**Machine Learning – Train**

Training is the process of analyzing input data by using the parameters of a predefined model. From this analysis, the model learns the patterns, and saves them in the form of a trained model.

**Create and use machine learning models**

The typical workflow for machine learning includes these phases:

* Choosing a suitable algorithm, and setting initial options.
* Training the model on compatible data.
* Creating predictions by using new data, based on the patterns in the model.
* Evaluating the model to determine if the predictions are accurate, how much error there is, and if there is any overfitting.

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, which can be modified, added, or removed, without breaking the rest of your experiment.

The modules in this category support training for different types of models. During training, the data is analyzed by the machine learning algorithm. This algorithm analyzes the distribution and type of the data, compiles statistics, and creates patterns that can be used later for prediction.

**Model training**

When Machine Learning is training a model, rows with missing values are skipped. Therefore, if you want to fix the values manually, use imputation, or specify a different method for handling missing values, use the Clean Missing Data module before training on the dataset.

We recommend that you use the Edit Metadata module to fix any other issues with the data. You might need to mark the label column, change data types, or correct column names.

**Choose the right trainer**

The method that you use to train a model depends on the type of model you are creating, and the type of data that the model requires. For example, Machine Learning provides modules specifically for training anomaly detection models, recommendation models, and more. Check the list of training modules to determine which one is correct for your scenario.

If you are not sure of the best parameters to use when training a model, use one of the modules provided for parameter sweeping and validation:

* **Tune Model Hyperparameters** can perform a parameter sweep on almost all classification and regression models. It trains multiple models, and then returns the best model.
* The **Sweep Clustering** module supports model tuning during the training process, and is intended for use only with clustering models. You can specify a range of centroids, and train on data while automatically detecting the best parameters.
* The **Cross-Validate Model** module is also useful for model optimization, but does not return a trained model. Instead, it provides metrics that you can use to determine the best model.

**Use pretrained models**

Machine Learning includes some models that are pretrained, such as the Pretrained Cascade Image Classification module. You can use these models for scoring without additional data input.

Also, some modules (such as Time Series Anomaly Detection) do not generate a trained model in the iLearner format. But they do take training data and create a model internally, which can then be used to make predictions. To use these, you just configure the parameters and provide data.

**Save a snapshot of a trained model**

If you want to save or export the model, right-click the training module, and select Save as Trained Model. The model is exported to the iLearner format and saved in your workspace, under Trained Models. Trained models can be re-used in other experiments, or connected to other modules for scoring.

You can also use the Load Trained Model module in an experiment to retrieve a stored model.

**List of modules**

The Train category includes these modules:

* ***Sweep Clustering:*** Performs a parameter sweep on a clustering model to determine the optimum parameter settings, and trains the best model.
* ***Train Anomaly Detection Model:*** Trains an anomaly detector model and labels data from a training set.
* ***Train Clustering Model:*** Trains a clustering model and assigns data from the training set to clusters.
* ***Train Matchbox Recommender:*** Trains a Bayesian recommender by using the Matchbox algorithm.
* ***Train Model***: Trains a classification or regression model from a training set.
* ***Tune Model Hyperparameters***: Performs a parameter sweep on a regression or classification model to determine the optimum parameter settings, and trains the best model.

**Sweep Clustering**

To train a model using a parameter sweep. A parameter sweep is a way of finding the best hyperparameters for a model, given a set of data.

The Sweep Clustering module is designed specifically for clustering models. You provide a clustering model as input, together with a dataset. The module iterates over a set of parameters you specify, building and testing models with different parameters, until it finds the model with the best set of clusters. It automatically computes the best configuration, and then trains a model using that configuration. It also returns a set of metrics describing the models that were tested, and a set of cluster assignments based on the best model.

