagemaker:*Apps",
                "sagemaker:CreateApp",
                "sagemaker:DeleteApp",
                "sagemaker:DescribeApp",
                "sagemaker:ListAlgorithms",
                "sagemaker:ListApplications"
            ],
            "Resource": "*"
        }
    ]
}
```

```

"sagemaker:*AutoMLJob",
"sagemaker:*AutoMLJobs",
"sagemaker:*CodeRepositories",
"sagemaker:*CodeRepository",
"sagemaker:*CompilationJob",
"sagemaker:*CompilationJobs",
"sagemaker:*Endpoint",
"sagemaker:*EndpointConfig",
"sagemaker:*EndpointConfigs",
"sagemaker:*EndpointWeightsAndCapacities",
"sagemaker:*Endpoints",
"sagemaker:*Environment",
"sagemaker:*EnvironmentVersion",
"sagemaker:*EnvironmentVersions",
"sagemaker:*Environments",
"sagemaker:*Experiment",
"sagemaker:*Experiments",
"sagemaker:*FlowDefinitions",
"sagemaker:*HumanLoop",
"sagemaker:*HumanLoops",
"sagemaker:*HumanTaskUi",
"sagemaker:*HumanTaskUis",
"sagemaker:*HyperParameterTuningJob",
"sagemaker:*HyperParameterTuningJobs",
"sagemaker:*LabelingJob",
"sagemaker:*LabelingJobs",
"sagemaker:*Metrics",
"sagemaker:*Model",
"sagemaker:*ModelPackage",
"sagemaker:*ModelPackages",
"sagemaker:*Models",
"sagemaker:*MonitoringExecutions",
"sagemaker:*MonitoringSchedule",
"sagemaker:*MonitoringSchedules",
"sagemaker:*NotebookInstance",
"sagemaker:*NotebookInstanceLifecycleConfig",
"sagemaker:*NotebookInstanceLifecycleConfigs",
"sagemaker:*NotebookInstanceUrl",
"sagemaker:*NotebookInstances",
"sagemaker:*ProcessingJob",
"sagemaker:*ProcessingJobs",
"sagemaker:*RenderUiTemplate",
"sagemaker:*Search",
"sagemaker:*SearchSuggestions",
"sagemaker:*Tags",
"sagemaker:*TrainingJob",
"sagemaker:*TrainingJobs",
"sagemaker:*TransformJob",
"sagemaker:*TransformJobs",
"sagemaker:*Trial",
"sagemaker:*TrialComponent",
"sagemaker:*TrialComponents",
"sagemaker:*Trials",
"sagemaker:*Workteam",
"sagemaker:*Workteams"
],
"Resource": "*"
},
{
"Effect": "Allow",
"Action": [
"sagemaker:*FlowDefinition"
],
"Resource": "*",
"Condition": {
"StringEqualsIfExists": {

```

```
        "sagemaker:WorkteamType": [
            "private-crowd",
            "vendor-crowd"
        ]
    }
},
{
    "Effect": "Allow",
    "Action": [
        "application-autoscaling:DeleteScalingPolicy",
        "application-autoscaling:DeleteScheduledAction",
        "application-autoscaling:DeregisterScalableTarget",
        "application-autoscaling:DescribeScalableTargets",
        "application-autoscaling:DescribeScalingActivities",
        "application-autoscaling:DescribeScalingPolicies",
        "application-autoscaling:DescribeScheduledActions",
        "application-autoscaling:PutScalingPolicy",
        "application-autoscaling:PutScheduledAction",
        "application-autoscaling:RegisterScalableTarget",
        "aws-marketplace:ViewSubscriptions",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DescribeAlarms",
        "cloudwatch:GetMetricData",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch>ListMetrics",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:PutMetricData",
        "codecommit:BatchGetRepositories",
        "codecommit>CreateRepository",
        "codecommit:GetRepository",
        "codecommit>ListBranches",
        "codecommit>ListRepositories",
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp:CreateGroup",
        "cognito-idp:CreateUserPool",
        "cognito-idp:CreateUserPoolClient",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp>ListGroups",
        "cognito-idp>ListIdentityProviders",
        "cognito-idp>ListUserPoolClients",
        "cognito-idp>ListUserPools",
        "cognito-idp>ListUsers",
        "cognito-idp>ListUsersInGroup",
        "cognito-idp:UpdateUserPool",
        "cognito-idp:UpdateUserPoolClient",
        "ec2>CreateNetworkInterface",
        "ec2>CreateNetworkInterfacePermission",
        "ec2>CreateVpcEndpoint",
        "ec2>DeleteNetworkInterface",
        "ec2>DeleteNetworkInterfacePermission",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcEndpoints",
        "ec2:DescribeVpcs",
        "ecr:BatchCheckLayerAvailability",
```

```

        "ecr:BatchGetImage",
        "ecr>CreateRepository",
        "ecr:Describe*",
        "ecr:GetAuthorizationToken",
        "ecr:GetDownloadUrlForLayer",
        "elastic-inference:Connect",
        "elasticfilesystem:DescribeFileSystems",
        "elasticfilesystem:DescribeMountTargets",
        "fsx:DescribeFileSystems",
        "glue:CreateJob",
        "glue>DeleteJob",
        "glue:GetJob",
        "glue:GetJobRun",
        "glue:GetJobRuns",
        "glue:GetJobs",
        "glue:ResetJobBookmark",
        "glue:StartJobRun",
        "glue:UpdateJob",
        "groundtruthlabeling:*",
        "iam>ListRoles",
        "kms:DescribeKey",
        "kms>ListAliases",
        "lambda>ListFunctions",
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "logs:PutLogEvents",
        "sns>ListTopics"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogDelivery",
        "logs>DeleteLogDelivery",
        "logs:DescribeResourcePolicies",
        "logs:GetLogDelivery",
        "logs>ListLogDeliveries",
        "logs:PutResourcePolicy",
        "logs:UpdateLogDelivery"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:CompleteLayerUpload",
        "ecr:BatchDeleteImage",
        "ecr:UploadLayerPart",
        "ecr>DeleteRepositoryPolicy",
        "ecr:InitiateLayerUpload",
        "ecr>DeleteRepository",
        "ecr:PutImage"
    ],
    "Resource": "arn:aws:ecr:*:repository/*sagemaker*"
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit:GitPull",
        "codecommit:GitPush"
    ],

```

```

    "Resource": [
        "arn:aws:codemcommit:*::*sagemaker*",
        "arn:aws:codemcommit:*::*SageMaker*",
        "arn:aws:codemcommit:*::*Sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>ListSecrets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>DescribeSecret",
        "secretsmanager>GetSecretValue",
        "secretsmanager>CreateSecret"
    ],
    "Resource": [
        "arn:aws:secretsmanager:*::*secret:AmazonSageMaker-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>DescribeSecret",
        "secretsmanager>GetSecretValue"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "secretsmanager:ResourceTag/SageMaker": "true"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "robomaker>CreateSimulationApplication",
        "robomaker>DescribeSimulationApplication",
        "robomaker>DeleteSimulationApplication"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "robomaker>CreateSimulationJob",
        "robomaker>DescribeSimulationJob",
        "robomaker>CancelSimulationJob"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3>DeleteObject",
        "s3:AbortMultipartUpload",

```

```

        "s3:GetBucketCors",
        "s3:PutBucketCors"
    ],
    "Resource": [
        "arn:aws:s3::::*SageMaker*",
        "arn:aws:s3::::*Sagemaker*",
        "arn:aws:s3::::*sagemaker*",
        "arn:aws:s3::::*aws-glue*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3>CreateBucket",
        "s3:GetBucketLocation",
        "s3>ListBucket",
        "s3>ListAllMyBuckets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3GetObject"
    ],
    "Resource": "*",
    "Condition": {
        "StringEqualsIgnoreCase": {
            "s3:ExistingObjectTag/SageMaker": "true"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda::::function:*SageMaker*",
        "arn:aws:lambda::::function:*sagemaker*",
        "arn:aws:lambda::::function:*Sagemaker*",
        "arn:aws:lambda::::function:*LabelingFunction*"
    ]
},
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::::role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": "iam>CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:AWSServiceName": "robomaker.amazonaws.com"
        }
    }
}
{

```

```

        "Effect": "Allow",
        "Action": [
            "sns:Subscribe",
            "sns>CreateTopic"
        ],
        "Resource": [
            "arn:aws:sns:*::*SageMaker*",
            "arn:aws:sns:*::*Sagemaker*",
            "arn:aws:sns:*::*sagemaker*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::*:role/*",
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": [
                    "sagemaker.amazonaws.com",
                    "glue.amazonaws.com",
                    "robomaker.amazonaws.com",
                    "states.amazonaws.com"
                ]
            }
        }
    }
]
}

```

Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam>ListGroupsForUser",
                "iam>ListAttachedUserPolicies",
                "iam>ListUserPolicies",
                "iam GetUser"
            ],
            "Resource": [
                "arn:aws:iam::*:user/${aws:username}"
            ]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam GetPolicy",
                "iam>ListAttachedGroupPolicies",
                "iam>ListGroupPolicies",
                "iam>ListPolicyVersions",

```

```
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

Control Creation of Amazon SageMaker Resources with Condition Keys

Control fine-grained access to allow the creation of Amazon SageMaker resources by using Amazon SageMaker-specific condition keys. For information about using condition keys in IAM policies, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

The following table lists the Amazon SageMaker condition keys. The condition keys, along with related API actions, and links to relevant documentation are also listed in [Condition Keys for Amazon SageMaker](#) in the *IAM User Guide*.

The following examples show how to use the Amazon SageMaker condition keys to control access.

Topics

- [Control Access to Amazon SageMaker Resources by Using File System Condition Keys \(p. 764\)](#)
- [Restrict Training to a Specific VPC \(p. 766\)](#)
- [Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows \(p. 767\)](#)
- [Enforce Encryption of Input Data \(p. 768\)](#)
- [Enforce Encryption of Notebook Instance Storage Volume \(p. 768\)](#)
- [Enforce Network Isolation for Training Jobs \(p. 769\)](#)
- [Enforce a Specific Instance Type for Training Jobs \(p. 769\)](#)
- [Enforce a Specific EI Accelerator for Training Jobs \(p. 770\)](#)
- [Enforce Disabling Internet Access and Root Access for Creating Notebook Instances \(p. 770\)](#)

Control Access to Amazon SageMaker Resources by Using File System Condition Keys

Amazon SageMaker training provides a secure infrastructure for the training algorithm to run in, but for some cases you may want increased defense in depth. For example, you minimize the risk of running untrusted code in your algorithm, or you have specific security mandates in your organization. For these scenarios, you can use the service-specific condition keys in the Condition element of an IAM policy to scope down the user to specific file systems, directories, access modes (read-write, read-only) and security groups.

Topics

- [Restrict an IAM User to Specific Directories and Access Modes \(p. 764\)](#)
- [Restrict an IAM User to a Specific File System \(p. 765\)](#)

Restrict an IAM User to Specific Directories and Access Modes

The policy below restricts an IAM user to the `/sagemaker/xgboost-dm/train` and `/sagemaker/xgboost-dm/validation` directories of an EFS file system to `ro` (read-only) `AccessMode`:

Note

When a directory is allowed, all of its subdirectories are also accessible by the training algorithm. POSIX permissions are ignored.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToElasticFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/train"
                }
            }
        },
        {
            "Sid": "AccessToElasticFileSystemValidation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/validation"
                }
            }
        }
    ]
}
```

[Restrict an IAM User to a Specific File System](#)

To prevent a malicious algorithm using a user space client from accessing any file system directly in your account, you can restrict networking traffic by allowing ingress from a specific security group. In the following example, the IAM user can only use the specified security group to access the file system:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToLustreFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
```

```
        "sagemaker:FileSystemId": "fs-12345678",
        "sagemaker:FileSystemAccessMode": "ro",
        "sagemaker:FileSystemType": "FSxLustre",
        "sagemaker:FileSystemDirectoryPath": "/fsx/sagemaker/xgboost/train"
    },
    "ForAllValues:StringEquals": {
        "sagemaker:VpcSecurityGroupIds": [
            "sg-12345678"
        ]
    }
}
]
```

Although the above example can restrict an algorithm to a specific file system, it does not prevent an algorithm from accessing any directory within that file system using the user space client. To mitigate this, you can:

- Ensure that the file system only contains data that you trust your IAM users to access
 - Create an IAM role that restricts your IAM users to launching training jobs with algorithms from approved ECR repositories

For more information on how to use roles with Amazon SageMaker, see [Amazon SageMaker Roles](#).

Restrict Training to a Specific VPC

Restrict an AWS user to creating training jobs from within a Amazon VPC. When a training job is created within a VPC, you can use VPC flow logs to monitor all traffic to and from the training cluster. For information about using VPC flow logs, see [VPC Flow Logs](#) in the *Amazon Virtual Private Cloud User Guide*.

The following policy enforces that a training job is created by an IAM user calling [CreateTrainingJob](#) from within a VPC:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowFromVpc",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:CreateTrainingJob",  
                "sagemaker:CreateHyperParameterTuningJob"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "ForAllValues:StringEquals": {  
                    "sagemaker:VpcSubnets": ["subnet-a1234"],  
                    "sagemaker:VpcSecurityGroupIds": ["sg12345", "sg-67890"]  
                },  
                "Null": {  
                    "sagemaker:VpcSubnets": "false",  
                    "sagemaker:VpcSecurityGroupIds": "false"  
                }  
            }  
        }  
    ]  
}
```

Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows

Amazon SageMaker Ground Truth and Amazon Augmented AI work teams fall into one of three [workforce types](#): public (with Amazon Mechanical Turk), private, and vendor. To restrict IAM user access to a specific work team using one of these types or the work team ARN, use the `sagemaker:WorkteamType` and/or the `sagemaker:WorkteamArn` condition keys. For the `sagemaker:WorkteamType` condition key, use [string condition operators](#). For the `sagemaker:WorkteamArn` condition key, use [Amazon Resource Name \(ARN\) condition operators](#). If the user attempts to create a labeling job with a restricted work team, Amazon SageMaker returns an access denied error.

The policies below demonstrate different ways to use the `sagemaker:WorkteamType` and `sagemaker:WorkteamArn` condition keys with appropriate condition operators and valid condition values.

The following example uses the `sagemaker:WorkteamType` condition key with the `StringEquals` condition operator to restrict access to a public work team. It accepts condition values in the following format: `workforcetype-crowd`, where `workforcetype` can equal `public`, `private`, or `vendor`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:WorkteamType": "public-crowd"
                }
            }
        }
    ]
}
```

The following policies show how to restrict access to a public work team using the `sagemaker:WorkteamArn` condition key. The first shows how to use it with a valid IAM regex-variant of the work team ARN and the `ArnLike` condition operator. The second shows how to use it with the `ArnEquals` condition operator and the work team ARN.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "sagemaker:WorkteamArn": "arn:aws:sagemaker:*:::workteam/public-crowd/*"
                }
            }
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "sagemaker:WorkteamArn": "arn:aws:sagemaker:us-west-2:394669845002:workteam/public-crowd/default"
                }
            }
        }
    ]
}
```

Enforce Encryption of Input Data

The following policy restricts an IAM user to specify a AWS KMS key to encrypt input data when creating training, hyperparameter tuning, and labeling jobs by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob",
                "sagemaker:CreateLabelingJob",
                "sagemaker:CreateFlowDefiniton"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "sagemaker:VolumeKmsKey": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
                }
            }
        }
    ]
}
```

Enforce Encryption of Notebook Instance Storage Volume

The following policy restricts an IAM user to specify a AWS KMS key to encrypt the attached storage volume when creating or updating a notebook instance by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance"
```

```

        ],
        "Resource": "*",
        "Condition": {
            "ArnLike": {
                "sagemaker:VolumeKmsKey": "*key/volume-kms-key-12345"
            }
        }
    ]
}

```

Enforce Network Isolation for Training Jobs

The following policy restricts an IAM user to enable network isolation when creating training jobs by using the `sagemaker:NetworkIsolation` condition key:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceIsolation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "sagemaker:NetworkIsolation": "true"
                }
            }
        }
    ]
}

```

Enforce a Specific Instance Type for Training Jobs

The following policy restricts an IAM user to use a specific instance type when creating training jobs by using the `sagemaker:InstanceTypes` condition key:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceInstanceType",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringLike": {
                    "sagemaker:InstanceTypes": ["ml.c5.*"]
                }
            }
        }
    ]
}

```

Enforce a Specific EI Accelerator for Training Jobs

The following policy restricts an IAM user to use a specific elastic inference (EI) accelerator, if an accelerator is provided, when creating or updating notebook instances and when creating endpoint configurations by using the `sagemaker:AcceleratorTypes` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceAcceleratorType",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance",
                "sagemaker:UpdateNotebookInstance",
                "sagemaker:CreateEndpointConfig"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sagemaker:AcceleratorTypes": ["ml.eia1.medium"]
                }
            }
        }
    ]
}
```

Enforce Disabling Internet Access and Root Access for Creating Notebook Instances

You can disable both internet access and root access to notebook instances to help make them more secure. For information about controlling root access to a notebook instance, see [Control Root Access to a Notebook Instance \(p. 255\)](#). For information about disabling internet access for a notebook instance, see [Connect a Notebook Instance to Resources in a VPC \(p. 798\)](#).

The following policy requires an IAM user to disable network access when creating instance, and disable root access when creating or updating a notebook instance.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LockDownCreateNotebookInstance",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:DirectInternetAccess": "Disabled",
                    "sagemaker:RootAccess": "Disabled"
                },
                "Null": {
                    "sagemaker:VpcSubnets": "false",
                    "sagemaker:VpcSecurityGroupIds": "false"
                }
            }
        },
        {
            "Sid": "LockDownUpdateNotebookInstance",
            "Effect": "Allow",
            "Action": [
                "sagemaker:UpdateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:DirectInternetAccess": "Disabled",
                    "sagemaker:RootAccess": "Disabled"
                },
                "Null": {
                    "sagemaker:VpcSubnets": "false",
                    "sagemaker:VpcSecurityGroupIds": "false"
                }
            }
        }
    ]
}
```

```
        "Effect": "Allow",
        "Action": [
            "sagemaker:UpdateNotebookInstance"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "sagemaker:RootAccess": "Disabled"
            }
        }
    }
}
```

Control Access to the Amazon SageMaker API by Using Identity-based Policies

To control access to Amazon SageMaker API calls and calls to Amazon SageMaker hosted endpoints, use identity-based IAM policies.

Topics

- [Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC \(p. 771\)](#)

Restrict Access to Amazon SageMaker API and Runtime to Calls from Within Your VPC

If you set up an interface endpoint in your VPC, individuals outside the VPC can still connect to the Amazon SageMaker API and runtime over the internet unless you attach an IAM policy that restricts access to calls coming from within the VPC to all users and groups that have access to your Amazon SageMaker resources. For information about creating a VPC interface endpoint for the Amazon SageMaker API and runtime, see [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 800\)](#).

Important

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

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

Note

This policy allows connections only to callers within a subnet where you created an interface endpoint.

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

```

        }
    ]
}
}
```

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

```

{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedNotebookInstanceUrl"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        ]
    }
}
```

Limit Access to Amazon SageMaker API and Runtime Calls by IP Address

To allow access to Amazon SageMaker API calls and runtime invocations only from IP addresses in a list that you specify, attach an IAM policy that denies access to the API unless the call comes from an IP address in the list to every AWS Identity and Access Management user, group, or role used to access the API or runtime. For information about creating IAM policies, see [Creating IAM Policies](#) in the *AWS Identity and Access Management User Guide*. To specify the list of IP addresses that you want to have access to the API call, use the `IpAddress` condition operator and the `aws:SourceIP` condition context key. For information about IAM condition operators, see [IAM JSON Policy Elements: Condition Operators](#) in the *AWS Identity and Access Management User Guide*. For information about IAM condition context keys, see [AWS Global Condition Context Keys](#).

