**Machine Learning - Initialize Model**

**Create and use machine learning models in Machine Learning Studio (Classic)**

The typical workflow for machine learning includes these phases:

* Choose a suitable algorithm and set initial options.
* Train the model by using compatible data.
* Create predictions by using new data based on the patterns in the model.
* Evaluate the model to determine whether the predictions are accurate, the amount of error, and whether overfitting occurs.

Machine Learning Studio (classic) supports a flexible, customizable framework for machine learning. Each task in this process is performed by a specific type of module. Modules can be modified, added, or removed without breaking the rest of your experiment.

Use the modules in this category to select an initial algorithm. Then, configure detailed parameters based on the specific model type. You can then apply this model specification to a set of data.

**Creating models**

Machine Learning provides many state-of-the art machine learning algorithms to help you build analytical models. Each algorithm is packaged in its own module. To create a customized model:

1. Choose a model by category.

Algorithms are grouped by specific types of predictive tasks. Examples include regression, classification, and image recognition. Your first task is to identify the general category of machine learning task to perform, and then to select an algorithm.

1. Configure algorithm parameters.

Use the Properties pane in each module to set parameters. Parameters control how the model learns from data.

1. Train the model on data.

After you configure the model, connect a dataset. Then, use one of the training modules to run data through the algorithms that you want to use.

You can use Tune Model Hyperparameters to iterate over all possible parameters and determine the optimal configuration for your task and data.

1. Predict, score, or evaluate.

After you build and train a model, typically your next step is to use one of the scoring modules to generate predictions based on the model.

You can use the modules for model evaluation to measure the accuracy of the model based on the scores that you generate.

**Anomaly Detection**

This introduces the modules provided in Machine Learning Studio (classic) for anomaly detection. Anomaly detection encompasses many important tasks in machine learning:

* Identifying transactions that are potentially fraudulent.
* Learning patterns that indicate that a network intrusion has occurred.
* Finding abnormal clusters of patients.
* Checking values entered into a system.

Because anomalies are rare events by definition, it can be difficult to collect a representative sample of data to use for modeling. The algorithms included in this category have been especially designed to address the core challenges of building and training models by using imbalanced data sets.

**Anomaly detection modules**

Machine Learning Studio (classic) provides the following modules that you can use to create an anomaly detection model. Just drag the module into your experiment to begin working with the model.

* One-Class Support Vector Machine
* PCA-Based Anomaly Detection

After setting model parameters, you must train the model by using a labeled data set and the Train Anomaly Detection Model training module. The result is a trained model that you can use to test new data. To do this, use the all-purpose Score Model module.

***Time Series Anomaly*** ***Detection*** is a new module that's a bit different from the other anomaly detection models. The Time Series Anomaly Detection module is designed for time series data. It's intended to be used to analyze trends over time. The algorithm identifies potentially anomalous trends in the time series data. It flags deviations from the trend's direction or magnitude.

**One-Class Support Vector Machine**

This module is particularly useful in scenarios where you have a lot of "normal" data and not many cases of the anomalies you are trying to detect. For example, if you need to detect fraudulent transactions, you might not have many examples of fraud that you could use to train a typical classification model, but you might have many examples of good transactions.

You use the One-Class Support Vector Model module to create the model, and then train the model using the Train Anomaly Detection Model. The dataset that you use for training can contain all or mostly normal cases.

You can then apply different metrics to identify potential anomalies. For example, you might use a large dataset of good transactions to identify cases that possibly represent fraudulent transactions.

Support vector machines (SVMs) are supervised learning models that analyze data and recognize patterns, and that can be used for both classification and regression tasks.

Typically, the SVM algorithm is given a set of training examples labeled as belonging to one of two classes. An SVM model is based on dividing the training sample points into separate categories by as wide a gap as possible, while penalizing training samples that fall on the wrong side of the gap. The SVM model then makes predictions by assigning points to one side of the gap or the other.

