**Classification modules**

Classification is a machine learning method that uses data to determine the category, type, or class of an item or row of data. For example, you can use classification to:

* Classify email filters as spam, junk, or good.
* Determine whether a patient's lab sample is cancerous.
* Categorize customers by their propensity to respond to a sales campaign.
* Identify sentiment as positive or negative.

Classification tasks are frequently organized by whether a classification is binary (either A or B) or multiclass (multiple categories that can be predicted by using a single model).

**Create a classification model**

To create a classification model, or classifier, first, select an appropriate algorithm. Consider these factors:

* How many classes or different outcomes do you want to predict?
* What is the distribution of the data?
* How much time can you allow for training?

Machine Learning Studio (classic) provides multiple classification algorithms. When you use the One-Vs-All algorithm, you can even apply a binary classifier to a multiclass problem.

After you choose an algorithm and set the parameters by using the modules in this section, train the model on labeled data. Classification is a supervised machine learning method. It always requires labeled training data.

When training is finished, you can evaluate and tune the model. When you're satisfied with the model, use the trained model for scoring with new data.

**List of modules**

The Classification category includes the following modules:

* ***Multiclass Decision Forest:*** Creates a multiclass classification model by using the decision forest algorithm.
* ***Multiclass Decision Jungle:*** Creates a multiclass classification model by using the decision jungle algorithm.
* ***Multiclass Logistic Regression:*** Creates a multiclass logistic regression classification model.
* ***Multiclass Neural Network:*** Creates a multiclass classification model by using a neural network algorithm.
* ***One-vs-All Multiclass:*** Creates a multiclass classification model from an ensemble of binary classification models.
* ***Two-Class Averaged Perceptron:*** Creates an averaged perceptron binary classification model.
* ***Two-Class Bayes Point Machine:*** Creates a Bayes point machine binary classification model.
* ***Two-Class Boosted Decision Tree:*** Creates a binary classifier by using a boosted decision tree algorithm.
* ***Two-Class Decision Forest:*** Creates a two-class classification model by using the decision forest algorithm.
* ***Two-Class Decision Jungle:*** Creates a two-class classification model by using the decision jungle algorithm.
* ***Two-Class Locally Deep Support Vector Machine:*** Creates a binary classification model by using the locally deep Support Vector Machine algorithm.
* ***Two-Class Logistic Regression:*** Creates a two-class logistic regression model.
* ***Two-Class Neural Network:*** Creates a binary classifier by using a neural network algorithm.
* ***Two-Class Support Vector Machine***: Creates a binary classification model by using the Support Vector Machine algorithm.

**Multiclass Decision Forest**

To create a machine learning model based on the decision forest algorithm. A decision forest is an ensemble model that very rapidly builds a series of decision trees while learning from tagged data.

**Decision forests**

The decision forest algorithm is an ensemble learning method for classification. The algorithm works by building multiple decision trees and then voting on the most popular output class. Voting is a form of aggregation, in which each tree in a classification decision forest 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 have a greater weight in the final decision of the ensemble.

Decision trees in general are non-parametric models, meaning they support data with varied distributions. In each tree, a sequence of simple tests is run for each class, increasing the levels of a tree structure until a leaf node (decision) is reached.

Decision trees have many advantages:

* They can represent non-linear decision boundaries.
* They are efficient in computation and memory usage during training and prediction.
* They perform integrated feature selection and classification.
* They are resilient in the presence of noisy features.

The decision forest classifier in Machine Learning Studio (classic) consists of an ensemble of decision trees. Generally, ensemble models provide better coverage and accuracy than single decision trees.

**How to configure Multiclass Decision Forest**

1. Add the Multiclass Decision Forest module to your experiment in Studio (classic). You can find this module under Machine Learning, Initialize Model, and Classification.
2. Double-click the module to open the Properties pane.
3. For Resampling method, choose the method used to create the individual trees. You can choose from bagging or replication.

* **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. For more information, see the Wikipedia entry for Bootstrap aggregating.
* **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, creating diverse trees.

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

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

1. **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 might increase.

This value also controls the number of trees displayed in the results, when visualizing the trained model. To see or print a single tree, you can set the value to 1; however, this means that only one tree can be produced (the tree with the initial set of parameters), and no further iterations are performed.

1. **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.
2. **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.
3. **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.
4. **Allow unknown values for categorical features:** Select this 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 present in the training data.
5. Connect 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 the Create trainer mode option to Parameter Range, use the Tune Model Hyperparameters module. With this option, the trainer can iterate over multiple combinations of the settings and determine the parameter values that produce the best model.