**How to configure Sweep Clustering**

1. Add the ***Sweep Clustering*** module to your experiment in Studio (classic). You can find this module under Machine Learning, in the Train category.
2. Add the ***K-Means Clustering*** module and your training dataset to the experiment, and connect them both to the ***Sweep Clustering*** module.
3. Configure the ***K-Means Clustering*** module to use a parameter sweep as follows:
4. Set Create trainer mode to Parameter Range.
5. Use the ***Range Builder*** (or manually type multiple values) for each parameter to set the range of values to iterate over.
6. ***Initialization for sweep:*** Specify how the K-means algorithm should find the initial cluster centroids. Multiple algorithms are provided for randomly initializing and then testing centroids. If your training dataset contains a label column, even with partial values, you can use those values for centroids. Use the Assign Label Mode option to indicate how the label values are used.
7. ***Number of seeds to sweep:*** Indicate how many different random starting seeds to try when doing the parameter sweep.
8. Choose the metric to use when measuring cluster similarity.
9. **Iterations:** Specify the total number of iterations that the K-means algorithm should perform. These iterations are used to optimize the selection of the cluster centroids.
10. If you are using a label column to initialize the sweep, use the ***Assign Label Mode*** option to specify how the values in the label column should be handled.

* ***Fill missing values:*** If your label column contains some missing values, use this option to impute categories based on the cluster the data point is assigned to.
* ***Overwrite from closest to center***: Generates label values for all data points assigned to a cluster, using the label of the point that is closest to the center of the cluster.
* ***Ignore label column:*** Select this option if you don’t want to perform either of the above operations.

1. In the Sweep Clustering module, use the option, For Metric for measuring clustering result, to specify the mathematical method to use when estimating the fit of the trained clustering model:

* **Simplified Silhouette**: This metric captures the tightness of data points within each cluster. It is computed as a combination of the similarity of each row with its cluster and its similarity with next closest cluster. If the cluster has only 1 row, the prorated distance to next closest centroid is calculated instead, to avoid getting 0 as the result. “Simplified” refers to the fact that the distance to cluster centroid is used as a simple similarity measure. In general, a higher score is better. The average value over the dataset indicates how well the data has been clustered. If there are too many or too few clusters, some clusters will have lower silhouette values than the rest.
* **Davies-Bouldin:** This metric aims to identify the smallest set of clusters with the least scatter. Because the metric is defined as a ratio of scatter within each cluster over cluster separation, a lower value means that clustering is better. The best clustering model minimizes this metric. To calculate the Davies-Bouldin metric, the average row to centroid distance is computed per cluster. For each pair of clusters, the sum of those averages is divided by the distance between centroids. Maximal value over all other clusters is selected for each cluster and averaged over all clusters.
* **Dunn:** This metric aims to identify the smallest set of most compact clusters. Generally, a higher value for this metric indicates better clustering. To calculate the Dunn metric, the minimal centroid-to-centroid distance is divided by the maximal distance of each data point to its cluster center.
* **Average deviation:** This metric is computed by taking the average distance from each data point to its cluster center. The value decreases as the number of centroids increases; therefore, it is not useful when sweeping to find the number of centroids. This metric is recommended for use when you are choosing the best centroid initialization seed.

1. ***Specify parameter sweeping mode:*** Select an option that defines the combinations of values that are used when training, and how they are chosen:

* *Entire grid:* All values within the given range are tried and evaluated. This option is usually more computationally expensive.
* *Random sweep:* Use this option to limit the number of runs. The clustering model is built and evaluated using a combination of values chosen randomly from the allowed range of parameter values.

1. ***Maximum number of runs on random sweep:*** Set this option if you choose the Random sweep option. Type a value to limit the maximum number of iterations when testing sets of randomly chosen parameters. If you select a random sweep, use the Random seed option to specify the initial random seed values, on which to begin creating the centroids. One advantage of using a parameter sweep to create a clustering model is that you can easily test multiple seed values to mitigate the known sensitivity of clustering models to the initial seed value.
2. Click ***Column Set***, and choose the columns to use when building the clusters. By default, all feature columns are used when building and testing the clustering model. You can include a label column, if present in your dataset. If a label is present, you can use it to guide selection of centroids, use the label as a feature, or ignore the label. Set these options for label handling the Kmeans Clustering module as described in Step 3 above.
3. ***Check for Append or Uncheck for Result Only***: Use this option to control which columns are returned in the results. By default, the module returns the original columns of the training dataset together with the results. If you deselect this option. only the cluster assignments are returned.
4. Add the ***Assign Data to Clusters*** module to your experiment.
5. Connect the output labeled Best Trained Model to the Trained Model input of ***Assign Data to Clusters.***
6. Add the dataset intended for evaluation, and connect it to the Dataset port of the ***Assign Data to Clusters*** module.
7. Add the ***Evaluate Model*** module, and connect it to ***Assign Data to Clusters***. Optionally, you can connect an evaluation dataset.
8. Run the experiment.

