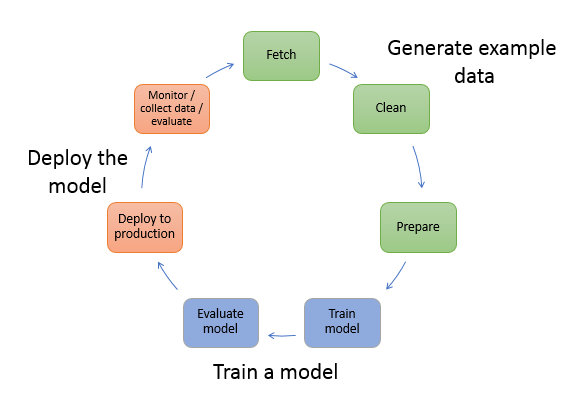
**Machine Learning with Amazon SageMaker – Fast Walk**



Explore, Analyze, and Process Data

To preprocess data use one of the following methods:

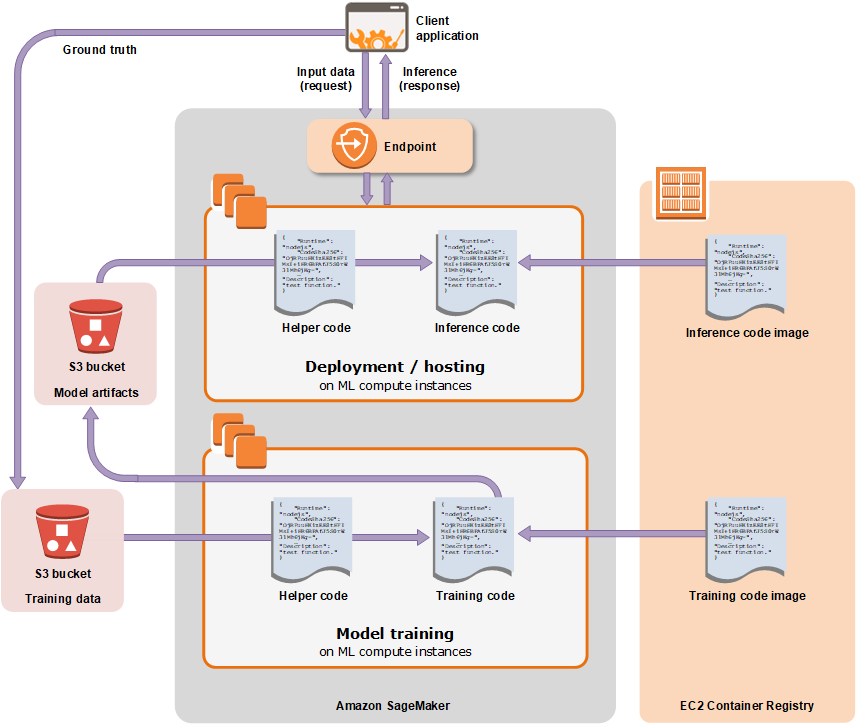
* Use a Jupyter notebook on a Amazon SageMaker notebook instance. You can also use the notebook instance to do the following:
* Write code to create model training jobs
* Deploy models to SageMaker hosting
* Test or validate your models
* You can use a model to transform data by using SageMaker batch transform.

What Is Fairness and Model Explainability for Machine Learning Predictions?

**Amazon SageMaker Clarify** helps improve your machine learning (ML) models by detecting potential bias and helping explain the predictions that models make.

The fairness and explainability functionality provided by SageMaker Clarify provides components that help AWS customers build less biased and more understandable machine learning models

Train a Model with Amazon SageMaker



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

To train a model in 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 SageMaker to use for model training. Compute resources are ML compute instances that are managed by 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 [Docker Registry Paths for SageMaker Built-in Algorithms](https://docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-algo-docker-registry-paths.html).

When you create a training job with the API, SageMaker replicates the **entire dataset** on ML compute instances **by default**. To make 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](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_S3DataSource.html).

Deploy a Model in Amazon SageMaker

After you train your model, you can deploy it using Amazon SageMaker to get predictions in any of the following ways:

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

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

* **Create a model in SageMaker**—By creating a model, you tell 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](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateModel.html)** API.

