**Classification modules**

**Two-Class Averaged Perceptron**

To create a machine learning model based on the averaged perceptron algorithm. This classification algorithm is a supervised learning method, and requires a tagged dataset, which includes a label column. You can train the model by providing the model and the tagged dataset as an input to Train Model or Tune Model Hyperparameters. The trained model can then be used to predict values for the new input examples.

**Averaged perceptron models**

The averaged perceptron method is an early and very simple version of a neural network. In this approach, inputs are classified into several possible outputs based on a linear function, and then combined with a set of weights that are derived from the feature vector—hence the name "perceptron." The simpler perceptron models are suited to learning linearly separable patterns, whereas neural networks (especially deep neural networks) can model more complex class boundaries. However, perceptrons are faster, and because they process cases serially, perceptrons can be used with continuous training.

**How to configure Two-Class Averaged Perceptron**

1. Add the Two-Class Averaged Perceptron module to your experiment in Studio (classic).
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* **Single Parameter:** If you know how you want to configure the model, provide a specific set of values as arguments.
* **Parameter Range:** If you are not sure of the best parameters, find the optimal parameters by specifying multiple values and using the Tune Model Hyperparameters module to find the optimal configuration. The trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

1. For Learning rate, specify a value for the learning rate. The learning rate values controls the size of the step that is used in stochastic gradient descent each time the model is tested and corrected.

By making the rate smaller, you test the model more often, with the risk that you might get stuck in a local plateau. By making the step larger, you can converge faster, at the risk of overshooting the true minima.

1. For Maximum number of iterations, type the number of times you want the algorithm to examine the training data.

Stopping early often provides better generalization. Increasing the number of iterations improves fitting, at the risk of overfitting.

1. For Random number seed, optionally type an integer value to use as the seed. Using a seed is recommended if you want to ensure reproducibility of the experiment across runs.
2. Select the Allow unknown categorical levels option to create a group for unknown values in the training and validation sets. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.

If you deselect this option, the model can accept only the values that are contained in the training data.

1. Connect a training dataset, and one of the training modules:

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

**Results**

After training is complete:

* To see a summary of the model's parameters, together with the feature weights learned from training, right-click the output of Train Model or Tune Model Hyperparameters.

**Two-Class Bayes Point Machine**

To create an untrained binary classification model.The algorithm in this module uses a Bayesian approach to linear classification called the "Bayes Point Machine". This algorithm efficiently approximates the theoretically optimal Bayesian average of linear classifiers (in terms of generalization performance) by choosing one "average" classifier, the Bayes Point. Because the Bayes Point Machine is a Bayesian classification model, it is not prone to overfitting to the training data.

**How to configure Two-Class Bayes Point Machine**

1. In Machine Learning Studio (classic), add the Two-Class Bayes Point Machine module to your experiment. You can find the module under Machine Learning, Initialize Model, Classification.

For Number of training iterations, type a number to specify how often the message-passing algorithm iterates over the training data. Typically, the number of iterations should be set to a value in the range 5 – 100.The higher the number of training iterations, the more accurate the predictions; however, training will be slower. For most datasets, the default setting of 30 training iterations is sufficient for the algorithm to make accurate predictions. Sometimes accurate predictions can be made by using fewer iterations. For datasets with highly correlated features, you might benefit from more training iterations.

1. Select the option, Include bias, if you want a constant feature or bias to be added to each instance in training and prediction. Including a bias is necessary when the data does not already contain a constant feature.
2. Select the option, Allow unknown values in categorical features, to create a group for unknown values. If you deselect this option, the model can accept only the values that are contained in the training data.If you select this option and allow unknown values, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.
3. Add an instance of the Train Model module, and your training data.
4. Connect the training data and the output of the Two-Class Bayes Point Machine module to the Train Model module, and choose the label column.
5. Run the experiment.

**Results**

After training is complete, right-click the output of the Train Model module to view the results:

* To see a summary of the model's parameters, together with the feature weights learned from training, select Visualize.
* To save the model for later use, right-click the output of Train MOdel, and select Save as Trained Model.
* To make predictions, use the trained model as an input to the Score Model module.