1. Run the experiment.

**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. To see the rules for each node, click each tree to drill down into the splits.

**Multiclass Decision Jungle**

To create a machine learning model that is based on a supervised learning algorithm called decision jungles. You define the model and its parameters using this module and then connect a labeled training data set to train the model using one of the training modules. The trained model can be used to predict a target that has multiple values

**Decision jungles**

Decision jungles are a recent extension to decision forests. 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 a better generalization performance than a decision tree, albeit at the cost of a somewhat higher training time.
* Decision jungles are non-parametric models, which 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 Multiclass Decision Jungle Model**

1. Add the Multiclass Decision Jungle module to your experiment in Studio (classic). You can find this module under Machine Learning, Initialize Model, and Classification.
2. Double-click the module to open the Properties pane.
3. Resampling method, choose the method for creating multiple trees, either bagging or replication.

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

Each tree in a decision forest outputs a Gaussian distribution by way of 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:** Select this option to use replication. In this method, each tree is trained on exactly the same input data. The determination of which split predicate is used for each tree node remains random, so diverse trees are created.

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

* Single Parameter: Use this option when you know how you want to configure the model.
* Parameter Range: Use this option if you are not sure of the best parameters, and want to use a parameter sweep.

1. Number of decision DAGs: Indicate the maximum number of graphs that can be created in the ensemble.
2. Maximum depth of the decision DAGs: Specify the maximum depth of each graph.
3. Maximum width of the decision DAGs: Specify the maximum width of each graph.
4. Number of optimization steps per decision DAG layer: Indicate how many iterations over the data to perform when building each DAG.
5. Allow unknown values for categorical features: Select this option to create a group for unknown values in testing or validation data. 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 values that were present in the training data.
6. Connect 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 the Tune Model Hyperparameters module. With this option, the algorithm iterates over multiple combinations of the settings you provided and determines the combination of values that produces the best model.

1. Run the experiment.

**Results**

After training is complete:

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

Multiclass Logistic Regression

To create a logistic regression model that can be used to predict multiple values. Classification using logistic regression is a supervised learning method, and therefore requires a labeled dataset. You train the model by providing the model and the labeled dataset as an input to a module such as Train Model or Tune Model Hyperparameters. The trained model can then be used to predict values for new input examples.

Machine Learning Studio (classic) also provides a Two-Class Logistic Regression module, which is suited for classification of binary or dichotomous variables.

**Multiclass logistic regression**

Logistic regression is a well-known method in statistics that is used to predict the probability of an outcome, and is particularly popular for classification tasks. The algorithm predicts the probability of occurrence of an event by fitting data to a logistic function. For details about this implementation, see the Technical Notes section. In multiclass logistic regression, the classifier can be used to predict multiple outcomes.

**How to configure a Multiclass Logistic Regression**

1. Add the Multiclass Logistic Regression module to the experiment.
2. Specify how you want the model to be trained, by setting the Create trainer mode option.

* 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 use a parameter sweep.

1. ***Optimization tolerance,*** specify the threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model.
2. ***L1 regularization weight, L2 regularization weight:*** Type a value to use for the regularization parameters L1 and L2. A non-zero value is recommended for both.

Regularization is a method for preventing overfitting by penalizing models with extreme coefficient values. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. An accurate model with extreme coefficient values would be penalized more, but a less accurate model with more conservative values would be penalized less.

L1 and L2 regularization have different effects and uses. L1 can be applied to sparse models, which is useful when working with high-dimensional data. In contrast, L2 regularization is preferable for data that is not sparse. This algorithm supports a linear combination of L1 and L2 regularization values: that is, if x = L1 and y = L2, ax + by = c defines the linear span of the regularization terms.

Different linear combinations of L1 and L2 terms have been devised for logistic regression models, such as elastic net regularization.

1. ***Memory size for L-BFGS:*** Specify the amount of memory to use for L-BFGS optimization. This parameter indicates the number of past positions and gradients to store for the computation of the next step.

L-BFGS stands for limited memory Broyden-Fletcher-Goldfarb-Shanno, and it is an optimization algorithm that is popular for parameter estimation. This optimization parameter limits the amount of memory that is used to compute the next step and direction. When you specify less memory, training is faster but less accurate.