Sometimes oversampling is used to replicate the existing samples so that you can create a two-class model, but it is impossible to predict all the new patterns of fraud or system faults from limited examples. Moreover, collection of even limited examples can be expensive.

Therefore, in one-class SVM, the support vector model is trained on data that has only one class, which is the “normal” class. It infers the properties of normal cases and from these properties can predict which examples are unlike the normal examples. This is useful for anomaly detection because the scarcity of training examples is what defines anomalies: that is, typically there are very few examples of the network intrusion, fraud, or other anomalous behavior.

**How to configure One-Class SVM**

1. Add the One-Class Support Vector Model module to your experiment in Studio (classic). You can find the module under Machine Learning - Initialize, in the Anomaly Detection category.
2. Double-click the One-Class Support Vector Model module to open the Properties pane.
3. For Create trainer mode, select an option that indicates how the model should be trained:

* ***Single Parameter:*** Use this option if you know how you want to configure the model, and provide a specific set of values as arguments.
* ***Parameter Range:*** Use this option if you are not sure of the best parameters, and want to perform a parameter sweep to find the optimal configuration.

1. η: Type a value that represents the upper bound on the fraction of outliers. This parameter corresponds to the nu-property described in this paper. The nu-property lets you control the trade-off between outliers and normal cases.
2. ε (epsilon): Type a value to use as the stopping tolerance. The stopping tolerance, affects the number of iterations used when optimizing the model, and depends on the stopping criterion value. When the value is exceeded, the trainer stops iterating on a solution.
3. Connect a training dataset, and one of the training modules:

* If you set Create Trainer mode to Single Parameter, use the Train Anomaly Detection Model module.
* If you set Create Trainer mode to Parameter Range, use the Tune Model Hyperparameters module.

1. Run the experiment.

**Results**

The module returns a trained anomaly detection model. You can either save the model in your workspace, or you can connect the Score Model module and use the trained model to detect possible anomalies. If you trained the model using a parameter sweep, make a note of the optimal parameter settings to use when configuring a model for use in production.

**PCA-Based Anomaly Detection**

To create an anomaly detection model based on Principal Component Analysis (PCA). This module helps you build a model in scenarios where it is easy to obtain training data from one class, such as valid transactions, but difficult to obtain sufficient samples of the targeted anomalies.

For example, to detect fraudulent transactions, very often you don't have enough examples of fraud to train on, but have many examples of good transactions. The PCA-Based Anomaly Detection module solves the problem by analyzing available features to determine what constitutes a "normal" class, and applying distance metrics to identify cases that represent anomalies. This let you train a model using existing imbalanced data.

**Principal Component Analysis**

Principal Component Analysis, which is frequently abbreviated to PCA, is an established technique in machine learning. PCA is frequently used in exploratory data analysis because it reveals the inner structure of the data and explains the variance in the data.

PCA works by analyzing data that contains multiple variables. It looks for correlations among the variables and determines the combination of values that best captures differences in outcomes. These combined feature values are used to create a more compact feature space called the principal components.

For anomaly detection, each new input is analyzed, and the anomaly detection algorithm computes its projection on the eigenvectors, together with a normalized reconstruction error. The normalized error is used as the anomaly score. The higher the error, the more anomalous the instance is.

**How to configure PCA Anomaly Detection**

1. Add the PCA-Based Anomaly Detection module to your experiment in Studio (classic). You can find this module under Machine Learning, Initialize Model, in the Anomaly Detection category.
2. In the Properties pane for the PCA-Based Anomaly Detection module, click the Training mode option, and indicate whether you want to train the model using a specific set of parameters, or use a parameter sweep to find the best parameters.

* ***Single Parameter:*** Select this option if you know how you want to configure the model and provide a specific set of values as arguments.
* ***Parameter Range:*** Select this option if you are not sure of the best parameters and want to use a parameter sweep, using the Tune Model Hyperparameters module. The trainer iterates over a range of settings you specify, and determines the combination of settings that produces the optimal results.

