Amazon **Athena** is an interactive query service that makes it easy to analyze the data stored in Amazon S3 (or any data lake) using standard **SQL**. Athena is serverless. Athena also can be used for extract, transform and load (ETL) jobs for data processing.

**CloudTrail** could be used to enable governance, compliance and operational **auditing**.  It can also be used to create visibility into user and resource activity and also security analysis and troubleshooting.

The best solution to support both ad-hoc querying of data via SQL and also to allow that same data to be sent to an ML pipeline would be AWS Athena and AWS Glue.  AWS **Athena** can do ad-hoc queries and AWS **Glue** can do the ETL.

RDS, S3, and DynamoDB all have the ability to take snapshots.

AWS Spot instances can save up to 90% from on demand.

Both **DeepLense** and **Step** **Functions** have AWS **Lambda** embedded as part of their service.

A data lake can store structured and unstructured data, can be used for analytics and ML, and also work on data without data movement.  Additionally, it is low-cost storage.

**Time-series Analytics** is a best practice **Kinesis** streaming use case.

**Descriptive statistics** are a tool for **identifying the central tendency** and also the **measures of variability**.

Box plots, histograms and density plots are all used to show shape and **distribution** of data sets.

AWS Comprehend to get sentiment analysis.

AWS Sagemaker is designed to work with Amazon S3 data and allows for easy data visualization because it includes common Python libraries.

Validation set is a third split that can reduce overfitting. It is used after the model is trained, and allows you to select which model performs best on validation set, then it can be double-checked on the test set.

Amazon SageMaker XGBoost can train data in either CSV or LibSVM format. Label should be in the **1st column**. It should have **not** a header row.

First, we will **convert** our categorical features into numeric features, then **split** the data into training, validation and test sets.

**Early stopping** is a simple technique for preventing neural networks from training too far, and learning patterns in the training data that **can't be generalized**. **Dropout** regularization forces the learning to be spread out amongst the artificial neurons, further **preventing overfitting**. **Removing layers**, rather than adding them, might also help **prevent an overly complex model** from being created - as would using fewer features, not more.

Your automatic **hyperparameter tuning job in SageMaker** is consuming more resources than you would like, and coming at a high cost. What are TWO techniques that might **reduce this cost**?

Since the tuning process learns from each incremental step, too much concurrency can actually hinder that learning. Logarithmic ranges tend to find optimal values more quickly than linear ranges. Inference pipelines are a thing, but have nothing to do with this problem. So we are going with Use **logarithmic** **scales** on your parameter ranges & **Use less concurrency while tuning**.

**Top of Form**

**Deep learning is better suited to the imputation of categorical data**. Square footage is **numerical**, which is better served by **kNN**. While simply dropping rows of missing data or using the mean values are a lot easier, they won't result in the best results.

Bottom of Form

Top of Form

Bottom of Form

Top of Form

The SageMakerEstimator classes allow tight integration between Spark and SageMaker for several models including XGBoost, and offers the simplest solution. You can't deploy SageMaker to an EMR cluster, and XGBoost actually requires LibSVM or CSV input, not RecordIO.

Bottom of Form

Top of Form

Bottom of Form

Top of Form

SageMaker Neo is designed for compiling models using TensorFlow and other frameworks to edge devices such as Nvidia Jetson. The low latency requirement requires an edge solution, where the classification is being done within the vehicle itself and not over the air. Rekognition (which doesn't have an "edge mode," but does integrate with DeepLens) can't handle the very specific classification task of identifying different street signs and what they mean.

With Pipe input mode in Amazon SageMaker, your dataset is streamed directly to your training instances instead of being downloaded first. This means that your training jobs start sooner, finish quicker, and need less disk space. Amazon SageMaker algorithms have been engineered to be fast and highly scalable.

With Pipe input mode, your data is fed on-the-fly into the algorithm container without involving any disk I/O. This approach shortens the lengthy download process and dramatically reduces startup time. It also offers generally better read throughput than File input mode. This is because your data is fetched from Amazon S3 by a highly optimized multi-threaded background process. It also allows you to train on datasets that are much larger than the 16 TB Amazon Elastic Block Store (EBS) volume size limit.

**SMOTE** is an oversampling **technique** that generates synthetic samples from the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier.