**Results**

The Sweep Clustering module outputs three different results:

* ***Best Trained Model***. A trained model that you can use for scoring and evaluation. Right-click and select Save as Trained Model to capture the optimized clustering model and use it for scoring.
* ***Results dataset.*** A set of cluster assignments, based on the optimized model.

| **Column name** | **Description** |
| --- | --- |
| **Assignments** | This value indicates the cluster to which each data point has been assigned. The clusters in the trained model are labeled with 0-based indexes. |
| **DistancesToClusterCenter no.1**  **DistancesToClusterCenter no.n** | This value indicates how close the data point is to the center of each cluster.  A column is created for each cluster created in the optimized model.  You can constrain the number of clusters by using the **Number of centroids** option. |

By default, you can return the columns from the training dataset along with the results, to make it easier to review and interpret the cluster assignments.

* **Sweep results**. A dataset containing the following evaluation metrics for the clusters:

| **Column name** | **Description** |
| --- | --- |
| **Cluster metric** | A value that indicates the average cluster quality for that run. The runs are ordered by the best score. |
| **Number of centroids** | The number of clusters that were created in this particular iteration of the sweep |
| **Index of run** | An identifier for each iteration |

**Train Anomaly Detection Model**

The module takes as input a set of model parameters for anomaly detection model, such as that produced by the One-Class Support Vector Machine module, and an unlabeled dataset. It returns a trained anomaly detection model, together with a set of labels for the training data.

**How to configure Train Anomaly Detection Model**

1. Add the Train Anomaly Detection Model module to your experiment in Studio (classic). You can find the module under Machine Learning, in the Train category.
2. Connect one of the modules designed for anomaly detection, such as PCA-Based Anomaly Detection or One-Class Support Vector Machine. Other types of models are not supported; on running the experiment you will get the error: All models must have the same learner type.
3. Configure the anomaly detection module by choosing the label column and setting other parameters specific to the algorithm.
4. Attach a training dataset to the right-hand input of Train Anomaly Detection Model.
5. Run the experiment.

**Results**

After training is complete:

* To view the model's parameters, right-click the module and select Visualize.
* To create predictions, use Score Model with new input data.
* To save a snapshot of the trained model, right-click the Trained Model output, and select Save As.

**Train Clustering Model**

The module takes an untrained clustering model that you have already configured using the K-Means Clustering module, and trains the model using a labeled or unlabeled data set. The module creates both a trained model that you can use for prediction, and a set of cluster assignments for each case in the training data.

**How to use Train Clustering Model**

1. Add the Train Clustering Model module to your experiment in Studio (classic).
2. Add the K-Means Clustering module, or another custom module that creates a compatible clustering model, and set the parameters of the clustering model.
3. Attach a training dataset to the right-hand input of Train Clustering Model.
4. In Column Set, select the columns from the dataset to use in building clusters. Be sure to select columns that make good features: for example, avoid using IDs or other columns that have unique values, or columns that have all the same values. If a label is available, you can either use it as a feature, or leave it out.
5. Select the option, Check for Append or Uncheck for Result Only, if you want to output the training data together with the new cluster label. If you deselect this option, only the cluster assignments are output.
6. Run the experiment, or click the Train Clustering Model module and select Run Selected.

