**Regression**

**Linear regression**

To create a linear regression model for use in an experiment. Linear regression attempts to establish a linear relationship between one or more independent variables and a numeric outcome, or dependent variable. You use this module to define a linear regression method, and then train a model using a labeled dataset. 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.

Linear regression is a common statistical method, which has been adopted in machine learning and enhanced with many new methods for fitting the line and measuring error. In the most basic sense, regression refers to prediction of a numeric target. Linear regression is still a good choice when you want a very simple model for a basic predictive task. Linear regression also tends to work well on high-dimensional, sparse data sets lacking complexity.

Machine Learning Studio (classic) supports a variety of regression models, in addition to linear regression. However, the term "regression" can be interpreted loosely, and some types of regression provided in other tools are not supported in Studio (classic).

* The ***classic regression*** problem involves a single independent variable and a dependent variable. This is called simple regression. This module supports simple regression.
* ***Multiple linear regression*** involves two or more independent variables that contribute to a single dependent variable. Problems in which multiple inputs are used to predict a single numeric outcome are also called *multivariate linear regression.* The Linear Regression module can solve these problems, as can most of the other regression modules in Studio (classic).
* ***Multi-label regression*** is the task of predicting multiple dependent variables within a single model. For example, in multi-label logistic regression, a sample can be assigned to multiple different labels. (This is different from the task of predicting multiple levels within a single class variable). This type of regression is not supported in Machine Learning. To predict multiple variables, create a separate learner for each output that you wish to predict. For years statisticians have been developing increasingly advanced methods for regression. This is true even for linear regression. This module supports two methods to measure error and fit the regression line: ordinary least squares method, and gradient descent.
* **Gradient descent** is a method that minimizes the amount of error at each step of the model training process. There are many variations on gradient descent and its optimization for various learning problems has been extensively studied. If you choose this option for Solution method, you can set a variety of parameters to control the step size, learning rate, and so forth. This option also supports use of an integrated parameter sweep.
* ***Ordinary least squares*** is one of the most commonly used techniques in linear regression. For example, least squares is the method that is used in the Analysis Toolpak for Microsoft Excel.Ordinary least squares refers to the loss function, which computes error as the sum of the square of distance from the actual value to the predicted line, and fits the model by minimizing the squared error. This method assumes a strong linear relationship between the inputs and the dependent variable.

**How to configure Linear Regression**

This module supports two methods for fitting a regression model, with very different options:

* Create a regression model using online gradient descent

Gradient descent is a better loss function for models that are more complex, or that have too little training data given the number of variables. This option also supports a parameter sweep, if you train the model using Tune Model Hyperparameters to automatically optimize the model parameters.

* Fit a regression model using ordinary least squares

For small datasets, it is best to select ordinary least squares. This should give very similar results to Excel.

**Create a regression model using ordinary least squares**

1. Add the Linear Regression Model module to your experiment in Studio (classic).

You can find this module in the Machine Learning category. Expand Initialize Model, expand Regression, and then drag the Linear Regression Model module to your experiment.

1. In the Properties pane, in the Solution method dropdown list, select Ordinary Least Squares. This option specifies the computation method used to find the regression line.
2. In L2 regularization weight, type the value to use as the weight for L2 regularization. We recommend that you use a non-zero value to avoid overfitting.
3. Select the option, Include intercept term, if you want to view the term for the intercept. Deselect this option if you don't need to review the regression formula.
4. For Random number seed, you can optionally type a value to seed the random number generator used by the model.

Using a seed value is useful if you want to maintain the same results across different runs of the same experiment. Otherwise, the default is to use a value from the system clock.

1. Deselect the option, Allow unknown categorical levels, if you want missing values to raise an error.

If this option is selected, an additional level is created for each categorical column. Any levels in the test dataset that were not present in the training dataset are mapped to this additional level.