**Note**

The S3 bucket where the model artifacts are stored **must be in the same region** as the model that you are creating.

* **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 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, SageMaker launches them in multiple Availability Zones. This ensures continuous availability. SageMaker manages deploying the instances. For more information, see the [CreateEndpointConfig](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateEndpointConfig.html) API.

* **Create an HTTPS endpoint**—Provide the endpoint configuration to 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](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateEndpoint.html)** API. To get inferences from the model, client applications send requests to the SageMaker Runtime HTTPS endpoint. For more information about the API, see the **[InvokeEndpoint](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_InvokeEndpoint.html)** API.

**Note**

When you create an endpoint, 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.

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

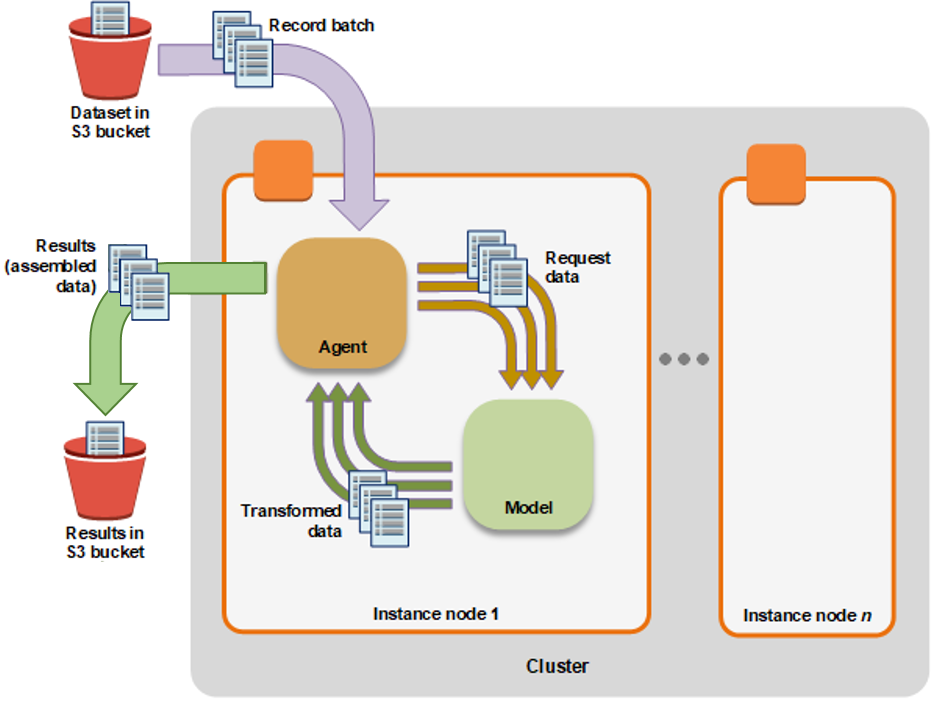
* Typically, a client application sends requests to the SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.
* You can deploy a model trained with SageMaker to your own deployment target.
* You can deploy multiple variants of a model to **the same 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](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_ProductionVariant.html).
* You can configure a ProductionVariant to use Application Auto Scaling
* 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**. SageMaker implements the changes without any downtime. For more information see, [UpdateEndpoint](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_UpdateEndpoint.html) and [UpdateEndpointWeightsAndCapacities](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_UpdateEndpointWeightsAndCapacities.html).
* 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.

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 SageMaker hosted endpoints provide



To perform a batch transform, create a batch transform job using either the 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 SageMaker to use for the transform job. *Compute resources* are machine learning (ML) compute instances that are managed by SageMaker.
* The path to the S3 bucket where you want to store the output of the job.
* The name of the SageMaker model that you want to use to create inferences. You must use a model that you have already created either with the [CreateModel](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateModel.html) operation or the console.