For example, the following policy allows access to the [CreateTrainingJob](#) only from IP addresses in the ranges 192.0.2.0-192.0.2.255 and 203.0.113.0-203.0.113.255:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sagemaker>CreateTrainingJob",
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```

```
        ]
    }
}
]
```

Limit Access to a Notebook Instance by IP Address

To allow access to a notebook instance only from IP addresses in a list that you specify, attach an IAM policy that denies access to

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

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

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "aws:SourceIp": [
            "192.0.2.0/24",
            "203.0.113.0/24"
          ]
        }
      }
    }
  ]
}
```

The policy restricts access to both the call to `CreatePresignedNotebookInstanceUrl` and to the URL that the call returns. The policy also restricts access to opening a notebook instance in the console and is enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance.

Note

Using this method to filter by IP address is incompatible when [connecting to Amazon SageMaker through a VPC interface endpoint](#). For information about restricting access to a notebook instance when connecting through a VPC interface endpoint, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 802\)](#).

Control Access to Amazon SageMaker Resources by Using Tags

Control access to groups of Amazon SageMaker resources by attaching tags to the resources and specifying `ResourceTag` conditions in IAM policies.

Note

Tag-based policies don't work to restrict the following API calls:

- ListAlgorithms
- ListCodeRepositories
- ListCompilationJobs
- ListEndpointConfigs
- ListEndpoints
- ListFlowDefinitions
- ListHumanTaskUis
- ListHyperparameterTuningJobs
- ListLabelingJobs
- ListLabelingJobsForWorkteam
- ListModelPackages
- ListModels
- ListNotebookInstanceLifecycleConfigs
- ListNotebookInstances
- ListSubscribedWorkteams
- ListTags
- ListProcessingJobs
- ListTrainingJobs
- ListTrainingJobsForHyperParameterTuningJob
- ListTransformJobs
- ListWorkteams
- Search

For example, suppose you've defined two different IAM groups, named `DevTeam1` and `DevTeam2`, in your AWS account. Suppose also that you've created 10 notebook instances, 5 of which are used for one project, and 5 of which are used for a second project. You want to allow members of `DevTeam1` to make API calls on notebook instances used for the first project, and members of `DevTeam2` to make API calls on notebook instances used for the second project.

To control access to API calls (example)

1. Add a tag with the key `Project` and value `A` to the notebook instances used for the first project. For information about adding tags to Amazon SageMaker resources, see [AddTags](#).
2. Add a tag with the key `Project` and value `B` to the notebook instances used for the second project.
3. Create an IAM policy with a `ResourceTag` condition that denies access to the notebook instances used for the second project, and attach that policy to `DevTeam1`. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `B`:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sagemaker:*",  
            "Resource": "*"  
        },  
        {  
            "Effect": "Deny",  
            "Action": "sagemaker:*",  
            "Resource": "*",  
            "Condition": {  
                "StringLike": {"aws:tag/Project": "B"}  
            }  
        }  
    ]  
}
```

```

        "StringEquals": {
            "sagemaker:ResourceTag/Project": "B"
        }
    },
{
    "Effect": "Deny",
    "Action": [
        "sagemaker:AddTags",
        "sagemaker:DeleteTags"
    ],
    "Resource": "*"
}
]
}

```

For information about creating IAM policies and attaching them to identities, see [Controlling Access Using Policies](#) in the *AWS Identity and Access Management User Guide*.

4. Create an IAM policy with a ResourceTag condition that denies access to the notebook instances used for the first project, and attach that policy to DevTeam2. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of Project and a value of A:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:ResourceTag/Project": "A"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "sagemaker:AddTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "*"
        }
    ]
}

```

Require the Presence or Absence of Tags for API Calls

Require the presence or absence of specific tags or specific tag values by using RequestTag condition keys in an IAM policy. For example, if you want to require that every endpoint created by any member of an IAM group to be created with a tag with the key environment and value dev, create a policy as follows:

```
{
}
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:CreateEndpoint",
            "Resource": "arn:aws:sagemaker:*:*:endpoint/*",
            "Condition": {
                "StringNotEquals": {
                    "aws:RequestTag/environment": "dev"
                }
            }
        }
    ]
}

```

Use Tags with Hyperparameter Tuning Jobs

You can add tags to a hyperparameter tuning job when you create the tuning job by specifying the tags as the `Tags` parameter when you call [CreateHyperParameterTuningJob](#). If you do this, the tags you specify for the hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches.

If you add tags to a hyperparameter tuning job by calling [AddTags](#), the tags you add are also added to any training jobs that the hyperparameter tuning job launches after you call [AddTags](#), but are not added to training jobs the hyperparameter tuning jobs launched before you called [AddTags](#). Similarly, when you remove tags from a hyperparameter tuning job by calling [DeleteTags](#), those tags are not removed from training jobs that the hyperparameter tuning job launched previously. Because of this, the tags associated with training jobs can be out of sync with the tags associated with the hyperparameter tuning job that launched them. If you use tags to control access to a hyperparameter tuning job and the training jobs it launches, you might want to keep the tags in sync. To make sure the tags associated with training jobs stay sync with the tags associated with the hyperparameter tuning job that launched them, first call [ListTrainingJobsForHyperParameterTuningJob](#) for the hyperparameter tuning job to get a list of the training jobs that the hyperparameter tuning job launched. Then, call [AddTags](#) or [DeleteTags](#) for the hyperparameter tuning job and for each of the training jobs in the list of training jobs to add or delete the same set of tags for all of the jobs. The following Python example demonstrates this:

```

tuning_job_arn =
    smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName='MyTuningJob')
    ['HyperParameterTuningJobArn']
smclient.add_tags(ResourceArn=tuning_job_arn, Tags=[{'Key': 'Env', 'Value': 'Dev'}])
training_jobs = smclient.list_training_jobs_for_hyper_parameter_tuning_job(
    HyperParameterTuningJobName='MyTuningJob')['TrainingJobSummaries']
for training_job in training_jobs:
    time.sleep(1) # Wait for 1 second between calls to avoid being throttled
    smclient.add_tags(ResourceArn=training_job['TrainingJobArn'], Tags=[{'Key': 'Env',
    'Value': 'Dev'}])

```

Amazon SageMaker 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](#), [CreateHyperParameterTuningJob](#), [CreateProcessingJob](#), [CreateTrainingJob](#), and [CreateModel](#).

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:

```
{  
    "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 broad, to allow for any actions you might want to perform in Amazon SageMaker. For a listing of the policy including information about the reasons for adding many of the permissions, see [AmazonSageMakerFullAccess Policy \(p. 789\)](#). If you prefer to create custom policies and manage permissions to scope the permissions only to the actions you need to perform with the execution role, see the following topics.

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

Topics

- [CreateNotebookInstance API: Execution Role Permissions \(p. 777\)](#)
- [CreateHyperParameterTuningJob API: Execution Role Permissions \(p. 780\)](#)
- [CreateProcessingJob API: Execution Role Permissions \(p. 782\)](#)
- [CreateTrainingJob API: Execution Role Permissions \(p. 785\)](#)
- [CreateModel API: Execution Role Permissions \(p. 787\)](#)
- [AmazonSageMakerFullAccess Policy \(p. 789\)](#)

CreateNotebookInstance API: Execution Role Permissions

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

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:*",  
                "ecr:GetAuthorizationToken",  
                "ecr:GetDownloadUrlForLayer",  
                "ecr:BatchGetImage",  
                "lambda:InvokeFunction"  
            ]  
        }  
    ]  
}
```

```

    "ecr:BatchCheckLayerAvailability",
    "ecr:SetRepositoryPolicy",
    "ecr:CompleteLayerUpload",
    "ecr:BatchDeleteImage",
    "ecr:UploadLayerPart",
    "ecr:DeleteRepositoryPolicy",
    "ecr:InitiateLayerUpload",
    "ecr:DeleteRepository",
    "ecr:PutImage",
    "ecr>CreateRepository",
    "cloudwatch:PutMetricData",
    "cloudwatch:GetMetricData",
    "cloudwatch:GetMetricStatistics",
    "cloudwatch>ListMetrics",
    "logs>CreateLogGroup",
    "logs>CreateLogStream",
    "logs:DescribeLogStreams",
    "logs:PutLogEvents",
    "logs:GetLogEvents",
    "s3>CreateBucket",
    "s3>ListBucket",
    "s3:GetBucketLocation",
    "s3GetObject",
    "s3:PutObject",
    "s3:DeleteObject",
    "robomaker>CreateSimulationApplication",
    "robomaker:DescribeSimulationApplication",
    "robomaker>DeleteSimulationApplication",
    "robomaker>CreateSimulationJob",
    "robomaker:DescribeSimulationJob",
    "robomaker:CancelSimulationJob",
    "ec2>CreateVpcEndpoint",
    "ec2:DescribeRouteTables",
    "fsx:DescribeFileSystem",
    "elasticfilesystem:DescribeMountTargets"
],
"Resource": "*"
},
{
"Effect": "Allow",
"Action": [
    "codecommit:GitPull",
    "codecommit:GitPush"
],
"Resource": [
    "arn:aws:codecommit:*::*:sagemaker*",
    "arn:aws:codecommit:*::*:SageMaker*",
    "arn:aws:codecommit:*::*:Sagemaker*"
]
},
{
"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 restricting "Resource": "*", as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "cloudwatch:PutMetricData",
                "logs>CreateLogGroup",
                "logs>CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3>DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object1",
                "arn:aws:s3:::outputbucket/path",
                "arn:aws:s3:::inputbucket/object2",
                "arn:aws:s3:::inputbucket/object3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:::repository/my-repo1",
                "arn:aws:ecr:::repository/my-repo2",
                "arn:aws:ecr:::repository/my-repo3"
            ]
        }
    ]
}
```

```

        "arn:aws:ecr:::repository/my-repo3"  

    ]  

}  

}
]
```

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 `s3GetObject`, `s3PutObject`, and `s3DeleteObject` permissions as follows:
 - Scope to the following values that you specify in a `CreateTrainingJob` request:

```
    InputDataConfig.DataSource.S3DataSource.S3Uri
```

```
    OutputDataConfig.S3OutputPath
```

- Scope to the following values that you specify in a `CreateModel` request:

```
    PrimaryContainer.ModelDataUrl
```

```
    SupplementalContainers.ModelDataUrl
```

- Scope `ecr` permissions as follows:
 - Scope to the `AlgorithmSpecification.TrainingImage` value that you specify in a `CreateTrainingJob` request.
 - Scope to the `PrimaryContainer.Image` value that you specify in a `CreateModel` request:

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

CreateHyperParameterTuningJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateHyperParameterTuningJob` API request, you can attach the following permission policy to the role:

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs>CreateLogStream",
        "logs:PutLogEvents",
        "logs>CreateLogGroup",
        "logs:DescribeLogStreams",
        "s3:GetObject",
        "s3:PutObject",
        "s3>ListBucket",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

```
        ]
    }
}
```

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

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3::::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3GetObject",
                "s3PutObject"
            ],
            "Resource": [
                "arn:aws:s3::::inputbucket/object",
                "arn:aws:s3::::outputbucket/path"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr>BatchCheckLayerAvailability",
                "ecr>GetDownloadUrlForLayer",
                "ecr>BatchGetImage"
            ],
            "Resource": "arn:aws:ecr::::repository/my-repo"
        }
    ]
}
```

If the training container associated with the hyperparameter tuning job needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the s3>ListBucket permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the s3GetObject and s3PutObject permissions to the following objects that you specify in the input and output data configuration in a `CreateHyperParameterTuningJob` request:

- ```
InputDataConfig.DataSource.S3DataSource.S3Uri
OutputDataConfig.S3OutputPath
• Scope Amazon ECR permissions to the registry path (AlgorithmSpecification.TrainingImage) that you specify in a CreateHyperParameterTuningJob request.
```

The `cloudwatch` and `logs` actions are applicable for "\*" resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your hyperparameter tuning job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:CreateNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2:DeleteNetworkInterfacePermission",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
]
}
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Decrypt"
]
}
```

If you specify a KMS key in the output configuration of your hyperparameter tuning job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt"
]
}
```

If you specify a volume KMS key in the resource configuration of your hyperparameter tuning job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms>CreateGrant"
]
}
```

## CreateProcessingJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateProcessingJob` API request, you can attach the following permission policy to the role:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:PutMetricData",
 "logs>CreateLogStream",
 "logs:PutLogEvents",
 "logs>CreateLogGroup",
 "logs:DescribeLogStreams",
 "s3:GetObject",
 "s3:PutObject",
 "s3>ListBucket",
 "ecr:GetAuthorizationToken",
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": "*"
 }
]
}
```

Instead of specifying "Resource": "\*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:PutMetricData",
 "logs>CreateLogStream",
 "logs:PutLogEvents",
 "logs>CreateLogGroup",
 "logs:DescribeLogStreams",
 "ecr:GetAuthorizationToken"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3>ListBucket"
],
 "Resource": [
 "arn:aws:s3:::inputbucket"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject"
],
 "Resource": [
 "arn:aws:s3:::inputbucket/object",
 "arn:aws:s3:::outputbucket/path"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:ListBucket"
],
 "Resource": [
 "arn:aws:s3:::inputbucket"
]
 }
]
}
```

```

 "Action": [
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": "arn:aws:ecr:::repository/my-repo"
]
}

```

If `CreateProcessingJob.AppSpecification.ImageUri` 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 `ProcessingInputs` in a `CreateProcessingJob` request.
- Scope the `s3GetObject` and `s3PutObject` permissions to the objects that will be downloaded or uploaded in the `ProcessingInputs` and `ProcessingOutputConfig` in a `CreateProcessingJob` request.
- Scope Amazon ECR permissions to the registry path (`AppSpecification.ImageUri`) that you specify in a `CreateProcessingJob` request.

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your processing job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:CreateNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2:DeleteNetworkInterfacePermission",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
]
}
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Decrypt"
]
}
```

If you specify a KMS key in the output configuration of your processing job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt"
]
}
```

```
]
}
```

If you specify a volume KMS key in the resource configuration of your processing job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:CreateGrant"
]
}
```

## 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:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:PutMetricData",
 "logs:CreateLogStream",
 "logs:PutLogEvents",
 "logs:CreateLogGroup",
 "logs:DescribeLogStreams",
 "s3:GetObject",
 "s3:PutObject",
 "s3>ListBucket",
 "ecr:GetAuthorizationToken",
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": "*"
 }
]
}
```

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

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "cloudwatch:PutMetricData",
 "logs:CreateLogStream",
 "logs:PutLogEvents",
 "logs:CreateLogGroup",
 "logs:DescribeLogStreams",
 "ecr:GetAuthorizationToken"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "s3:GetObject",
 "s3:PutObject",
 "s3>ListBucket",
 "ecr:BatchGetImage"
],
 "Resource": "arn:aws:s3:::mybucket/*"
 }
]
}
```

```

 "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.
- Scope the `s3GetObject` and `s3PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateTrainingJob` request:

`InputDataConfig.DataSource.S3DataSource.S3Uri`

`OutputDataConfig.S3OutputPath`

- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateTrainingJob` request.

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your training job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "ec2>CreateNetworkInterface",
 "ec2>CreateNetworkInterfacePermission",
 "ec2>DeleteNetworkInterface",
 "ec2>DeleteNetworkInterfacePermission",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
]
}
```

```
"ec2:DescribeDhcpOptions",
"ec2:DescribeSubnets",
"ec2:DescribeSecurityGroups"
```

If your input is encrypted using server-side encryption with an AWS KMS–managed key (SSE-KMS), add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Decrypt"
]
}
```

If you specify a KMS key in the output configuration of your training job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms:Encrypt"
]
}
```

If you specify a volume KMS key in the resource configuration of your training job, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
 "kms>CreateGrant"
]
}
```

## 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:

```
{
 "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:

```
{
 "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:GetObject"
],
 "Resource": [
 "arn:aws:s3:::inputbucket/object",
 "arn:aws:s3:::inputbucket/object"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "ecr:BatchCheckLayerAvailability",
 "ecr:GetDownloadUrlForLayer",
 "ecr:BatchGetImage"
],
 "Resource": [
 "arn:aws:ecr:::repository/my-repo",
 "arn:aws:ecr:::repository/my-repo"
]
 }
]
}
```

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.

If you specify a private VPC for your model, add the following permissions:

```
{
 "Effect": "Allow",
 "Action": [
```

```
"ec2:CreateNetworkInterface",
"ec2:CreateNetworkInterfacePermission",
"ec2:DeleteNetworkInterface",
"ec2:DeleteNetworkInterfacePermission",
"ec2:DescribeNetworkInterfaces",
"ec2:DescribeVpcs",
"ec2:DescribeDhcpOptions",
"ec2:DescribeSubnets",
"ec2:DescribeSecurityGroups"
```

## AmazonSageMakerFullAccess Policy

The [AmazonSageMakerFullAccess](#) managed policy includes all of the necessary permissions to perform most actions in Amazon SageMaker. You can use attach this policy to any role that you pass to an Amazon SageMaker execution role. You can also create more narrowly-scoped policies if you want more granular control of the permissions that you grant to your execution role.

The following list explains why some of the categories of permissions in the [AmazonSageMakerFullAccess](#) policy are needed.

`application-autoscaling`

Needed for automatically scaling an Amazon SageMaker real-time inference endpoint.

`aws-marketplace`

Needed to view AWS AI Marketplace subscriptions.

`cloudwatch`

Needed to post CloudWatch metrics, interact with alarms, and upload CloudWatch Logs logs in your account.

`codecommit`

Needed for AWS CodeCommit integration with Amazon SageMaker notebook instances.

`cognito`

Needed for Amazon SageMaker Ground Truth to define your private workforce and work teams.

`ec2`

Needed to manage elastic network interfaces when you specify a Amazon VPC for your Amazon SageMaker jobs and notebook instances.

`ec2:DescribeVpcs`

All Amazon SageMaker services launch Amazon EC2 instances and require this permission set.

`ecr`

Needed to pull and store Docker artifacts for training and inference. This is required only if you use your own container in Amazon SageMaker.

`elastic-inference`

Needed to integrate Amazon Elastic Inference with Amazon SageMaker.

`glue`

Needed for inference pipeline pre-processing from within Amazon SageMaker notebook instances.

`groundtruthlabeling`

Needed for Amazon SageMaker Ground Truth.

`iam>ListRoles`

Needed to give the Amazon SageMaker console access to list available roles.

`kms`

Needed to give the Amazon SageMaker console access to list the available AWS KMS keys.

`logs`

Needed to allow Amazon SageMaker jobs and endpoints to publish log streams.

