**Regression modules**

Regression is a methodology used widely in fields ranging from engineering to education. For example, you might use regression to predict the value of a house based on regional data, or to create projections about future enrollment. Regression tasks are supported in many tools: for example, Excel provides "What If" analysis, forecasting over time, and the Analysis ToolPak for traditional regression. The modules for regression in Machine Learning Studio (classic) each incorporate a different method, or algorithm, for regression. In general, a regression algorithm tries to learn the value of a function for a particular instance of data. You might predict someone's height by using a height function, or predict the probability of hospital admission based on medical test values. Regression algorithms can incorporate input from multiple features, by determining the contribution of each feature of the data to the regression function.

**List of modules**

* Bayesian Linear Regression: Creates a Bayesian linear regression model.
* Boosted Decision Tree Regression: Creates a regression model by using the Boosted Decision Tree algorithm.
* Decision Forest Regression: Creates a regression model by using the decision forest algorithm.
* Fast Forest Quantile Regression: Creates a quantile regression model.
* Linear Regression: Creates a linear regression model.
* Neural Network Regression: Creates a regression model by using a neural network algorithm.
* Ordinal Regression: Creates an ordinal regression model.
* Poisson Regression: Creates a regression model that assumes data has a Poisson distribution.

**Bayesian Linear Regression**

To define a regression model based on Bayesian statistics. After you have defined the model parameters, you must train the model using a tagged dataset and the Train Model module. The trained model can then be used to make predictions. Alternatively, the untrained model can be passed to Cross-Validate Model for cross-validation against a labeled data set.

**Bayesian regression**

In statistics, the Bayesian approach to regression is often contrasted with the frequentist approach. The Bayesian approach uses linear regression supplemented by additional information in the form of a prior probability distribution. Prior information about the parameters is combined with a likelihood function to generate estimates for the parameters. In contrast, the frequentist approach, represented by standard least-square linear regression, assumes that the data contains sufficient measurements to create a meaningful model.

**How to configure Bayesian Regression**

1. Add the ***Bayesian Linear Regression*** module to your experiment. You can find the this module under Machine Learning, Initialize, in the Regression category.
2. ***Regularization weight:*** Type a value to use for regularization. Regularization is used to prevent overfitting. This weight corresponds to L2.
3. Allow unknown categorical levels: Select this option to create a grouping for unknown values. The model can accept only the values contained in the training data. The model might be less precise on known values but provide better predictions for new (unknown) values.
4. Connect a training dataset, and one of the training modules. This model type has no parameters that can be changed in a parameter sweep, so although you can train the model using Tune Model Hyperparameters, it cannot automatically optimize the model.
5. Select the single numeric column that you want to model or predict.
6. Run the experiment.

**Results**

After training is complete:

* To see a summary of the model's parameters, right-click the output of the Train Model module and select Visualize.
* To create predictions, use the trained model as an input to Score Model.

**Boosted Decision Tree Regression**

To create an ensemble of regression trees using boosting. Boosting means that each tree is dependent on prior trees. The algorithm learns by fitting the residual of the trees that preceded it. Thus, boosting in a decision tree ensemble tends to improve accuracy with some small risk of less coverage. This regression method is a supervised learning method, and therefore requires a labeled dataset. The label column must contain numerical values. boosted regression trees

Boosting is one of several classic methods for creating ensemble models, along with bagging, random forests, and so forth. In Machine Learning Studio (classic), boosted decision trees use an efficient implementation of the MART gradient boosting algorithm. Gradient boosting is a machine learning technique for regression problems. It builds each regression tree in a step-wise fashion, using a predefined loss function to measure the error in each step and correct for it in the next. Thus the prediction model is actually an ensemble of weaker prediction models. In regression problems, boosting builds a series of trees in a step-wise fashion, and then selects the optimal tree using an arbitrary differentiable loss function. The gradient boosting method can also be used for classification problems by reducing them to regression with a suitable loss function. For more information about the boosted trees implementation for classification tasks, see Two-Class Boosted Decision Tree.

**How to configure Boosted Decision Tree Regression**

1. Add the Boosted Decision Tree module to your experiment. You can find this module under Machine Learning, Initialize, under the Regression category.
2. 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 specific set of values as arguments.
* Parameter Range: Select this option if you are not sure of the best parameters, and want to run a parameter sweep. Select a range of values to iterate over, and the Tune Model Hyperparameters iterates over all possible combinations of the settings you provided to determine the hyperparameters that produce the optimal results.

