Regression For Machine Learning

January 4, 2018

Kyle Hewitt

1 Introduction

Regression Machine Learning is a way to create predictive models from exsisting data to predict outcomes of new, never before seen data. This style of machine learning uses exsisting data to train and access the quality of the predicted model by splitting the exsisting data into training, validation and testing sets. As we will see when exploring farther into the different regression models, it is very easy to overfit a model to the training data so that the model looks like its perfect for the information its given but when new data is introducted, the predictions can be wildly wrong and sparatic. The idea of Regression models is to use the training data to create a model and then use the validation and testing sets to verify that these models are not overfit and how well they estimate on data outside the training set. There are different error metrics used throughout to test this idea of how well the model fits such as Residual Sum of Squares and RSME. In our experimentation we will be using the gradiant decent and coodinate decent algorithms in order to find the optimal model parameters, but it is worth mentioning that most of the problems addressed can be solved in closed form as well. Techniques such as Linear Regression, Regularization through Ridge and Lasso Regression, and Nearest Neighbor/Kernal Regression will be addressed.

`All the code presented here can be referenced at https://github.com/redonelima/regression`

1.1 Notation and software packages used

1.1.1 Software Packages Used

Throughout there will usually be two implementations of each idea, one that uses GraphLab and SFrame and another that will be implemented myself. What to expect from this is to see the behavior with these exsisting tools and then xperimentation in self implementation. Other tools can be used such as sypi and pandas, but this course uses GraphLab because it is good for large datasets because it doesn't require the entire data set to be in memory, instead it only pulls the data into memory as it is needed. This causes the import of GraphLab and the import of the Frame to take a while to load, however, once this is completed operations run faster and large-scale data sets can be used. In the example of the kc_house_data.gl set has 17,258 house entries, each with 21 features.

The Numpy library is used for complex matrix calculations. Numpy is a python library which is written in fortran for extremely efficent matrix calculations. These calculations could be done manually in python, but given large data sets, this would be very slow and inefficent. The MathPlotLib python library is used to visulize data throughout. Graphlab also has functions like .show() to visualize data.

The following import will be used throughout most of the code examples thoughout:

```
In [1]: import graphlab
        import matplotlib.pyplot as plt
        %matplotlib inline
        graphlab.canvas.set_target('ipynb', port=None)
        sales = graphlab.SFrame('kc_house_data.gl/')
        train_data,test_data = sales.random_split(.8,seed=0)
```

```
[INFO] graphlab.cython.cy_server: GraphLab Create v2.1 started. Logging: /tmp/graphlab_server_1515096051

This non-commercial license of GraphLab Create for academic use is assigned to khewitt08@live.com and w
```

2 Regression

2.1 What is Regression?

From a high level perspective, regression is the way to learn about a relationship between ***features*** and ***predictions***. Throughout this case study approtch we will be looking at the use-case of predicting housing prices using regressive learning but the scope of regression goes well beyond this, and many of the techniques used throughout the experiments have impacts on other areas of machine learning besides regression such as bias-varience tradeoff which will be covered in depth later on. In the context of regression, the ***features*** of a model are considered the independant variables and those are derived from the data. For example, for our housing predictive case, the features could be squarefeet, number of bathrooms, lot size, ect. These independant variables are what determine the value of our dependant, ***predictive*** values, which in our example would be the value or predicted selling price of the home. It is the responsibility of the regressive model to learn the relationship between the independant and dependant variables.

3 Linear Regression

3.1 Simple Linear Regression

The idea of simple linear regression is to fit a line to a dataset. In the simplest form, the regression line is simply a line and intercept. Even if the complexity of the model pushes into a higher order polynomial, the main problem is figuring out which line approximates the best prediction to the test data. In the examples, a dataset of house sales in Kings County Seattle and their features will be used. We start by splitting the dataset into a training set and a test set. We do this because we can fit our model to fit the data perfectly but then any variation in future predictions will be very far from the actual value and the housing price won't be well represented. Therefore we must have a portion of our data serperated from the training set to test the quality of our model.
In a simple linear model, the obvoius equation of the model will be,

$$f(x) = w_0 + w_1 x \tag{1}$$

where the equation for the true point can be represented by the equation,

$$y_i = w_0 + w_1 x_i + \epsilon \tag{2}$$

where ϵ represents the noise that causes variation from the model. This noise will be covered more in depth in the section covering performance assessments.
The next point to address is to figure out how to form a metric to assess the quality of the model. The first way we will address this is called the Residual Sum of Squares (RSS), which is the squared sum of the difference between the real data point and the estimated point given by the prediction model.

$$RSS(w_0, w_1) = \sum_{i=1}^{N} (y_i - \hat{y_i})^2 \tag{3}$$

Where $\hat{y_i} = w_0 + w_1 x_i$ is the value predicted from the model

In the housing example, the most obvious feature to use would be the square footage of the house, represented by the column 'sqft living' in the dataset. In the simple slope-intercept form, the w_1 slope term represents the change in price of the house for every unit of squarefeet.