**Results**

After training has completed:

* To view the cluster and their separation in a graph, right-click the Results dataset output and select Visualize. The graph represents the principal components of the cluster, rather than the actual values. See Principal Component Analysis for more information.
* To view the values in the dataset, add an instance of the Convert to Dataset module, and connect it to the Results dataset output. Run the Convert to Dataset module to get a copy of the data that you can view or download.
* To save the trained model for later re-use, right-click the module, select Trained model, and click Save As Trained Model.
* To generate scores from the model, use Assign Data to Clusters.

**Train Matchbox Recommender**

The recommendation algorithm in Machine Learning is based on the Matchbox model, developed by Microsoft Research.

The Train Matchbox Recommender module reads a dataset of user-item-rating triples and, optionally, some user and item features. It returns a trained Matchbox recommender. You can then use the trained model to generate recommendations, find related users, or find related items, by using the Score Matchbox Recommender module.

**Recommendation models and the Matchbox recommender**

The main aim of a recommendation system is to recommend one or more items to users of the system. Examples of an item could be a movie, restaurant, book, or song. A user could be a person, group of persons, or other entity with item preferences.

There are two principal approaches to recommender systems.

* The first is the ***content-based approach,*** which makes use of features for both users and items. Users may be described by properties such as age and gender, and items may be described by properties such as author and manufacturer. Typical examples of content-based recommendation systems can be found on social matchmaking sites.
* The second approach is ***collaborative filtering***, which uses only identifiers of the users and the items and obtains implicit information about these entities from a (sparse) matrix of ratings given by the users to the items. We can learn about a user from the items they have rated and from other users who have rated the same items.

The Matchbox recommender combines these approaches, using collaborative filtering with a content-based approach. It is therefore considered a hybrid recommender.

How this works: When a user is relatively new to the system, predictions are improved by making use of the feature information about the user, thus addressing the well-known "cold-start" problem. However, once you have collected a sufficient number of ratings from a particular user, it is possible to make fully personalized predictions for them based on their specific ratings rather than on their features alone. Hence, there is a smooth transition from content-based recommendations to recommendations based on collaborative filtering. Even if user or item features are not available, Matchbox will still work in its collaborative filtering mode.

**How to configure Train Matchbox Recommender**

* Prepare the training data
* Train the model

***Prepare data***

Before trying to use the module, it is essential that your data be in the format expected by the recommendation model. A training data set of user-item-rating triples is required, but you can also include user features and item features (if available), in separate datasets. To divide source data into training and testing datasets, use the Recommender Split option in the Split Data module.

**Required dataset of user-item-ratings**

It is very important that the input data used for training contain the right type of data in the correct format:

* The first column must contain user identifiers.
* The second column must contain item identifiers.
* The third column contains the rating for the user-item pair. Rating values must be either numeric or categorical.

During training, the rating values cannot all be the same. Moreover, if numeric, the difference between the minimum and the maximum rating values must be less than 100, and ideally not greater than 20.

The Restaurant ratings dataset in Machine Learning Studio (classic) (click Saved Datasets and then Samples) demonstrates the expected format:

| **userID** | **placeID** | **rating** |
| --- | --- | --- |
| U1077 | 135085 | 2 |
| U1077 | 135038 | 2 |

From this sample, you can see that a single user has rated two separate restaurants.

**User features dataset (optional)**

The dataset of **user features** must contain identifiers for users, and use the same identifiers that were provided in the first column of the users-items-ratings dataset. The remaining columns can contain any number of features that describe the users.

For an example, see the **Restaurant customer** dataset in Machine Learning Studio (classic). A typical set of user features looks like this:

| **userID** | **ambience** | **Dress\_preference** | **transport** | **smoker** |
| --- | --- | --- | --- | --- |
| U1004 | family | informal | On foot | FALSE |
| U1005 | friends | No preference | Car owner | TRUE |

**Item features dataset (optional)**

The dataset of item features must contain item identifiers in its first column. The remaining columns can contain any number of descriptive features for the items.