**Two-Class Boosted Decision Tree**

To create a machine learning model that is based on the boosted decision trees algorithm. A boosted decision tree is an ensemble learning method in which the second tree corrects for the errors of the first tree, the third tree corrects for the errors of the first and second trees, and so forth. Predictions are based on the entire ensemble of trees together that makes the prediction. Generally, when properly configured, boosted decision trees are the easiest methods with which to get top performance on a wide variety of machine learning tasks. However, they are also one of the more memory-intensive learners, and the current implementation holds everything in memory. Therefore, a boosted decision tree model might not be able to process the very large datasets that some linear learners can handle.

**How to configure Two-Class Boosted Decision Tree**

This module creates an untrained classification model. Because classification is a supervised learning method, to train the model, you need a tagged dataset that includes a label column with a value for all rows.

You can train this type of model by using either the Train Model or Tune Model Hyperparameters modules.

1. In Machine Learning Studio (classic), add the Boosted Decision Tree 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 how you want to configure the 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 using the Tune Model Hyperparameters module. You provide some range of values, and the trainer iterates over multiple combinations of the settings to determine the combination of values that produces the best result.

1. For Maximum number of leaves per tree, indicate the maximum number of terminal nodes (leaves) that can be created in any tree. By increasing this value, you potentially increase the size of the tree and get better precision, at the risk of overfitting and longer training time.
2. For Minimum number of samples per leaf node, indicate the number of cases required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

1. For Learning rate, type a number between 0 and 1 that defines the step size while learning.

The learning rate determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too small, training takes longer to converge on the best solution.

1. For Number of trees constructed, indicate the total number of decision trees to create in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time will increase.

This value also controls the number of trees displayed when visualizing the trained model. if you want to see or print a single tree, set the value to 1. However, when you do so, only one tree is produced (the tree with the initial set of parameters) and no further iterations are performed.

1. For Random number seed, optionally type a non-negative integer to use as the random seed value. Specifying a seed ensures reproducibility across runs that have the same data and parameters.

The random seed is set by default to 0, which means the initial seed value is obtained from the system clock. Successive runs using a random seed can have different results.

1. Select Allow unknown categorical levels option to create a group for unknown values in the training and validation sets. If you deselect this option, the model can accept only the values that are contained in the training data. If you allow unknown values, the model might be less precise for known values, but likely can provide better predictions for new (unknown) values.
2. Train the model.

* If you set Create trainer mode to Single Parameter, connect a tagged dataset and the Train Model module.
* If you set Create trainer mode to Parameter Range, connect a tagged dataset and train the model by using Tune Model Hyperparameters.

**Results**

After training is complete:

* To see the tree that was created on each iteration, right-click Train Model module and select Trained model to visualize. If you use Tune Model Hyperparameters, right click the module and select Trained best model to visualize the best model. Click each tree to drill down into the splits and see the rules for each node.
* To use the model for scoring, connect it to Score Model, to predict values for new input examples.

**Tips**

* To train a boosted decision tree model, you must provide multiple data instances. An error is generated during the training process if the dataset contains too few rows.
* If your data has missing values, you must add indicators for the features.
* In general, boosted decision trees yield better results when features are somewhat related. If features have a large degree of entropy (that is, they are not related), they share little or no mutual information, and ordering them in a tree does not yield a lot of predictive significance. If this is not the case, you might try a random forests model.

Boosting also works well when you have many more examples than features because the model is prone to overfitting.

* Do not normalize the dataset. Because the treatment of features is a simple, non-parametric, less-than or greater-than comparison, normalization or any form of non-monotonic transformation function might have little effect.
* Features are discretized and binned prior to training, so only a relatively small set of threshold candidates are considered, even for continuous features.

**Implementation details**

The boosted decision tree algorithm in Machine Learning uses the following boosting method:

1. Start with an empty ensemble of weak learners.
2. For each training example, get the current output of the ensemble. This is the sum of the outputs of all weak learners in the ensemble.
3. Calculate the gradient of the loss function for each example.

This depends on whether the task is a binary classification problem or a regression problem.

* In a binary classification model, the log-loss is used, much like in logistic regression.
* In a regression model, the squared loss is used, and the gradient is the current output, minus the target).

1. Use the examples to fit a weak learner, using the gradient just defined as the target function.
2. Add that weak learner to the ensemble with a strength indicated by the learning rate, and if desired, go to Step 2.

In this implementation, the weak learners are the least-squares regression trees, based on the gradients calculated in Step 3. The trees are subject to the following restrictions:

* They are trained up to a maximum number of leaves.
* Each leaf has a minimum number of examples to guard against overfitting.
* Each decision node is a single feature that is compared against some threshold. If that feature is less than or equal to the threshold, it goes down one path, and if it is greater than the threshold, it goes down the other path.
* Each leaf node is a constant value.