## 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 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](#)

Required Permissions (API Action): `sagemaker:AddTags`

Resources: \*

### API Operation: [CreateEndpoint](#)

Required Permissions (API Action): `sagemaker:CreateEndpoint`

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

### API Operation: [CreateEndpointConfig](#)

Required Permissions (API Action): `sagemaker:CreateEndpointConfig`

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

### API Operation: [CreateModel](#)

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

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

### API Operation: [CreateLabelingJob](#)

Required Permissions (API Action): `sagemaker:CreateLabelingJob, iam:PassRole`

Resources: `arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName`

### API Operation:

#### [CreateNotebookInstance](#)

Required Permissions (API Action): `sagemaker:CreateNotebookInstance, iam:PassRole, ec2:CreateNetworkInterface, ec2:AttachNetworkInterface, ec2:ModifyNetworkInterfaceAttribute, ec2:DescribeAvailabilityZones, ec2:DescribeInternetGateways, ec2:DescribeSecurityGroups, ec2:DescribeSubnets, ec2:DescribeVpcs, kms>CreateGrant`

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

### API Operation: [CreateTrainingJob](#)

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

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

### API Operation: [CreateWorkteam](#)

Required Permissions (API Action): `sagemaker>CreateWorkteam, sagemaker:CreateWorkteam, cognito-idp:DescribeUserPoolClient, cognito-idp:UpdateUserPool, cognito-idp:DescribeUserPool, cognito-idp:UpdateUserPoolClient`

Resources: `arn:aws:sagemaker:region:account-id:workteam/private-crowd/work team name, arn:aws:sagemaker:region:account-id:workteam/vendor-crowd/work team name, arn:aws:sagemaker:region:account-id:workteam/public-crowd/work team name`

### API Operation: [DeleteEndpoint](#)

Required Permissions (API Action): `sagemaker>DeleteEndpoint`

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

**API Operation:** [DeleteEndpointConfig](#)

Required Permissions (API Action): `sagemaker:DeleteEndpointConfig`

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

**API Operation:** [DeleteModel](#)

Required Permissions (API Action): `sagemaker:DeleteModel`

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

**API Operation:** [DeleteNotebookInstance](#)

Required Permissions (API Action): `sagemaker:DeleteNotebookInstance, ec2:DeleteNetworkInterface, ec2:DetachNetworkInterface, ec2:DescribeAvailabilityZones, ec2:DescribeInternetGateways, ec2:DescribeSecurityGroups, ec2:DescribeSubnets, ec2:DescribeVpcs`

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

**API Operation:** [DeleteTags](#)

Required Permissions (API Action): `sagemaker:DeleteTags`

Resources: \*

**API Operation:** [DeleteWorkteam](#)

Required Permissions (API Action): `sagemaker:DeleteWorkteam`

Resources: `arn:aws:sagemaker:region:account-id:workteam/*`

**API Operation:** [DescribeEndpoint](#)

Required Permissions (API Action): `sagemaker:DescribeEndpoint`

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

**API Operation:** [DescribeEndpointConfig](#)

Required Permissions (API Action): `sagemaker:DescribeEndpointConfig`

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

**API Operation:** [DescribeLabelingJob](#)

Required Permissions (API Action): `sagemaker:DescribeLabelingJob`

Resources: `arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName`

**API Operation:** [DescribeModel](#)

Required Permissions (API Action): `sagemaker:DescribeModel`

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

**API Operation:**

[DescribeNotebookInstance](#)

Required Permissions (API Action): `sagemaker:DescribeNotebookInstance`

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

**API Operation:** [DescribeSubscribedWorkteam](#)

Required Permissions (API Action): sagemaker:DescribeSubscribedWorkteam, aws-marketplace:ViewSubscriptions

Resources: arn:aws:sagemaker:*region:account-id:workteam/\**

**API Operation:** [DescribeTrainingJob](#)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

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

**API Operation:** [DescribeWorkteam](#)

Required Permissions (API Action): sagemaker:DescribeWorkteam

Resources: arn:aws:sagemaker:*region:account-id:workteam/\**

**API Operation:**

[CreatePresignedNotebookInstanceUrl](#)

Required Permissions (API Action): sagemaker>CreatePresignedNotebookInstanceUrl

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

**API Operation:** [runtime\\_InvokeEndpoint](#)

Required Permissions (API Action): sagemaker:InvokeEndpoint

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

**API Operation:** [ListEndpointConfigs](#)

Required Permissions (API Action): sagemaker>ListEndpointConfigs

Resources: \*

**API Operation:** [ListEndpoints](#)

Required Permissions (API Action): sagemaker>ListEndpoints

Resources: \*

**API Operation:** [ListLabelingJobs](#)

Required Permissions (API Action): sagemaker>ListLabelingJobs

Resources: \*

**API Operation:** [ListLabelingJobsForWorkteam](#)

Required Permissions (API Action): sagemaker>ListLabelingJobsForWorkteam

Resources: \*

**API Operation:** [ListModels](#)

Required Permissions (API Action): sagemaker>ListModels

Resources: \*

**API Operation:** [ListNotebookInstances](#)

Required Permissions (API Action): sagemaker>ListNotebookInstances

Resources: \*

**API Operation:** [ListSubscribedWorkteams](#)

Required Permissions (API Action): `sagemaker>ListSubscribedWorkteam`, `aws-marketplace\ViewSubscriptions`

Resources: `arn:aws:sagemaker:region:account-id:workteam/*`

**API Operation:** [ListTags](#)

Required Permissions (API Action): `sagemaker>ListTags`

Resources: \*

**API Operation:** [ListTrainingJobs](#)

Required Permissions (API Action): `sagemaker>ListTrainingJobs`

Resources: \*

**API Operation:** [ListWorkteams](#)

Required Permissions (API Action): `sagemaker>ListWorkteams`

Resources: `arn:aws:sagemaker:region:account-id:workteam/*`

**API Operation:**

[StartNotebookInstance](#)

Required Permissions (API Action): `sagemaker StartNotebookInstance`,  
`iam:PassRole`, `ec2>CreateNetworkInterface`, `ec2:AttachNetworkInterface`,  
`ec2:ModifyNetworkInterfaceAttribute`, `ec2:DescribeAvailabilityZones`,  
`ec2:DescribeInternetGateways`, `ec2:DescribeSecurityGroups`,  
`ec2:DescribeSubnets`, `ec2:DescribeVpcs`, `kms>CreateGrant`

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

**API Operation:** [StopLabelingJob](#)

Required Permissions (API Action): `sagemaker:StopLabelingJob`

Resources: `arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName`

**API Operation:**

[StopNotebookInstance](#)

Required Permissions (API Action): `sagemaker:StopNotebookInstance`

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

**API Operation:** [StopTrainingJob](#)

Required Permissions (API Action): `sagemaker:StopTrainingJob`

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

**API Operation:** [UpdateEndpoint](#)

Required Permissions (API Action): `sagemaker:UpdateEndpoints`

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

**API Operation:**

[UpdateNotebookInstance](#)

Required Permissions (API Action): `sagemaker:UpdateNotebookInstance`, `iam:PassRole`

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

**API Operation:** [UpdateWorkteam](#)

Required Permissions (API Action): sagemaker:UpdateWorkteam

Resources: arn:aws:sagemaker:*region*:*account-id*:workteam/\*

## Troubleshooting Amazon SageMaker Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon SageMaker and IAM.

### Topics

- [I Am Not Authorized to Perform an Action in Amazon SageMaker \(p. 795\)](#)
- [I Am Not Authorized to Perform iam:PassRole \(p. 795\)](#)
- [I Want to View My Access Keys \(p. 796\)](#)
- [I'm an Administrator and Want to Allow Others to Access Amazon SageMaker \(p. 796\)](#)
- [I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources \(p. 796\)](#)

## I Am Not Authorized to Perform an Action in Amazon SageMaker

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a training job but does not have `sagemaker:sagemaker:DescribeTrainingJob` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not
 authorized to perform: sagemaker:DescribeTrainingJob on resource: my-example-
 widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `TrainingJob` resource using the `sagemaker:DescribeTrainingJob` action.

## I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to Amazon SageMaker.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon SageMaker. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

## I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

### Important

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing Access Keys](#) in the *IAM User Guide*.

## I'm an Administrator and Want to Allow Others to Access Amazon SageMaker

To allow others to access Amazon SageMaker, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in Amazon SageMaker.

To get started right away, see [Creating Your First IAM Delegated User and Group](#) in the *IAM User Guide*.

## I Want to Allow People Outside of My AWS Account to Access My Amazon SageMaker Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon SageMaker supports these features, see [How Amazon SageMaker Works with IAM \(p. 751\)](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing Access to an IAM User in Another AWS Account That You Own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing Access to AWS Accounts Owned by Third Parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing Access to Externally Authenticated Users \(Identity Federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM Roles Differ from Resource-based Policies](#) in the *IAM User Guide*.

# Logging and Monitoring

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 [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 731\)](#).

Amazon CloudWatch Logs enables you to monitor, store, and access your log files from Amazon EC2 instances, AWS CloudTrail, and other sources. 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. 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 [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 738\)](#).

AWS CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon SageMaker. Using the information collected by CloudTrail, you can determine the request that was made to Amazon SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, [Log Amazon SageMaker API Calls with AWS CloudTrail \(p. 739\)](#).

**Note**

CloudTrail does not monitor calls to `runtime_InvokeEndpoint`.

You can create rules in Amazon CloudWatch Events to react to status changes in status in an Amazon SageMaker training, hyperparameter tuning, or batch transform job. For more information, see [React to Amazon SageMaker Job Status Changes with CloudWatch Events \(p. 742\)](#).

# Compliance Validation for Amazon SageMaker

Third-party auditors assess the security and compliance of Amazon SageMaker as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see [AWS Services in Scope by Compliance Program](#). For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using Amazon SageMaker is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- [AWS Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [AWS Config](#) – This AWS service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [AWS Security Hub](#) – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

## Resilience in Amazon SageMaker

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

In addition to the AWS global infrastructure, Amazon SageMaker offers several features to help support your data resiliency and backup needs.

## Infrastructure Security in Amazon SageMaker

As a managed service, Amazon SageMaker is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to access Amazon SageMaker through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

### Topics

- [Connect a Notebook Instance to Resources in a VPC \(p. 798\)](#)
- [Training and Inference Containers Run in Internet-Free Mode \(p. 799\)](#)
- [Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities \(p. 800\)](#)
- [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 800\)](#)
- [Give Amazon SageMaker Processing Jobs Access to Resources in Your Amazon VPC \(p. 805\)](#)
- [Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 808\)](#)
- [Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 811\)](#)
- [Give Batch Transform Jobs Access to Resources in Your Amazon VPC \(p. 814\)](#)

## Connect a Notebook Instance to Resources in a VPC

Amazon SageMaker notebook instances are internet-enabled by default. This allows you to download popular packages and notebooks, customize your development environment, and work efficiently. However, if you connect a notebook instance to your VPC, the notebook instance provides an additional avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the computer (in the form of a publicly available notebook or a publicly available source code library) could access your data. If you do not want Amazon SageMaker to provide internet access to your notebook instance, you can disable direct internet access when you specify a VPC for your notebook instance. If you disable direct internet access, the notebook instance won't be able to train or host models unless your VPC has an interface endpoint (PrivateLink) or a NAT gateway and your security groups allow outbound connections. For information about creating a VPC interface endpoint to use

PrivateLink for your notebook instance, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 802\)](#). For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*. For information about security groups, see [Security Groups for Your VPC](#).

## Notebook Instances Provide the Best Experience for a Single User

An Amazon SageMaker notebook instance is designed to work best for an individual user. It is designed to give data scientists and other users the most power for managing their development environment. A notebook instance user has root access for installing packages and other pertinent software. We recommend that you exercise judgement when granting individuals access to notebook instances that are attached to a VPC that contains sensitive information. For example, you might grant a user access to a notebook instance with an IAM policy, as in the following example:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",
 "Resource": "arn:aws:sagemaker:region:account-id:notebook-instance/
myNotebookInstance"
 }
]
}
```

## Training and Inference Containers Run in Internet-Free Mode

Amazon SageMaker training and deployed inference containers are internet-enabled by default. This allows containers to access external services and resources on the public internet as part of your training and inference workloads. However, this offers an avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the container (in the form of a publicly available source code library) could access your data and transfer it to a remote host. If you use an Amazon VPC by specifying a value for the `VpcConfig` parameter when you call [CreateTrainingJob](#), [CreateHyperParameterTuningJob](#), or [CreateModel](#), you can protect your data and resources by managing security groups and restricting internet access from your VPC. However, this comes at the cost of additional network configuration, and has the risk of configuring your network incorrectly. If you do not want Amazon SageMaker to provide external network access to your training or inference containers, you can enable network isolation when you create your training job or model by setting the value of the `EnableNetworkIsolation` parameter to `True` when you call [CreateTrainingJob](#), [CreateHyperParameterTuningJob](#), or [CreateModel](#). If you enable network isolation, the containers are not able to make any outbound network calls, even to other AWS services such as Amazon S3. Additionally, no AWS credentials are made available to the container runtime environment. In the case of a training job with multiple instances, network inbound and outbound traffic is limited to the peers of each training container. Amazon SageMaker still performs download and upload operations against Amazon S3 using your Amazon SageMaker Execution Role in isolation from the training or inference container. Network isolation is required for training jobs and models run using resources from AWS Marketplace. Network isolation can be used in conjunction with a VPC. In this scenario, download and upload of customer data and model artifacts are routed via your VPC subnet. However, the training and inference containers themselves continue to be isolated from the network, and do not have access to any resource within your VPC or on the internet.

Network isolation is not supported by the following managed Amazon SageMaker containers as they require access to Amazon S3:

- Chainer
- PyTorch
- Scikit-learn
- Amazon SageMaker Reinforcement Learning

## Amazon SageMaker Scans AWS Marketplace Training and Inference Containers for Security Vulnerabilities

To meet our security requirements, algorithms and model packages listed in AWS Marketplace are scanned for Common Vulnerabilities and Exposures (CVE). CVE is a list of publicly known information about security vulnerability and exposure. The National Vulnerability Database (NVD) provides CVE details such as severity, impact rating, and fix information. Both CVE and NVD are available for public consumption and free for security tools and services to use. For more information, see [http://cve.mitre.org/about/faqs.html#what\\_is\\_cve](http://cve.mitre.org/about/faqs.html#what_is_cve).

## Connect to Amazon SageMaker Through a VPC Interface Endpoint

You can connect directly to the Amazon SageMaker API or to the Amazon SageMaker Runtime through an [interface endpoint](#) in your Virtual Private Cloud (VPC) instead of connecting over the internet. When you use a VPC interface endpoint, communication between your VPC and the Amazon SageMaker API or Runtime is conducted entirely and securely within the AWS network.

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

The VPC interface endpoint connects your VPC directly to the Amazon SageMaker API or Runtime without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the Amazon SageMaker API or Runtime.

You can create an interface endpoint to connect to Amazon SageMaker or to Amazon SageMaker Runtime with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see [Creating an Interface Endpoint](#).

*After you have created a VPC endpoint, you can use the following example CLI commands that use the endpoint-url parameter to specify interface endpoints to the Amazon SageMaker API or Runtime:*

```
aws sagemaker list-notebook-instances --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker list-training-jobs --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker-runtime invoke-endpoint --endpoint-
url VPC_Endpoint_ID.runtime.sagemaker.Region.vpce.amazonaws.com \
--endpoint-name Endpoint_Name \
--body "Endpoint_Body" \
--content-type "Content_Type" \
Output_File
```

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint URL. The Amazon SageMaker API DNS hostname that the CLI and Amazon SageMaker SDK use by default

(<https://api.sagemaker.Region.amazonaws.com>) resolves to your VPC endpoint. Similarly, the Amazon SageMaker Runtime DNS hostname that the CLI and Amazon SageMaker Runtime SDK use by default (<https://runtime.sagemaker.Region.amazonaws.com>) resolves to your VPC endpoint.

The Amazon SageMaker API and Runtime support VPC endpoints in all AWS Regions where both [Amazon VPC](#) and [Amazon SageMaker](#) are available. Amazon SageMaker supports making calls to all of its [Operations](#) inside your VPC. The result `AuthorizedUrl` from the `CreatePresignedNotebookInstanceUrl` is not supported by Private Link. For information about how to enable PrivateLink for the authorized URL that users use to connect to a notebook instance, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 802\)](#).

To learn more about AWS PrivateLink, see the [AWS PrivateLink documentation](#). Refer to [VPC Pricing](#) for the price of VPC Endpoints. To learn more about VPC and Endpoints, see [Amazon VPC](#). For information about how to use identity-based AWS Identity and Access Management policies to restrict access to the Amazon SageMaker API and runtime, see [Control Access to the Amazon SageMaker API by Using Identity-based Policies \(p. 771\)](#).

## Create a VPC Endpoint Policy for Amazon SageMaker

You can create a policy for Amazon VPC endpoints for Amazon SageMaker to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

**Note**

VPC endpoint policies aren't supported for Federal Information Processing Standard (FIPS) Amazon SageMaker runtime endpoints for [runtime\\_InvokeEndpoint](#).

The following example VPC endpoint policy specifies that all users who have access to the VPC interface endpoint are allowed to invoke the Amazon SageMaker hosted endpoint named `myEndpoint`.

```
{
 "Statement": [
 {
 "Action": "sagemaker:InvokeEndpoint",
 "Effect": "Allow",
 "Resource": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/myEndpoint",
 "Principal": "*"
 }
]
}
```

In this example, the following are denied:

- Other Amazon SageMaker API actions, such as `sagemaker:CreateEndpoint` and `sagemaker:CreateTrainingJob`.
- Invoking Amazon SageMaker hosted endpoints other than `myEndpoint`.

**Note**

In this example, users can still take other Amazon SageMaker API actions from outside the VPC. For information about how to restrict API calls to those from within the VPC, see [Control Access to the Amazon SageMaker API by Using Identity-based Policies \(p. 771\)](#).

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

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

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

### Note

Before you create an interface VPC endpoint to connect to a notebook instance, create an interface VPC endpoint to connect to the Amazon SageMaker API. That way, when users call [CreatePresignedNotebookInstanceUrl](#) to get the URL to connect to the notebook instance, that call also goes through the interface VPC endpoint. For information, see [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 800\)](#).

You can create an interface endpoint to connect to your notebook instance with either the AWS console or AWS Command Line Interface (AWS CLI) commands. For instructions, see [Creating an Interface Endpoint](#). Make sure that you create an interface endpoint for all of the subnets in your VPC from which you want to connect to the notebook instance.

When you create the interface endpoint, specify `aws.sagemaker.region.notebook` as the service name. After you create a VPC endpoint, enable private DNS for your VPC endpoint. Anyone using the Amazon SageMaker API, the AWS CLI, or the console to connect to the notebook instance from within the VPC will connect to the notebook instance through the VPC endpoint instead of the public internet.

Amazon SageMaker notebook instances support VPC endpoints in all AWS Regions where both [Amazon VPC](#) and [Amazon SageMaker](#) are available.

### Topics

- [Connect Your Private Network to Your VPC \(p. 802\)](#)
- [Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances \(p. 802\)](#)
- [Restrict Access to Connections from Within Your VPC \(p. 803\)](#)

## Connect Your Private Network to Your VPC

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

## Create a VPC Endpoint Policy for Amazon SageMaker Notebook Instances

You can create a policy for Amazon VPC endpoints for Amazon SageMaker notebook instances to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the [Amazon VPC User Guide](#).

The following example of a VPC endpoint policy specifies that all users that have access to the endpoint are allowed to access the notebook instance named `myNotebookInstance`.