1. 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. Minimum number of samples per leaf node: Indicate the minimum 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. 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.
2. 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 increases.

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

1. Random number seed: Type an optional non-negative integer to use as the random seed value. Specifying a seed ensures reproducibility across runs that have the same data and parameters. By default, the random seed is set to 0, which means the initial seed value is obtained from the system clock.
2. Allow unknown categorical levels: Select this 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. The model might be less precise for known values, but it can provide better predictions for new (unknown) values.
3. Add a training dataset, and one of the training modules:
4. If you set Create trainer mode option to Single Parameter, use the Train Model module.
5. If you set Create trainer mode to Parameter Range, use the Tune Model Hyperparameters module.
6. 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. 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.
* To save a snapshot of the trained model, right-click the Trained model output of the training module and select Save As. The copy of the trained model that you save is not updated on successive runs of the experiment.

**Implementation details**The ensemble of trees is produced by computing, at each step, a regression tree that approximates the gradient of the loss function, and adding it to the previous tree with coefficients that minimize the loss of the new tree. The output of the ensemble produced by MART on a given instance is the sum of the tree outputs.

* For binary classification problem, the output is converted to probability by using some form of calibration.
* For regression problems, the output is the predicted value of the function.
* For ranking problems, the instances are ordered by the output value of the ensemble.

**Module parameters**

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

**Fast Forest Quantile Regression**

To create a regression model that can predict values for a specified number of quantiles. Quantile regression is useful if you want to understand more about the distribution of the predicted value, rather than get a single mean prediction value. This method has many applications, including:

* Predicting prices
* Estimating student performance or applying growth charts to assess child development
* Discovering predictive relationships in cases where there is only a weak relationship between variables

This regression algorithm is a supervised learning method, which means it requires a tagged dataset that includes a label column. Because it is a regression algorithm, the label column must contain only numerical values.

**Quantile regression**

There are many different types of regression. In the most basic sense, regression means fitting a model to a target expressed as a numeric vector. However, statisticians have been developing increasingly advanced methods for regression. The simplest definition of quantile is a value that divides a set of data into equal-sized groups; thus, the quantile values mark the boundaries between groups. Statistically speaking, quantiles are values taken at regular intervals from the inverse of the cumulative distribution function (CDF) of a random variable. Whereas linear regression models attempt to predict the value of a numeric variable using a single estimate, the mean, sometimes you need to predict the range or entire distribution of the target variable. Techniques such as Bayesian regression and quantile regression have been developed for this purpose. Quantile regression helps you understand the distribution of the predicted value. Tree-based quantile regression models, such as the one used in this module, have the additional advantage that they can be used to predict non-parametric distributions.

**How to configure Fast\_Forest Quantile Regression**

You configure the properties of the regression model using this module, and then train it using one of the training modules.

Configuration steps differ considerably dependng on whether you are providing a fixed set of parameters, or setting up a parameter sweep.

* To create a quantile regression model using fixed parameters
* To create a quantile regression model using a parameter sweep

**Create a quantile regression model using fixed parameters**

Assuming you know how you want to configure the model, you can provide a specific set of values as arguments. When you train the model, use Train Model.

1. Add the Fast Forest Quantile Regression module to your experiment in Studio (classic).
2. Set the Create trainer mode option to Single Parameter.
3. For Number of Trees, type the maximum number of trees that can be created in the ensemble. If you create more trees, it generally leads to greater accuracy, but at the cost of longer training time.
4. For Number of Leaves, type the maximum number of leaves, or terminal nodes, that can be created in any tree.
5. For Minimum number of training instances required to form a leaf , specify the minimum number of examples 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
6. For Bagging fraction, specify a number between 0 and 1 that represents the fraction of samples to use when building each group of quantiles. Samples are chosen randomly, with replacement.
7. For Feature fraction, type a number between 0 and 1 that indicates the fraction of total features to use when building any particular tree. Features are always chosen randomly.
8. For Split fraction, type a number between 0 and 1 that represents the fraction of features to use in each split of the tree. The features used are always chosen randomly.
9. For Quantile sample count, type the number of cases to evaluate when estimating the quantiles.
10. For Quantiles to be estimated, type a comma-separated list of the quantiles for which you want the model to train and create predictions. For example, if you want to build a model that estimates for quartiles, you would type 0.25, 0.5, 0.75.
11. Optionally, type a value for Random number seed to seed the random number generator used by the model. The default is 0, meaning a random seed is chosen. You should provide a value if you need to reproduce results across successive runs on the same data.
12. Select the Allow unknown categorical levels option to create a group for unknown values. If you deselect it, the model can accept only the values that are contained in the training data. If you select this option, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.
13. Connect a training dataset, select a single label column, and connect Train Model.
14. Run the experiment.