For an example, see the Restaurant feature data dataset, provided in Machine Learning Studio (classic) (click **Saved Datasets** and then **Samples**). A typical set of item features (in this case, the item is a restaurant) might look like this:

| **placeID** | **alcohol** | **Smoking\_area** | **price** | **Rambience** |
| --- | --- | --- | --- | --- |
| 135106 | Wine-Beer | none | low | family |
| 132667 | No\_Alcohol\_Served | permitted | medium | casual |

**Train the model**

1. Add the Train Matchbox Recommender module to your experiment in Studio (classic), and connect it to the training data.
2. If you have a separate dataset of either user features and/or item features, connect them to the Train Matchbox Recommender module.

* User features dataset: Connect the dataset that describes users to the second input.
* Item features dataset: Connect the dataset that describes items to the third input.

1. For Number of training batches, type the number of batches for dividing the data during training.

Based on this value, the dataset of user-item-rating triples is divided into multiple parts or batches during training. Because Train Matchbox Recommender runs batches in parallel, we recommend that the number of training batches be set to the number of available cores, if the entire training data fits into memory. Otherwise, the number of training batches should be set to the lowest multiple of the number of available cores for which the training data does fit into memory. By default, the training data is split into four (4) batches. Only the dataset of user-item-rating triples is split. User or item features are not split because features do not need to be split.

1. For Number of traits, type the number of latent traits that should be learned for each user and item.

The higher the number of traits, the more accurate the predictions will typically be; however, training will be slower. The number of traits usually lies in the range 2 - 20.

1. For Number of recommendation algorithm iterations, indicate how many times the algorithm should process the input data.

The Matchbox recommender is trained using a message-passing algorithm that can iterate multiple times over the input data. The higher this number, the more accurate the predictions; however, training is slower. Usually, the number of iterations is in the range 1 - 10.

1. Run the experiment, or select just the Train Matchbox Recommender module and select Run selected.

**Train Model**

Training takes place after you have defined a model and set its parameters, and requires tagged data. You can also use Train Model to retrain an existing model with new data.

**How the training process works**

In Machine Learning, creating and using a machine learning model is typically a three-step process.

1. You configure a model, by choosing a particular type of algorithm, and defining its parameters or hyperparameters. Choose any of the following model types:

* Classification models, based on neural networks, decision trees, and decision forests, and other algorithms.
* Regression models, which can include standard linear regression, or which use other algorithms, including neural networks and Baysian regression.

1. Provide a dataset that is labeled, and has data compatible with the algorithm. Connect both the data and the model to Train Model.

What training produces is a specific binary format, the iLearner, that encapsulates the statistical patterns learned from the data. You cannot directly modify or read this format; however, other modules in Studio (classic) can use this trained model.

1. After training is completed, use the trained model with one of the scoring modules, to make predictions on new data.

**Supervised and unsupervised training**

You might have heard the terms supervised or unsupervised learning. Training a classification or regression model with Train Model is a classic example of supervised machine learning. That means you must provide a dataset that contains historical data from which to learn patterns. The data should contain both the outcome (label) you are trying to predict, and related factors (variables). The machine learning model needs the outcomes to determine the features that best predict the outcomes. During the training process, the data are sorted by outcomes and the algorithm extracts statistical patterns to build the model.

Unsupervised learning indicates either that the outcome is unknown, or you choose not to use known labels. For example, clustering algorithms usually employ unsupervised learning methods, but can use labels if available. Another example is topic modeling using LDA. You cannot use Train Model with these algorithms.

**How to use Train Model**

1. In Machine Learning Studio (classic), configure a classification model or regression model models. You can also train a custom model created by using Create R Model.
2. Add the Train Model module to the experiment. You can find this module under the Machine Learning category. Expand Train, and then drag the Train Model module into your experiment.
3. On the left input, attach the untrained mode. Attach the training dataset to the right-hand input of Train Model.The training dataset must contain a label column. Any rows without labels are ignored.
4. For Label column, click Launch column selector, and choose a single column that contains outcomes the model can use for training.

* For classification problems, the label column must contain either categorical values or discrete values. Some examples might be a yes/no rating, a disease classification code or name, or an income group. If you pick a noncategorical column, the module will return an error during training.
* For regression problems, the label column must contain numeric data that represents the response variable. Ideally the numeric data represents a continuous scale.