```
{
 "Statement": [
 {
 "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",
 "Effect": "Allow",
 "Resource": "arn:aws:sagemaker:us-west-2:123456789012:notebook-instance/
myNotebookInstance",
 "Principal": "*"
 }
]
}
```

Access to other notebook instances is denied.

## Restrict Access to Connections from Within Your VPC

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

### Important

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

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

### Note

This policy allows connections only to callers within a subnet where you created an interface endpoint.

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

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

```
{
```

```
"Id": "notebook-example-1",
"Version": "2012-10-17",
"Statement": [
 {
 "Sid": "Enable Notebook Access",
 "Effect": "Allow",
 "Action": [
 "sagemaker>CreatePresignedNotebookInstanceUrl",
 "sagemaker:DescribeNotebookInstance"
],
 "Resource": "*",
 "Condition": {
 "ForAnyValue:StringEquals": {
 "aws:sourceVpce": [
 "vpce-111bbccc",
 "vpce-111bbddd"
]
 }
 }
 }
]
```

Both of these policy examples assume that you have also created an interface endpoint for the Amazon SageMaker API. For more information, see [Connect to Amazon SageMaker Through a VPC Interface Endpoint \(p. 800\)](#). In the second example, one of the values for aws:SourceVpce is the ID of the interface endpoint for the notebook instance. The other is the ID of the interface endpoint for the Amazon SageMaker API.

The policy examples here include

`DescribeNotebookInstance` because typically you would call `DescribeNotebookInstance` to make sure that the `NotebookInstanceStatus` is `InService` before you try to connect to it. For example:

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

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

{
 "AuthorizedUrl": "https://mynotebookinstance.notebook.us-west-2.sagemaker.aws?
authToken=AuthToken"
}
```

For both of these calls, if you did not enable private DNS hostnames for your VPC endpoint, or if you are using a version of the AWS SDK that was released before August 13, 2018, you must specify the

endpoint URL in the call. For example, the call to `create-presigned-notebook-instance-url` would be:

```
aws sagemaker create-presigned-notebook-instance-url
--notebook-instance-name myNotebookInstance --endpoint-url
VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

## Connect Your Private Network to Your VPC

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

## Give Amazon SageMaker Processing Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs processing jobs in an Amazon Virtual Private Cloud by default. However, processing containers access AWS resources—such as the Amazon S3 buckets where you store data—over the internet.

To control access to your data and processing containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your processing containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your processing containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create processing jobs by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your processing containers to connect to resources in your VPC. For information about ENIs, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

## Configure a Processing Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `NetworkConfig.VpcConfig` request parameter of the [CreateProcessingJob](#) API, or provide this information when you create a processing job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your processing containers. The ENIs provide your processing containers with a network connection within your VPC that is not connected to the internet. They also enable your processing job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateProcessingJob`:

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

```
"sg-0123456789abcdef0"
]
}
```

## Configure Your Private VPC for Amazon SageMaker Processing

When configuring the private VPC for your Amazon SageMaker processing jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 806\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 806\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 806\)](#)
- [Configure Route Tables \(p. 807\)](#)
- [Configure the VPC Security Group \(p. 807\)](#)
- [Connect to Resources Outside Your VPC \(p. 807\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a processing job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC so that processing containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your processing containers to access the buckets where you store your data. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

#### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 806\)](#).

### Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

## Restrict Package Installation on the Processing Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the processing container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

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

{
 "Statement": [
 { "Sid": "AmazonLinux2AMIRRepositoryAccess",
 "Principal": "*",
 "Action": [
 "s3:GetObject"
],
 "Effect": "Deny",
 "Resource": [
 "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
]
 }
]
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your processing jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In distributed processing, you must allow communication between the different containers in the same processing job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

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

- If your processing job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC](#)

[Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.

- If your processing job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give Amazon SageMaker Training Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud by default. However, training containers access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To control access to your data and training containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create training jobs by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your training containers to connect to resources in your VPC. For information about ENIs, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

### Note

For training jobs, you can configure only subnets with a default tenancy VPC in which your instance runs on shared hardware. For more information on the tenancy attribute for VPCs, see [Dedicated Instances](#).

## Configure a Training Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateTrainingJob](#) API, or provide this information when you create a training job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your training containers. The ENIs provide your training containers with a network connection within your VPC that is not connected to the internet. They also enable your training job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateTrainingJob`:

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

## Configure Your Private VPC for Amazon SageMaker Training

When configuring the private VPC for your Amazon SageMaker training jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 809\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 809\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 809\)](#)
- [Configure Route Tables \(p. 810\)](#)
- [Configure the VPC Security Group \(p. 810\)](#)
- [Connect to Resources Outside Your VPC \(p. 810\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a training job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC so that training containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your training data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your training containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

#### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 809\)](#).

### Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Training Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository,

create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

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

{
 "Statement": [
 { "Sid": "AmazonLinux2AMIRRepositoryAccess",
 "Principal": "*",
 "Action": [
 "s3:GetObject"
],
 "Effect": "Deny",
 "Resource": [
 "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
]
 }
]
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your training jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In distributed training, you must allow communication between the different containers in the same training job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

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

- If your training job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your training job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow

outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give Amazon SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC

Amazon SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access AWS resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To avoid making your data and model containers accessible over the internet, we recommend that you create a private VPC and configure it to control access to them. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

### Configure a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateModel](#) API, or provide this information when you create a model in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your model to connect to resources in your private VPC.

#### Note

You must create at least two subnets in different availability zones in your private VPC, even if you have only one hosting instance.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

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

### Configure Your Private VPC for Amazon SageMaker Hosting

When configuring the private VPC for your Amazon SageMaker models, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

#### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 812\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 812\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to Amazon S3 \(p. 812\)](#)
- [Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies \(p. 813\)](#)
- [Configure Route Tables \(p. 814\)](#)
- [Connect to Resources Outside Your VPC \(p. 814\)](#)

## Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each model instance. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

## Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

### To create an Amazon S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the AWS Region where your VPC resides.
4. For **VPC**, choose the VPC that you want to use for this endpoint.
5. For **Configure route tables**, choose the route tables that the endpoint will use. The VPC service automatically adds a route to each route table that you choose that points Amazon S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the Amazon S3 service by any user or service within the VPC. To restrict access further, choose **Custom**. For more information, see [Use a Custom Endpoint Policy to Restrict Access to Amazon S3 \(p. 812\)](#).

## Use a Custom Endpoint Policy to Restrict Access to Amazon S3

The default endpoint policy allows full access to Amazon Simple Storage Service (Amazon S3) for any user or service in your VPC. To further restrict access to Amazon S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#).

You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Model Container with a Custom Endpoint Policy

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from those repositories, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{
 "Statement": [
```

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

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

## Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies

The `SageMakerFullAccess` managed policy includes the permissions that you need to use models configured for Amazon VPC access with an endpoint. These permissions allow Amazon SageMaker to create an elastic network interface and attach it to model containers running in a VPC. If you use your own IAM policy, you must add the following permissions to that policy to use models configured for VPC access.

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:DescribeVpcEndpoints",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeVpcs",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DeleteNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2>CreateNetworkInterfacePermission",
 "ec2>CreateNetworkInterface"
],
 "Resource": "*"
 }
]
}
```

For more information about the `SageMakerFullAccess` managed policy, see [AmazonSageMakerFullAccess Policy \(p. 789\)](#).

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your models resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, models that use that VPC do not have access to resources outside your VPC. If your model needs access to resources outside your VPC, provide access with one of the following options:

- If your model needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your model needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give Batch Transform Jobs Access to Resources in Your Amazon VPC

Amazon SageMaker runs batch transform jobs in an Amazon Virtual Private Cloud by default. However, model containers access AWS resources—such as the Amazon S3 buckets where you store your data and model artifacts—over the internet.

To control access to your model containers and data, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your model containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your model containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. You then specify the same model when you create a batch transform job. When you specify the subnets and security groups, Amazon SageMaker creates *elastic network interfaces* (ENIs) that are associated with your security groups in one of the subnets. ENIs allow your model containers to connect to resources in your VPC. For information about ENIs, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

## Configure a Batch Transform Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [`CreateModel`](#) API, or provide this information when you create a model in the Amazon SageMaker console. Then specify the same model in the `ModelName` request parameter of the [`CreateTransformJob`](#) API, or in the **Model name** field when you create a transform job in the Amazon SageMaker console. Amazon SageMaker uses this information to create ENIs and attach them to your model containers. The ENIs provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your transform job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

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

If you are creating a model using the `CreateModel` API operation, the IAM execution role that you use to create your model must include the permissions described in [CreateModel API: Execution Role Permissions \(p. 787\)](#), including the following permissions required for a private VPC.

When creating a model in the console, if you select **Create a new role** in the **Model Settings** section, the [AmazonSageMakerFullAccess](#) policy used to create the role will already contain these permissions. If you select **Enter a custom IAM role ARN or Use existing role**, the role ARN that you specify must have an execution policy attached with the following permissions.