1. The tree-building algorithm greedily selects the feature and threshold for which a split minimizes the squared loss with regard to the gradient calculated in Step 3. The selection of the split is subject to a minimum number of training examples per leaf.

**Module parameters**

The algorithm repeatedly splits until it reaches the maximum number of leaves, or until no valid split is available.

| **Name** | **Range** | **Type** | **Default** | **Description** |
| --- | --- | --- | --- | --- |
| Maximum number of leaves per tree | >=1 | Integer | 20 | Specify the maximum number of leaves allowed per tree |
| Minimum number of samples per leaf node | >=1 | Integer | 10 | Specify the minimum number of cases required to form a leaf |
| Learning rate | [double.Epsilon;1.0] | Float | 0.2 | Specify the initial learning rate |
| Number of trees constructed | >=1 | Integer | 100 | Specify the maximum number of trees that can be created during training |
| Random number seed | Any | Integer |  | Type a value to seed the random number generator that is used by the model. Leave it blank for the default. |
| Allow unknown categorical levels | Any | Boolean | True | If True, an additional level is created for each categorical column. Any levels in the test dataset that are not available in the training dataset are mapped to this additional level. |

The untrained model can also be passed to Cross-Validate Model for cross-validation against a labeled data set.

**Two-Class Decision Forest**

To create a machine learning model based on the decision forests algorithm. Decision forests are fast, supervised ensemble models. This module is a good choice if you want to predict a target with a maximum of two outcomes. If you are not sure how to configure a decision tree model for the best results, we recommend that you use the Tune Model Hyperparameters module to train and test multiple models. tuning iterates over multiple possibilities and finds the optimal solution for you.

**Decision forests**

This decision forest algorithm is an ensemble learning method intended for classification tasks. Ensemble methods are based on the general principle that rather than relying on a single model, you can get better results and a more generalized model by creating multiple related models and combining them in some way. Generally, ensemble models provide better coverage and accuracy than single decision trees. There are many ways to create individual models and combine them in an ensemble. This particular implementation of a decision forest works by building multiple decision trees and then voting on the most popular output class. Voting is one of the better-known methods for generating results in an ensemble model.

* Many individual classification trees are created, using the entire dataset, but different (usually randomized) starting points. This differs from the random forest approach, in which the individual decision trees might only use some randomized portion of the data or features.
* Each tree in the decision forest tree outputs a non-normalized frequency histogram of labels.
* The aggregation process sums these histograms and normalizes the result to get the “probabilities” for each label.
* The trees that have high prediction confidence will have a greater weight in the final decision of the ensemble.

Decision trees in general have many advantages for classification tasks:

* They can capture non-linear decision boundaries.
* You can train and predict on lots of data, as they are efficient in computation and memory usage.
* Feature selection is integrated in the training and classification processes.
* Trees can acommodate noisy data and many features.
* They are non-parametric models, meaning they can handle data with varied distributions.

However, simple decision trees can overfit on data, and are less generalizable than tree ensembles.

**How to configure Two-Class Decision Forest**

1. Add the Two-Class Decision Forest module to your experiment in Machine Learning Studio (classic), and open the Properties pane of the module.
2. For Resampling method, choose the method used to create the individual trees. You can choose from Bagging or Replicate.

* **Bagging:** Bagging is also called bootstrap aggregating. In this method, each tree is grown on a new sample, created by randomly sampling the original dataset with replacement until you have a dataset the size of the original.

The outputs of the models are combined by voting, which is a form of aggregation. Each tree in a classification decision forest outputs an un-

normalised frequency histogram of labels. The aggregation is to sum these histograms and normalise to get the “probabilities” for each label. In this manner, the trees that have high prediction confidence will have a greater weight in the final decision of the ensemble.

* **Replicate:** In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse.

1. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know how you want to configure the 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 Tune Model Hyperparameters module to find the optimal configuration. The trainer iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

1. For Number of decision trees, type the maximum number of decision trees that can be created in the ensemble. By creating more decision trees, you can potentially get better coverage, but training time increases.
2. For Maximum depth of the decision trees, type a number to limit the maximum depth of any decision tree. Increasing the depth of the tree might increase precision, at the risk of some overfitting and increased training time.
3. For Number of random splits per node, type the number of splits to use when building each node of the tree. A split means that features in each level of the tree (node) are randomly divided.
4. For Minimum number of samples per leaf node, indicate the minimum number of cases that are required to create any terminal node (leaf) in a tree.