Monitoring a Model in Production

After you deploy a model into your production environment, use Amazon SageMaker model monitor to continuously monitor the quality of your machine learning models in real time. Amazon **CloudWatch** model monitor enables you to set up an automated alert triggering system when there are deviations in the model quality, such as data drift and anomalies. Amazon **CloudWatch Logs** collects log files of monitoring the model status and notifies when the quality of your model hits certain thresholds that you preset. AWS **CloudTrail** stores the log files to an Amazon S3 bucket you specify. Early and pro-active detection of model deviations through AWS model monitor products enables you to take prompt actions to maintain and improve the quality of your deployed model.

Use Amazon SageMaker Notebook Instances

An *Amazon SageMaker notebook instance* is a ML compute instance running the Jupyter Notebook App. 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 SageMaker hosting, and test or validate your models.

To create a notebook instance, use either the SageMaker console or the  [CreateNotebookInstance](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateNotebookInstance.html) API.

A best practice when using a 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 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.

**Associate Git Repositories with 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.

**Notebook Instance Metadata**

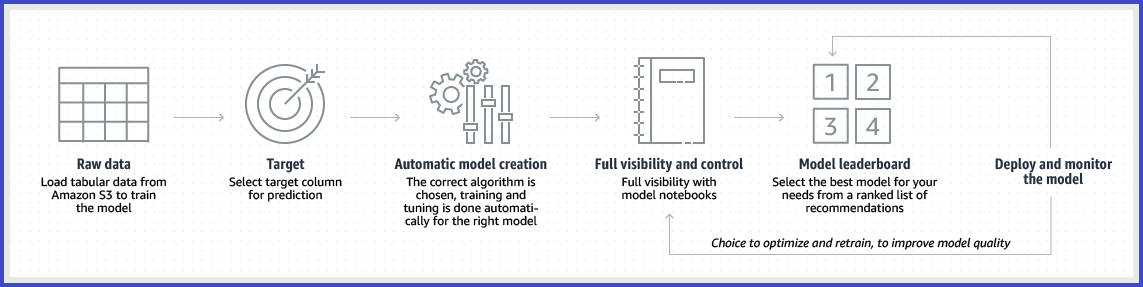
When you create a notebook instance, Amazon SageMaker creates a JSON file on the instance at the location /opt/ml/metadata/resource-metadata.json that contains the ResourceName and ResourceArn of the notebook instance.

**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 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.

 Automate model development with Amazon SageMaker Autopilot

Amazon SageMaker Autopilot is a feature-set that automates key tasks of an automatic machine learning (**AutoML**) process. It explores your data, selects the algorithms relevant to your problem type, and prepares the data to facilitate model training and tuning. It simplifies your machine learning experience by automating these key tasks that constitute an AutoML process. It ranks all of the optimized models tested by their performance. It finds the best performing model that you can deploy at a fraction of the time normally required.



Autopilot currently supports **regression** and **binary** and **multiclass classification**. It also only supports **tabular data** formatted in files with comma-separated values.

**Autopilot algorithm support**

Autopilot supports three types of machine learning algorithms to address machine learning problems:

* [Linear Learner Algorithm](https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html): a supervised learning algorithms used for solving either classification or regression problems.
* [XGBoost Algorithm](https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost.html): a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models.
* Deep Learning Algorithm: multilayer perceptron (MLP), a feedforward artificial neural network that can handle data that is not linear separable.

**Note**

You do not need to specify an algorithm to use for your machine learning problem. Autopilot automatically selects the appropriate algorithm to train.

Label Data

To train a machine learning model, you need a large, high-quality, labeled dataset. You can label your data using **Amazon SageMaker Ground Truth**. Choose from one of the Ground Truth [built-in task types](https://docs.aws.amazon.com/sagemaker/latest/dg/sms-task-types.html) or create your own [custom labeling workflow](https://docs.aws.amazon.com/sagemaker/latest/dg/sms-custom-templates.html). To improve the accuracy of your data labels and reduce the total cost of labeling your data, use Ground Truth enhanced data labeling features like [automated data labeling](https://docs.aws.amazon.com/sagemaker/latest/dg/sms-automated-labeling.html) and [annotation consolidation](https://docs.aws.amazon.com/sagemaker/latest/dg/sms-annotation-consolidation.html).

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.

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.