```
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface",
 "ec2:CreateNetworkInterfacePermission",
 "ec2:DeleteNetworkInterface",
 "ec2:DeleteNetworkInterfacePermission",
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeDhcpOptions",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
```

## Configure Your Private VPC for Amazon SageMaker Batch Transform

When configuring the private VPC for your Amazon SageMaker batch transform jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 815\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 816\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 816\)](#)
- [Configure Route Tables \(p. 817\)](#)
- [Configure the VPC Security Group \(p. 817\)](#)
- [Connect to Resources Outside Your VPC \(p. 817\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a transform job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

## Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.region.s3**, where **region** is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 816\)](#).

## Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Model Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

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

{
 "Statement": [
 { "Sid": "AmazonLinux2AMIRepositoryAccess",
 "Principal": "*",
 "Action": [
 "s3:GetObject"
]
 }
]
}
```

```
 "s3:GetObject"
],
 "Effect": "Deny",
 "Resource": [
 "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
]
}
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your batch transform jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In distributed batch transform, you must allow communication between the different containers in the same batch transform job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, batch transform jobs that use that VPC do not have access to resources outside your VPC. If your batch transform job needs access to resources outside your VPC, provide access with one of the following options:

- If your batch transform job needs access to an AWS service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(AWS PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your batch transform job needs access to an AWS service that doesn't support interface VPC endpoints or to a resource outside of AWS, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

# Limits and Supported Regions

For Amazon SageMaker service limits, see [Amazon SageMaker Limits](#).

You can request a quota increase using Service Quotas or the AWS Support Center. To request an increase, see [AWS Service Quotas](#) in the *AWS General Reference*.

For a list of the AWS Regions supporting Amazon SageMaker, see [Amazon SageMaker Regions](#).

## Topics

- [Supported Instance Types and Availability Zones \(p. 818\)](#)

# Supported Instance Types and Availability Zones

The tables in this topic list the availability of instance types for each AWS Region and Availability Zone for the following Amazon SageMaker components.

- Notebook instances – listed as `Notebook` in the tables
- Training jobs – listed as `Training` in the tables
- Batch transform jobs – listed as `Batch` in the tables
- Hosted endpoints – listed as `Endpoint` in the tables

Not all components are supported on each instance type in each Availability Zone.

There is one table for each AWS Region. In each table, the Amazon SageMaker components that support each instance type are listed for each Availability Zone. If no components support the instance type in an Availability Zone, the cell contains "None". If all components support the instance type in an Availability Zone, the cell contains "All".

To create a component on a specific instance type, you must specify the Availability Zone ID, which is listed in the header row of each table, for example, `use1-az1`. Availability Zone names, for example, `us-east-1a`, don't map directly to Availability Zone IDs. For different AWS accounts, the same Availability Zone name might refer to a different Availability Zone ID.

For more information, see [Regions and Availability Zones](#) in the *Amazon EC2 User Guide* and [AZ IDs for Your Resources](#) in the *AWS RAM User Guide*.

## Topics

- [Component Support for Instances in US East \(Ohio\) us-east-2 \(p. 819\)](#)
- [Component Support for Instances in US East \(N. Virginia\) us-east-1 \(p. 821\)](#)
- [Component Support for Instances in US West \(N. California\) us-west-1 \(p. 824\)](#)
- [Component Support for Instances in US West \(Oregon\) us-west-2 \(p. 826\)](#)
- [Component Support for Instances in Asia Pacific \(Hong Kong\) ap-east-1 \(p. 828\)](#)
- [Component Support for Instances in Asia Pacific \(Mumbai\) ap-south-1 \(p. 830\)](#)
- [Component Support for Instances in Asia Pacific \(Seoul\) ap-northeast-2 \(p. 833\)](#)
- [Component Support for Instances in Asia Pacific \(Singapore\) ap-southeast-1 \(p. 835\)](#)
- [Component Support for Instances in Asia Pacific \(Sydney\) ap-southeast-2 \(p. 837\)](#)
- [Component Support for Instances in Asia Pacific \(Tokyo\) ap-northeast-1 \(p. 839\)](#)
- [Component Support for Instances in Canada \(Central\) ca-central-1 \(p. 841\)](#)
- [Component Support for Instances in EU \(Frankfurt\) eu-central-1 \(p. 844\)](#)
- [Component Support for Instances in EU \(Ireland\) eu-west-1 \(p. 846\)](#)

- Component Support for Instances in EU (London) eu-west-2 (p. 848)
- Component Support for Instances in EU (Paris) eu-west-3 (p. 850)
- Component Support for Instances in EU (Stockholm) eu-north-1 (p. 852)
- Component Support for Instances in Middle East (Bahrain) me-south-1 (p. 854)
- Component Support for Instances in South America (Sao Paulo) sa-east-1 (p. 856)
- Component Support for Instances in AWS GovCloud (US-Gov-West) us-gov-west-1 (p. 859)

## Component Support for Instances in US East (Ohio) us-east-2

| Instance Type  | use2-az1                     | use2-az2                     | use2-az3                     |
|----------------|------------------------------|------------------------------|------------------------------|
| ml.t2.medium   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.large    | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.xlarge   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.2xlarge  | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t3.medium   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.large    | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.xlarge   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.2xlarge  | Notebook                     | Notebook                     | Notebook                     |
| ml.m4.xlarge   | All                          | All                          | All                          |
| ml.m4.2xlarge  | All                          | All                          | All                          |
| ml.m4.4xlarge  | All                          | All                          | All                          |
| ml.m4.10xlarge | All                          | All                          | All                          |
| ml.m4.16xlarge | Notebook, Training,<br>Batch | All                          | All                          |
| ml.m5.large    | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint |
| ml.m5.xlarge   | All                          | All                          | All                          |
| ml.m5.2xlarge  | All                          | All                          | All                          |
| ml.m5.4xlarge  | All                          | All                          | All                          |
| ml.m5.12xlarge | All                          | All                          | All                          |
| ml.m5.24xlarge | All                          | All                          | All                          |
| ml.r5.large    | None                         | None                         | None                         |
| ml.r5.xlarge   | None                         | None                         | None                         |
| ml.r5.2xlarge  | None                         | None                         | None                         |

| Instance Type    | use2-az1                     | use2-az2 | use2-az3                     |
|------------------|------------------------------|----------|------------------------------|
| ml.r5.4xlarge    | None                         | None     | None                         |
| ml.r5.12xlarge   | None                         | None     | None                         |
| ml.r5.24xlarge   | None                         | None     | None                         |
| ml.c4.large      | Endpoint                     | Endpoint | Endpoint                     |
| ml.c4.xlarge     | All                          | All      | All                          |
| ml.c4.2xlarge    | All                          | All      | All                          |
| ml.c4.4xlarge    | All                          | All      | All                          |
| ml.c4.8xlarge    | All                          | All      | All                          |
| ml.c5.large      | Endpoint                     | Endpoint | Endpoint                     |
| ml.c5.xlarge     | All                          | All      | All                          |
| ml.c5.2xlarge    | All                          | All      | All                          |
| ml.c5.4xlarge    | All                          | All      | All                          |
| ml.c5.9xlarge    | All                          | All      | All                          |
| ml.c5.18xlarge   | All                          | All      | All                          |
| ml.c5d.xlarge    | Notebook                     | Notebook | Notebook                     |
| ml.c5d.2xlarge   | Notebook                     | Notebook | Notebook                     |
| ml.c5d.4xlarge   | Notebook                     | Notebook | Notebook                     |
| ml.c5d.9xlarge   | Notebook                     | Notebook | Notebook                     |
| ml.c5d.18xlarge  | Notebook                     | Notebook | Notebook                     |
| ml.p2.xlarge     | All                          | All      | All                          |
| ml.p2.8xlarge    | All                          | All      | All                          |
| ml.p2.16xlarge   | Notebook, Training,<br>Batch | All      | Notebook, Training,<br>Batch |
| ml.p3.2xlarge    | All                          | All      | None                         |
| ml.p3.8xlarge    | All                          | All      | None                         |
| ml.p3.16xlarge   | All                          | All      | None                         |
| ml.p3dn.24xlarge | None                         | None     | None                         |
| ml.g4dn.xlarge   | None                         | None     | None                         |
| ml.g4dn.2xlarge  | None                         | None     | None                         |
| ml.g4dn.4xlarge  | None                         | None     | None                         |
| ml.g4dn.8xlarge  | None                         | None     | None                         |

| Instance Type    | use2-az1           | use2-az2           | use2-az3 |
|------------------|--------------------|--------------------|----------|
| ml.g4dn.12xlarge | None               | None               | None     |
| ml.g4dn.16xlarge | None               | None               | None     |
| ml.eia1.medium   | Notebook, Endpoint | Notebook, Endpoint | None     |
| ml.eia1.large    | Notebook, Endpoint | Notebook, Endpoint | None     |
| ml.eia1.xlarge   | Notebook, Endpoint | Notebook, Endpoint | None     |
| ml.eia2.medium   | Notebook, Endpoint | Notebook, Endpoint | None     |
| ml.eia2.large    | Notebook, Endpoint | Notebook, Endpoint | None     |
| ml.eia2.xlarge   | Notebook, Endpoint | Notebook, Endpoint | None     |

## Component Support for Instances in US East (N. Virginia) us-east-1

| Instance Type  | use1-az1           | use1-az2           | use1-az3           | use1-az4           | use1-az5           | use1-az6           |
|----------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| ml.t2.medium   | Notebook, Endpoint |
| ml.t2.large    | Notebook, Endpoint |
| ml.t2.xlarge   | Notebook, Endpoint |
| ml.t2.2xlarge  | Notebook, Endpoint |
| ml.t3.medium   | Notebook           | Notebook           | None               | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.t3.large    | Notebook           | Notebook           | None               | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.t3.xlarge   | Notebook           | Notebook           | None               | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.t3.2xlarge  | Notebook           | Notebook           | None               | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.m4.xlarge   | All                | All                | All                | All                | All                | All                |
| ml.m4.2xlarge  | All                | All                | All                | All                | All                | All                |
| ml.m4.4xlarge  | All                | All                | All                | All                | All                | All                |
| ml.m4.10xlarge | All                | All                | All                | All                | All                | All                |
| ml.m4.16xlarge | All                | All                | All                | All                | All                | All                |

| Instance Type  | use1-az1                        | use1-az2              | use1-az3 | use1-az4                        | use1-az5                        | use1-az6                        |
|----------------|---------------------------------|-----------------------|----------|---------------------------------|---------------------------------|---------------------------------|
| ml.m5.large    | Training,<br>Batch,<br>Endpoint | Training,<br>Batch    | None     | Training,<br>Batch,<br>Endpoint | Training,<br>Batch              | Training,<br>Batch,<br>Endpoint |
| ml.m5.xlarge   | All                             | All                   | None     | All                             | All                             | All                             |
| ml.m5.2xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.m5.4xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.m5.12xlarge | All                             | All                   | None     | All                             | All                             | All                             |
| ml.m5.24xlarge | All                             | All                   | None     | All                             | All                             | All                             |
| ml.r5.large    | None                            | None                  | None     | None                            | None                            | None                            |
| ml.r5.xlarge   | None                            | None                  | None     | None                            | None                            | None                            |
| ml.r5.2xlarge  | None                            | None                  | None     | None                            | None                            | None                            |
| ml.r5.4xlarge  | None                            | None                  | None     | None                            | None                            | None                            |
| ml.r5.12xlarge | None                            | None                  | None     | None                            | None                            | None                            |
| ml.r5.24xlarge | None                            | None                  | None     | None                            | None                            | None                            |
| ml.c4.large    | Endpoint                        | Endpoint              | Endpoint | Endpoint                        | Endpoint                        | Endpoint                        |
| ml.c4.xlarge   | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c4.2xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c4.4xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c4.8xlarge  | All                             | All                   | None     | All                             | Notebook,<br>Endpoint           | All                             |
| ml.c5.large    | Endpoint                        | Endpoint              | None     | Endpoint                        | Endpoint                        | Endpoint                        |
| ml.c5.xlarge   | All                             | All                   | Batch    | All                             | All                             | All                             |
| ml.c5.2xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c5.4xlarge  | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c5.9xlarge  | All                             | All                   | None     | All                             | Notebook,<br>Batch,<br>Endpoint | All                             |
| ml.c5.18xlarge | All                             | All                   | None     | All                             | All                             | All                             |
| ml.c5d.xlarge  | Notebook,<br>Endpoint           | Notebook,<br>Endpoint | None     | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           |
| ml.c5d.2xlarge | Notebook,<br>Endpoint           | Notebook,<br>Endpoint | None     | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           |
| ml.c5d.4xlarge | Notebook,<br>Endpoint           | Notebook,<br>Endpoint | None     | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           | Notebook,<br>Endpoint           |

| Instance Type    | use1-az1                  | use1-az2                  | use1-az3                  | use1-az4           | use1-az5           | use1-az6                  |
|------------------|---------------------------|---------------------------|---------------------------|--------------------|--------------------|---------------------------|
| ml.c5d.9xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | None                      | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint        |
| ml.c5d.18xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | None                      | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint        |
| ml.p2.xlarge     | All                       | All                       | Notebook, Training, Batch | All                | Notebook, Endpoint | All                       |
| ml.p2.8xlarge    | Notebook, Training, Batch | All                       | Notebook, Training, Batch | All                | None               | All                       |
| ml.p2.16xlarge   | Notebook, Training, Batch | Notebook, Training, Batch | Notebook, Training, Batch | All                | None               | Notebook, Training, Batch |
| ml.p3.2xlarge    | All                       | Training                  | None                      | None               | All                | Notebook, Training, Batch |
| ml.p3.8xlarge    | All                       | Training                  | None                      | None               | All                | All                       |
| ml.p3.16xlarge   | All                       | Training                  | None                      | None               | All                | All                       |
| ml.p3dn.24xlarge | Training                  | None                      | None                      | None               | None               | Training                  |
| ml.g4dn.xlarge   | None                      | None                      | None                      | None               | None               | None                      |
| ml.g4dn.2xlarge  | None                      | None                      | None                      | None               | None               | None                      |
| ml.g4dn.4xlarge  | None                      | None                      | None                      | None               | None               | None                      |
| ml.g4dn.8xlarge  | None                      | None                      | None                      | None               |                    |                           |
| ml.g4dn.12xlarge | None                      | None                      | None                      | None               |                    |                           |
| ml.g4dn.16xlarge | None                      | None                      | None                      | None               | None               | None                      |
| ml.eia1.medium   | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |
| ml.eia1.large    | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |
| ml.eia1.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |
| ml.eia2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |
| ml.eia2.large    | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |
| ml.eia2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | None                      | None               | None               | Notebook, Endpoint        |

## Component Support for Instances in US West (N. California) us-west-1

| Instance Type  | usw1-az1                  | usw1-az3                  |
|----------------|---------------------------|---------------------------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium   | Notebook                  | Notebook                  |
| ml.t3.large    | Notebook                  | Notebook                  |
| ml.t3.xlarge   | Notebook                  | Notebook                  |
| ml.t3.2xlarge  | Notebook                  | Notebook                  |
| ml.m4.xlarge   | All                       | All                       |
| ml.m4.2xlarge  | All                       | All                       |
| ml.m4.4xlarge  | All                       | All                       |
| ml.m4.10xlarge | All                       | All                       |
| ml.m4.16xlarge | All                       | All                       |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | All                       |
| ml.m5.2xlarge  | All                       | All                       |
| ml.m5.4xlarge  | All                       | All                       |
| ml.m5.12xlarge | All                       | All                       |
| ml.m5.24xlarge | All                       | All                       |
| ml.r5.large    | None                      | None                      |
| ml.r5.xlarge   | None                      | None                      |
| ml.r5.2xlarge  | None                      | None                      |
| ml.r5.4xlarge  | None                      | None                      |
| ml.r5.12xlarge | None                      | None                      |
| ml.r5.24xlarge | None                      | None                      |
| ml.c4.large    | Endpoint                  | Endpoint                  |
| ml.c4.xlarge   | All                       | All                       |
| ml.c4.2xlarge  | All                       | All                       |

| Instance Type    | usw1-az1 | usw1-az3 |
|------------------|----------|----------|
| ml.c4.4xlarge    | All      | All      |
| ml.c4.8xlarge    | All      | All      |
| ml.c5.large      | Endpoint | Endpoint |
| ml.c5.xlarge     | All      | All      |
| ml.c5.2xlarge    | All      | All      |
| ml.c5.4xlarge    | All      | All      |
| ml.c5.9xlarge    | All      | All      |
| ml.c5.18xlarge   | All      | All      |
| ml.c5d.xlarge    | Notebook | Notebook |
| ml.c5d.2xlarge   | Notebook | Notebook |
| ml.c5d.4xlarge   | Notebook | Notebook |
| ml.c5d.9xlarge   | Notebook | Notebook |
| ml.c5d.18xlarge  | Notebook | Notebook |
| ml.p2.xlarge     | None     | None     |
| ml.p2.8xlarge    | None     | None     |
| ml.p2.16xlarge   | None     | None     |
| ml.p3.2xlarge    | None     | None     |
| ml.p3.8xlarge    | None     | None     |
| ml.p3.16xlarge   | None     | None     |
| ml.p3dn.24xlarge | None     | None     |
| ml.g4dn.xlarge   | None     | None     |
| ml.g4dn.2xlarge  | None     | None     |
| ml.g4dn.4xlarge  | None     | None     |
| ml.g4dn.8xlarge  | None     | None     |
| ml.g4dn.12xlarge | None     | None     |
| ml.g4dn.16xlarge | None     | None     |
| ml.eia1.medium   | None     | None     |
| ml.eia1.large    | None     | None     |
| ml.eia1.xlarge   | None     | None     |
| ml.eia2.medium   | None     | None     |
| ml.eia2.large    | None     | None     |

| Instance Type  | usw1-az1 | usw1-az3 |
|----------------|----------|----------|
| ml.eia2.xlarge | None     | None     |

## Component Support for Instances in US West (Oregon) us-west-2

| Instance Type  | usw2-az1                  | usw2-az2                  | usw2-az3                  | usw2-az4 |
|----------------|---------------------------|---------------------------|---------------------------|----------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t3.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t3.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t3.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.t3.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        | None     |
| ml.m4.xlarge   | All                       | All                       | All                       | None     |
| ml.m4.2xlarge  | All                       | All                       | All                       | None     |
| ml.m4.4xlarge  | All                       | All                       | All                       | None     |
| ml.m4.10xlarge | All                       | All                       | All                       | None     |
| ml.m4.16xlarge | All                       | All                       | All                       | None     |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint | None     |
| ml.m5.xlarge   | All                       | All                       | All                       | None     |
| ml.m5.2xlarge  | All                       | All                       | All                       | None     |
| ml.m5.4xlarge  | All                       | All                       | All                       | None     |
| ml.m5.12xlarge | All                       | All                       | All                       | None     |
| ml.m5.24xlarge | All                       | All                       | All                       | None     |
| ml.r5.large    | None                      | None                      | None                      | None     |

| Instance Type   | usw2-az1                     | usw2-az2                        | usw2-az3                     | usw2-az4 |
|-----------------|------------------------------|---------------------------------|------------------------------|----------|
| ml.r5.xlarge    | None                         | None                            | None                         | None     |
| ml.r5.2xlarge   | None                         | None                            | None                         | None     |
| ml.r5.4xlarge   | None                         | None                            | None                         | None     |
| ml.r5.12xlarge  | None                         | None                            | None                         | None     |
| ml.r5.24xlarge  | None                         | None                            | None                         | None     |
| ml.c4.large     | Endpoint                     | Endpoint                        | Endpoint                     | None     |
| ml.c4.xlarge    | All                          | All                             | All                          | None     |
| ml.c4.2xlarge   | All                          | All                             | All                          | None     |
| ml.c4.4xlarge   | All                          | All                             | All                          | None     |
| ml.c4.8xlarge   | All                          | All                             | All                          | None     |
| ml.c5.large     | Endpoint                     | Endpoint                        | Endpoint                     | None     |
| ml.c5.xlarge    | All                          | All                             | All                          | None     |
| ml.c5.2xlarge   | All                          | All                             | All                          | None     |
| ml.c5.4xlarge   | All                          | All                             | All                          | None     |
| ml.c5.9xlarge   | All                          | All                             | All                          | None     |
| ml.c5.18xlarge  | All                          | All                             | All                          | None     |
| ml.c5d.xlarge   | Notebook,<br>Endpoint        | Notebook,<br>Endpoint           | Notebook,<br>Endpoint        | None     |
| ml.c5d.2xlarge  | Notebook,<br>Endpoint        | Notebook,<br>Endpoint           | Notebook,<br>Endpoint        | None     |
| ml.c5d.4xlarge  | Notebook,<br>Endpoint        | Notebook,<br>Endpoint           | Notebook,<br>Endpoint        | None     |
| ml.c5d.9xlarge  | Notebook,<br>Endpoint        | Notebook,<br>Endpoint           | Notebook,<br>Endpoint        | None     |
| ml.c5d.18xlarge | Notebook,<br>Endpoint        | Notebook,<br>Endpoint           | Notebook,<br>Endpoint        | None     |
| ml.p2.xlarge    | All                          | All                             | All                          | None     |
| ml.p2.8xlarge   | Notebook,<br>Training, Batch | Notebook,<br>Training, Batch    | Notebook,<br>Training, Batch | None     |
| ml.p2.16xlarge  | Notebook,<br>Training, Batch | Notebook,<br>Training, Batch    | Notebook,<br>Training, Batch | None     |
| ml.p3.2xlarge   | All                          | Notebook,<br>Training, Endpoint | All                          | None     |
| ml.p3.8xlarge   | All                          | Notebook,<br>Training, Endpoint | All                          | None     |

| Instance Type    | usw2-az1           | usw2-az2                     | usw2-az3           | usw2-az4 |
|------------------|--------------------|------------------------------|--------------------|----------|
| ml.p3.16xlarge   | All                | Notebook, Training, Endpoint | All                | None     |
| ml.p3dn.24xlarge | Training           | None                         | Training           | None     |
| ml.g4dn.xlarge   | None               | None                         | None               | None     |
| ml.g4dn.2xlarge  | None               | None                         | None               | None     |
| ml.g4dn.4xlarge  | None               | None                         | None               | None     |
| ml.g4dn.8xlarge  | None               | None                         | None               | None     |
| ml.g4dn.12xlarge | None               | None                         | None               | None     |