1. Number of components to use in PCA, Range for number of PCA components: Specify the number of output features, or components, that you want to output.

The decision of how many components to include is an important part of experiment design using PCA. General guidance is that you should not include the same number of PCA components as there are variables. Instead, you should start with a smaller number of components and increase them until some criteria is met.

If you are unsure of what the optimum value might be, we recommend that you train the anomaly detection model using the Parameter Range option.

The best results are obtained when the number of output components is less than the number of feature columns available in the dataset.

1. Specify the amount of oversampling to perform during randomized PCA training. In anomaly detection problems, imbalanced data makes it difficult to apply standard PCA techniques. By specifying some amount of oversampling, you can increase the number of target instances.

If you specify 1, no oversampling is performed. If you specify any value higher than 1, additional samples are generated to use in training the model.

There are two options, depending on whether you are using a parameter sweep or not:

* **Oversampling parameter for randomized PCA**: Type a single whole number that represents the ratio of oversampling of the minority class over the normal class. (Available when using the Single parameter training method.)
* **Range for the oversampling parameter used in randomized PCA:** Type a series of numbers to try, or use the Range Builder to select values using a slider. (Available only when using the Parameter range training method.)

1. Enable input feature mean normalization: Select this option to normalize all input features to a mean of zero. Normalization or scaling to zero is generally recommended for PCA because the goal of PCA is to maximize variance among variables. This option is selected by default. Deselect this option if values have already been normalized using a different method or scale.
2. Connect a tagged training dataset, and one of the training modules:

* If you set the Create trainer mode option to Single Parameter, use the Train Anomaly Detection Model module.
* If you set the Create trainer mode option to Parameter Range, use the Tune Model Hyperparameters module.

1. Run the experiment, or select the module and click Run selected.

**Results**

When training is complete, you can either save the trained model, or connect it to the Score Model module to predict anomaly scores.

To evaluate the results of an anomaly detection models requires some additional steps:

1. Ensure that a score column is available in both datasets

If you try to evaluate an anomaly detection model and get the error, "There is no score column in scored dataset to compare", it means you are using a typical evaluation dataset that contains a label column but no probability scores. You need to choose a dataset that matches the schema output for anomaly detection models, which includes a Scored Labels and Scored Probabilities column.

1. Ensure that label columns are marked

Sometimes the metadata associated with the label column is removed in the experiment graph. If this happens, when you use the Evaluate Model module to compare the results of two anomaly detection models, you might get the error, "There is no label column in scored dataset", or "There is no label column in the scored dataset to compare".

You can avoid this error by adding the Edit Metadata module before the Evaluate Model module. Use the column selector to choose the class column, and in the Fields dropdown list, select Label.

1. Normalize scores from different model types

Predictions from the PCA anomaly detection model always are in the range [0,1]. In contrast, output from the One-Class SVM module is uncalibrated scores that are possibly unbounded.

Therefore, if you are comparing models based on different algorithms, you must always normalize scores. See the example in the Azure AI Gallery for an example of normalization among different anomaly detection models.

**Clustering**

**What is clustering?**

Clustering, in machine learning, is a method of grouping data points into similar clusters. It is also called segmentation. Over the years, many clustering algorithms have been developed. Almost all clustering algorithms use the features of individual items to find similar items. For example, you might apply clustering to find similar people by demographics. You might use clustering with text analysis to group sentences with similar topics or sentiment.

Clustering is called a non-supervised learning technique because it can be used in unlabeled data. Indeed, clustering is a useful first step for discovering new patterns, and requires little prior knowledge about how the data might be structured or how items are related. Clustering is often used for exploration of data prior to analysis with other more predictive algorithms.

**How to create a clustering model**

In Machine Learning Studio (classic), you can use clustering with either labeled or unlabeled data.