Prepare and Analyze Datasets

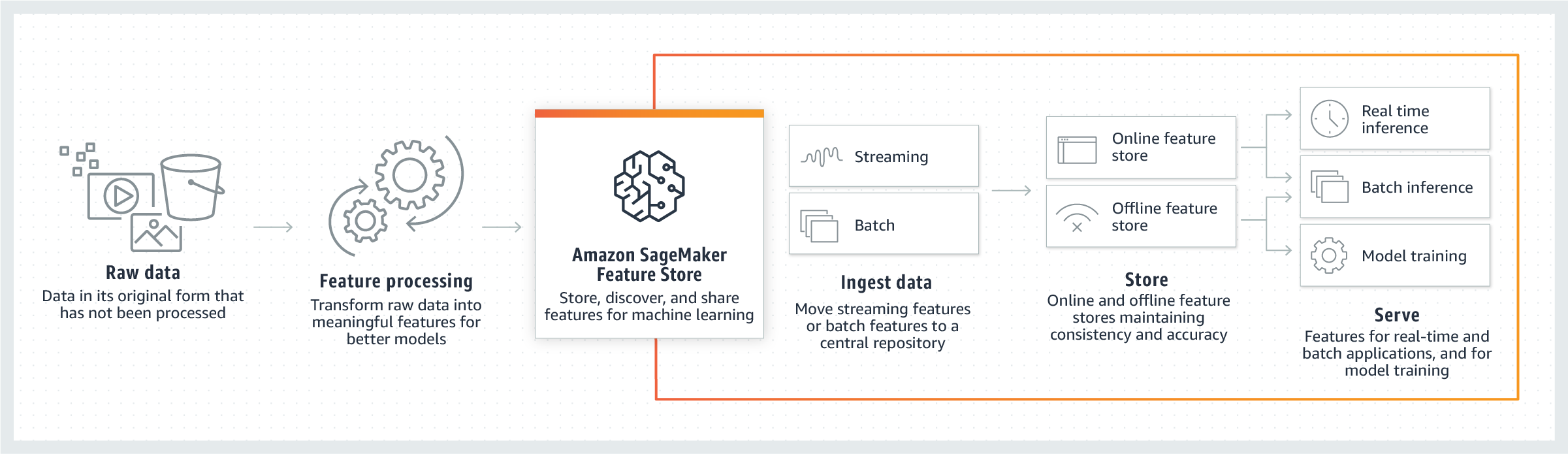
Import, prepare, transform, visualize and analyze data with **Amazon SageMaker Data Wrangler**. You can integrate Data Wrangler into your machine learning workflows to simplify and streamline data pre-processing and feature engineering **using little to no coding**. You can also add your own Python scripts and transformations to customize your data prep workflow.

Amazon SageMaker Data Wrangler (Data Wrangler) is **a feature of SageMaker Studio** that provides an end-to-end solution to import, prepare, transform, featurize, and analyze data. You can integrate a Data Wrangler data flow into your machine learning (ML) workflows to simplify and streamline data pre-processing and feature engineering using little to no coding. You can also add your own Python scripts and transformations to customize a Data Wrangler data prep workflow.

To analyze data and evaluate machine learning models on Amazon SageMaker, use **Amazon SageMaker Processing**. With Processing, you can use a simplified, managed experience on SageMaker to run your data processing workloads, such as feature engineering, data validation, model evaluation, and model interpretation. You can also use the Amazon SageMaker Processing APIs during the experimentation phase and after the code is deployed in production to evaluate performance.

Create, Store, and Share Features with Amazon SageMaker Feature Store

The machine learning (ML) development process often begins with extracting data signals also known as *features* from data to train ML models. **Amazon SageMaker Feature Store** makes it easy for data scientists, machine learning engineers, and general practitioners to create, share, and manage features for machine learning (ML) development. Feature Store accelerates this process by reducing repetitive data processing and curation work required to convert raw data into features for training an ML algorithm.