Many developers want to implement the famous Amazon model that was used to power the "**People who bought this also bought these items**" feature on  
Amazon.com. This model is based on a method called **Collaborative Filtering**. It takes items such as movies, books, and products that were rated highly by a set of users and recommending them to other users who also gave them high ratings. This method works well in domains where explicit ratings or implicit user actions can be gathered and analyzed.

Bottom of Form

Top of Form

Bottom of Form

You can use Amazon S3 bucket policies to control access to buckets from specific virtual private cloud (VPC) (VPC) **endpoints**, or specific VPCs.

A VPC endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. The VPC endpoint routes requests to Amazon S3 and routes responses back to the VPC.

During mini-batch training of a neural network for a classification problem, a Data Scientist notices that **training accuracy** **oscillates**.  
What is the MOST likely cause of this issue?

Ans: **The learning rate is very high.**

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.

Bottom of Form

Top of Form

Bottom of Form

An **ROC** curve (receiver operating characteristic curve) is a graph showing the **performance** of a classification model at all **classification thresholds**.

How Your **Container** Should Respond to **Inference** Requests

To obtain inferences, the client application sends a POST request to the SageMaker endpoint.

SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

* SageMaker **strips all POST headers** **except** those supported by **InvokeEndpoint**. 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.

What is normalization and standardization in machine learning?

* **Normalization** typically means rescales the values into a range of [0,1].
* **Standardization** typically means rescales data to have a mean of 0 and a standard deviation of 1 (unit variance).

<https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/>

The **residual plot** will be give whether the **target** value is **overestimated or underestimated.**

A positive residual indicates that the model is **underestimating** the target (the actual target is **larger** than the **predicted** target). A negative residual indicates an **overestimation** (the actual target is **smaller** than the **predicted** target).

<https://docs.aws.amazon.com/machine-learning/latest/dg/regression-model-insights.html>

MinMaxScaler preserves the shape of the original distribution. It doesn’t meaningfully change the information embedded in the original data.

Note that MinMaxScaler doesn’t reduce the importance of outliers.

The default range for the feature returned by MinMaxScaler is 0 to 1.

**RobustScaler** transforms the feature vector by **subtracting the median** and then dividing by the interquartile range (75% value — 25% value).

**Use RobustScaler if you want to reduce the effects of outliers**, relative to MinMaxScaler.

**StandardScaler** standardizes a feature by **subtracting the mean** and then scaling to unit **variance**. Unit variance means dividing all the values by the **standard deviation.**

StandardScaler makes the mean of the distribution 0. About 68% of the values will lie be between -1 and 1.

StandardScaler does distort the relative distances between the feature values, so it’s generally my second choice in this family of transformations.

* Use MinMaxScaler as the default if you are transforming a feature. It’s non-distorting.
* You could use RobustScaler if you have outliers and want to reduce their influence. However, you might be better off removing the outliers, instead.
* Use StandardScaler if you need a relatively normal distribution.
* Use Normalizer sparingly — it normalizes sample rows, not feature columns. It can use l2 or l1 normalization.

**AUC is scale**-**invariant**. It measures how well predictions are ranked, rather than their absolute values. **AUC** is classification-threshold-**invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

**Athena** performs much more efficiently and at lower cost when using columnar format such as **Parquet or ORC**, and **Kinesis Firehose** has the ability to **convert JSON data to Parquet or ORC format** on the fly.

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](https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateHyperParameterTuningJob.html) when you create the tuning job.

In this JSON object, you specify:

* The **ranges of hyperparameters** that you want to tune.
* The **limits of the resource** that the hyperparameter tuning job can consume.
* The **objective metric for the hyperparameter** tuning job. An *objective metric* is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

An Amazon Kinesis Data Streams producer is an application that puts user data records into a Kinesis data stream (also called data ingestion). The Kinesis Producer Library (**KPL**) simplifies producer application development, allowing developers to achieve high write throughput to a Kinesis data stream.

XGBoost Hyperparameters >>> Very Important

Working with Visual Types in Amazon QuickSight

<https://docs.aws.amazon.com/quicksight/latest/user/working-with-visual-types.html>

**The KPL can incur an additional processing delay** of up to RecordMaxBufferedTime within the library (user-configurable). Larger values of RecordMaxBufferedTime results in higher packing efficiencies and better performance.