1. Add the Train Model module to your experiment, and connect a labeled dataset.
2. Run the experiment.

**Results for ordinary least squares model**

After training is complete:

* To view the model's parameters, right-click the trainer output and select Visualize.
* To make predictions, connect the trained model to the Score Model module, along with a dataset of new values.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Create a regression model using online gradient descent**

1. Add the Linear Regression Model module to your experiment in Studio (classic).

You can find this module in the Machine Learning category. Expand Initialize Model, expand Regression, and drag the Linear Regression Model module to your experiment

1. In the Properties pane, in the Solution method dropdown list, choose Online Gradient Descent as the computation method used to find the regression line.
2. For Create trainer mode, indicate whether you want to train the model with a predefined set of parameters, or if you want to optimize the model by using a parameter sweep.

* Single Parameter: If you know how you want to configure the linear regression network, you can provide a specific set of values as arguments.
* Parameter Range: If you want the algorithm to find the best parameters for you, set Create trainer mode option to Parameter Range. You can then specify multiple values for the algorithm to try.

1. For Learning rate, specify the initial learning rate for the stochastic gradient descent optimizer.
2. For Number of training epochs, type a value that indicates how many times the algorithm should iterate through examples. For datasets with a small number of examples, this number should be large to reach convergence.
3. Normalize features: If you have already normalized the numeric data used to train the model, you can deselect this option. By default, the module normalizes all numeric inputs to a range between 0 and 1.
4. In L2 regularization weight, type the value to use as the weight for L2 regularization. We recommend that you use a non-zero value to avoid overfitting.
5. Select the option, Average final hypothesis, to average the final hypothesis.

In regression models, hypothesis testing means using some statistic to evaluate the probability of the null hypothesis, which states that there is no linear correlation between a dependent and independent variable. In many regression problems, you must test a hypothesis involving more than one variable.This option is enabled by default, meaning the algorithm tests a combination of the parameters where two or more parameters are involved.

1. Select the option, Decrease learning rate, if you want the learning rate to decrease as iterations progress.
2. For Random number seed, you can optionally type a value to seed the random number generator used by the model. Using a seed value is useful if you want to maintain the same results across different runs of the same experiment.
3. Deselect the option, Allow unknown categorical levels, if you want missing values to raise an error. When this option is selected, an additional level is created for each categorical column. Any levels in the test dataset not present in the training dataset are mapped to this additional level.
4. Add a labeled dataset and one of the training modules.

If you are not using a parameter sweep, use the Train Model module. To have the algorithm find the best parameters for you, train the model using Tune Model Hyperparameters.

1. Run the experiment.

**Results for online gradient descent**

After training is complete:

* To make predictions, connect the trained model to the Score Model module, together with new input data.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Neural Network Regression**

To create a regression model using a customizable neural network algorithm. Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

Neural network regression is a supervised learning method, and therefore requires a tagged dataset, which includes a label column. Because a regression model predicts a numerical value, the label column must be a numerical data type. 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.

**How to configure Neural Network Regression**

Neural networks can be extensively customized. This section describes how to create a model using two methods:

* Create a neural network model using the default architecture

If you accept the default neural network architecture, use the Properties pane to set parameters that control the behavior of the neural network, such as the number of nodes in the hidden layer, learning rate, and normalization.

Start here if you are new to neural networks. The module supports many customizations, as well as model tuning, without deep knowledge of neural networks.

* Define a custom architecture for a neural network

Use this option if you want to add extra hidden layers, or fully customize the network architecture, its connections, and activation functions.

This option is best if you are already somewhat familiar with neural networks. You use the Net# language to define the network architecture.

**Create a neural network model using the default architecture**

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

* 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. Then, specify a range of values and use the Tune Model Hyperparameters module to iterate over the combinations and find the optimal configuration.

1. In Hidden layer specification, select Fully-connected case. This option creates a model using the default neural network architecture, which for a neural network regression model, has these attributes:

* The network has exactly one hidden layer.
* The output layer is fully connected to the hidden layer and the hidden layer is fully connected to the input layer.
* The number of nodes in the hidden layer can be set by the user (default value is 100).

