CS 4860: Home Work Assignment 1: Regression

Cameron Stacey
University of Colorado Colorado Springs
cstacey@uccs.edu

Abstract

In this homework assignment, I will be deriving the mathematics, analyzing the meaning, and implementing the algorithms for two different types of regression: least squares linear regression and logistic regression. I will also be looking at how different tools such as Weka, and R are extremely useful for machine learning and data analyzation, as well as the importance of dataset selection and optimization.

Introduction

This homework assignment will let me explore one of the main foundations of machine learning methods: regression.

Least Squares Linear Regression

This section explores analysis and implementation of a least squares linear regression algorithm. After defining the objective function and deriving the equation of a straight line that optimizes it, I'll report the results of my least squares linear regression implementation as well as testing it with the "Combined Cycle Power Plant" dataset stated in the assignment. I utilized the Numerical Methods for Engineers textbook for the derivation (Chapra and Canale, 2015).

Objective Function

The objective function that needs to be solved for Least Squares Linear Regression is a simple straight line equation:

$$y = a_0 + a_1 x + e$$

In this case, a_0 and a_1 are coefficients that represent the intercept and the slope, respectively, and e is the value of the error between the model and observations, which can be expressed by solving the given equation for e:

$$e = y - a_0 - a_1 x$$

Straight Line Optimization Derivation

The best way to find the best fit line for a set of linear data is to minimize the sum of the squares of the residuals between the measured y values and the predicted y values from our best fit line. This can be expressed mathematically through the following equation:

$$S_r = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

The values for a_0 and a_1 can be found by differentiating the equation with respect to each coefficient separately:

$$\frac{\partial S_r}{\partial a_0} = -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)$$

$$\frac{\partial S_r}{\partial a_1} = -2 \sum_{i=1}^{n} [(y_i - a_0 - a_1 x_i) x_i]$$

Setting these derivatives equal to zero will result in a minimized S_r function. These can then be simplified into two linear equations:

$$n a_0 + (\sum_{i=1}^{n} x_i) a_1 = \sum_{i=1}^{n} y_i$$

$$(\sum_{i=1}^{n} x_i) a_0 + (\sum_{i=1}^{n} x_i^2) a_1 = \sum_{i=1}^{n} x_i y_i$$

Solving for a_1 and a_0, we get:

$$a_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$a_0 = \bar{y} - a_1 \bar{x}$$

Where $\bar{y}$ and $\bar{x}$ are the means of y and x respectively.

This example is using the simple case of having one independent variable influencing the dependent variable. Almost all machine learning problems have more than one independent variable, or feature, involved in the calculation, but I kept it to the simplified case in my derivation so that it is understandable and easy to follow.

Implementation and Test Results

I implemented a least squares linear regression using Python and Scikit-learn. I also used the pandas library to create a dataframe that is easy to work with. The linear regression came up with this equation to estimate potential energy (PE):

$$PE = -1.98AT - 0.23V + 0.06AP - 0.16RH + 454.62$$

I also calculated the Mean Squared Error to be 20.7674, and the coefficient of determination R^2 to be 0.9287, which means that the prediction is extremely accurate to the actual values for potential energy.

Logistic Regression

This section explores analysis and implementation of a logistic regression function for binary classification. I will derive the optimization function needed and then explain the implementation of my logistic regression binary classification algorithm, while testing it on the Iris and Wisconsin Breast Cancer dataset listed in the assignment. I used Stanford's UFLDL tutorial that was handed out in class for my optimization function derivation (Ng et al. 2013), and the Numerical Methods for Engineers textbook for explanation of the Newton-Raphson and Gradient Ascent/Descent methods (Chapra and Canale, 2015).

Optimization Function Derivation

Logistic regression is very similar to linear regression, except that it minimizes the value of the output into the range [0, 1]. We start with the logistic, or sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{(-z)}}$$

The goal is to have the output be large when the classification result is 1, and small when the result is 0. We can analyze how well our logistic function is doing that using the following cost function:

$$J(\theta) = -\sum_i (y^i log(h_\theta(x^i)) + (1 - y^i) log(1 - h_\theta(x^i)))$$

We now need to minimize $J(\theta)$, which we can do by using the same method for minimization in linear regression. We compute the derivative of $J(\theta)$ with respect to θ_j:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1} x_j^i (h_\theta(x^i) - y^i)$$

Written in vector form, the entire gradient is expressed as:

$$\nabla_\theta J(\theta) = \sum_{i=1} x^i (h_\theta(x^i) - y^i)$$

Newton-Raphson and Gradient Ascent/Descent Methods

The Newton-Raphson method is an iterative method where every successive iteration provides a better estimate of the root of the function than the last. If the initial guess of the root is x_i, a tangent can be extended from the point $(x_i, f(x_i))$. The point where the tangent line intersects with the x axis is usually a more accurate estimate of the root. The Newton-Raphson method is based off the Taylor series, With each iteration equal to:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

A multi-dimensional approach looks like this:

$$\vec{x}_{i+1} = \vec{x}_i - \nabla f(\vec{x})[H(f(\vec{x}))]^{-1}$$

For each iteration, the error is roughly equivalent to the square of the previous error. There are numerous options for stopping cases, typically when the error is small enough or when the algorithm goes through a specific amount of iterations, which is a smart choice if you are unsure of how well the Newton-Raphson Method will work with your data.