Examples might be a credit risk score, the projected time to failure for a hard drive, or the forecasted number of calls to a call center on a given day or time. If you do not choose a numeric column, you might get an error.

* If you do not specify which label column to use, Machine Learning will try to infer which is the appropriate label column, by using the metadata of the dataset. If it picks the wrong column, use the column selector to correct it.

1. Run the experiment. If you have a lot of data, this can take a while.

**Results**

After the model is trained:

* To view the model parameters and feature weights, right-click the output and select Visualize.
* To use the model in other experiments, right-click the model and select Save Model. Type a name for the model. This saves the model as a snapshot that is not updated by repeated runs of the experiment.
* To use the model in predicting new values, connect it to the Score Model module, together with new input data.

**Tune Model Hyperparameters**

The module builds and tests multiple models, using different combinations of settings, and compares metrics over all models to get the combination of settings. The terms parameter and hyperparameter can be confusing. The model's parameters are what you set in the properties pane. Basically, this module performs a parameter sweep over the specified parameter settings, and learns an optimal set of hyperparameters, which might be different for each specific decision tree, dataset, or regression method. The process of finding the optimal configuration is sometimes called tuning.

The module support two methods for finding the optimum settings for a model:

* ***Integrated train and tune:*** You configure a set of parameters to use, and then let the module iterate over multiple combinations, measuring accuracy until it finds a "best" model. With most learner modules, you can choose which parameters should be changed during the training process, and which should remain fixed. Depending on how long you want the tuning process to run, you might decide to exhaustively test all combinations, or you could shorten the process by establishing a grid of parameter combinations and testing a randomized subset of the parameter grid.
* ***Cross validation with tuning***: With this option, you divide your data into some number of folds and then build and test models on each fold. This method provides the best accuracy and can help find problems with the dataset; however, it takes longer to train.

Both methods generate a trained model that you can save for re-use.

**How to configure Tune Model Hyperparameters**

Generally, learning the optimal hyperparameters for a given machine learning model requires considerable experimentation. This module supports both the initial tuning process, and cross-validation to test model accuracy:

* Find optimal model parameters using a parameter sweep
* Perform cross-validation during a parameter sweep

**Train a model using a parameter sweep**

This section describes how to perform a basic parameter sweep, which trains a model by using the Tune Model Hyperparameters module.

1. Add the Tune Model Hyperparameters module to your experiment in Studio (classic).
2. Connect an untrained model (a model in the iLearner format) to the leftmost input.
3. Set the Create trainer mode option to Parameter Range and use the Range Builder to specify a range of values to use in the parameter sweep.

Almost all the classification and regression modules support an integrated parameter sweep. For those learners that do not support configuring a parameter range, only the available parameter values can be tested. You can manually set the value for one or more parameters, and then sweep over the remaining parameters. This might save some time.

1. Add the dataset you want to use for training and connect it to the middle input of Tune Model Hyperparameters. Optionally, if you have a tagged dataset, you can connect it to the rightmost input port (Optional validation dataset). This lets you measure accuracy while training and tuning.
2. In the Properties pane of Tune Model Hyperparameters, choose a value for Parameter sweeping mode. This option controls how the parameters are selected.

* Entire grid: When you select this option, the module loops over a grid predefined by the system, to try different combinations and identify the best learner. This option is useful for cases where you don't know what the best parameter settings might be and want to try all possible combination of values. You can also reduce the size of the grid and run a random grid sweep. Research has shown that this method yields the same results, but is more efficient computationally.
* Random sweep: When you select this option, the module will randomly select parameter values over a system-defined range. You must specify the maximum number of runs that you want the module to execute. This option is useful for cases where you want to increase model performance using the metrics of your choice but still conserve computing resources.

1. For Label column, launch the column selector to choose a single label column.
2. Choose a single metric to use when ranking the models.

When you run a parameter sweep, all applicable metrics for the model type are calculated and are returned in the Sweep results report. Separate metrics are used for regression and classification models.