The Amazon Kinesis Data Streams API **PutRecords** call is the **best choice** for **processing in real-time** since it sends its data synchronously and does not have the processing delay of the Producer Library.

Using ORC files improve performance when Hive is reading, writing and processing data. Also, AWS Glue supports ORC for output.

As documented in Amazon Kinesis Data Streams API, titled PutRecord “the request accepts the following data in JSON format: *Data*, *ExplicitHashKey*, *PartitionKey*, *SequenceNumberForOrdering* and *StreamName*”.

How can you most effectively load data from Hadoop cluster into your SageMaker model for training?

The **SageMaker Spark Library** that makes it so you can easily train models using data frames in your Spark clusters.

Using **k-fold cross validation will randomly split your data**. By **sequentially splitting the data you preserve the time element of your observations**.

In order to get proper generalization from your data, you need to **randomize** it.

SimpleImputer transformer default strategy is **mean**.

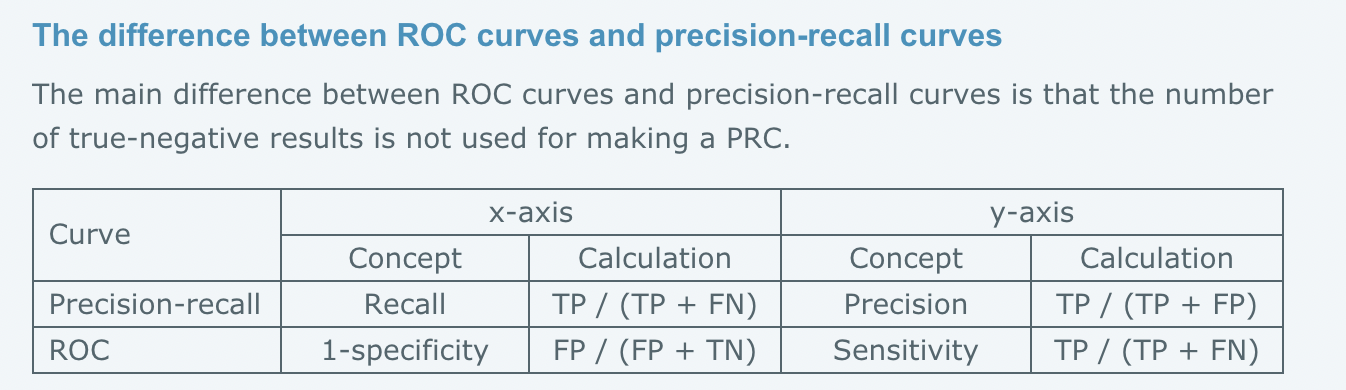
The OneHotEncoder transformer has the following methodologies you can use to drop one of the categories per feature: None, first, array.

In case of **discrete classification** problem, when using the **Linear Learner** algorithm, you set the **predictor\_type hyperparameters to binary\_classifier**.

Kinesis Data Analytics works really well for near-real time and RFC for anomaly detection.

**Random Forest** algorithm is well known to **increase the prediction accuracy** and **prevent overfitting** that occurs with a single decision tree.

The main difference between ROC curves and precision-recall (PR) curves is that the number of **true-negative results is not used for making a PR curve**.



In XGBoost hyperparameters, **num\_class** and **num\_round** used in case objective is set to ***softprob***.

The **Time Series Cross Validation** technique is the correct choice for cross **validating a time series dataset**. Time series cross validation uses forward chaining where the origin of the forecast moves forward in time. Day n is training data and day n+1 is test data.

**K-Means** is used to find **discrete groupings in data**. It is mostly used on numeric data that is **continuous**.

**Low learning rate** in image classification algorithm will make the model learn **more slowly** and be **less sensitive to outliers**.

When using k-fold for cross-validation the variance of the estimate is reduced as you increase k.

If you have relatively equal error rates for all k-fold rounds it is an indication that you have properly randomized your test data, therefore reducing the chance of bias.

In Linear Learner Algorithm, for binary classification; the model produces a **score** denoting the strength of the prediction **AND** a **predicted\_label** denoting complete or not complete.

* **For binary classification**, predicted\_label is 0 or 1, and score is a single floating point number that indicates how strongly the algorithm believes that the label should be 1.
* **For multiclass classification**, the predicted\_class will be an integer from 0 to num\_classes-1, and 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.