Training a Model

**Choose an Algorithm**

There are currently three basic paradigms for machine learning used to address various problem types:

* [Supervised learning](https://docs.aws.amazon.com/sagemaker/latest/dg/algorithms-choose.html#algorithms-choose-supervised-learning)
* [Unsupervised learning](https://docs.aws.amazon.com/sagemaker/latest/dg/algorithms-choose.html#algorithms-choose-unsupervised-learning)
* [Reinforcement learning](https://docs.aws.amazon.com/sagemaker/latest/dg/algorithms-choose.html#algorithms-choose-reinforcement-learning)

Machine learning paradigms use algorithmic methods to address their various problem types. The algorithms provide recipes for solving these problems.

**Choose an algorithm implementation**

After choosing an algorithm, you must decide which implementation of it you want to use. Amazon SageMaker supports three implementation options that require increasing levels of effort.

* **Built-in algorithms** **require the least effort** and scale if the data set is large and significant resources are needed to train and deploy the model.
* If there is no built-in solution that works, try to develop one that uses **pre-made images for machine and deep learning frameworks** for supported frameworks such as Scikit-Learn, TensorFlow, PyTorch, MXNet, or Chainer.
* If you need to run custom packages or use any code which isn’t a part of a supported framework or available via PyPi, then you need to build **your own custom Docker image** that is configured to install the necessary packages or software. **The custom image must also be pushed to an online repository like the Amazon Elastic Container Service.**

**Use a built-in algorithm**

When choosing an algorithm for your type of problem and data, the easiest option is to use one of Amazon SageMaker's built-in algorithms. These built-in algorithms come with two major benefits.

* The built-in algorithms require no coding to start running experiments. The only inputs you need to provide are the data, hyperparameters, and compute resources. This allows you to run experiments more quickly, with less overhead for tracking results and code changes.
* The built-in algorithms come with parallelization across multiple compute instances and GPU support right out of the box for all applicable algorithms (some algorithms may not be included due to inherent limitations). If you have a lot of data with which to train your model, most built-in algorithms can easily scale to meet the demand. Even if you already have a pre-trained model, it may still be easier to use its corollary in SageMaker and input the hyper-parameters you already know than to port it over, using script mode on a supported framework.

Problem types for the basic machine learning paradigms

**Supervised learning**

* Binary classification
* Multiclass classification
* Regression

**Unsupervised learning**

Principal component and cluster analyses are two of the main methods commonly deployed for **preprocessing data.**

Here is a short list of problem types that can be addressed by unsupervised learning:

* **Dimension reduction** is typically part of a data exploration step used to determine the most relevant features to use for model construction. The idea is to transform data from a high-dimensional, sparsely populated space into a low-dimensional space that retains most significant properties of the original data. This provides relief for the curse of dimensionality that can arise with sparsely populated, high-dimensional data on which statistical analysis becomes problematic. It can also be used to help understand data, reducing high-dimensional data to a lower dimensionality that can be visualized.
* **Cluster analysis** is a class of techniques that are used to classify objects or cases into groups called clusters. 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 features or attributes that you want the algorithm to use to determine similarity, select a distance function to measure similarity, and specify the number of clusters to use in the analysis.
* **Anomaly detection** is the identification of rare items, events, or observations in a data set which raise suspicions because they differ significantly from the rest of the data. The identification of anomalous items can be used, for example, to detect bank fraud or medical errors. Anomalies are also referred to as outliers, novelties, noise, deviations, and exceptions.
* **Density estimation** is the construction of estimates of unobservable underlying probability density functions based on observed data. A natural use of density estimates is for data exploration. Density estimates can discover features such as skewness and multimodality in the data. The most basic form of density estimation is a rescaled histogram.

**Common Data Formats for Training**

**Amazon SageMaker requires that a CSV file does not 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 **'content\_type=text/csv;label\_size=0'**.

Using RecordIO Format

**Most Amazon SageMaker algorithms work best when you use the optimized protobuf [recordIO](https://mxnet.apache.org/api/architecture/note_data_loading" \l "data-format) data format for training**. Using this format allows you to take advantage of ***Pipe mode***. In *Pipe mode*, your training job streams data directly from Amazon Simple Storage Service (Amazon S3). Streaming can provide faster start times for training jobs and better throughput. This is in contrast to *File mode*, in which your data from Amazon S3 is stored on the training instance volumes. File mode uses disk space to store both your final model artifacts and your full training dataset. By streaming in your data directly from Amazon S3 in Pipe mode, you reduce the size of Amazon Elastic Block Store volumes of your training instances. Pipe mode needs only enough disk space to store your final model artifacts.