However, the metric you choose determines how the models are ranked. Only the top model, as ranked by the chosen metric, is output as a trained model to use for scoring.

1. For Random seed, type a number to use when initializing the parameter sweep.

If you are training a model that supports an integrated parameter sweep, you can also set a range of seed values to use and iterate over the random seeds as well. This can be useful for avoiding bias introduced by seed selection.

1. Run the experiment.

**Results of hyperparameter tuning**

When training is complete:

* To view a set of accuracy metrics for the best model, right-click the module, select Sweep results, and then select Visualize. All accuracy metrics applicable to the model type are output, but the metric that you selected for ranking determines which model is considered "best". Metrics are generated only for the top-ranked model.
* To view the settings derived for the "best" model, right-click the module, select Trained best model, and then click Visualize. The report includes parameter settings and feature weights for the input columns.
* To use the model for scoring in other experiments, without having to repeat the tuning process, right-click the model output and select Save as Trained Model.

**Perform cross-validation with a parameter sweep**

This section describes how to combine a parameter sweep with cross-validation. This process takes longer, but you can specify the number of folds, and you get the maximum amount of information about your dataset and the possible models.

1. Add the Partition and Sample module to your experiment, and connect the training data.
2. Choose the Assign to Folds option and specify some number of folds to divide the data into. If you don't specify a number, by default 10 folds are used. Rows are apportioned randomly into these folds, without replacement.
3. To balance the sampling on some column, set the Stratified split to TRUE, and then select the strata column. For example, if you have an imbalanced dataset, you might want to divide the dataset such that each fold gets the same number of minority cases.
4. Add the Tune Model Hyperparameters module to your experiment.
5. Connect one of the machine learning modules in this category to the left-hand input of Tune Model Hyperparameters.
6. In the Properties pane for the learner, set the Create trainer mode option to Parameter Range and use the Range Builder to specify a range of values to use in the parameter sweep. You don’t need to specify a range for all values. You can manually set the value for some parameters, and then sweep over the remaining parameters. This might save some time.
7. Connect the output of Partition and Sample to the labeled Training dataset input of Tune Model Hyperparameters.
8. Optionally, you can connect a validation dataset to the rightmost input of Tune Model Hyperparameters. For cross-validation, you need only a training dataset.
9. In the Properties pane of Tune Model Hyperparameters, indicate whether you want to perform a random sweep or a grid sweep. A grid sweep is exhaustive, but more time-consuming. A random parameter search can get good results without taking quite so much time.

***Maximum number of runs on random sweep:*** If you choose a random sweep, you can specify how many times the model should be trained, using a random combination of parameter values.

***Maximum number of runs on random grid:*** This option also controls the number of iterations over a random sampling of parameter values, but the values are not generated randomly from the specified range; instead, a matrix is created of all possible combinations of parameter values and a random sampling is taken over the matrix. This method is more efficient and less prone to regional oversampling or undersampling.

1. Choose a single label column.
2. Choose a single metric to use in ranking the model. Many metrics are computed, so select the most important one to use in ordering the results.
3. For Random seed, type a number to use when initializing the parameter sweep.

If you are training a model that supports an integrated parameter sweep, you can also set a range of seed values to use and iterate over the random seeds as well. This is optional, but can be useful for avoiding bias introduced by seed selection.

1. Add the Cross-Validate Model module. Connect the output of Partition and Sample to the Dataset input, and connect the output of Tune Model Hyperparameters to the Untrained model input.
2. Run the experiment.

**Results of cross-validation**

When cross-validation is complete:

* To view the evaluation results, right-click the module, select Evaluation results by fold, and then select Visualize.
* The accuracy metrics are calculated from the cross-validation pass, and may vary slightly depending on how many folds you selected.
* To see how the dataset was divided, and how the "best" model would score each row in the dataset, right-click the module, select Scored results, and then select Visualize.
* If you save this dataset for later re-use, the fold assignments are preserved.
* To get the parameter settings for the "best" model, right-click Tune Model Hyperparameters