| ml.g4dn.16xlarge | None               | None                         | None               | None     |
| ml.eia1.medium   | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |
| ml.eia1.large    | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |
| ml.eia1.xlarge   | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |
| ml.eia2.medium   | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |
| ml.eia2.large    | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |
| ml.eia2.xlarge   | Notebook, Endpoint | Notebook, Endpoint           | Notebook, Endpoint | None     |

## Component Support for Instances in Asia Pacific (Hong Kong) ap-east-1

| Instance Type | ape1-az1 | ape1-az2 | ape1-az3 |
|---------------|----------|----------|----------|
| ml.t2.medium  | None     | None     | None     |
| ml.t2.large   | None     | None     | None     |
| ml.t2.xlarge  | None     | None     | None     |
| ml.t2.2xlarge | None     | None     | None     |
| ml.t3.medium  | Notebook | Notebook | Notebook |
| ml.t3.large   | Notebook | Notebook | Notebook |
| ml.t3.xlarge  | Notebook | Notebook | Notebook |
| ml.t3.2xlarge | Notebook | Notebook | Notebook |

| <b>Instance Type</b> | <b>ape1-az1</b>              | <b>ape1-az2</b>              | <b>ape1-az3</b>              |
|----------------------|------------------------------|------------------------------|------------------------------|
| ml.m4.xlarge         | None                         | None                         | None                         |
| ml.m4.2xlarge        | None                         | None                         | None                         |
| ml.m4.4xlarge        | None                         | None                         | None                         |
| ml.m4.10xlarge       | None                         | None                         | None                         |
| ml.m4.16xlarge       | None                         | None                         | None                         |
| ml.m5.large          | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint |
| ml.m5.xlarge         | All                          | All                          | All                          |
| ml.m5.2xlarge        | All                          | All                          | All                          |
| ml.m5.4xlarge        | All                          | All                          | All                          |
| ml.m5.12xlarge       | All                          | All                          | All                          |
| ml.m5.24xlarge       | All                          | All                          | All                          |
| ml.r5.large          | None                         | None                         | None                         |
| ml.r5.xlarge         | None                         | None                         | None                         |
| ml.r5.2xlarge        | None                         | None                         | None                         |
| ml.r5.4xlarge        | None                         | None                         | None                         |
| ml.r5.12xlarge       | None                         | None                         | None                         |
| ml.r5.24xlarge       | None                         | None                         | None                         |
| ml.c4.large          | None                         | None                         | None                         |
| ml.c4.xlarge         | None                         | None                         | None                         |
| ml.c4.2xlarge        | None                         | None                         | None                         |
| ml.c4.4xlarge        | None                         | None                         | None                         |
| ml.c4.8xlarge        | None                         | None                         | None                         |
| ml.c5.large          | Endpoint                     | Endpoint                     | Endpoint                     |
| ml.c5.xlarge         | All                          | All                          | All                          |
| ml.c5.2xlarge        | All                          | All                          | All                          |
| ml.c5.4xlarge        | All                          | All                          | All                          |
| ml.c5.9xlarge        | All                          | All                          | All                          |
| ml.c5.18xlarge       | All                          | All                          | All                          |
| ml.c5d.xlarge        | Notebook                     | Notebook                     | Notebook                     |
| ml.c5d.2xlarge       | Notebook                     | Notebook                     | Notebook                     |

| <b>Instance Type</b> | <b>ape1-az1</b> | <b>ape1-az2</b> | <b>ape1-az3</b> |
|----------------------|-----------------|-----------------|-----------------|
| ml.c5d.4xlarge       | Notebook        | Notebook        | Notebook        |
| ml.c5d.9xlarge       | Notebook        | Notebook        | Notebook        |
| ml.c5d.18xlarge      | Notebook        | Notebook        | Notebook        |
| ml.p2.xlarge         | None            | None            | None            |
| ml.p2.8xlarge        | None            | None            | None            |
| ml.p2.16xlarge       | None            | None            | None            |
| ml.p3.2xlarge        | None            | None            | None            |
| ml.p3.8xlarge        | None            | None            | None            |
| ml.p3.16xlarge       | None            | None            | None            |
| ml.p3dn.24xlarge     | None            | None            | None            |
| ml.g4dn.xlarge       | None            | None            | None            |
| ml.g4dn.2xlarge      | None            | None            | None            |
| ml.g4dn.4xlarge      | None            | None            | None            |
| ml.g4dn.8xlarge      | None            | None            | None            |
| ml.g4dn.12xlarge     | None            | None            | None            |
| ml.g4dn.16xlarge     | None            | None            | None            |
| ml.eia1.medium       | None            | None            | None            |
| ml.eia1.large        | None            | None            | None            |
| ml.eia1.xlarge       | None            | None            | None            |
| ml.eia2.medium       | None            | None            | None            |
| ml.eia2.large        | None            | None            | None            |
| ml.eia2.xlarge       | None            | None            | None            |

## Component Support for Instances in Asia Pacific (Mumbai) ap-south-1

| <b>Instance Type</b> | <b>aps1-az1</b>    | <b>aps1-az2</b> | <b>aps1-az3</b>    |
|----------------------|--------------------|-----------------|--------------------|
| ml.t2.medium         | Notebook, Endpoint | None            | Notebook, Endpoint |
| ml.t2.large          | Notebook, Endpoint | None            | Notebook, Endpoint |
| ml.t2.xlarge         | Notebook, Endpoint | None            | Notebook, Endpoint |
| ml.t2.2xlarge        | Notebook, Endpoint | None            | Notebook, Endpoint |

| Instance Type  | aps1-az1                     | aps1-az2 | aps1-az3                     |
|----------------|------------------------------|----------|------------------------------|
| ml.t3.medium   | None                         | None     | None                         |
| ml.t3.large    | None                         | None     | None                         |
| ml.t3.xlarge   | None                         | None     | None                         |
| ml.t3.2xlarge  | None                         | None     | None                         |
| ml.m4.xlarge   | All                          | None     | All                          |
| ml.m4.2xlarge  | All                          | None     | All                          |
| ml.m4.4xlarge  | All                          | None     | All                          |
| ml.m4.10xlarge | All                          | None     | All                          |
| ml.m4.16xlarge | All                          | None     | All                          |
| ml.m5.large    | Training, Batch,<br>Endpoint | Batch    | Training, Batch,<br>Endpoint |
| ml.m5.xlarge   | All                          | Batch    | All                          |
| ml.m5.2xlarge  | All                          | Batch    | All                          |
| ml.m5.4xlarge  | All                          | Batch    | All                          |
| ml.m5.12xlarge | All                          | Batch    | All                          |
| ml.m5.24xlarge | All                          | Batch    | All                          |
| ml.r5.large    | None                         | None     | None                         |
| ml.r5.xlarge   | None                         | None     | None                         |
| ml.r5.2xlarge  | None                         | None     | None                         |
| ml.r5.4xlarge  | None                         | None     | None                         |
| ml.r5.12xlarge | None                         | None     | None                         |
| ml.r5.24xlarge | None                         | None     | None                         |
| ml.c4.large    | Endpoint                     | None     | Endpoint                     |
| ml.c4.xlarge   | All                          | Batch    | All                          |
| ml.c4.2xlarge  | All                          | Batch    | All                          |
| ml.c4.4xlarge  | All                          | Batch    | All                          |
| ml.c4.8xlarge  | All                          | Batch    | All                          |
| ml.c5.large    | Endpoint                     | None     | Endpoint                     |
| ml.c5.xlarge   | All                          | None     | All                          |
| ml.c5.2xlarge  | All                          | None     | All                          |
| ml.c5.4xlarge  | All                          | None     | All                          |

| Instance Type    | aps1-az1 | aps1-az2 | aps1-az3 |
|------------------|----------|----------|----------|
| ml.c5.9xlarge    | All      | None     | All      |
| ml.c5.18xlarge   | All      | None     | All      |
| ml.c5d.xlarge    | None     | None     | None     |
| ml.c5d.2xlarge   | None     | None     | None     |
| ml.c5d.4xlarge   | None     | None     | None     |
| ml.c5d.9xlarge   | None     | None     | None     |
| ml.c5d.18xlarge  | None     | None     | None     |
| ml.p2.xlarge     | All      | None     | All      |
| ml.p2.8xlarge    | All      | None     | All      |
| ml.p2.16xlarge   | All      | None     | All      |
| ml.p3.2xlarge    | None     | None     | None     |
| ml.p3.8xlarge    | None     | None     | None     |
| ml.p3.16xlarge   | None     | None     | None     |
| ml.p3dn.24xlarge | None     | None     | None     |
| ml.g4dn.xlarge   | None     | None     | None     |
| ml.g4dn.2xlarge  | None     | None     | None     |
| ml.g4dn.4xlarge  | None     | None     | None     |
| ml.g4dn.8xlarge  | None     | None     | None     |
| ml.g4dn.12xlarge | None     | None     | None     |
| ml.g4dn.16xlarge | None     | None     | None     |
| ml.eia1.medium   | None     | None     | None     |
| ml.eia1.large    | None     | None     | None     |
| ml.eia1.xlarge   | None     | None     | None     |
| ml.eia2.medium   | None     | None     | None     |
| ml.eia2.large    | None     | None     | None     |
| ml.eia2.xlarge   | None     | None     | None     |

## Component Support for Instances in Asia Pacific (Seoul) ap-northeast-2

| Instance Type  | apne2-az1                 | apne2-az2 | apne2-az3                 |
|----------------|---------------------------|-----------|---------------------------|
| ml.t2.medium   | Notebook, Endpoint        | None      | Notebook, Endpoint        |
| ml.t2.large    | Notebook, Endpoint        | None      | Notebook, Endpoint        |
| ml.t2.xlarge   | Notebook, Endpoint        | None      | Notebook, Endpoint        |
| ml.t2.2xlarge  | Notebook, Endpoint        | None      | Notebook, Endpoint        |
| ml.t3.medium   | None                      | None      | None                      |
| ml.t3.large    | None                      | None      | None                      |
| ml.t3.xlarge   | None                      | None      | None                      |
| ml.t3.2xlarge  | None                      | None      | None                      |
| ml.m4.xlarge   | All                       | None      | All                       |
| ml.m4.2xlarge  | All                       | None      | All                       |
| ml.m4.4xlarge  | All                       | None      | All                       |
| ml.m4.10xlarge | All                       | None      | All                       |
| ml.m4.16xlarge | All                       | None      | All                       |
| ml.m5.large    | Training, Batch, Endpoint | None      | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | None      | All                       |
| ml.m5.2xlarge  | All                       | None      | All                       |
| ml.m5.4xlarge  | All                       | None      | All                       |
| ml.m5.12xlarge | All                       | None      | All                       |
| ml.m5.24xlarge | All                       | None      | All                       |
| ml.r5.large    | None                      | None      | None                      |
| ml.r5.xlarge   | None                      | None      | None                      |
| ml.r5.2xlarge  | None                      | None      | None                      |
| ml.r5.4xlarge  | None                      | None      | None                      |
| ml.r5.12xlarge | None                      | None      | None                      |
| ml.r5.24xlarge | None                      | None      | None                      |
| ml.c4.large    | Endpoint                  | None      | Endpoint                  |
| ml.c4.xlarge   | All                       | None      | All                       |
| ml.c4.2xlarge  | All                       | None      | All                       |

| Instance Type    | apne2-az1          | apne2-az2 | apne2-az3          |
|------------------|--------------------|-----------|--------------------|
| ml.c4.4xlarge    | All                | None      | All                |
| ml.c4.8xlarge    | All                | None      | All                |
| ml.c5.large      | Endpoint           | None      | Endpoint           |
| ml.c5.xlarge     | All                | None      | All                |
| ml.c5.2xlarge    | All                | None      | All                |
| ml.c5.4xlarge    | All                | None      | All                |
| ml.c5.9xlarge    | All                | None      | All                |
| ml.c5.18xlarge   | All                | None      | All                |
| ml.c5d.xlarge    | Notebook           | None      | Notebook           |
| ml.c5d.2xlarge   | Notebook           | None      | Notebook           |
| ml.c5d.4xlarge   | Notebook           | None      | Notebook           |
| ml.c5d.9xlarge   | Notebook           | None      | Notebook           |
| ml.c5d.18xlarge  | Notebook           | None      | Notebook           |
| ml.p2.xlarge     | All                | None      | All                |
| ml.p2.8xlarge    | All                | None      | All                |
| ml.p2.16xlarge   | All                | None      | All                |
| ml.p3.2xlarge    | All                | None      | All                |
| ml.p3.8xlarge    | All                | None      | All                |
| ml.p3.16xlarge   | All                | None      | All                |
| ml.p3dn.24xlarge | None               | None      | None               |
| ml.g4dn.xlarge   | None               | None      | None               |
| ml.g4dn.2xlarge  | None               | None      | None               |
| ml.g4dn.4xlarge  | None               | None      | None               |
| ml.g4dn.8xlarge  | None               | None      | None               |
| ml.g4dn.12xlarge | None               | None      | None               |
| ml.g4dn.16xlarge | None               | None      | None               |
| ml.eia1.medium   | Notebook, Endpoint | None      | Notebook, Endpoint |
| ml.eia1.large    | Notebook, Endpoint | None      | Notebook, Endpoint |
| ml.eia1.xlarge   | Notebook, Endpoint | None      | Notebook, Endpoint |
| ml.eia2.medium   | Notebook, Endpoint | None      | Notebook, Endpoint |
| ml.eia2.large    | Notebook, Endpoint | None      | Notebook, Endpoint |

| Instance Type  | <b>apne2-az1</b>   | <b>apne2-az2</b> | <b>apne2-az3</b>   |
|----------------|--------------------|------------------|--------------------|
| ml.eia2.xlarge | Notebook, Endpoint | None             | Notebook, Endpoint |

## Component Support for Instances in Asia Pacific (Singapore) ap-southeast-1

| Instance Type  | <b>apse1-az1</b>          | <b>apse1-az2</b>          | <b>apse1-az3</b>          |
|----------------|---------------------------|---------------------------|---------------------------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.large    | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.xlarge   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.2xlarge  | Notebook                  | Notebook                  | Notebook                  |
| ml.m4.xlarge   | All                       | All                       | All                       |
| ml.m4.2xlarge  | All                       | All                       | All                       |
| ml.m4.4xlarge  | All                       | All                       | Notebook, Endpoint        |
| ml.m4.10xlarge | All                       | All                       | Notebook, Endpoint        |
| ml.m4.16xlarge | All                       | All                       | Notebook, Endpoint        |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | All                       | All                       |
| ml.m5.2xlarge  | All                       | All                       | All                       |
| ml.m5.4xlarge  | All                       | All                       | All                       |
| ml.m5.12xlarge | All                       | All                       | All                       |
| ml.m5.24xlarge | All                       | All                       | All                       |
| ml.r5.large    | None                      | None                      | None                      |
| ml.r5.xlarge   | None                      | None                      | None                      |
| ml.r5.2xlarge  | None                      | None                      | None                      |
| ml.r5.4xlarge  | None                      | None                      | None                      |
| ml.r5.12xlarge | None                      | None                      | None                      |
| ml.r5.24xlarge | None                      | None                      | None                      |

| Instance Type    | <b>apse1-az1</b>          | <b>apse1-az2</b>          | <b>apse1-az3</b> |
|------------------|---------------------------|---------------------------|------------------|
| ml.c4.large      | Endpoint                  | Endpoint                  | Endpoint         |
| ml.c4.xlarge     | All                       | All                       | All              |
| ml.c4.2xlarge    | All                       | All                       | All              |
| ml.c4.4xlarge    | All                       | All                       | All              |
| ml.c4.8xlarge    | All                       | All                       | All              |
| ml.c5.large      | Endpoint                  | Endpoint                  | Endpoint         |
| ml.c5.xlarge     | All                       | All                       | All              |
| ml.c5.2xlarge    | All                       | All                       | All              |
| ml.c5.4xlarge    | All                       | All                       | All              |
| ml.c5.9xlarge    | All                       | All                       | All              |
| ml.c5.18xlarge   | All                       | All                       | All              |
| ml.c5d.xlarge    | Notebook                  | Notebook                  | Notebook         |
| ml.c5d.2xlarge   | Notebook                  | Notebook                  | Notebook         |
| ml.c5d.4xlarge   | Notebook                  | Notebook                  | Notebook         |
| ml.c5d.9xlarge   | Notebook                  | Notebook                  | Notebook         |
| ml.c5d.18xlarge  | Notebook                  | Notebook                  | Notebook         |
| ml.p2.xlarge     | All                       | All                       | None             |
| ml.p2.8xlarge    | All                       | All                       | None             |
| ml.p2.16xlarge   | All                       | All                       | None             |
| ml.p3.2xlarge    | Notebook, Training, Batch | Notebook, Training, Batch | None             |
| ml.p3.8xlarge    | Notebook, Training, Batch | Notebook, Training, Batch | None             |
| ml.p3.16xlarge   | Notebook, Training, Batch | Notebook, Training, Batch | None             |
| ml.p3dn.24xlarge | None                      | None                      | None             |
| ml.g4dn.xlarge   | None                      | None                      | None             |
| ml.g4dn.2xlarge  | None                      | None                      | None             |
| ml.g4dn.4xlarge  | None                      | None                      | None             |
| ml.g4dn.8xlarge  | None                      | None                      | None             |
| ml.g4dn.12xlarge | None                      | None                      | None             |
| ml.g4dn.16xlarge | None                      | None                      | None             |

| Instance Type  | apse1-az1 | apse1-az2 | apse1-az3 |
|----------------|-----------|-----------|-----------|
| ml.eia1.medium | None      | None      | None      |
| ml.eia1.large  | None      | None      | None      |
| ml.eia1.xlarge | None      | None      | None      |
| ml.eia2.medium | None      | None      | None      |
| ml.eia2.large  | None      | None      | None      |
| ml.eia2.xlarge | None      | None      | None      |

## Component Support for Instances in Asia Pacific (Sydney) ap-southeast-2

| Instance Type  | apse2-az1                 | apse2-az2                 | apse2-az3          |
|----------------|---------------------------|---------------------------|--------------------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint |
| ml.t3.medium   | Notebook                  | Notebook                  | None               |
| ml.t3.large    | Notebook                  | Notebook                  | None               |
| ml.t3.xlarge   | Notebook                  | Notebook                  | None               |
| ml.t3.2xlarge  | Notebook                  | Notebook                  | None               |
| ml.m4.xlarge   | All                       | All                       | All                |
| ml.m4.2xlarge  | All                       | All                       | All                |
| ml.m4.4xlarge  | All                       | All                       | All                |
| ml.m4.10xlarge | All                       | All                       | All                |
| ml.m4.16xlarge | All                       | All                       | All                |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint | None               |
| ml.m5.xlarge   | All                       | All                       | None               |
| ml.m5.2xlarge  | All                       | All                       | None               |
| ml.m5.4xlarge  | All                       | All                       | None               |
| ml.m5.12xlarge | All                       | All                       | None               |
| ml.m5.24xlarge | All                       | All                       | None               |
| ml.r5.large    | None                      | None                      | None               |

| <b>Instance Type</b> | <b>apse2-az1</b>          | <b>apse2-az2</b>          | <b>apse2-az3</b>          |
|----------------------|---------------------------|---------------------------|---------------------------|
| ml.r5.xlarge         | None                      | None                      | None                      |
| ml.r5.2xlarge        | None                      | None                      | None                      |
| ml.r5.4xlarge        | None                      | None                      | None                      |
| ml.r5.12xlarge       | None                      | None                      | None                      |
| ml.r5.24xlarge       | None                      | None                      | None                      |
| ml.c4.large          | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.c4.xlarge         | All                       | All                       | Notebook, Training, Batch |
| ml.c4.2xlarge        | All                       | All                       | Notebook, Training, Batch |
| ml.c4.4xlarge        | All                       | All                       | All                       |
| ml.c4.8xlarge        | All                       | All                       | All                       |
| ml.c5.large          | Endpoint                  | Endpoint                  | None                      |
| ml.c5.xlarge         | All                       | All                       | None                      |
| ml.c5.2xlarge        | All                       | All                       | None                      |
| ml.c5.4xlarge        | All                       | All                       | None                      |
| ml.c5.9xlarge        | All                       | All                       | None                      |
| ml.c5.18xlarge       | All                       | All                       | None                      |
| ml.c5d.xlarge        | Notebook                  | Notebook                  | None                      |
| ml.c5d.2xlarge       | Notebook                  | Notebook                  | None                      |
| ml.c5d.4xlarge       | Notebook                  | Notebook                  | None                      |
| ml.c5d.9xlarge       | Notebook                  | Notebook                  | None                      |
| ml.c5d.18xlarge      | Notebook                  | Notebook                  | None                      |
| ml.p2.xlarge         | All                       | All                       | None                      |
| ml.p2.8xlarge        | All                       | Notebook, Training, Batch | None                      |
| ml.p2.16xlarge       | Notebook, Training, Batch | All                       | None                      |
| ml.p3.2xlarge        | Notebook, Training, Batch | None                      | None                      |
| ml.p3.8xlarge        | Notebook, Training, Batch | Batch                     | None                      |
| ml.p3.16xlarge       | Notebook, Training, Batch | Batch                     | None                      |

| <b>Instance Type</b> | <b>apse2-az1</b> | <b>apse2-az2</b> | <b>apse2-az3</b> |
|----------------------|------------------|------------------|------------------|
| ml.p3dn.24xlarge     | None             | None             | None             |
| ml.g4dn.xlarge       | None             | None             | None             |
| ml.g4dn.2xlarge      | None             | None             | None             |
| ml.g4dn.4xlarge      | None             | None             | None             |
| ml.g4dn.8xlarge      | None             | None             | None             |
| ml.g4dn.12xlarge     | None             | None             | None             |
| ml.g4dn.16xlarge     | None             | None             | None             |
| ml.eia1.medium       | None             | None             | None             |
| ml.eia1.large        | None             | None             | None             |
| ml.eia1.xlarge       | None             | None             | None             |
| ml.eia2.medium       | None             | None             | None             |
| ml.eia2.large        | None             | None             | None             |
| ml.eia2.xlarge       | None             | None             | None             |

## Component Support for Instances in Asia Pacific (Tokyo) ap-northeast-1

| <b>Instance Type</b> | <b>apne1-az1</b>      | <b>apne1-az2</b>      | <b>apne1-az3</b> | <b>apne1-az4</b>      |
|----------------------|-----------------------|-----------------------|------------------|-----------------------|
| ml.t2.medium         | Notebook,<br>Endpoint | Notebook,<br>Endpoint | None             | Notebook,<br>Endpoint |
| ml.t2.large          | Notebook,<br>Endpoint | Notebook,<br>Endpoint | None             | Notebook,<br>Endpoint |
| ml.t2.xlarge         | Notebook,<br>Endpoint | Notebook,<br>Endpoint | None             | Notebook,<br>Endpoint |
| ml.t2.2xlarge        | Notebook,<br>Endpoint | Notebook,<br>Endpoint | Endpoint         | Notebook,<br>Endpoint |
| ml.t3.medium         | Notebook              | Notebook              | None             | Notebook              |
| ml.t3.large          | Notebook              | Notebook              | None             | Notebook              |
| ml.t3.xlarge         | Notebook              | Notebook              | None             | Notebook              |
| ml.t3.2xlarge        | Notebook              | Notebook              | None             | Notebook              |
| ml.m4.xlarge         | All                   | All                   | Endpoint         | All                   |
| ml.m4.2xlarge        | All                   | All                   | Endpoint         | All                   |
| ml.m4.4xlarge        | All                   | All                   | Endpoint         | All                   |

| <b>Instance Type</b> | <b>apne1-az1</b>          | <b>apne1-az2</b>          | <b>apne1-az3</b> | <b>apne1-az4</b>          |
|----------------------|---------------------------|---------------------------|------------------|---------------------------|
| ml.m4.10xlarge       | All                       | All                       | Endpoint         | All                       |
| ml.m4.16xlarge       | All                       | All                       | Endpoint         | All                       |
| ml.m5.large          | Training, Batch, Endpoint | Training, Batch, Endpoint | None             | Training, Batch, Endpoint |