**Use a parameter sweep to create a quantile regression model**

If you are not sure of the optimal parameters for the model, you can configure a parameter sweep, and provide a range of values as arguments. When you train the model, use the Tune Model Hyperparameters module.

1. Add the Fast Forest Quantile Regression module to your experiment in Studio (classic).
2. Set the Create trainer mode option to Parameter Range.

A parameter sweep is recommended if you are not sure of the best parameters. By specifying multiple values and using the Tune Model Hyperparameters module to train the model, you can find the optimal set of parameters for your data. After choosing a parameter sweep, for each property that is tunable, you can set either a single value, or multiple values. For example, you might decide to fix the number of trees, but randomly change other values that control the way each tree is built.

* If you type a single value, that value is used across all iterations of the sweep, even if other values change.
* Type a comma-separated list of discrete values to use. These values are used in combination with other properties.
* Use the Range Builder to define a range of continuous values.

During the training process, the Tune Model Hyperparameters module iterates over various combinations of the values to build the best model.

1. For Maximum number of leaves per tree, type the total number of leaves, or terminal nodes, to allow in each tree.
2. For Number of trees constructed, type the number of iterations to perform when constructing the ensemble. By creating more trees, you can potentially get better coverage, at the expense of increased training time.
3. For Minimum number of sample per leaf node, indicate how many cases are required to create a leaf node. 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. In Range for bagging fraction, type the fraction of samples to use when building each group of quantiles. Samples are chosen randomly, with replacement. Each fraction should be a number between 0 and 1. Separate multiple fractions, by using commas.
5. In Range for feature fraction, type the fraction of total features to use when building each group of quantiles. Features are chosen randomly. Each fraction should be a number between 0 and 1; separate multiple fractions by using commas.
6. In Range for split fraction, specify some fraction of features to use in each group of quantiles. The actual features used are chosen randomly.Each fraction should be a number between 0 and 1; separate multiple fractions by using commas.
7. In Sample count used to estimate the quantiles, indicate how many samples should be evaluated when estimating the quantiles. If you type a number greater than the number of available samples, all samples are used.
8. In Required quantile values, type a comma-separated list of the quantiles on which you want the model to train. For example, if you want to build a model that estimates quartiles, you would type `0.25, 0.5, 0.75
9. In Random number seed, type a value to seed the random number generator used by the model. Use of a seed is useful in order to reproduce duplicate runs.The default is 0, meaning a random seed is chosen.
10. Select the Allow unknown values for categorical features option to create a group for unknown values in the training or validation sets. If you deselect this option, the model can accept only the values that are contained in the training data. If you select this option, the model might be less precise for known values, but it can provide better predictions for new (unknown) values.
11. Connect a training dataset, select the label column, and connect the Tune Model Hyperparameters module.
12. Run the experiment.

**Results**

After training is complete:

To see the final hyperparameters of the optimized model, right-click the output of Tune Model Hyperparameters and select Visualize.

**Implementation details**

The Fast Forest Quantile Regression module in Machine Learning is an implementation of random forest quantile regression using decision trees. Random forests can be helpful to avoid overfitting that can occur with decision trees. A decision tree is a binary tree-like flow chart, where at every interior node, one decides which of the two child nodes to continue to, based on the value of one of the features of the input. In each leaf node, a value is returned. In the interior nodes, the decision is based on the test ``x≤v`, where x is the value of the feature in the input sample and v is one of the possible values of this feature. The functions that can be produced by a regression tree are all the piece-wise constant functions. In a random forest, an ensemble of trees is created by using bagging to select a subset of random samples and features of the training data, and then fit a decision tree to each subset of data. Unlike the random forest algorithm, which averages out the output of the all the trees, Fast Forest Quantile Regression keeps all the predicted labels in trees specified by the parameter Quantile sample count and outputs the distribution, so that the user can view the quantile values for the given instance.