* In unlabeled data, the clustering algorithm determines which data points are closest together, and creates clusters around a central point, or centroid. You can then use the cluster ID as a temporary label for the group of data.
* If the data has labels, you can use the label to drive the number of clusters, or use the label as just another feature.

After you have configured the clustering algorithm, you train it on data by using either the Train Clustering Model or Sweep Clustering modules.

When the model is trained, use it to predict cluster membership for new data points. For example, if you have used clustering to group customers by purchasing behavior, you can use the model to predict the purchasing behavior of new customers.

**K-Means Clustering**

K-means is one of the simplest and the best known unsupervised learning algorithms, and can be used for a variety of machine learning tasks, such as detecting abnormal data, clustering of text documents, and analysis of a dataset prior to using other classification or regression methods. To create a clustering model, you add this module to your experiment, connect a dataset, and set parameters such as the number of clusters you expect, the distance metric to use in creating the clusters, and so forth. After you have configured the module hyperparameters, connect the untrained model to the Train Clustering Model or the Sweep Clustering modules to train the model on the input data that you provide. Because the K-means algorithm is an unsupervised learning method, a label column is optional.

* If your data includes a label, you can use the label values to guide selection of the clusters and optimize the model.
* If your data has no label, the algorithm creates clusters representing possible categories, based solely on the data.

In general, clustering uses iterative techniques to group cases in a dataset into clusters that contain similar characteristics. These groupings are useful for exploring data, identifying anomalies in the data, and eventually for making predictions. Clustering models can also help you identify relationships in a dataset that you might not logically derive by browsing or simple observation. For these reasons, clustering is often used in the early phases of machine learning tasks, to explore the data and discover unexpected correlations.

When you configure a clustering model using the k-means method, you must specify a target number k indicating the number of centroids you want in the model. The centroid is a point that is representative of each cluster. The K-means algorithm assigns each incoming data point to one of the clusters by minimizing the within-cluster sum of squares.

When processing the training data, the K-means algorithm begins with an initial set of randomnly chosen centroids, which serve as starting points for each cluster, and applies Lloyd's algorithm to iteratively refine the locations of the centroids. The K-means algorithm stops building and refining clusters when it meets one or more of these conditions:

* The centroids stabilize, meaning that cluster assignments for individual points no longer change and the algorithm has converged on a solution.
* The algorithm completed running the specified number of iterations.

After completing the training phase, you use the Assign Data to Clusters module to assign new cases to one of the clusters that was found by the k-means algorithm. Cluster assignment is performed by computing the distance between the new case and the centroid of each cluster. Each new case is assigned to the cluster with the nearest centroid.

**How to configure K-Means Clustering**

1. Add the K-Means Clustering module to your experiment.
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know the exact parameters you want to use in the clustering model, you can provide a specific set of values as arguments.
* Parameter Range: If you are not sure of the best parameters, you can find the optimal parameters by specifying multiple values and using the Sweep Clustering module to find the optimal configuration.The trainer iterates over multiple combinations of the settings you provided and determine the combination of values that produces the optimal clustering results.

1. For Number of Centroids, type the number of clusters you want the algorithm to begin with.The model is not guaranteed to produce exactly this number of clusters. The algorithn starts with this number of data points and iterates to find the optimal configuration.

If you are performing a parameter sweep, the name of the property changes to Range for Number of Centroids. You can use the Range Builder to specify a range, or you can type a series of numbers representing different numbers of clusters to create when initializing each model.

1. The properties Initialization or Initialization for sweep are used to specify the algorithm that is used to define the initial cluster configuration.

* ***First N:*** Some initial number of data points are chosen from the data set and used as the initial means. Also called the Forgy method.
* ***Random:*** The algorithm randomly places a data point in a cluster and then computes the initial mean to be the centroid of the cluster's randomly assigned points. Also called the random partition method.
* ***K-Means++:*** This is the default method for initializing clusters.