XGBoost, for example, only supports text/csv from this list, but also supports text/libsvm.

**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.

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 Hyperparameters (Required)

The hyperparameters for the BlazingText algorithm depend on which mode you use: Word2Vec (unsupervised) and Text Classification (supervised).

|  |  |
| --- | --- |
| **Parameter Name** | **Description** |
| mode | The Word2vec architecture used for training.  **Required**  Valid values: batch\_skipgram, skipgram, or cbow |

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.

The BlazingText Text Classification algorithm (supervised mode), also reports on a single metric during training: the validation:accuracy.

**DeepAR Algorithm**

Time-series forecasting

DeepAR Hyperparameters (Required)

|  |
| --- |
| context\_length |

epochs

|  |
| --- |
| prediction\_length |
| time\_freq |

**Factorization Machines Algorithm**

A factorization machine is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. Factorization machines are a good choice for tasks dealing with **high dimensional sparse datasets**, such as click prediction and item recommendation.

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.

Factorization Machines Hyperparameters

|  |
| --- |
| feature\_dim |
| num\_factors |
| predictor\_type |

**Image Classification Algorithm**

The recommended input format for the Amazon SageMaker image classification algorithms is Apache MXNet [RecordIO](https://mxnet.apache.org/api/faq/recordio). However, you can also use raw images in **.JPG or .PNG** format.

When using the RecordIO content type in pipe mode, you must set the S3DataDistributionType of the S3DataSource to FullyReplicated

Image Classification Hyperparameters

|  |
| --- |
| num\_classes |
| num\_training\_samples |

Metrics Computed by the Image Classification Algorithm

|  |
| --- |
| validation:accuracy |

**IP Insights**

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.

SageMaker IP insights ingests historical data as **(entity, IPv4 Address) pairs** and learns the IP usage patterns of each entity.

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**.

IP Insights Hyperparameters

|  |
| --- |
| num\_entity\_vectors |
| vector\_dim |

Metrics Computed by the IP Insights Algorithm

|  |
| --- |
| validation:discriminator\_auc |

**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.

Both recordIO-wrapped-protobuf and CSV formats are supported for training.

K-Means Hyperparameters

|  |
| --- |
| feature\_dim |

k

Metrics Computed by the K-Means Algorithm

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** |  |
| test:msd | **Mean squared distances** between each record in the test set and the closest center of the model. |  |
| test:ssd | **Sum of the squared distances** between each record in the test set and the closest center of the model. |  |

**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**.

For training inputs, k-NN supports text/csv and application/x-recordio-protobuf data formats.

k-NN Hyperparameters

|  |
| --- |
| feature\_dim |

k

|  |
| --- |
| predictor\_type |

sample\_size

|  |
| --- |
| dimension\_reduction\_target |

Metrics Computed by the k-NN Algorithm

***classifier*** specifies a classification task and computes test:accuracy

***regressor*** specifies a regression task and computes test:mse.

**Latent Dirichlet Allocation (LDA) Algorithm**

LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus.

LDA supports both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats. For CSV

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

LDA Hyperparameters

|  |
| --- |
| num\_topics |

feature\_dim

|  |
| --- |
| mini\_batch\_size |

Metrics Computed by the LDA Algorithm

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** |  |
| test:pwll | Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model. |  |

**Linear Learner Algorithm**

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.

**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.

Linear learner hyperparameters

|  |
| --- |
| num\_classes |
| predictor\_type |

Metrics computed by the linear learner algorithm

|  |
| --- |
| test:objective\_loss |
| test:binary\_classification\_ accuracy |
| test:binary\_f\_beta |
| test:precision |
| test:recall |

**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.

**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.

both recordIO-wrapped-protobuf (dense and sparse) and CSV file formats

NTM Hyperparameters

|  |
| --- |
| feature\_dim |
| num\_topics |

Metrics Computed by the NTM Algorithm

|  |
| --- |
| validation:total\_loss |

**Object2Vec Algorithm**

The object in each pair can be asymmetric. For example, the pairs can be (token, sequence) or (token, token) or (sequence, sequence).