The Gradient Ascent/Descent Method is also iterative in nature, and is best explained using an example of climbing a hill. A strategy to get to the top would be to start walking in the direction where the slope is maximized at your starting point. However, that direction will most likely not always be the direction of the maximum slope. Thus, we recalculate the direction whenever we get to a point where the slope becomes zero. That process is repeated until we reach the top of the hill. Gradient descent is similar, but instead you find where the slope is minimized, and continue from there.

Like the Newton-Raphson Method, you start at a random spot x_0. For a defined and differentiable function $F(x)$, each slope minimization calculation can be expressed as:

$$x_{n+1} = x_n - \gamma \nabla F(x_n), n \geq 0$$

Where γ is the step size, calculated by finding the length of the slope decrease, or using the Barzilai-Borwein method:

$$\gamma_n = \frac{(x_n - x_{n-1})^T [\nabla F(x_n) - \nabla F(x_{n-1})]}{||\nabla F(x_n) - \nabla F(x_{n-1})||^2}$$

Implementation and Test Results

I implemented a logistic regression algorithm in Python using the Gradient Ascent Method. The algorithm accepts a user defined amount of steps, as well as the learning rate, and computes the sigmoid of the dot product of the feature set with the weights to predict the coefficients. It then calculates the error by subtracting the predicted values from the target, and computes the gradient of the dot product of the calculated difference and the features. That gradient is then multiplied by the learning rate and added to the weights. This process is repeated for however many steps the user has decided to run.

In order to check goodness of fit, the algorithm checks the value of the log likelihood function, the function we are trying to maximize, as it essentially is a measure of observing the data we actually have. Since the function is concave, we can conclude that there is only one maximum, and thus it is the global maximum. Thus, maximizing this function means we are closing in on the global maximum, which is the goal behind the Gradient Ascent Method. The value of that function prints periodically to ensure it is continuing to be maximized.

The algorithm worked on both the Iris and Wisconsin Breast Cancer Datasets. This is the equation I was able to calculate for predicting a flower to be of the Iris-setosa class using the Iris dataset, specifying 500,000 steps and a learning rate of $5 \cdot 10^{-5}$:

$$y = 1.0748SL + 3.4506SW - 5.3220PL - 2.5126PW$$

And here is the equation I calculated for the Wisconsin Breast Cancer Dataset, predicting if the sample is benign or malignant using 100,000 steps and a learning rate of $5 \cdot 10^{-5}$:

$$y = -83.7506x_1 - 11.1633x_2 - ... + 2.7806x_{30}$$

I condensed the equation due to the dataset having 30 different features. On both datasets, the log likelihood function reached a point of maximization, certifying that the algorithm works as intended.

Weka and R Results

I wrote a program using R to find the equation of the least squares linear regression line for the "Combined Cycle Power Plant" dataset. I performed the *lm* function which creates a linear model, specified PE, the energy output, as the dependent variable, and the other variables (AT, V, AP, and AH) as the features. I then used the *summary* command to obtain information about the linear fit, including the coefficients. I achieved the correct coefficients for each feature and was able to construct this equation:

$$PE = -1.98AT - 0.23V + 0.06AP - 0.16RH + 464.61$$

I also calculated a R^2 goodness of fit value from R of 0.9287, which means that the data fit the linear model to a very high accuracy of 93%.

For the Iris dataset, I ran a logistic regression in Weka, which classified the flowers correctly 96% of the time. The feature that has the greatest impact on flower type is petal width, which we know because it had the biggest coefficient of -43.877. The feature with the least impact, and thus the smallest coefficient, is sepal width. I also wrote a program in R using logistic regression to predict if a flower was within the iris setosa class, finding the coefficients for each attribute.

Weka and R implementations of the Wisconsin Breast Cancer dataset were also successful. The logistic classifier in Weka gave me the coefficients of each feature, which allowed me to create this equation to predict if the sample was benign or malignant:

$$benign = -0.5313x_1 - 0.0069x_2 - 0.3301x_3$$

$$-0.2393x_4 - 0.0676x_5 - 0.4068x_6 - 0.4093x_7$$

$$-0.1463x_8 - 0.5488x_9 + 9.6727$$

I also did a logistic regression in R which offered similar results, although I used a larger dataset within the R implementation, so the coefficients were slightly different.

Conclusion

Through this homework assignment, I learned a lot about two main types of regression, how to implement them from scratch, and how to take advantage of libraries and other tools to perform regression. My knowledge of the mathematics behind these algorithms has greatly expanded as well. I also learned the importance and difficulty of optimizing datasets for inputting into programs, as it was much more complicated than I initially thought it would be. By implementing the algorithms manually, I got insight as to how the direct implementations of the methods worked, such as the ones in scikit.

References

Steven Chapra and Raymond Canale. *Numerical Methods for Engineers, 7th Edition*. McGraw-Hill Education, New York, New York, 2015.

Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, and Sameep Tandon. *Logistic Regression Tutorial*. Stanford University, 2013.