3.2 Gradient Decent

So the next step in finding the lowest error is minimizing the RSS. To do this we have to realize that a plot of every possible fit line will lead to a convex with the optimal point at the minimum. The gradient of a function is the derivitive vector in the increasing direction. So in order to find the minumum would be to subtract the gradient. This is known as the gradient decent algorthim. Given some function $g(w)$, when we take the derivitive with respect to w, then when $\frac{dg(w)}{w}$ is negitive, it means we need to increase w to reach the minimum and when the sign switches it means we need to decrease w.
This algorithm, known as the gradient decent algorithm, starts at a point and takes steps in the opposite direction of the gradiant. This step size, denoted as η controls how far each iteration of the gradient decent steps towards the minimum. In the case where the derivitive of $g(x)$ is large, then the steps could be very large and may jump back and forth across the minimum and take a very long time to converge. On the other end, if η is too small, then it may take a very long time to converge. This means that step size must be choosen carefully.
There are two popular choices in picking a step size.

1. Fixed step size
   $\eta = 0.1$

2. Decreasing step size
   $\eta_t = \frac{\alpha}{t}$
   $\eta_t = \frac{\alpha}{\sqrt{t}}$

In a closed form solution, we would know that we would reach convergence when $\frac{dg(w)}{dw}$ is zero. Since in a real world situation, with our step size η, it could be impossible to reach zero within a reasonable amount of time. So we have to have looser convergence criteria. Therefore, a threshold criteria ε has to be established so that when $\frac{dg(w)}{dw} < \varepsilon$ we can say we're close enough within convergence to say that we can consider it convered.

This gradient decent algorithm should look like:

```
while $\frac{dg(w)}{dw} \geq \varepsilon$:
$w^{(t+1)} = w^t - \eta \frac{dg(w^t)}{dw}$
```

The final part of the gradient decent algorithm for our linear regression model is finding $\frac{dg(w)}{dw}$ where $g(w_i) = (y_i - w_0 + w_1 x_i)^2$ and since the sum of the derivitive is the derivitive of the sum we can take the derivitive of a single value with respect to w_0 and w_1 to create the gradient vector.

$$\bigtriangledown RSS(w_0, w_1) = \begin{vmatrix} -2\sum_{i=1}^{N}[y_i - \hat{y}_i] \\ \\ -2\sum_{i=1}^{N}[y_i - \hat{y}_i]x_i \end{vmatrix} \tag{4}$$

This ends up being a vector because even in our linear equation we have two variables. This will be a very important concept when we look at multiple regression in the next section where we start adding other features like number of bathrooms or bedrooms ontop of square footage. Given this vector, we can used a closed form solution to solve for each of the unknowns in closed form:

$$\hat{w}_0 = \frac{\sum y_i}{N} - \hat{w}_1 \frac{\sum x_i}{N} \tag{5}$$

$$\hat{w}_1 = \frac{\sum y_i x_i - \frac{\sum y_i x_i}{N}}{\sum x_i^2 - \frac{\sum x_i \sum x_i}{N}} \tag{6}$$

Now that we have these closed form solutions for these two variables. There's a lot of similarities between these two equations so we can pull out a few main calculations that we need, namely:

1. $\sum y_i$ 2. $\sum x_i$ 3. $\sum y_i x_i$ 4. $\sum x_i^2$

Using these, we can create a function for calculating simple linear regression.

```
In [2]: def simple_linear_regression(input_feature, output):
            # compute the sum of input_feature and output
            feature_sum = input_feature.sum()
            output_sum = output.sum()
            num_inputs = output.size()

            # compute the product of the output and the input_feature and its sum
            product_sum = (input_feature *output).sum()

            # compute the squared value of the input_feature and its sum
            squared_sum = (input_feature ** 2).sum()
            # use the formula for the slope
            numerator = product_sum - ((float(1)/num_inputs) * (feature_sum*output_sum))
```

```
        denominator = squared_sum - ((float(1)/num_inputs) * (feature_sum*feature_sum))
        slope = numerator/denominator
        # use the formula for the intercept
        intercept = output.mean() - slope * input_feature.mean()
        return (intercept, slope)
```

Now we can test this function by giving it input that we know the outcome.
We are going to make a feature and then put the output exactly on a line. This results in both our slope and interscept being 1. It will also be used to test our model since all the points will fall on a line it means out RSS will be exactly zero.

```
In [3]: test_feature = graphlab.SArray(range(5))
        test_output = graphlab.SArray(1 + 1*test_feature)
        (test_intercept, test_slope) =  simple_linear_regression(
            test_feature, test_output)
        print "Intercept: " + str(test_intercept)
        print "Slope: " + str(test_slope)
```

```
Intercept: 1.0
Slope: 1.0
```

As expected, we see that both slope and intercept of our simple test case are 1.
Next, its time to use our simple linear regression model on some actual data. As mentioned earlier our feature for the focus of this model is going to be the square footage of the house and our predictive $\hat{y}$ is going to be the sales price of the house.

```
In [4]: sqft_intercept, sqft_slope = simple_linear_regression(
            train_data['sqft_living'], train_data['price'])

        print "Intercept: " + str(sqft_intercept)
        print "Slope: " + str(sqft_slope)
```

```
Intercept: -47116.0765749
Slope: 281.958838568
```

From the above code we can see that we have an intercept approximately $-47,116$ and a slope of approximately 282.
What this means for us is that a house with no square feet (an empty lot) would actually cost the seller $\$47,000$ for someone to take. Clearly, this intercept has very little significance for our purposes.
However, the slope value of 281.96 is significant and it means that for every square foot extra a house has, the sales price of the house increases by about \$282. Clearly this is not the most accurate model, since there are many other factors that can play into the value of a home, but that's the reason we're calling this a 'simple' model.