Content-type: application/jsonlines

Object2Vec Hyperparameters

|  |
| --- |
| enc0\_max\_seq\_len |
| enc0\_vocab\_size |

Metrics Computed by the Object2Vec Algorithm

**Regressor Metrics Computed by the Object2Vec Algorithm**

|  |
| --- |
| test:mean\_squared\_error |

**Classification Metrics Computed by the Object2Vec Algorithm**

|  |
| --- |
| test:accuracy |
| test:cross\_entropy |

**Object Detection Algorithm**

It uses the [Single Shot multibox Detector (SSD)](https://arxiv.org/pdf/1512.02325.pdf) framework and supports two base networks: [VGG](https://arxiv.org/pdf/1409.1556.pdf) and [ResNet](https://arxiv.org/pdf/1603.05027.pdf)

The 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.

Object Detection Hyperparameters

num\_classes

|  |
| --- |
| num\_training\_samples |

Metrics Computed by the Object Detection Algorithm

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** |  |
| validation:mAP | Mean Average Precision (mAP) computed on the validation set. |  |

**Principal Component Analysis (PCA) Algorithm**

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.

Both recordIO-wrapped-protobuf and CSV formats are supported for training

PCA Hyperparameters

feature\_dim

|  |
| --- |
| mini\_batch\_size |
| num\_components |

**Random Cut Forest (RCF) Algorithm**

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set.

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

RCF Hyperparameters

|  |
| --- |
| feature\_dim |

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** |  |
| test:f1 | F1-score on the test dataset, based on the difference between calculated labels and actual labels. |  |

**Semantic Segmentation Algorithm**

The SageMaker semantic segmentation algorithm provides a fine-grained, **pixel-level** approach to developing computer vision applications.

Input data is JPG format

**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).

Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

SageMaker seq2seq expects data in RecordIO-Protobuf format with integer tokens.

Currently Amazon SageMaker seq2seq is **only** supported on **GPU** instance types

Metrics Computed by the Sequence-to-Sequence Algorithm

|  |  |  |
| --- | --- | --- |
| **Metric Name** | **Description** |  |
| validation:accuracy | Accuracy computed on the validation dataset. |  |
| validation:bleu | [Bleu﻿](https://en.wikipedia.org/wiki/BLEU) score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the bleu\_sample\_size parameter to specify the subsample. |  |
| validation:perplexity | [Perplexity](https://en.wikipedia.org/wiki/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. |  |

**XGBoost Algorithm**

For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.

Note

For CSV **training**, the algorithm assumes that the **target variable is in the first column** and that the **CSV does not have a header record**.

For CSV **inference**, the algorithm assumes that **CSV input does not have the label column**.

XGBoost Hyperparameters

|  |
| --- |
| num\_class |
| num\_round |

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.

Amazon SageMaker Debugger

Debug, monitor, and profile **training jobs** in real time, detect non-converging conditions, optimize resource utilization by eliminating bottlenecks, improve training time and reduce costs of your machine learning models using Amazon SageMaker Debugger.

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.

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**.

Bayesian Search

**Bayesian search treats hyperparameter tuning like a**[***[regression]***](https://docs.aws.amazon.com/general/latest/gr/glos-chap.html#%5Bregression%5D)**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**. Thus this method will take longer than the Random Search method.

Augmented Manifest

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.

Augmented manifests can only support **Pipe** input mode

An augmented manifest file must be formatted in [JSON Lines](http://jsonlines.org/) format.

Inference

**Deploy Models for Inference**

To set up a persistent endpoint to get predictions from your models, use Amazon SageMaker hosting services

To get predictions for an entire dataset, use SageMaker batch transform

**Deploy an Inference Pipeline**

An *inference pipeline* is a 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 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.

What is SageMaker Neo?

Neo is a capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

Neo automatically optimizes Gluon, Keras, MXNet, PyTorch, TensorFlow, TensorFlow-Lite, and ONNX models for inference on Android, Linux, and Windows machines based on processors from Ambarella, ARM, Intel, Nvidia, NXP, Qualcomm, Texas Instruments, and Xilinx. Neo is tested with computer vision models available in the model zoos across the frameworks.