How would you best use AWS Glue to build the data schema needed to classify the data?

**Use Glue crawlers to crawl your data**. (the best way to build the schema for your data is to use a Glue crawler that leverages a classifier or multiple classifiers).

Key Performance Indicator

A KPI is usually a single value that relates to a particular area or function and is a reflection of how well you are doing in that area or function. This varies from business to business and function to function. Here are some popular KPIs that companies like to track:

* **Net Promoter Score (NPS)**: How likely is it for a customer to recommend your product or service to a friend?
* **Customer Profitability Score (CPS)**: How much profit does a customer bring to your business after deducting customer acquisition and customer retention costs?
* **Conversion Rate**: How many leads get converted to customers?
* **Relative Market Share**: How big is your slice of the pie compared to your competitors in the market?
* **Net Profit Margin**: The percent of your revenue which is net profit.

KPIs are best represented using KPI charts.

Amazon Kinesis Data Analytics is very efficient service for taking streams from Kinesis Data Streams and transforming them with SQL or Flink.

**Quantile Binning Transformation**

The quantile binning processor takes two inputs, a numerical variable and a parameter called *bin number*, and outputs a categorical variable. The purpose is to discover non-linearity in the variable's distribution by grouping observed values together.

A **scatter chart** shows a multiple distribution, i.e., two or three measures for a dimension.

A histogram is an accurate representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable.

Use line charts to compare changes in measured values over a period of time.

**Term Frequency – Inverse Document Frequency** determines how important a word is in a document by giving weights to words that are common and less common in the document.

The **Bag-of-Words** NLP algorithm creates tokens of the input document text and outputs a statistical depiction of the text. The statistical depiction, such as a histogram, shows the count of each word in the document.

For most data lake environments, we recommend using **user polices**, so that permissions to access data assets can also be tied to user roles and permissions for data processing and analytics services and tools that your data lake users will use.

The lambda timeout value is 3 seconds. For many Kinesis Data Firehose implementations, 3 seconds is not enough time to execute the transformation function.

Kinesis Data Firehose supports Amazon S3 server-side encryption with AWS Key Management Service (AWS KMS) for encrypting delivered data in Amazon S3.

In Kinesis Data Firehose, you are required to create IAM role when creating delivery stream.

Use AWS Glue for data preprocessing, Save the data in Amazon S3 in **Parquet** format.

Standard scaler, it performs scaling and shifting/centering.

Max absolute scaler, this would scale each column by its max value, but would not shift/center the data.

Normalizer, this would perform row normalization.

Standard scaler is used to scale numerical data.

T-SNE is used to reduce the dimensionality of the data.

Heatmaps show relationships between two variables, but is not enough to check for overall distribution or skewness in the data.

**Scatterplot** can **help check for outliers**, **but it won’t show the skewness of the data**.

**Box Plot** and **Histogram** are **good for outliers and overall distribution and skewness of the feature.**

**Grid Search** The traditional way of performing hyperparameter optimization has been grid search or a parameter *sweep*, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. **A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set**.

Optimizers can be used to improve the training performance, and helps in convergence;

* 1. **Adam** (Adaptive Momentum) which can help the model converge faster and get out of being stuck in local minima.
  2. **Adagrad** is an algorithm for gradient-based optimization that adapts the learning rate to the parameters by performing smaller update, and in turn, helps with convergence.
  3. **RMSProp** uses a moving average of squared gradients to normalize the gradient itself, which helps with faster convergence.

***HyperparameterTuner()*** class defines interaction with Amazon SageMaker hyperparameter tuning jobs. It also supports deploying the resulting models.

**De-register** the endpoint as a scalable target, then, update the endpoint using a new endpoint configuration with the latest model Amazon S3 path, then, finally register the endpoint as a scalable target again.

Using a new endpoint configuration ONLY **will not have Auto Scaling enabled**.

**VolumeKmsKeyId** in Amazon SageMaker training job, helps in encrypting data on the training job instance storage not on Amazon S3.

Blue/Green Deployments and Canary Deployment

<https://docs.aws.amazon.com/wellarchitected/latest/machine-learning-lens/standard-deployment.html>

The Generative Adversarial Networks (**GANs**) technique **generates unique observations** that more closely resemble the real minority observations without being so similar that they are almost identical.

