**Logistic Regression for Machine Learning**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on April 1, 2016 in [**Machine Learning Algorithms**](https://machinelearningmastery.com/category/machine-learning-algorithms/)

Logistic regression is another technique borrowed by machine learning from the field of statistics.

It is the go-to method for binary classification problems (problems with two class values). In this post you will discover the logistic regression algorithm for machine learning.

After reading this post you will know:

* The many names and terms used when describing logistic regression (like log odds and logit).
* The representation used for a logistic regression model.
* Techniques used to learn the coefficients of a logistic regression model from data.
* How to actually make predictions using a learned logistic regression model.
* Where to go for more information if you want to dig a little deeper.

This post was written for developers interested in applied machine learning, specifically predictive modeling. You do not need to have a background in linear algebra or statistics. Let’s get started.

## Logistic Function

Logistic regression is named for the function used at the core of the method, the logistic function.The [logistic function](https://en.wikipedia.org/wiki/Logistic_function), also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e-value)

Where e is the [base of the natural logarithms](https://en.wikipedia.org/wiki/E_(mathematical_constant)) (Euler’s number or the EXP() function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.



Logistic Function

Now that we know what the logistic function is, let’s see how it is used in logistic regression.

## Representation Used for Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression.

Input values (x) are combined linearly using weights or coefficient values (referred to as the Greek capital letter Beta) to predict an output value (y). A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.

Below is an example logistic regression equation:

y = e(b0 + b1\*x) / (1 + e(b0 + b1\*x))

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data.

The actual representation of the model that you would store in memory or in a file are the coefficients in the equation (the beta value or b’s).

## Logistic Regression Predicts Probabilities (Technical Interlude)

Logistic regression models the probability of the default class (e.g. the first class).

For example, if we are modeling people’s sex as male or female from their height, then the first class could be male and the logistic regression model could be written as the probability of male given a person’s height, or more formally:

P(sex=male|height)

Written another way, we are modeling the probability that an input (X) belongs to the default class (Y=1), we can write this formally as:

P(X) = P(Y=1|X)

We’re predicting probabilities? I thought logistic regression was a classification algorithm?

Note that the probability prediction must be transformed into a binary values (0 or 1) in order to actually make a probability prediction. More on this later when we talk about making predictions.

Logistic regression is a linear method, but the predictions are transformed using the logistic function. The impact of this is that we can no longer understand the predictions as a linear combination of the inputs as we can with linear regression, for example, continuing on from above, the model can be stated as:

p(X) = e(b0 + b1\*X) / (1 + e(b0 + b1\*X))

I don’t want to dive into the math too much, but we can turn around the above equation as follows (remember we can remove the e from one side by adding a natural logarithm (ln) to the other):

ln(p(X) / 1 – p(X)) = b0 + b1 \* X

This is useful because we can see that the calculation of the output on the right is linear again (just like linear regression), and the input on the left is a log of the probability of the default class.

This ratio on the left is called the odds of the default class (it’s historical that we use odds, for example, odds are used in horse racing rather than probabilities). Odds are calculated as a ratio of the probability of the event divided by the probability of not the event, e.g. 0.8/(1-0.8) which has the odds of 4. So we could instead write:

ln(odds) = b0 + b1 \* X

Because the odds are log transformed, we call this left hand side the log-odds or the probit. It is possible to use other types of functions for the transform (which is out of scope\_, but as such it is common to refer to the transform that relates the linear regression equation to the probabilities as the link function, e.g. the probit link function.

We can move the exponent back to the right and write it as:

odds = e(b0 + b1 \* X)

All of this helps us understand that indeed the model is still a linear combination of the inputs, but that this linear combination relates to the log-odds of the default class.

## Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using maximum-likelihood estimation.

[Maximum-likelihood estimation](https://en.wikipedia.org/wiki/Maximum_likelihood) is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).

The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

We are not going to go into the math of maximum likelihood. It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data. This is often implemented in practice using efficient numerical optimization algorithm (like the [Quasi-newton method](https://en.wikipedia.org/wiki/Quasi-Newton_method)).

When you are learning logistic, you can implement it yourself from scratch using the much simpler gradient descent algorithm.

## Making Predictions with Logistic Regression

Making predictions with a logistic regression model is as simple as plugging in numbers into the logistic regression equation and calculating a result.

Let’s make this concrete with a specific example.

Let’s say we have a model that can predict whether a person is male or female based on their height (completely fictitious). Given a height of 150cm is the person male or female.

We have learned the coefficients of b0 = -100 and b1 = 0.6. Using the equation above we can calculate the probability of male given a height of 150cm or more formally P(male|height=150). We will use EXP() for e, because that is what you can use if you type this example into your spreadsheet:

y = e(b0 + b1\*X) / (1 + e(b0 + b1\*X))

y = EXP(-100 + 0.6\*150) / (1 + EXP(-100 + 0.6\*X))

y = 0.0000453978687

Or a probability of near zero that the person is a male.

In practice we can use the probabilities directly. Because this is classification and we want a crisp answer, we can snap the probabilities to a binary class value, for example:

0 if p(male) < 0.5

1 if p(male) >= 0.5

Now that we know how to make predictions using logistic regression, let’s look at how we can prepare our data to get the most from the technique.

## Prepare Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

* **Binary Output Variable**: This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
* **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
* **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
* **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
* **Fail to Converge**: It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).