Because the number of nodes in the input layer is determined by the number of features in the training data, in a regression model there can be only one node in the output layer.

1. For Number of hidden nodes, type the number of hidden nodes. The default is one hidden layer with 100 nodes. (This option is not available if you define a custom architecture using Net#.)
2. For Learning rate, type a value that defines 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.
3. For Number of learning iterations, specify the maximum number of times the algorithm processes the training cases.
4. For The initial learning weights diameter , type a value that determines the node weights at the start of the learning process.
5. For The momentum, type a value to apply during learning as a weight on nodes from previous iterations.
6. For The type of normalizer, choose one of the following methods to use for feature normalization:

Binning normalizer: Binning creates groups of equal size, and then normalizes every value in each group to be divided by the total number of groups.

* Gaussian normalizer: Gaussian normalization 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, and then, for each instance, subtracting the mean value and dividing by the square root of the variance (the standard deviation).
* Min-Max normalizer: Min-max normalization 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. Select the option, Shuffle examples, to change the order of cases between iterations. If you deselect this option, cases are processed in exactly the same order each time you run the experiment.
2. For Random number seed, you can optionally type a value to use as the seed. Specifying a seed value is useful when you want to ensure repeatability across runs of the same experiment.
3. Select the option Allow unknown categorical levels to create a grouping for unknown values. 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.
4. Connect a training datset 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.

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, 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.

**Ordinal Regression**

Ordinal regression is used when the label or target column contains numbers, but the numbers represent a ranking or order rather than a numeric measurement. Predicting ordinal numbers requires a different algorithm than predicting the values of numbers on a continuous scale, because the numbers assigned to represent rank order do not have intrinsic scale. For example, to predict students’ test scores, you would use a standard regression model, because students’ test scores vary on a continuous scale and can be measured. However, to predict their class ranking, you must use an ordinal regression model.

**How to configure Ordinal Regression**

This module solves a ranking problem as a series of related classification problems. Therefore, the algorithm creates a series of extended training examples using a binary model for each rank, and trains against that extended set. This operation can be computationally expensive.

1. Add the Ordinal Regression Model module to your experiment in Studio (classic). You can find this module under Machine Learning - Initialize, in the Regression category.
2. Add a module that supports binary classification, and configure the model. There are several two-class modules in the classification category.
3. Connect the binary classification model as an input to the Ordinal Regression Model module.
4. Additional parameters are not required on the Ordinal Regression Model; the algorithm has been pre-configured with the most effective parameters for solving a ranking problem.
5. Connect a training dataset and the Train Model module.
6. In the Train Model module, select the column that contains the rank values.The rank values must be numerical values, but they need not be integers or positive numbers, as long as they represent a sequence. For purposes of processing, the ranks are assumed to have the order 1 to K, where 1 is lowest rank, and K is the highest rank. However, the Train Model module can work even if the semantics of your scale are reversed.For example, if in your original survey, 1 was the highest score and 5 is the lowest, it does not affect the processing of the model.
7. Run the experiment.

**Results**

After training is complete:

* To make predictions, connect the trained model, along with new data, to the Score Model module.
* To perform cross-validation against a labeled data set, connect the untrained model to Cross-Validate Model.

**Restrictions on input data**

You can use any numeric column as the target of an ordinal regression model, but in practice you should use only data that represents some sort of order or ranking. The intervals between ranks are assumed to be unknown and the size of the interval does not matter to the model; however, the model assumes that the sequence of ranks follows the natural ordering of numbers. The model itself does not assign any meaning to a particular scale. In other words, you might create one model in which 1 is a good rank and 10 is the worst, and in another model assume that 10 is the desired rank and 1 is the worst.