| ml.m5.xlarge         | All                       | All                       | None             | All                       |
| ml.m5.2xlarge        | All                       | All                       | None             | All                       |
| ml.m5.4xlarge        | All                       | All                       | None             | All                       |
| ml.m5.12xlarge       | All                       | All                       | None             | All                       |
| ml.m5.24xlarge       | All                       | All                       | None             | All                       |
| ml.r5.large          | None                      | None                      | None             | None                      |
| ml.r5.xlarge         | None                      | None                      | None             | None                      |
| ml.r5.2xlarge        | None                      | None                      | None             | None                      |
| ml.r5.4xlarge        | None                      | None                      | None             | None                      |
| ml.r5.12xlarge       | None                      | None                      | None             | None                      |
| ml.r5.24xlarge       | None                      | None                      | None             | None                      |
| ml.c4.large          | Endpoint                  | Endpoint                  | None             | Endpoint                  |
| ml.c4.xlarge         | All                       | Notebook, Endpoint        | Endpoint         | All                       |
| ml.c4.2xlarge        | All                       | Notebook, Endpoint        | Endpoint         | All                       |
| ml.c4.4xlarge        | All                       | Notebook, Endpoint        | None             | All                       |
| ml.c4.8xlarge        | All                       | Notebook, Endpoint        | None             | All                       |
| ml.c5.large          | Endpoint                  | Endpoint                  | None             | Endpoint                  |
| ml.c5.xlarge         | All                       | All                       | None             | All                       |
| ml.c5.2xlarge        | All                       | All                       | None             | All                       |
| ml.c5.4xlarge        | All                       | All                       | None             | All                       |
| ml.c5.9xlarge        | All                       | All                       | None             | All                       |
| ml.c5.18xlarge       | All                       | All                       | None             | All                       |
| ml.c5d.xlarge        | Notebook                  | None                      | None             | Notebook                  |
| ml.c5d.2xlarge       | Notebook                  | None                      | None             | Notebook                  |
| ml.c5d.4xlarge       | Notebook                  | None                      | None             | Notebook                  |

| <b>Instance Type</b> | <b>apne1-az1</b>             | <b>apne1-az2</b> | <b>apne1-az3</b> | <b>apne1-az4</b>      |
|----------------------|------------------------------|------------------|------------------|-----------------------|
| ml.c5d.9xlarge       | Notebook                     | None             | None             | Notebook              |
| ml.c5d.18xlarge      | Notebook                     | None             | None             | Notebook              |
| ml.p2.xlarge         | All                          | None             | None             | All                   |
| ml.p2.8xlarge        | All                          | None             | None             | All                   |
| ml.p2.16xlarge       | All                          | None             | None             | All                   |
| ml.p3.2xlarge        | All                          | None             | None             | All                   |
| ml.p3.8xlarge        | All                          | None             | None             | All                   |
| ml.p3.16xlarge       | Notebook,<br>Training, Batch | None             | None             | All                   |
| ml.p3dn.24xlarge     | None                         | None             | None             | None                  |
| ml.g4dn.xlarge       | None                         | None             | None             | None                  |
| ml.g4dn.2xlarge      | None                         | None             | None             | None                  |
| ml.g4dn.4xlarge      | None                         | None             | None             | None                  |
| ml.g4dn.8xlarge      | None                         | None             | None             | None                  |
| ml.g4dn.12xlarge     | None                         | None             | None             | None                  |
| ml.g4dn.16xlarge     | None                         | None             | None             | None                  |
| ml.eia1.medium       | Notebook,<br>Endpoint        | None             | None             | Notebook,<br>Endpoint |
| ml.eia1.large        | Notebook,<br>Endpoint        | None             | None             | Notebook,<br>Endpoint |
| ml.eia1.xlarge       | Notebook,<br>Endpoint        | None             | None             | Notebook,<br>Endpoint |
| ml.eia2.medium       | None                         | None             | None             | None                  |
| ml.eia2.large        | None                         | None             | None             | None                  |
| ml.eia2.xlarge       | None                         | None             | None             | None                  |

## Component Support for Instances in Canada (Central) ca-central-1

| <b>Instance Type</b> | <b>cac1-az1</b>    | <b>cac1-az2</b>    |
|----------------------|--------------------|--------------------|
| ml.t2.medium         | Notebook, Endpoint | Notebook, Endpoint |
| ml.t2.large          | Notebook, Endpoint | Notebook, Endpoint |
| ml.t2.xlarge         | Notebook, Endpoint | Notebook, Endpoint |

| Instance Type  | cac1-az1                  | cac1-az2                  |
|----------------|---------------------------|---------------------------|
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium   | Notebook                  | Notebook                  |
| ml.t3.large    | Notebook                  | Notebook                  |
| ml.t3.xlarge   | Notebook                  | Notebook                  |
| ml.t3.2xlarge  | Notebook                  | Notebook                  |
| ml.m4.xlarge   | All                       | All                       |
| ml.m4.2xlarge  | All                       | All                       |
| ml.m4.4xlarge  | All                       | All                       |
| ml.m4.10xlarge | All                       | All                       |
| ml.m4.16xlarge | All                       | All                       |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | All                       |
| ml.m5.2xlarge  | All                       | All                       |
| ml.m5.4xlarge  | All                       | All                       |
| ml.m5.12xlarge | All                       | All                       |
| ml.m5.24xlarge | All                       | All                       |
| ml.r5.large    | None                      | None                      |
| ml.r5.xlarge   | None                      | None                      |
| ml.r5.2xlarge  | None                      | None                      |
| ml.r5.4xlarge  | None                      | None                      |
| ml.r5.12xlarge | None                      | None                      |
| ml.r5.24xlarge | None                      | None                      |
| ml.c4.large    | Endpoint                  | Endpoint                  |
| ml.c4.xlarge   | All                       | All                       |
| ml.c4.2xlarge  | All                       | All                       |
| ml.c4.4xlarge  | All                       | All                       |
| ml.c4.8xlarge  | All                       | All                       |
| ml.c5.large    | Endpoint                  | Endpoint                  |
| ml.c5.xlarge   | All                       | All                       |
| ml.c5.2xlarge  | All                       | All                       |
| ml.c5.4xlarge  | All                       | All                       |

| <b>Instance Type</b> | <b>cac1-az1</b> | <b>cac1-az2</b>           |
|----------------------|-----------------|---------------------------|
| ml.c5.9xlarge        | All             | All                       |
| ml.c5.18xlarge       | All             | All                       |
| ml.c5d.xlarge        | Notebook        | Notebook                  |
| ml.c5d.2xlarge       | Notebook        | Notebook                  |
| ml.c5d.4xlarge       | Notebook        | Notebook                  |
| ml.c5d.9xlarge       | Notebook        | Notebook                  |
| ml.c5d.18xlarge      | Notebook        | Notebook                  |
| ml.p2.xlarge         | None            | None                      |
| ml.p2.8xlarge        | None            | None                      |
| ml.p2.16xlarge       | None            | None                      |
| ml.p3.2xlarge        | None            | Notebook, Training, Batch |
| ml.p3.8xlarge        | None            | Notebook, Training, Batch |
| ml.p3.16xlarge       | None            | Notebook, Training, Batch |
| ml.p3dn.24xlarge     | None            | None                      |
| ml.g4dn.xlarge       | None            | None                      |
| ml.g4dn.2xlarge      | None            | None                      |
| ml.g4dn.4xlarge      | None            | None                      |
| ml.g4dn.8xlarge      | None            | None                      |
| ml.g4dn.12xlarge     | None            | None                      |
| ml.g4dn.16xlarge     | None            | None                      |
| ml.eia1.medium       | None            | None                      |
| ml.eia1.large        | None            | None                      |
| ml.eia1.xlarge       | None            | None                      |
| ml.eia2.medium       | None            | None                      |
| ml.eia2.large        | None            | None                      |
| ml.eia2.xlarge       | None            | None                      |

## Component Support for Instances in EU (Frankfurt) eu-central-1

| Instance Type  | euc1-az1                     | euc1-az2                     | euc1-az3                     |
|----------------|------------------------------|------------------------------|------------------------------|
| ml.t2.medium   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.large    | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.xlarge   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.2xlarge  | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t3.medium   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.large    | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.xlarge   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.2xlarge  | Notebook                     | Notebook                     | Notebook                     |
| ml.m4.xlarge   | All                          | All                          | All                          |
| ml.m4.2xlarge  | All                          | All                          | All                          |
| ml.m4.4xlarge  | All                          | All                          | All                          |
| ml.m4.10xlarge | All                          | All                          | All                          |
| ml.m4.16xlarge | All                          | All                          | All                          |
| ml.m5.large    | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint |
| ml.m5.xlarge   | All                          | All                          | All                          |
| ml.m5.2xlarge  | All                          | All                          | All                          |
| ml.m5.4xlarge  | All                          | All                          | All                          |
| ml.m5.12xlarge | All                          | All                          | All                          |
| ml.m5.24xlarge | All                          | All                          | All                          |
| ml.r5.large    | None                         | None                         | None                         |
| ml.r5.xlarge   | None                         | None                         | None                         |
| ml.r5.2xlarge  | None                         | None                         | None                         |
| ml.r5.4xlarge  | None                         | None                         | None                         |
| ml.r5.12xlarge | None                         | None                         | None                         |
| ml.r5.24xlarge | None                         | None                         | None                         |
| ml.c4.large    | Endpoint                     | Endpoint                     | Endpoint                     |
| ml.c4.xlarge   | All                          | All                          | All                          |
| ml.c4.2xlarge  | All                          | All                          | All                          |

| <b>Instance Type</b> | <b>euc1-az1</b> | <b>euc1-az2</b>           | <b>euc1-az3</b>           |
|----------------------|-----------------|---------------------------|---------------------------|
| ml.c4.4xlarge        | All             | All                       | All                       |
| ml.c4.8xlarge        | All             | All                       | All                       |
| ml.c5.large          | Endpoint        | Endpoint                  | Endpoint                  |
| ml.c5.xlarge         | All             | All                       | All                       |
| ml.c5.2xlarge        | All             | All                       | All                       |
| ml.c5.4xlarge        | All             | All                       | All                       |
| ml.c5.9xlarge        | All             | All                       | All                       |
| ml.c5.18xlarge       | All             | All                       | All                       |
| ml.c5d.xlarge        | Notebook        | Notebook                  | Notebook                  |
| ml.c5d.2xlarge       | Notebook        | Notebook                  | Notebook                  |
| ml.c5d.4xlarge       | Notebook        | Notebook                  | Notebook                  |
| ml.c5d.9xlarge       | Notebook        | Notebook                  | Notebook                  |
| ml.c5d.18xlarge      | Notebook        | Notebook                  | Notebook                  |
| ml.p2.xlarge         | None            | All                       | Notebook, Training, Batch |
| ml.p2.8xlarge        | None            | Notebook, Training, Batch | Notebook, Training, Batch |
| ml.p2.16xlarge       | Training        | Training, Batch           | Training, Batch           |
| ml.p3.2xlarge        | None            | All                       | All                       |
| ml.p3.8xlarge        | None            | All                       | All                       |
| ml.p3.16xlarge       | None            | All                       | All                       |
| ml.p3dn.24xlarge     | None            | None                      | None                      |
| ml.g4dn.xlarge       | None            | None                      | None                      |
| ml.g4dn.2xlarge      | None            | None                      | None                      |
| ml.g4dn.4xlarge      | None            | None                      | None                      |
| ml.g4dn.8xlarge      | None            | None                      | None                      |
| ml.g4dn.12xlarge     | None            | None                      | None                      |
| ml.g4dn.16xlarge     | None            | None                      | None                      |
| ml.eia1.medium       | None            | None                      | None                      |
| ml.eia1.large        | None            | None                      | None                      |
| ml.eia1.xlarge       | None            | None                      | None                      |
| ml.eia2.medium       | None            | None                      | None                      |

| Instance Type  | euc1-az1 | euc1-az2 | euc1-az3 |
|----------------|----------|----------|----------|
| ml.eia2.large  | None     | None     | None     |
| ml.eia2.xlarge | None     | None     | None     |

## Component Support for Instances in EU (Ireland) eu-west-1

| Instance Type  | euw1-az1                  | euw1-az2                  | euw1-az3                  |
|----------------|---------------------------|---------------------------|---------------------------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.large    | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.xlarge   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.2xlarge  | Notebook                  | Notebook                  | Notebook                  |
| ml.m4.xlarge   | All                       | All                       | All                       |
| ml.m4.2xlarge  | All                       | All                       | All                       |
| ml.m4.4xlarge  | All                       | Notebook, Training, Batch | All                       |
| ml.m4.10xlarge | All                       | All                       | All                       |
| ml.m4.16xlarge | All                       | All                       | All                       |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | All                       | All                       |
| ml.m5.2xlarge  | All                       | All                       | All                       |
| ml.m5.4xlarge  | All                       | All                       | All                       |
| ml.m5.12xlarge | All                       | All                       | All                       |
| ml.m5.24xlarge | All                       | All                       | All                       |
| ml.r5.large    | None                      | None                      | None                      |
| ml.r5.xlarge   | None                      | None                      | None                      |
| ml.r5.2xlarge  | None                      | None                      | None                      |
| ml.r5.4xlarge  | None                      | None                      | None                      |

| Instance Type    | <b>euw1-az1</b> | <b>euw1-azz</b>              | <b>euw1-az3</b>              |
|------------------|-----------------|------------------------------|------------------------------|
| ml.r5.12xlarge   | None            | None                         | None                         |
| ml.r5.24xlarge   | None            | None                         | None                         |
| ml.c4.large      | Endpoint        | Endpoint                     | Endpoint                     |
| ml.c4.xlarge     | All             | All                          | All                          |
| ml.c4.2xlarge    | All             | All                          | All                          |
| ml.c4.4xlarge    | All             | All                          | All                          |
| ml.c4.8xlarge    | All             | All                          | All                          |
| ml.c5.large      | Endpoint        | Endpoint                     | Endpoint                     |
| ml.c5.xlarge     | All             | All                          | All                          |
| ml.c5.2xlarge    | All             | All                          | All                          |
| ml.c5.4xlarge    | All             | All                          | All                          |
| ml.c5.9xlarge    | All             | All                          | All                          |
| ml.c5.18xlarge   | All             | All                          | All                          |
| ml.c5d.xlarge    | Notebook        | Notebook                     | Notebook                     |
| ml.c5d.2xlarge   | Notebook        | Notebook                     | Notebook                     |
| ml.c5d.4xlarge   | Notebook        | Notebook                     | Notebook                     |
| ml.c5d.9xlarge   | Notebook        | Notebook                     | Notebook                     |
| ml.c5d.18xlarge  | Notebook        | Notebook                     | Notebook                     |
| ml.p2.xlarge     | All             | All                          | All                          |
| ml.p2.8xlarge    | All             | Notebook, Training,<br>Batch | All                          |
| ml.p2.16xlarge   | All             | All                          | Notebook, Training,<br>Batch |
| ml.p3.2xlarge    | None            | All                          | All                          |
| ml.p3.8xlarge    | None            | All                          | All                          |
| ml.p3.16xlarge   | None            | All                          | All                          |
| ml.p3dn.24xlarge | None            | None                         | None                         |
| ml.g4dn.xlarge   | None            | None                         | None                         |
| ml.g4dn.2xlarge  | None            | None                         | None                         |
| ml.g4dn.4xlarge  | None            | None                         | None                         |
| ml.g4dn.8xlarge  | None            | None                         | None                         |
| ml.g4dn.12xlarge | None            | None                         | None                         |

| Instance Type    | euw1-az1 | euw1-az2           | euw1-az3           |
|------------------|----------|--------------------|--------------------|
| ml.g4dn.16xlarge | None     | None               | None               |
| ml.eia1.medium   | None     | Notebook, Endpoint | Notebook, Endpoint |
| ml.eia1.large    | None     | Notebook, Endpoint | Notebook, Endpoint |
| ml.eia1.xlarge   | None     | Notebook, Endpoint | Notebook, Endpoint |
| ml.eia2.medium   | None     | Notebook, Endpoint | Notebook, Endpoint |
| ml.eia2.large    | None     | Notebook, Endpoint | Notebook, Endpoint |
| ml.eia2.xlarge   | None     | Notebook, Endpoint | Notebook, Endpoint |

## Component Support for Instances in EU (London) eu-west-2

| Instance Type  | euw2-az1                     | euw2-az2                     | euw2-az3                     |
|----------------|------------------------------|------------------------------|------------------------------|
| ml.t2.medium   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.large    | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.xlarge   | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t2.2xlarge  | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint           |
| ml.t3.medium   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.large    | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.xlarge   | Notebook                     | Notebook                     | Notebook                     |
| ml.t3.2xlarge  | Notebook                     | Notebook                     | Notebook                     |
| ml.m4.xlarge   | All                          | All                          | All                          |
| ml.m4.2xlarge  | All                          | All                          | All                          |
| ml.m4.4xlarge  | All                          | All                          | All                          |
| ml.m4.10xlarge | All                          | All                          | All                          |
| ml.m4.16xlarge | All                          | All                          | All                          |
| ml.m5.large    | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint | Training, Batch,<br>Endpoint |
| ml.m5.xlarge   | All                          | All                          | All                          |
| ml.m5.2xlarge  | All                          | All                          | All                          |
| ml.m5.4xlarge  | All                          | All                          | All                          |
| ml.m5.12xlarge | All                          | All                          | All                          |
| ml.m5.24xlarge | All                          | All                          | All                          |

| Instance Type    | euw2-az1 | euw2-az2 | euw2-az3 |
|------------------|----------|----------|----------|
| ml.r5.large      | None     | None     | None     |
| ml.r5.xlarge     | None     | None     | None     |
| ml.r5.2xlarge    | None     | None     | None     |
| ml.r5.4xlarge    | None     | None     | None     |
| ml.r5.12xlarge   | None     | None     | None     |
| ml.r5.24xlarge   | None     | None     | None     |
| ml.c4.large      | Endpoint | Endpoint | Endpoint |
| ml.c4.xlarge     | All      | All      | All      |
| ml.c4.2xlarge    | All      | All      | All      |
| ml.c4.4xlarge    | All      | All      | All      |
| ml.c4.8xlarge    | All      | All      | All      |
| ml.c5.large      | Endpoint | Endpoint | Endpoint |
| ml.c5.xlarge     | All      | All      | All      |
| ml.c5.2xlarge    | All      | All      | All      |
| ml.c5.4xlarge    | All      | All      | All      |
| ml.c5.9xlarge    | All      | All      | All      |
| ml.c5.18xlarge   | All      | All      | All      |
| ml.c5d.xlarge    | Notebook | Notebook | Notebook |
| ml.c5d.2xlarge   | Notebook | Notebook | Notebook |
| ml.c5d.4xlarge   | Notebook | Notebook | Notebook |
| ml.c5d.9xlarge   | Notebook | Notebook | Notebook |
| ml.c5d.18xlarge  | Notebook | Notebook | Notebook |
| ml.p2.xlarge     | None     | None     | None     |
| ml.p2.8xlarge    | None     | None     | None     |
| ml.p2.16xlarge   | None     | None     | None     |
| ml.p3.2xlarge    | None     | All      | All      |
| ml.p3.8xlarge    | None     | All      | All      |
| ml.p3.16xlarge   | None     | All      | All      |
| ml.p3dn.24xlarge | None     | None     | None     |
| ml.g4dn.xlarge   | None     | None     | None     |
| ml.g4dn.2xlarge  | None     | None     | None     |

| Instance Type    | euw2-az1 | euw2-az2 | euw2-az3 |
|------------------|----------|----------|----------|
| ml.g4dn.4xlarge  | None     | None     | None     |
| ml.g4dn.8xlarge  | None     | None     | None     |
| ml.g4dn.12xlarge | None     | None     | None     |
| ml.g4dn.16xlarge | None     | None     | None     |
| ml.eia1.medium   | None     | None     | None     |
| ml.eia1.large    | None     | None     | None     |
| ml.eia1.xlarge   | None     | None     | None     |
| ml.eia2.medium   | None     | None     | None     |
| ml.eia2.large    | None     | None     | None     |
| ml.eia2.xlarge   | None     | None     | None     |

## Component Support for Instances in EU (Paris) eu-west-3

| Instance Type  | euw3-az1                  | euw3-az2                  | euw3-az3                  |
|----------------|---------------------------|---------------------------|---------------------------|
| ml.t2.medium   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.large    | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.xlarge   | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.2xlarge  | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.large    | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.xlarge   | Notebook                  | Notebook                  | Notebook                  |
| ml.t3.2xlarge  | Notebook                  | Notebook                  | Notebook                  |
| ml.m4.xlarge   | None                      | None                      | None                      |
| ml.m4.2xlarge  | None                      | None                      | None                      |
| ml.m4.4xlarge  | None                      | None                      | None                      |
| ml.m4.10xlarge | None                      | None                      | None                      |
| ml.m4.16xlarge | None                      | None                      | None                      |
| ml.m5.large    | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge   | All                       | All                       | All                       |
| ml.m5.2xlarge  | All                       | All                       | All                       |

| <b>Instance Type</b> | <b>euw3-az1</b> | <b>euw3-az2</b> | <b>euw3-az3</b> |
|----------------------|-----------------|-----------------|-----------------|
| ml.m5.4xlarge        | All             | All             | All             |
| ml.m5.12xlarge       | All             | All             | All             |
| ml.m5.24xlarge       | All             | All             | All             |
| ml.r5.large          | None            | None            | None            |
| ml.r5.xlarge         | None            | None            | None            |
| ml.r5.2xlarge        | None            | None            | None            |
| ml.r5.4xlarge        | None            | None            | None            |
| ml.r5.12xlarge       | None            | None            | None            |
| ml.r5.24xlarge       | None            | None            | None            |
| ml.c4.large          | None            | None            | None            |
| ml.c4.xlarge         | None            | None            | None            |
| ml.c4.2xlarge        | None            | None            | None            |
| ml.c4.4xlarge        | None            | None            | None            |
| ml.c4.8xlarge        | None            | None            | None            |
| ml.c5.large          | Endpoint        | Endpoint        | Endpoint        |
| ml.c5.xlarge         | All             | All             | All             |
| ml.c5.2xlarge        | All             | All             | All             |
| ml.c5.4xlarge        | All             | All             | All             |
| ml.c5.9xlarge        | All             | All             | All             |
| ml.c5.18xlarge       | All             | All             | All             |
| ml.c5d.xlarge        | Notebook        | Notebook        | Notebook        |
| ml.c5d.2xlarge       | Notebook        | Notebook        | Notebook        |
| ml.c5d.4xlarge       | Notebook        | Notebook        | Notebook        |
| ml.c5d.9xlarge       | Notebook        | Notebook        | Notebook        |
| ml.c5d.18xlarge      | Notebook        | Notebook        | Notebook        |