By increasing this value, you increase the threshold for creating new rules. For example, with the default value of 1, even a single case can cause a new rule to be created. If you increase the value to 5, the training data would have to contain at least 5 cases that meet the same conditions.

1. Select the Allow unknown values for categorical features option to create a group for unknown values in the training or validation sets. The model might be less precise for known values, but it can provide better predictions for new (unknown) values. If you deselect this option, the model can accept only the values that are contained in the training data.
2. Attach a labeled dataset, and one of the training modules:

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

**Results**

After training is complete:

* To see the tree that was created on each iteration, right-click Train Model module and select Trained model to visualize. If you use Tune Model Hyperparameters, right click the module and select Trained best model to visualize the best model. Click each tree to drill down into the splits and see the rules for each node.
* To save a snapshot of the model, right-click the Trained Model output, and select Save Model. The saved model is not updated on successive runs of the experiment.
* To use the model for scoring, add the Score Model module to an experiment.

**Tips**

If you have limited data, or if you want to minimize the time spent training the model, try these settings:

***Limited training set***

If the training set contains a limited number of instances:

* Create the decision forest by using a large number of decision trees (for example, more than 20).
* Use the Bagging option for resampling.
* Specify a large number of random splits per node (for example, more than 1,000).

***Limited training time***

If the training set contains a large number of instances and training time is limited:

* Create the decision forest by using fewer decision trees (for example, 5-10).
* Use the Replicate option for resampling.
* Specify a smaller number of random splits per node (for example, fewer than 100)

**Two-Class Decision Jungle**

To create a machine learning model that is based on a supervised ensemble learning algorithm called decision jungles. The Two-Class Decision Jungle module returns an untrained classifier. You then train this model on a labeled training data set, by using Train Model or Tune Model Hyperparameters. The trained model can then be used to make predictions.

**Decision jungles**

A decision jungle consists of an ensemble of decision directed acyclic graphs (DAGs).

Decision jungles have the following advantages:

* By allowing tree branches to merge, a decision DAG typically has a lower memory footprint and better generalization performance than a decision tree, albeit at the cost of somewhat longer training time.
* Decision jungles are non-parametric models that can represent non-linear decision boundaries.
* They perform integrated feature selection and classification and are resilient in the presence of noisy features.

**How to configure Two-Class Decision Jungle**

1. Add the Two-Class Decision Jungle module to your experiment in Studio (classic).
2. For Resampling method, choose the method used to create the individual trees. You can choose from Bagging or Replicate.

* Bagging: Select this option to use bagging, also called bootstrap aggregating

Each tree in a decision jungle outputs a Gaussian distribution as prediction. The aggregation is to find a Gaussian whose first two moments match the moments of the mixture of Gaussians given by combining all Gaussians returned by individual trees.

* Replicate: In replication, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random and the trees will be diverse.

1. Specify how you want the model to be trained, by setting the Create trainer mode option.

* Single Parameter: If you know how you want to configure the 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 Tune Model Hyperparameters module to find the optimal configuration. The trainer will iterate over multiple combinations of the settings you provided and determine the combination of values that produces the best model.

1. For Number of decision DAGs, indicate the maximum number of graphs that can be created in the ensemble.
2. For Maximum depth of the decision DAGs, indicate the maximum depth of each graph.
3. For Maximum width of the decision DAGs, indicate the maximum width of each graph.
4. In Number of optimization steps per decision DAG layer, indicate how many iterations over the data to perform when building each DAG.
5. Select the Allow unknown values for categorical features option to create a group for unknown values in testing or validation data. If you deselect it, the model can accept only the values that are contained in the training data. In the former case, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.
6. Add a tagged dataset to the experiment, and connect one of the training modules.

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

**Results**

After training is complete:

* To use the model for scoring, connect it to Score Model, to predict values for new input examples.

***Limited training set***

If your training set is small:

* Create the decision jungle by using a large number of decision DAGs (for example, more than 20).
* Use the Bagging option for resampling.
* Specify a large number of optimization steps per DAG layer (for example, more than 10,000).

***Limited training time***

If the training set is large but training time is limited:

* Create the decision jungle using a fewer number of decision DAGs (for example, 5-10
* Use the Replicate option for resampling.
* Specify a smaller number of optimization steps per DAG layer (for example, less than 2000.