**Ranking algorithm**

The training set (X,Y) consists of input vectors x and labels y. The labels represents ranks ranging from 1 to k in sequence: 1,2, … , K. The ranks are assumed to be ordered such that 1 is the lowest or the worst rank, and K is the best or highest rank. The crux of the algorithm lies in modifying the given input features X and labels Y to use extended examples, and then using a binary classifier to solve the ordinal regression problem. The binary classifier is trained to give a yes/no answer to the question, “Is the rank greater than r?”

For example, for each case in the training set there are K-1 extended examples, and the maximum observed rank is K. The extended features are formed by appending the ith row of a K-1 x K-1 identity matrix to the input features for all i. The labels are given +1 for the first r-1 rows if its rank is r and -1 to the rest.

**Sample calculations**

To illustrate how it works, let x1 be the training feature whose rank is 3, where the maximum observed rank is 5. The extended examples corresponding to this feature are as follows:

| **Case** | **Test** | **Resulting label** |
| --- | --- | --- |
| X11000 | Is rank greater than 1? | Yes; therefore +1 |
| X10100 | Is rank greater than 2? | Yes; therefore +1 |
| X10010 | Is rank greater than 3? | No; therefore no additional feature |
| X10001 | Is rank greater than 4? | No; therefore no additional feature |

**Poisson Regression**

Poisson regression is intended for use in regression models that are used to predict numeric values, typically counts. Therefore, you should use this module to create your regression model only if the values you are trying to predict fit the following conditions:

* The response variable has a Poisson distribution.
* Counts cannot be negative. The method will fail outright if you attempt to use it with negative labels.
* A Poisson distribution is a discrete distribution; therefore, it is not meaningful to use this method with non-whole numbers.

After you have set up the regression method, you must train the model using a dataset containing examples of the value you want to predict. The trained model can then be used to make predictions.

Poisson regression is a special type of regression analysis that is typically used to model counts. For example, Poisson regression would be useful in these scenarios:

* Modeling the number of colds associated with airplane flights
* Estimating the number of emergency service calls during an event
* Projecting the number of customer inquiries subsequent to a promotion
* Creating contingency tables

Because the response variable has a Poisson distribution, the model makes different assumptions about the data and its probability distribution than, say, least-squares regression. Therefore, Poisson models should be interpreted differently from other regression models.

**How to configure Poisson Regression**

1. Add the Poisson Regression module to your experiment in Studio (classic).

You can find this module under Machine Learning - Initialize, in the Regression category.

1. Add a dataset that contains training data of the correct type.

We recommend that you use Normalize Data to normalize the input dataset before using it to train the regressor.

1. In the Properties pane of the Poisson Regression module, 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, do a parameter sweep using the Tune Model Hyperparameters module. The trainer iterates over multiple values you specify to find the optimal configuration.

1. Optimization tolerance: Type a value that defines the tolerance interval during optimization. The lower the value, the slower and more accurate the fitting.
2. L1 regularization weight and L2 regularization weight: Type values to use for L1 and L2 regularization. Regularization adds constraints to the algorithm regarding aspects of the model that are independent of the training data. Regularization is commonly used to avoid overfitting.

* L1 regularization is useful if the goal is to have a model that is as sparse as possible.L1 regularization is done by subtracting the L1 weight of the weight vector from the loss expression that the learner is trying to minimize. The L1 norm is a good approximation to the L0 norm, which is the number of non-zero coordinates.
* L2 regularization prevents any single coordinate in the weight vector from growing too much in magnitude. L2 regularization is useful if the goal is to have a model with small overall weights.

In this module, you can apply a combination of L1 and L2 regularizations. By combining L1 and L2 regularization, you can impose a penalty on the magnitude of the parameter values. The learner tries to minimize the penalty, in a tradeoff with minimizing the loss.

1. Memory size for L-BFGS: Specify the amount of memory to reserve for model fitting and optimization.

L-BFGS is a specific method for optimization, based on the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. The method uses a limited amount of memory (L) to compute the next step direction. By changing this parameter, you can affect the number of past positions and gradients that are stored for computation of the next step.

1. Connect the training dataset and the untrained model to 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.

1. Run the experiment to train the model.