The K-means ++ algorithm was proposed in 2007 by David Arthur and Sergei Vassilvitskii to avoid poor clustering by the standard k-means algorithm. K-means ++ improves upon standard K-means by using a different method for choosing the initial cluster centers.

* ***K-Means++Fast***: A variant of the K-means ++ algorithm that was optimized for faster clustering.
* ***Evenly:*** Centroids are located equidistant from each other in the d-Dimensional space of n data points.
* ***Use label column:*** The values in the label column are used to guide the selection of centroids.

1. For Random number seed, optionally type a value to use as the seed for the cluster initialization. This value can have a significant effect on cluster selection. If you use a parameter sweep, you can specify that multiple initial seeds be created, to look for the best initial seed value. For Number of seeds to sweep, type the total number of random seed values to use as starting points.
2. For Metric, choose the function to use for measuring the distance between cluster vectors, or between new data points and the randomly chosen centroid. Machine Learning supports the following cluster distance metrics:

* ***Euclidean:*** The Euclidean distance is commonly used as a measure of cluster scatter for K-means clustering. This metric is preferred because it minimizes the mean distance between points and the centroids.
* ***Cosine:*** The cosine function is used to measure cluster similarity. Cosine similarity is useful in cases where you do not care about the length of a vector, only its angle.

1. For Iterations, type the number of times the algorithm should iterate over the training data before finalizing the selection of centroids. You can adjust this parameter to balance accuracy vs. training time.
2. For Assign label mode, choose an option that specifies how a label column, if present in the dataset, should be handled. Because K-means clustering is an unsupervised machine learning method, labels are optional. However, if your dataset already has a label column, you can use those values to guide selection of the clusters, or you can specify that the values be ignored.

* ***Ignore label column:*** The values in the label column are ignored and are not used in building the model.
* ***Fill missing values:*** The label column values are used as features to help build the clusters. If any rows are missing a label, the value is imputed by using other features.
* ***Overwrite from closest to center:*** The label column values are replaced with predicted label values, using the label of the point that is closest to the current centroid.

1. Train the model.

* If you set Create trainer mode to Single Parameter, add a tagged dataset and train the model by using the Train Clustering Model module.
* If you set Create trainer mode to Parameter Range, add a tagged dataset and train the model using Sweep Clustering. You can use the model trained using those parameters, or you can make a note of the parameter settings to use when configuring a learner.

**Results**

After you have finished configuring and training the model, you have a model that you can use to generate scores. However, there are multiple ways to train the model, and multiple ways to view and use the results:

**Capture a snapshot of the model in your workspace**

* If you used the Train Clustering Model module
* Right-click the Train Clustering Model module.
* Select Trained model and then click Save as Trained Model.
* If you used the Sweep Clustering module to train the model
* Right-click the Sweep Clustering module.
* Select Best Trained model and then click Save as Trained Model.

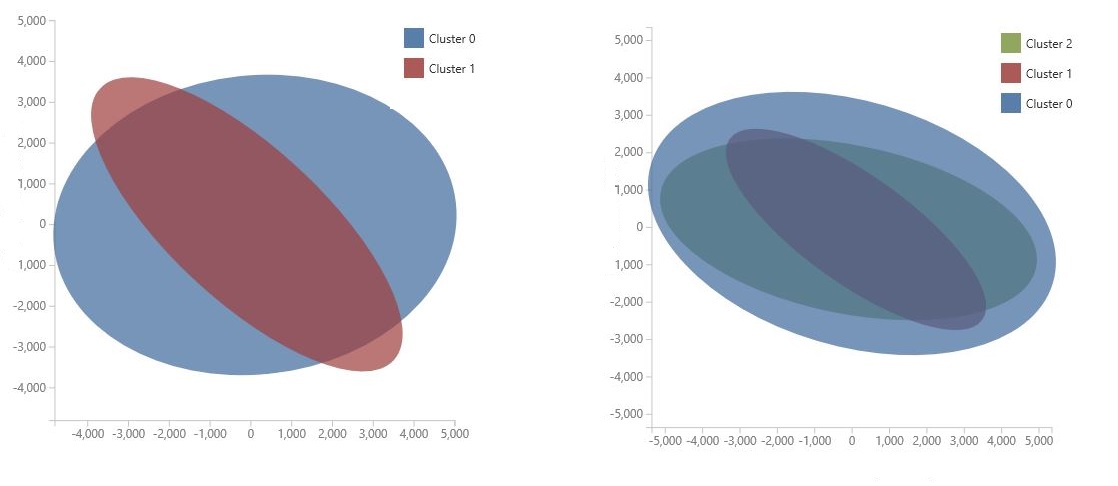
The saved model will represent the training data at the time you saved the model. If you later update the training data used in the experiment, it will not update the saved model.