Now that we have a model, we can start making predictions.

```
In [5]: def get_regression_predictions(input_feature, intercept, slope):
            # calculate the predicted values:
            return  slope*input_feature + intercept
```

Now that we have our model and our prediction function we can now use it to predict new houses based on the square footage feature. Lets look at a house that has a size of 2650 sqft.

```
In [6]: my_house_sqft = 2650
        estimated_price = get_regression_predictions(
            my_house_sqft, sqft_intercept, sqft_slope)
        print "The estimated price for a house with %d squarefeet is $%.2f" % (
            my_house_sqft, estimated_price)
```

```
The estimated price for a house with 2650 squarefeet is $700074.85
```

As we can see, using our predictive model a house with 2650 sqft is estimated to be sold for $ 700,000.

Lets take a look at the quality of our model. We will calculate our RSS, which we already know should be zero.

```
In [13]: def get_residual_sum_of_squares(input_feature, output, intercept, slope):
             # First get the predictions
             predicted_price = get_regression_predictions(input_feature, intercept,slope)
             # then compute the residuals (since we are squaring it doesn't matter which order you subt
             residual = output - (slope*input_feature+intercept)
             # square the residuals and add them up
             RSS = residual * residual
             return(RSS.sum())
```

```
In [14]: print get_residual_sum_of_squares(
             test_feature, test_output, test_intercept, test_slope)
         # should be 0.0
```

```
0.0
```

```
In [16]: rss_prices_on_sqft = get_residual_sum_of_squares(
             train_data['sqft_living'], train_data['price'],
             sqft_intercept, sqft_slope)
         print 'The RSS of predicting Prices based on Square Feet is : ' + str(
             rss_prices_on_sqft)
```

```
The RSS of predicting Prices based on Square Feet is : 1.20191835632e+15
```

We can use this to go the other way as well. Given a price, we can tell how big of a house that a buyer can afford.

```
In [17]: def inverse_regression_predictions(output, intercept, slope):
             return  (output-intercept)/slope
```

```
In [18]: my_house_price = 800000
         estimated_squarefeet = inverse_regression_predictions(
             my_house_price, sqft_intercept, sqft_slope)
         print "The estimated squarefeet for a house worth $%.2f is %d" % (
             my_house_price, estimated_squarefeet)
```

```
The estimated squarefeet for a house worth $800000.00 is 3004
```

So we can use this model to say that a buyer with $\$800,000$ could afford a house up to $3,004sqft$.

Now that we have a simple linear model on our housing data, we can change up the features and see how it fits a model when looking at bedrooms and see how that compares to the prediction of square footage.

```
In [19]: bedrooms_intercept, bedrooms_slope = simple_linear_regression(
             train_data['bedrooms'], train_data['price'])

         print "Intercept: " + str(bedrooms_intercept)
         print "Slope: " + str(bedrooms_slope)
```

```
Intercept: 109473.180469
Slope: 127588.952175
```

So based on our bedroom model, we can see that we get an increase in housing price of $ 127,589 per bedroom. So now that we have our two training models, its time to see how well these predict our test data.

```
In [20]: # Compute RSS when using bedrooms on TEST data:
         rss_prices_on_bedrooms_test = get_residual_sum_of_squares(
             test_data['bedrooms'], test_data['price'],
             bedrooms_intercept, bedrooms_slope)
         print 'The RSS of predicting Prices based on Bedrooms is : ' + str(
             rss_prices_on_bedrooms_test)
```

```
The RSS of predicting Prices based on Bedrooms is : 4.93364582868e+14
```

```
In [21]: # Compute RSS when using squarfeet on TEST data:
         rss_prices_on_sqft_test = get_residual_sum_of_squares(
             test_data['sqft_living'], test_data['price'],
             sqft_intercept, sqft_slope)
         print 'The RSS of predicting Prices based on Square Feet is : ' + str(
             rss_prices_on_sqft_test)
```

```
The RSS of predicting Prices based on Square Feet is : 2.75402936247e+14
```

Now that we have our RSS of our squarefoot model and our bedroom model, we can comare our two RSS and see that since the squarefootage model has a lower RSS on the testing data it does a better job of a predictive model. This is because we trained our model on the training data, so all of the houses in the testing set are all houses that the model has never seen. Since the RSS of squarefeet is lower, it means that the houses in the testing with the squarefeet feature set were all closer to the prediction model than those of the bedroom feature set.

3.3 Multiple Regression

TODO: talk about Multiple Regression

4 Assessing Performance