| ml.p2.xlarge         | None            | None            | None            |
| ml.p2.8xlarge        | None            | None            | None            |
| ml.p2.16xlarge       | None            | None            | None            |
| ml.p3.2xlarge        | None            | None            | None            |
| ml.p3.8xlarge        | None            | None            | None            |
| ml.p3.16xlarge       | None            | None            | None            |

| Instance Type    | euw3-az1 | euw3-az2 | euw3-az3 |
|------------------|----------|----------|----------|
| ml.p3dn.24xlarge | None     | None     | None     |
| ml.g4dn.xlarge   | None     | None     | None     |
| ml.g4dn.2xlarge  | None     | None     | None     |
| ml.g4dn.4xlarge  | None     | None     | None     |
| ml.g4dn.8xlarge  | None     | None     | None     |
| ml.g4dn.12xlarge | None     | None     | None     |
| ml.g4dn.16xlarge | None     | None     | None     |
| ml.eia1.medium   | None     | None     | None     |
| ml.eia1.large    | None     | None     | None     |
| ml.eia1.xlarge   | None     | None     | None     |
| ml.eia2.medium   | None     | None     | None     |
| ml.eia2.large    | None     | None     | None     |
| ml.eia2.xlarge   | None     | None     | None     |

## Component Support for Instances in EU (Stockholm) eu-north-1

| Instance Type  | eun1-az1 | eun1-az2 | eun1-az3 |
|----------------|----------|----------|----------|
| ml.t2.medium   | None     | None     | None     |
| ml.t2.large    | None     | None     | None     |
| ml.t2.xlarge   | None     | None     | None     |
| ml.t2.2xlarge  | None     | None     | None     |
| ml.t3.medium   | Notebook | Notebook | Notebook |
| ml.t3.large    | Notebook | Notebook | Notebook |
| ml.t3.xlarge   | Notebook | Notebook | Notebook |
| ml.t3.2xlarge  | Notebook | Notebook | Notebook |
| ml.m4.xlarge   | None     | None     | None     |
| ml.m4.2xlarge  | None     | None     | None     |
| ml.m4.4xlarge  | None     | None     | None     |
| ml.m4.10xlarge | None     | None     | None     |
| ml.m4.16xlarge | None     | None     | None     |

| <b>Instance Type</b> | <b>eun1-az1</b>           | <b>eun1-az2</b>           | <b>eun1-az3</b>           |
|----------------------|---------------------------|---------------------------|---------------------------|
| ml.m5.large          | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge         | All                       | All                       | All                       |
| ml.m5.2xlarge        | All                       | All                       | All                       |
| ml.m5.4xlarge        | All                       | All                       | All                       |
| ml.m5.12xlarge       | All                       | All                       | All                       |
| ml.m5.24xlarge       | All                       | All                       | All                       |
| ml.r5.large          | None                      | None                      | None                      |
| ml.r5.xlarge         | None                      | None                      | None                      |
| ml.r5.2xlarge        | None                      | None                      | None                      |
| ml.r5.4xlarge        | None                      | None                      | None                      |
| ml.r5.12xlarge       | None                      | None                      | None                      |
| ml.r5.24xlarge       | None                      | None                      | None                      |
| ml.c4.large          | None                      | None                      | None                      |
| ml.c4.xlarge         | None                      | None                      | None                      |
| ml.c4.2xlarge        | None                      | None                      | None                      |
| ml.c4.4xlarge        | None                      | None                      | None                      |
| ml.c4.8xlarge        | None                      | None                      | None                      |
| ml.c5.large          | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.c5.xlarge         | All                       | All                       | All                       |
| ml.c5.2xlarge        | All                       | All                       | All                       |
| ml.c5.4xlarge        | All                       | All                       | All                       |
| ml.c5.9xlarge        | All                       | All                       | All                       |
| ml.c5.18xlarge       | All                       | All                       | All                       |
| ml.c5d.xlarge        | Notebook                  | Notebook                  | Notebook                  |
| ml.c5d.2xlarge       | Notebook                  | Notebook                  | Notebook                  |
| ml.c5d.4xlarge       | Notebook                  | Notebook                  | Notebook                  |
| ml.c5d.9xlarge       | Notebook                  | Notebook                  | Notebook                  |
| ml.c5d.18xlarge      | Notebook                  | Notebook                  | Notebook                  |
| ml.p2.xlarge         | None                      | None                      | None                      |
| ml.p2.8xlarge        | None                      | None                      | None                      |

| Instance Type    | eun1-az1 | eun1-az2 | eun1-az3 |
|------------------|----------|----------|----------|
| ml.p2.16xlarge   | None     | None     | None     |
| ml.p3.2xlarge    | None     | None     | None     |
| ml.p3.8xlarge    | None     | None     | None     |
| ml.p3.16xlarge   | None     | None     | None     |
| ml.p3dn.24xlarge | None     | None     | None     |
| ml.g4dn.xlarge   | None     | None     | None     |
| ml.g4dn.2xlarge  | None     | None     | None     |
| ml.g4dn.4xlarge  | None     | None     | None     |
| ml.g4dn.8xlarge  | None     | None     | None     |
| ml.g4dn.12xlarge | None     | None     | None     |
| ml.g4dn.16xlarge | None     | None     | None     |
| ml.eia1.medium   | None     | None     | None     |
| ml.eia1.large    | None     | None     | None     |
| ml.eia1.xlarge   | None     | None     | None     |
| ml.eia2.medium   | None     | None     | None     |
| ml.eia2.large    | None     | None     | None     |
| ml.eia2.xlarge   | None     | None     | None     |

## Component Support for Instances in Middle East (Bahrain) me-south-1

| Instance Type | mes1-az1 | mes1-az2 | mes1-az3 |
|---------------|----------|----------|----------|
| ml.t2.medium  | None     | None     | None     |
| ml.t2.large   | None     | None     | None     |
| ml.t2.xlarge  | None     | None     | None     |
| ml.t2.2xlarge | None     | None     | None     |
| ml.t3.medium  | Notebook | Notebook | Notebook |
| ml.t3.large   | Notebook | Notebook | Notebook |
| ml.t3.xlarge  | Notebook | Notebook | Notebook |
| ml.t3.2xlarge | Notebook | Notebook | Notebook |
| ml.m4.xlarge  | None     | None     | None     |

| <b>Instance Type</b> | <b>mes1-az1</b>                 | <b>mes1-az2</b>                 | <b>mes1-az3</b>                 |
|----------------------|---------------------------------|---------------------------------|---------------------------------|
| ml.m4.2xlarge        | None                            | None                            | None                            |
| ml.m4.4xlarge        | None                            | None                            | None                            |
| ml.m4.10xlarge       | None                            | None                            | None                            |
| ml.m4.16xlarge       | None                            | None                            | None                            |
| ml.m5.large          | Training, Batch,<br>Endpoint    | Training, Batch,<br>Endpoint    | Training, Batch,<br>Endpoint    |
| ml.m5.xlarge         | All                             | All                             | All                             |
| ml.m5.2xlarge        | All                             | All                             | All                             |
| ml.m5.4xlarge        | All                             | All                             | All                             |
| ml.m5.12xlarge       | All                             | All                             | All                             |
| ml.m5.24xlarge       | All                             | All                             | All                             |
| ml.r5.large          | None                            | None                            | None                            |
| ml.r5.xlarge         | None                            | None                            | None                            |
| ml.r5.2xlarge        | None                            | None                            | None                            |
| ml.r5.4xlarge        | None                            | None                            | None                            |
| ml.r5.12xlarge       | None                            | None                            | None                            |
| ml.r5.24xlarge       | None                            | None                            | None                            |
| ml.c4.large          | None                            | None                            | None                            |
| ml.c4.xlarge         | Training                        | Training                        | Training                        |
| ml.c4.2xlarge        | Training                        | Training                        | Training                        |
| ml.c4.4xlarge        | Training                        | Training                        | Training                        |
| ml.c4.8xlarge        | Training                        | Training                        | Training                        |
| ml.c5.large          | Endpoint                        | Endpoint                        | Endpoint                        |
| ml.c5.xlarge         | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint |
| ml.c5.2xlarge        | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint |
| ml.c5.4xlarge        | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint |
| ml.c5.9xlarge        | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint |
| ml.c5.18xlarge       | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint | Notebook, Training,<br>Endpoint |
| ml.c5d.xlarge        | Notebook, Endpoint              | Notebook, Endpoint              | Notebook, Endpoint              |

| Instance Type    | mes1-az1           | mes1-az2           | mes1-az3           |
|------------------|--------------------|--------------------|--------------------|
| ml.c5d.2xlarge   | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.c5d.4xlarge   | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.c5d.9xlarge   | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.c5d.18xlarge  | Notebook, Endpoint | Notebook, Endpoint | Notebook, Endpoint |
| ml.p2.xlarge     | Training           | Training           | Training           |
| ml.p2.8xlarge    | Training           | Training           | Training           |
| ml.p2.16xlarge   | Training           | Training           | Training           |
| ml.p3.2xlarge    | Training           | Training           | Training           |
| ml.p3.8xlarge    | Training           | Training           | Training           |
| ml.p3.16xlarge   | Training           | Training           | Training           |
| ml.p3dn.24xlarge | None               | None               | None               |
| ml.g4dn.xlarge   | None               | None               | None               |
| ml.g4dn.2xlarge  | None               | None               | None               |
| ml.g4dn.4xlarge  | None               | None               | None               |
| ml.g4dn.8xlarge  | None               | None               | None               |
| ml.g4dn.12xlarge | None               | None               | None               |
| ml.g4dn.16xlarge | None               | None               | None               |
| ml.eia1.medium   | None               | None               | None               |
| ml.eia1.large    | None               | None               | None               |
| ml.eia1.xlarge   | None               | None               | None               |
| ml.eia2.medium   | None               | None               | None               |
| ml.eia2.large    | None               | None               | None               |
| ml.eia2.xlarge   | None               | None               | None               |

## Component Support for Instances in South America (Sao Paulo) sa-east-1

| Instance Type | sae1-az1           | sae1-az2 | sae1-az3           |
|---------------|--------------------|----------|--------------------|
| ml.t2.medium  | Notebook, Endpoint | Endpoint | Notebook, Endpoint |
| ml.t2.large   | Notebook, Endpoint | Endpoint | Notebook, Endpoint |
| ml.t2.xlarge  | Notebook, Endpoint | Endpoint | Notebook, Endpoint |

| Instance Type  | sae1-az1                     | sae1-az2 | sae1-az3                     |
|----------------|------------------------------|----------|------------------------------|
| ml.t2.2xlarge  | Notebook, Endpoint           | Endpoint | Notebook, Endpoint           |
| ml.t3.medium   | Notebook                     | None     | Notebook                     |
| ml.t3.large    | Notebook                     | None     | Notebook                     |
| ml.t3.xlarge   | Notebook                     | None     | Notebook                     |
| ml.t3.2xlarge  | Notebook                     | None     | Notebook                     |
| ml.m4.xlarge   | All                          | Endpoint | All                          |
| ml.m4.2xlarge  | All                          | Endpoint | All                          |
| ml.m4.4xlarge  | All                          | Endpoint | All                          |
| ml.m4.10xlarge | All                          | None     | All                          |
| ml.m4.16xlarge | All                          | Endpoint | All                          |
| ml.m5.large    | Training, Batch,<br>Endpoint | None     | Training, Batch,<br>Endpoint |
| ml.m5.xlarge   | All                          | None     | All                          |
| ml.m5.2xlarge  | All                          | None     | All                          |
| ml.m5.4xlarge  | All                          | None     | All                          |
| ml.m5.12xlarge | All                          | None     | All                          |
| ml.m5.24xlarge | All                          | None     | All                          |
| ml.r5.large    | None                         | None     | None                         |
| ml.r5.xlarge   | None                         | None     | None                         |
| ml.r5.2xlarge  | None                         | None     | None                         |
| ml.r5.4xlarge  | None                         | None     | None                         |
| ml.r5.12xlarge | None                         | None     | None                         |
| ml.r5.24xlarge | None                         | None     | None                         |
| ml.c4.large    | Endpoint                     | None     | Endpoint                     |
| ml.c4.xlarge   | All                          | None     | All                          |
| ml.c4.2xlarge  | All                          | None     | All                          |
| ml.c4.4xlarge  | All                          | None     | All                          |
| ml.c4.8xlarge  | All                          | None     | All                          |
| ml.c5.large    | Endpoint                     | None     | Endpoint                     |
| ml.c5.xlarge   | All                          | None     | All                          |
| ml.c5.2xlarge  | All                          | None     | All                          |

| Instance Type    | sae1-az1 | sae1-az2 | sae1-az3 |
|------------------|----------|----------|----------|
| ml.c5.4xlarge    | All      | None     | All      |
| ml.c5.9xlarge    | All      | None     | All      |
| ml.c5.18xlarge   | All      | None     | All      |
| ml.c5d.xlarge    | None     | None     | None     |
| ml.c5d.2xlarge   | None     | None     | None     |
| ml.c5d.4xlarge   | None     | None     | None     |
| ml.c5d.9xlarge   | None     | None     | None     |
| ml.c5d.18xlarge  | None     | None     | None     |
| ml.p2.xlarge     | None     | None     | None     |
| ml.p2.8xlarge    | None     | None     | None     |
| ml.p2.16xlarge   | None     | None     | None     |
| ml.p3.2xlarge    | None     | None     | None     |
| ml.p3.8xlarge    | None     | None     | None     |
| ml.p3.16xlarge   | None     | None     | None     |
| ml.p3dn.24xlarge | None     | None     | None     |
| ml.g4dn.xlarge   | None     | None     | None     |
| ml.g4dn.2xlarge  | None     | None     | None     |
| ml.g4dn.4xlarge  | None     | None     | None     |
| ml.g4dn.8xlarge  | None     | None     | None     |
| ml.g4dn.12xlarge | None     | None     | None     |
| ml.g4dn.16xlarge | None     | None     | None     |
| ml.eia1.medium   | None     | None     | None     |
| ml.eia1.large    | None     | None     | None     |
| ml.eia1.xlarge   | None     | None     | None     |
| ml.eia2.medium   | None     | None     | None     |
| ml.eia2.large    | None     | None     | None     |
| ml.eia2.xlarge   | None     | None     | None     |

## Component Support for Instances in AWS GovCloud (US-Gov-West) us-gov-west-1

| <b>Instance Type</b> | <b>usgw1-az1</b>          | <b>usgw1-az2</b>          | <b>usgw1-az3</b>          |
|----------------------|---------------------------|---------------------------|---------------------------|
| ml.t2.medium         | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.large          | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.xlarge         | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t2.2xlarge        | Notebook, Endpoint        | Notebook, Endpoint        | Notebook, Endpoint        |
| ml.t3.medium         | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.t3.large          | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.t3.xlarge         | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.t3.2xlarge        | Endpoint                  | Endpoint                  | Endpoint                  |
| ml.m4.xlarge         | All                       | All                       | All                       |
| ml.m4.2xlarge        | All                       | All                       | All                       |
| ml.m4.4xlarge        | All                       | All                       | All                       |
| ml.m4.10xlarge       | Notebook, Training, Batch | Notebook, Training, Batch | All                       |
| ml.m4.16xlarge       | Notebook, Training, Batch | All                       | All                       |
| ml.m5.large          | Training, Batch, Endpoint | Training, Batch, Endpoint | Training, Batch, Endpoint |
| ml.m5.xlarge         | All                       | All                       | All                       |
| ml.m5.2xlarge        | All                       | All                       | All                       |
| ml.m5.4xlarge        | All                       | All                       | All                       |
| ml.m5.12xlarge       | All                       | All                       | All                       |
| ml.m5.24xlarge       | All                       | All                       | All                       |
| ml.r5.large          | None                      | None                      | None                      |
| ml.r5.xlarge         | None                      | None                      | None                      |
| ml.r5.2xlarge        | None                      | None                      | None                      |
| ml.r5.4xlarge        | None                      | None                      | None                      |
| ml.r5.12xlarge       | None                      | None                      | None                      |
| ml.r5.24xlarge       | None                      | None                      | None                      |
| ml.c4.large          | Endpoint                  | Endpoint                  | None                      |

| <b>Instance Type</b> | <b>usgw1-az1</b>             | <b>usgw1-az2</b>             | <b>usgw1-az3</b>   |
|----------------------|------------------------------|------------------------------|--------------------|
| ml.c4.xlarge         | All                          | All                          | All                |
| ml.c4.2xlarge        | All                          | All                          | All                |
| ml.c4.4xlarge        | All                          | All                          | All                |
| ml.c4.8xlarge        | All                          | All                          | All                |
| ml.c5.large          | Endpoint                     | Endpoint                     | Endpoint           |
| ml.c5.xlarge         | All                          | All                          | All                |
| ml.c5.2xlarge        | All                          | All                          | All                |
| ml.c5.4xlarge        | All                          | All                          | All                |
| ml.c5.9xlarge        | All                          | All                          | All                |
| ml.c5.18xlarge       | All                          | All                          | All                |
| ml.c5d.xlarge        | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint |
| ml.c5d.2xlarge       | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint |
| ml.c5d.4xlarge       | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint |
| ml.c5d.9xlarge       | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint |
| ml.c5d.18xlarge      | Notebook, Endpoint           | Notebook, Endpoint           | Notebook, Endpoint |
| ml.p2.xlarge         | Endpoint                     | Endpoint                     | Training           |
| ml.p2.8xlarge        | Endpoint                     | Endpoint                     | Training           |
| ml.p2.16xlarge       | Endpoint                     | Endpoint                     | Training           |
| ml.p3.2xlarge        | Notebook, Batch,<br>Endpoint | Notebook, Batch,<br>Endpoint | Training           |
| ml.p3.8xlarge        | Notebook, Batch,<br>Endpoint | Notebook, Batch,<br>Endpoint | Training           |
| ml.p3.16xlarge       | Notebook, Batch              | Notebook, Batch              | Training           |
| ml.p3dn.24xlarge     | None                         | None                         | None               |
| ml.g4dn.xlarge       | None                         | None                         | None               |
| ml.g4dn.2xlarge      | None                         | None                         | None               |
| ml.g4dn.4xlarge      | None                         | None                         | None               |
| ml.g4dn.8xlarge      | None                         | None                         | None               |
| ml.g4dn.12xlarge     | None                         | None                         | None               |
| ml.g4dn.16xlarge     | None                         | None                         | None               |
| ml.eia1.medium       | None                         | None                         | None               |
| ml.eia1.large        | None                         | None                         | None               |

| Instance Type  | usgw1-az1 | usgw1-az2 | usgw1-az3 |
|----------------|-----------|-----------|-----------|
| ml.eia1.xlarge | None      | None      | None      |
| ml.eia2.medium | None      | None      | None      |
| ml.eia2.large  | None      | None      | None      |
| ml.eia2.xlarge | None      | None      | None      |

# API Reference Guide for Amazon SageMaker

## Overview

Amazon SageMaker provides APIs, SDKs, and a command line interface that you can use to create and manage notebook instances and train and deploy models.

- [Amazon SageMaker Python SDK](#) — Recommended!
- [Amazon SageMaker API Reference](#)
- [Amazon Augmented AI API Reference](#)
- [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](#)
- [AWS SDK for Python \(Boto\)](#)
- [AWS SDK for Ruby](#)
- [Amazon SageMaker Spark](#)

You can also get code examples from the Amazon SageMaker example notebooks GitHub repository.

- [Example notebooks](#)

## Programming Model for Amazon SageMaker

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 multiple AWS SDK languages (listed in the overview) and the [Amazon SageMaker Python SDK](#), a high-level Python library that you can use in your code to start model training jobs and deploy the resulting models.

- **The Amazon SageMaker Python SDK**—This 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 artifact, an endpoint configuration, then deploys the model on an endpoint.
  - If you use a custom framework script for model training, you call the `fit()` method. The method creates a .gzip file of your script, uploads it to an Amazon S3 location, and then runs it for model training, and other tasks. For more information, see [Use Machine Learning Frameworks with Amazon SageMaker \(p. 466\)](#).
- **The AWS SDKs** – The SDKs provide methods that correspond to the Amazon SageMaker API (see [Operations](#)). 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 the preceding list in the overview.

In [Get Started with Amazon SageMaker \(p. 22\)](#), you train and deploy a model using an algorithm provided by Amazon SageMaker. That exercise shows how to use both of these libraries. For more information, see [Get Started with Amazon SageMaker \(p. 22\)](#).

- **Integrate Amazon SageMaker into your Apache Spark workflow**—Amazon SageMaker provides a library for calling its APIs from Apache Spark. With it, you can use Amazon SageMaker-based estimators in an Apache Spark pipeline. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 466\)](#).

# Document History for Amazon SageMaker

| update-history-change                                         | update-history-description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | update-history-date |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| <a href="#">New features re:Invent 2019</a>                   | <a href="#">Amazon SageMaker Studio</a> , <a href="#">Amazon SageMaker Experiments</a> , <a href="#">Amazon SageMaker Autopilot</a> , <a href="#">Amazon SageMaker Debugger</a> , <a href="#">Amazon SageMaker Model Monitor</a>                                                                                                                                                                                                                                                                                                                                                                              | December 3, 2019    |
| <a href="#">New features re:Invent 2018</a>                   | <a href="#">Amazon SageMaker Ground Truth</a> , <a href="#">Using Elastic Inference in Amazon SageMaker</a> , <a href="#">Amazon SageMaker Resources in AWS Marketplace</a> , <a href="#">Amazon SageMaker Inference Pipelines</a> , <a href="#">Amazon SageMaker Neo</a> , <a href="#">Manage Machine Learning Experiments with Search</a> , <a href="#">Use Reinforcement Learning in Amazon SageMaker</a> , <a href="#">Associating Git Repositories with Amazon SageMaker Notebook Instances</a> , <a href="#">Semantic Segmentation</a> , <a href="#">Using Augmented Manifest Files in TrainingJobs</a> | November 28, 2018   |
| <a href="#">Configuring notebook instances</a>                | You can use shell scripts to configure notebook instances when you create or start them. For more information, see <a href="#">Customize a Notebook Instance</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                            | May 1, 2018         |
| <a href="#">Disable direct internet access</a>                | You can now disable direct internet access for notebook instances. For more information, see <a href="#">Notebook Instances Are Enabled with Internet Access by Default</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                 | March 15, 2018      |
| <a href="#">Application Auto Scaling support</a>              | Amazon SageMaker now supports Application Auto Scaling for production variants. For information, see <a href="#">Automatically Scaling Amazon SageMaker SageMaker Models</a>                                                                                                                                                                                                                                                                                                                                                                                                                                  | February 28, 2018   |
| <a href="#">TensorFlow 1.5 and MXNet 1.0 support (p. 864)</a> | Amazon SageMaker Deep Learning containers now support TensorFlow 1.5 and Apache MXNet 1.0.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | February 27, 2018   |

|                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                  |
|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <a href="#">BlazingText algorithm</a>                           | Amazon SageMaker now supports the <a href="#">BlazingText</a> algorithm.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | January 18, 2018 |
| <a href="#">KMS encryption support for training and hosting</a> | Amazon SageMaker now supports KMS encryption for hosting instances and training model artifacts at rest. You can specify a AWS Key Management Service key that Amazon SageMaker uses to encrypt data on the storage volume attached to a hosting endpoint by using the <code>KmsKeyId</code> request parameter in a call to <a href="#">CreateEndpointConfig</a> . You can specify an AWS KMS key that Amazon SageMaker uses to encrypt training model artifacts at rest by setting the <code>KmsKeyId</code> field of the <a href="#">OutputDataConfig</a> object you use to configure your training job. | January 17, 2018 |
| <a href="#">CloudTrail support</a>                              | Amazon SageMaker now supports <a href="#">logging with AWS CloudTrail</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | January 11, 2018 |
| <a href="#">DeepAR Forecasting algorithm</a>                    | Amazon SageMaker now supports the <a href="#">DeepAR</a> algorithm for time series forecasting.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | January 8, 2018  |

# AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.