**Visual representation of the clusters in the model**

* If you used the Train Clustering Model module
* Right-click the module, and select Results dataset.
* Select Visualize.
* If you used the Sweep Clustering module
* Add an instance of the Assign Data to Clusters module and generate scores using the Best Trained model.
* Right-click the Assign Data to Clusters module, select Results dataset, and select Visualize.

The chart is generated by using Principal Component Analysis, which is a technique in data science for compressing the feature space of a model. The chart shows some set of features, compressed into two dimensions, that best characterize the difference between the clusters. By visually reviewing the general size of the feature space for each cluster and how much the clusters overlap, you can get an idea of how well your model might perform.

For example, the following PCA charts represent the results from two models trained using the same data: the first was configured to output two clusters, and the second was configured to output three clusters. From these charts, you can see that increasing the number of clusters did not necessarily improve separation of the classes.



**List of data points and the clusters they belong to**

There are two options for viewing the dataset with results, depending on how you trained the model:

* If you used the Sweep Clustering module to train the model
* Use the checkbox in the Sweep Clustering module to specify whether you want to see the input data together with the results, or see just the results.
* When training is complete, right-click the module, and select Results dataset (output number 2)
* Click Visualize.
* If you used the Train Clustering Model module
* Add the Assign Data to Clusters module and connect the trained model to the left-hand input. Connect a dataset to the right-hand input.
* Add the Convert to Dataset module to your experiment and connect it to the output of Assign Data to Clusters.
* Use the checkbox in the Assign Data to Clusters module to specify whether you want to see the input data together with the results, or see just the results.
* Run the experiment, or run just the Convert to Dataset module.
* Right-click Convert to Dataset, select Results dataset, and click Visualize.

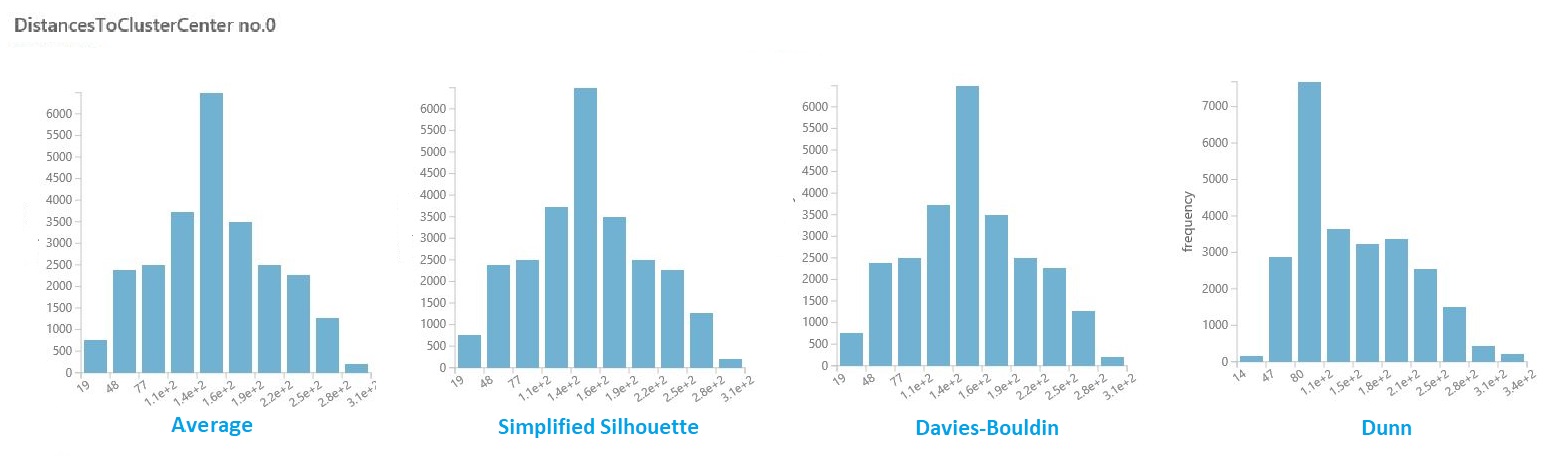
The output contains the input data columns first, if you included them, and the following columns for each row of input data:

* ***Assignment:*** The assignment is a value between 1 and n, where n is the total number of clusters in the model. Each row of data can be assigned to only one cluster.
* ***DistancesToClusterCenter no.n:*** This value measures the distance from the current data point to the centroid for the cluster. A separate column in output for each cluster in the trained model. The values for cluster distance are based on the distance metric you selected in the option, Metric for measuring cluster result. Even if you perform a parameter sweep on the clustering model, only one metric can be applied during the sweep. If you change the metric, you might get different distance values.

**Visualize intra-cluster distances**

In the dataset of results from the previous section, click the column of distances for each cluster. Studio (classic) displays a histogram that visualizes the distribution of distances for points within the cluster.

For example, the following histograms show the distribution of cluster distances from the same experiment, using four different metrics. All other settings for the parameter sweep were the same. Changing the metric resulted in a different number of clusters in one model.



**Tips for generating the best clustering model**

It is known that the seeding process used during clustering can significantly affect the model. Seeding means the initial placement of points into potental centroids.

For example, if the dataset contains many outliers, and an outlier is chosen to seed the clusters, no other data points would fit well with that cluster and the cluster could be a singleton: that is, a cluster with only one point.

There are various ways to avoid this problem:

* Use a parameter sweep to change the number of centroids and try multiple seed values.
* Create multiple models, varying the metric or iterating more.
* Use a method such as PCA to find variables that have a detrimental effect on clustering. See the Find similar companies sample for a demonstration of this technique.

In general, with clustering models, it is possible that any given configuration will result in a locally optimized set of clusters. In other words, the set of clusters returned by the model suits only the current data points, and is not generalizable to other data. If you used a different initial configuration, the K-means method might find a different, perhaps superior, configuration.

**Note**

Given a specific number of clusters (K) to find for a set of D-dimensional data points with N data points, the K-means algorithm builds the clusters as follows:

* The module initializes a K-by-D array with the final centroids that define the K clusters found.
* By default, the module assigns the first K data points in order to the K clusters.
* Starting with an initial set of K centroids, the method uses Lloyd's algorithm to iteratively refine the locations of the centroids.
* The algorithm terminates when the centroids stabilize or when a specified number of iterations are completed.
* A similarity metric (by default, Euclidean distance) is used to assign each data point to the cluster that has the closest centroid.

**Module parameters**

| **Name** | **Range** | **Type** | **Default** | **Description** |
| --- | --- | --- | --- | --- |
| Number of Centroids | >=2 | Integer | 2 | Number of Centroids |
| Metric | List (subset) | Metric | Euclidean | Selected metric |
| Initialization | List | Centroid initialization method | K-Means++ | Initialization algorithm |
| Iterations | >=1 | Integer | 100 | Number of iterations |