1. ***Random number seed:*** Type an integer value to use as the seed for the algorithm if you want the results to be repeatable over runs. Otherwise, a system clock value is used as the seed, which can produce slightly different results in runs of the same experiment.
2. ***Allow unknown categorical levels:*** Select this option to create an additional “unknown” level in each categorical column. Any values (levels) in the test dataset that are not present in the training dataset are mapped to this "unknown" level.
3. Connect a labeled dataset, and one of the train 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. With this option, you can specify multiple values, and the trainer iterates over multiple combinations of the settings to determine the combination of values that produces the best model.

1. Run the experiment.

**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 the Train Model module or Tune Model Hyperparameters, and select Visualize.

**Multiclass Neural Network**

To create a neural network model that can be used to predict a target that has multiple values. For example, neural networks of this kind might be used in complex computer vision tasks, such as digit or letter recognition, document classification, and pattern recognition.

Classification using neural networks is a supervised learning method, and therefore requires a tagged dataset that includes a label column.

You can train the model by providing the model and the tagged dataset as an input to Train Model or to Tune Model Hyperparameters. The trained model can then be used to predict values for the new input examples.

**Neural networks**

A neural network is a set of interconnected layers. The inputs are the first layer, and are connected to an output layer by an acyclic graph comprised of weighted edges and nodes.

Between the input and output layers you can insert multiple hidden layers. Most predictive tasks can be accomplished easily with only one or a few hidden layers. However, recent research has shown that deep neural networks (DNN) with many layers can be very effective in complex tasks such as image or speech recognition. The successive layers are used to model increasing levels of semantic depth.

The relationship between inputs and outputs is learned from training the neural network on the input data. The direction of the graph proceeds from the inputs through the hidden layer and to the output layer. All nodes in a layer are connected by the weighted edges to nodes in the next layer.

To compute the output of the network for a particular input, a value is calculated at each node in the hidden layers and in the output layer. The value is set by calculating the weighted sum of the values of the nodes from the previous layer. An activation function is then applied to that weighted sum.

**How to configure Multiclass Neural Network**

1. Add the MultiClass Neural Network module to your experiment in Studio (classic). You can find this module under Machine Learning, Initialize, in the Classification category.
2. Create trainer mode: Use this option to specify how you want the model to be trained:

* Single Parameter: Choose this option if you already know how you want to configure the model.
* Parameter Range: Choose this option if you are not sure of the best parameters, and want to use a parameter sweep. You then specify a range of values and use the Tune Model Hyperparameters module to iterate over the combinations and find the optimal configuration.

1. Hidden layer specification: Select the type of network architecture to create.

* ***Fully-connected case:*** Select this option to create a model using the default neural network architecture. For multiclass neural network models, the defaults are as follows:
* One hidden layer
* The output layer is fully connected to the hidden layer.
* The hidden layer is fully connected to the input layer.
* The number of nodes in the input layer is determined by the number of features in the training data.
* The number of nodes in the hidden layer can be set by the user. The default is 100.
* The number of nodes in the output layer depends on the number of classes.
* ***Custom definition script.*** Choose this option to create a custom neural network architecture, using the Net# language. You can define the number of hidden layers, their connections, and advanced options such as specifying the mappings between layers.

1. **Neural network definition:** If you selected the custom architecture option, use the text box to type or paste in statements written in the Net# language.
2. **Number of hidden nodes:** This option lets you customize the number of hidden nodes in the default architecture. Type the number of hidden nodes. The default is one hidden layer with 100 nodes.
3. **The learning rate:** Define the size of the step taken at each iteration, before correction.A larger value for learning rate can cause the model to converge faster, but it can overshoot local minima.
4. **Number of learning iterations**: Specify the maximum number of times the algorithm should process the training cases.
5. **The initial learning weights diameter**: Specify the node weights at the start of the learning process.
6. **The momentum:** Specify a weight to apply during learning to nodes from previous iterations.
7. **The type of normalizer:** Select the method to use for feature normalization. The following normalization methods are supported:

* ***Binning normalizer*:** The binning normalizer creates bins of equal size, and then normalizes every value in each bin, by dividing by the total number of bins.
* ***Gaussian normalizer:*** The Gaussian normalizer rescales the values of each feature to have mean 0 and variance 1. This is done by computing the mean and the variance of each feature. For each instance, the mean value is subtracted, and the result divided by the square root of the variance (the standard deviation).
* ***Min-max normalizer:*** The min-max normalizer linearly rescales every feature to the [0,1] interval. Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).
* ***Do not normalize:*** No normalization is performed.

1. **Shuffle examples**: Select this option to shuffle cases between iterations.

If you deselect this option, cases are processed in exactly the same order each time you run the experiment.