Neo consists of a compiler and a runtime.

SageMaker Edge Manager

Amazon SageMaker Edge Manager provides model management for edge devices so you can optimize, secure, monitor, and maintain machine learning models on fleets of edge devices such as smart cameras, robots, personal computers, and mobile devices.

With SageMaker Edge Manager, you can optimize, run, monitor, and update machine learning models across fleets of devices at the edge.

SageMaker Edge Manager uses SageMaker Neo to optimize your models for the target hardware in one click

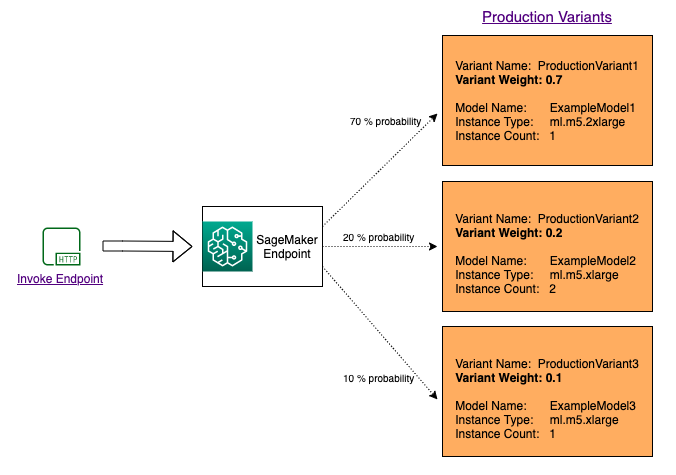
Use Amazon SageMaker Elastic Inference (EI)

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](https://docs.aws.amazon.com/sagemaker/latest/dg/how-it-works-hosting.html), 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 SageMaker [notebook instance](https://docs.aws.amazon.com/sagemaker/latest/dg/nbi.html) **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, Apache MXNet, and PyTorch.

Test models in production

In production ML workflows, data scientists and engineers frequently try to improve their models in various ways, such as by performing [Perform Automatic Model Tuning](https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html), training on additional or more-recent data, and improving feature selection. Performing A/B testing between a new model and an old model with production traffic can be an effective final step in the validation process for a new model. In A/B testing, you test different variants of your models and compare how each variant performs. If the newer version of the model delivers better performance than the previously-existing version, replace the old version of the model with the new version in production.



Deployment Best Practices

Deploy Multiple Instances Across Avalibility Zones

Using Amazon Augmented AI for Human Review

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 predictions** or a random sample of predictions.

Security

**Protect Data at Rest Using Encryption**

SageMaker uses the AWS Key Management Service (AWS KMS) to encrypt the notebooks and data. SageMaker uses AWS managed customer master keys (CMKs) by default.

**Protecting Data in Transit with Encryption**

Amazon SageMaker ensures that machine learning (ML) model artifacts and other system artifacts are encrypted in transit and at rest. Requests to the SageMaker API and console are made over a secure (SSL) connection.

**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 endpoin**t that allows access.

Best Practices for Hyperparameter Tuning

Choosing the Number of Hyperparameters

The computational complexity of a hyperparameter tuning job depends primarily on the number of hyperparameters whose range of values Amazon SageMaker has to search through during optimization. Although you can simultaneously specify up to 20 hyperparameters to optimize for a tuning job, **limiting your search to a much smaller number is likely to give you 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 get better results by limiting your search to a small range of values**. If you know that you get the best metric values within a subset of the possible range, consider limiting the range to that subset.

Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, SageMaker attempts to ﬁgure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they are in fact log-scaled, it might take some time for SageMaker to discover that fact. If you know that a hyperparameter is log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

Choosing the Best Number of Concurrent Training Jobs

When setting the resource limit [MaxParallelTrainingJobs](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_ResourceLimits.html" \l "MaxParallelTrainingJobs) for the maximum number of concurrent training jobs that a hyperparameter tuning job can launch, consider the following tradeoff. 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.