**SMOTE** technique creates new observations of the fraudulent. These synthetic observations **are almost identical to the original fraudulent** observations.

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 labels. It combines the results of multiple workers' 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.

**IoT Core** collects data from each shared bike, **IoT Analytics** retrieves messages from the shared bikes as they stream data, IoT Analytics also enriches the streaming data with your external data sources and sends the streaming data to your K-Means ML inference endpoint, **QuickSight** is then used to create your visualization.

**IoT Greengrass** is a service that you use to run local ML inference capabilities on connected devices.

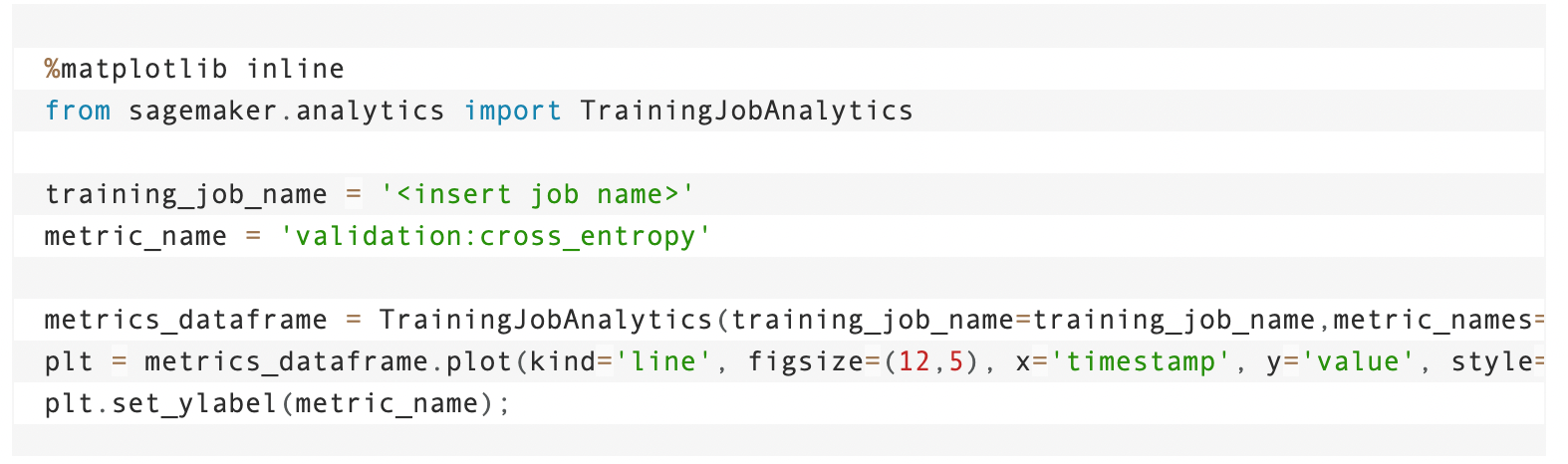
**The main advantage of** **random search is that all jobs can be run in parallel**. In contrast, **Bayesian optimization, the default tuning method, is a sequential algorithm** that learns from past trainings as the tuning job progresses. This highly limits the level of parallelism. ***The disadvantage of random search is that it typically requires running considerably more training jobs to reach a comparable model quality.***

So Bayesian Optimization approach to hyperparameter tuning results in less tuning job runs than the random search method.

Data scientists and developers can now quickly and easily access, monitor, and visualize metrics that are computed while training machine learning models on Amazon SageMaker. You can now specify the metrics you want to track by using the **AWS Management Console** for Amazon SageMaker or by using the Amazon **SageMaker Python SDK APIs**. After the model training starts, Amazon SageMaker will automatically monitor and stream the specified metrics in real time to the **Amazon CloudWatch** console for visualizing time-series curves, such as loss curves and accuracy curves. You can also access the metrics programmatically using Amazon SageMaker Python SDK APIs.

You can use the **regex patterns** that you see next to each metric to quickly parse and filter the metric values from your Amazon CloudWatch Log files created by Amazon SageMaker.