1. **Random number seed:** Type a value to use as the seed, if you want to ensure repeatability across runs of the same experiment.
2. **Allow unknown categorical levels:** Select this option to create a grouping for unknown values in the training and validation sets. The model might be less precise on known values but provide better predictions for new (unknown) values. If you deselect this option, the model can accept only the values contained in the training data.
3. Connect a training dataset and one of the training modules:

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

**Results**

After training is complete:

* To see a summary of the model's parameters, together with the feature weights learned from training, and other parameters of the neural network, right-click the output of Train Model or Tune Model Hyperparameters, and select Visualize.
* To save a snapshot of the trained model, right-click the Trained model output and select Save As Trained Model. This model is not updated on successive runs of the same experiment.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Customizing the neural network using script :**In Machine Learning Studio (classic), you can customize the architecture of a neural network model by using the Net# language. Customizations supported by the Net# language include:

* Specifying the number of hidden layers and the number of nodes in each layer
* Specifying mappings between layers
* Defining convolutions and weight-sharing bundles
* Choosing the activation function

A neural network model is defined by the structure of its graph, which includes these attributes:

* The number of hidden layers
* The number of nodes in each hidden layer
* How the layers are connected
* Which activation function is used
* Weights on the graph edges

In general, the network has these defaults:

* The first layer is always the input layer.
* The last layer is always the output layer.
* The number of nodes in the output layer should be equal to the number of classes.

You can define any number of intermediate layers: these are sometimes called hidden layers, because they are contained within the model, and they are not directly exposed as endpoints.

The Net# reference guide explains the syntax and provides sample network definitions. It explains how you can use Net# to add hidden layers and define the way that the different layers interact with each other.

For example, the following script uses the auto keyword, which sets the number of features automatically for input and output layers, and uses the default values for the hidden layer.

Text:

*input Data auto;*

*hidden Hidden auto from Data all;*

*output Result auto from Hidden all;*

**One-vs-All Multiclass**

To create a classification model that can predict multiple classes, using the "one vs. all" approach. This module is useful for creating models that predict three or more possible outcomes, when the outcome depends on continuous or categorical predictor variables. This method also lets you use binary classification methods for issues that require multiple output classes.

**one-vs.all models**

While some classification algorithms permit the use of more than two classes by design, others restrict the possible outcomes to one of two values (a binary, or two-class model). However, even binary classification algorithms can be adapted for multi-class classification tasks using a variety of strategies. This module implements the one vs. all method, in which a binary model is created for each of the multiple output classes. Each of these binary models for the individual classes is assessed against its complement (all other classes in the model) as though it were a binary classification issue. Prediction is then performed by running these binary classifiers, and choosing the prediction with the highest confidence score. In essence, an ensemble of individual models is created and the results are then merged, to create a single model that predicts all classes. Thus, any binary classifier can be used as the basis for a one-vs-all model.

For example, let’s say you configure a Two-Class Support Vector Machine model and provide that as input to the One-Vs-All Multiclass module. The module would create two-class support vector machine models for all members of the output class and then apply the one-vs-all method to combine the results for all classes.

**How to Configure the One-vs-All Classifier**

This module creates an ensemble of binary classification models to analyze multiple classes. Therefore, to use this module, you need to configure and train a binary classification model first. You then connect the binary model to One-Vs-All Multiclass module, and train the ensemble of models by using Train Model with a labeled training dataset. When you combine the models, even though the training dataset might have multiple class values, the One-Vs-All Multiclass creates multiple binary classification models, optimizes the algorithm for each class, and then merges the models.

1. Add the One-Vs-All Multiclass to your experiment in Studio (classic). You can find this module under Machine Learning - Initialize, in the Classification category.

The One-Vs-All Multiclass classifier has no configurable parameters of its own. Any customizations must be done in the binary classification model that is provided as input.

1. Add a binary classification model to the experiment, and configure that model. For example, you might use a Two-Class Support Vector Machine or Two-Class Boosted Decision Tree.

If you need help choosing the right algorithm, see these resources:

* Machine learning algorithm cheat sheet for Azure ML
* How to choose algorithms for Machine Learning Studio (classic)

1. Add the Train Model module to your experiment, and connect the untrained classifier that is the output of One-Vs-All Multiclass.
2. On the other input of Train Model, connect a labeled training data set that has multiple class values.
3. Run the experiment, or select Train Model and click Run Selected.

**Results**

After training is complete, you can use the model to make multiclass predictions. Alternatively, you can pass the untrained classifier to Cross-Validate Model for cross-validation against a labeled validation data set.