Using Amazon SageMaker Python SDK APIs to **visualize metrics**,





<https://aws.amazon.com/blogs/machine-learning/easily-monitor-and-visualize-metrics-while-training-models-on-amazon-sagemaker/>

K-Means has two valid metrics;

1. test:***ssd***
2. test:***msd***

Transformed records received by Kinesis Data Firehose from Lambda must contain the ***recordId***, ***result***, and ***data parameters***.

When you configure a Kinesis data stream as the data source of a Kinesis Data Firehose delivery stream, **Kinesis Data Firehose no longer stores the data at rest**. Instead, the data is stored in the data stream.

When you send data from your data producers to your data stream, **Kinesis Data Streams encrypts your data using an AWS Key Management Service (AWS KMS) key before storing the data at rest**. When your Kinesis Data Firehose delivery stream reads the data from your data stream, Kinesis Data Streams first decrypts the data and then sends it to Kinesis Data Firehose. Kinesis Data Firehose buffers the data in memory based on the buffering hints that you specify. It then delivers it to your destinations without storing the unencrypted data at rest.

You can use the Amazon **SageMaker model tracking** capability to search key model attributes such as hyperparameter values, the algorithm used, and tags associated with your team’s models. This SageMaker capability allows you to manage your team’s experiments at the scale of up to thousands of model experiments.

Use **customer owned KMS key**, in case your project requires encryption for regulatory **compliance** reasons.

Kinesis Firehose can invoke Lambda functions to transform incoming source data and deliver it to Amazon S3. Common transformation functions include transforming Apache Log and Syslog formats to standardized JSON and/or CSV formats. The JSON and CSV formats can then be directly queried using Amazon Athena.

Lake Formation then helps you collect and catalog data from databases and object storage, move the data into your new Amazon S3 data lake, clean and classify your data using machine learning algorithms, and secure access to your sensitive data.

When using AWS Glue FindMatches ML Transform, the labeling file must be encoded as UTF-8 without BOM (Byte Order Mark)

The inference request **serialization must be completed by your Lambda code**. The inference request is **deserialized by the algorithm** in the response to the inference request.

For a relationship between two variables, you could use the **scatter** chart. For a relationship between 3 variables a **bubble** chart is the best choice.

Factorization Machines solve a discrete recommendation.

A pairs plot is used to show the relationship between pairs of features as well as distribution of one of the variables in relation to other.

A covariance matrix shows the degree of correlation between two features.

Entropy represents the measure of randomness in your feature.

MAE (Mean Absolute Error) is a good metric for regression in case of outliers existing.

MLeap, MLib and SparkML Serving Container use Spark ML.

The decision threshold adjustment was developed to estimate the optimal decision threshold for specified **misclassification** costs and/or prior probabilities of the prevalence. When the class sizes are unequal, **a shift in a decision threshold to favor the minority class can increase minority class prediction**.

The **Bernoulli** Naïve Bayes algorithm is used in **document** classification tasks.

Your XGBoost model has high accuracy on its training set, but poor accuracy on its validation set, suggesting overfitting. the "subsample" parameter directly addresses overfitting, but other parameters such as eta, gamma, lambda, and alpha may also have an effect. Refer to

<https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost_hyperparameters.html>

**XGBoost** is a **CPU-only** algorithm, and won't benefit from the GPU's of a P3 or P2. It is also memory-bound, making **M4** a better choice than C4. It can be **parallelize**.

XGBoost Hyperparameters;

* num\_class; number of classes (Required if **objective** is set to *multi:softmax* or *multi:softprob*)
* num\_sample; The number of rounds to run the training (Required)
* alpha; **L1 Regularization**, increasing this value makes models more conservative
* eta; step size, prevent overfitting
* eval\_metric; rmse for regression & error for classification & map for ranking
* gamma; min loss reduction
* lambda; L2 regularization
* subsample; prevents overfitting

A "**vanishing** **gradient**" results from multiplying together many small derivates of the sigmoid activation function in multiple layers. **ReLU** does not have a small derivative, and avoids this problem.

**Ordinal Encoder** transform is better choice to fill missing value for feature, that has ordinal value, like rating H(High)>M(Medium)>L(Low)>N(No) or  size data L(Large)>M(Medium)>S(Small)

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.

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)

When using a **custom** algorithm, you need to ensure that the **desired** **metrics** are emitted to **stdout** **output**. You also need to include the metric definition and regex expression for the metric in the stdout output when defining the training job.

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.