Homework 2 - Linear Regression

Shrey Patel

January 19, 2024

1 Generating x and y

1.1 Generating Values

I started off this assignment by creating x and y. To start off, I used numpy's random number generator to create an array of 100 random numbers. I also used random seed to ensure I am returned the same values when I run the code each time. This just ensures consistency with the later regression models. Next, I made calculations for y using gaussian noise. The gaussian noise was used generating random numbers that follow a normal or gaussian distribution. I set the parameters to have a mean of 0 and standard deviation of x, and used length of x to ensure I am returned 100 values. Then, I added this gaussian noise to x to get y.

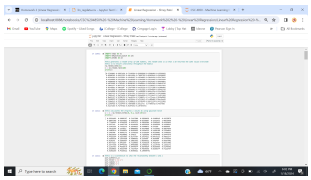

Figure 1: Image of generating values for x and y

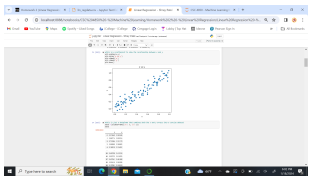

Figure 2: Image of scatterplot for x and y

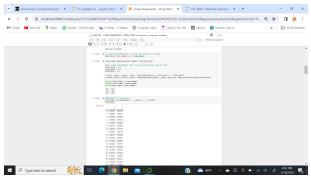

Figure 3: Image of sets split into 60-30-90 percents

1.2 Plots and Set Splits

Next, I created a scatterplot that plots the points of x and y. For further use of these values, I combined the arrays into one pandas dataframe, then saved it as a csv file. The final step before creating the regression models was to split the dataset into training, validation, and test sets. For the purpose of seeing different results, I tested out different splits. I will mostly be talking about 60-10-30 and an

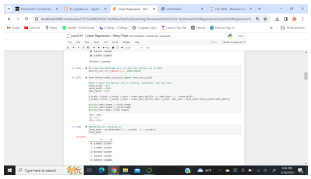

Figure 4: Image of sets split into 80-5-15 percents

unconventional 80-5-15 split. However, before I move on, I will say that the margin of errors decreased or increased depending upon the splits. I noticed that when the size of the training set increased, the error level decreased. When the validation split decreased, the margin of error increased. Different splits resulted in varying metric calculations. This is most likely due to the fact that the slope and intercept would be heavily impacted by the size of the training sets size.

2 The first regression function: Least Squares Method

2.1 Making the Least Squares Regression Function

To start off in making this function, I needed to look what I already had. I already had input and output values, x and y from training-validation-test splits. Since I have multiple values, I have to find the mean of the input and output values. Then, I started out creating an equation for the slope, m. First the numerator would be solved by subtracting the input that we are calculating for by the input mean. Then, that would be multiplied to the difference between the output and output mean. That would be the numerator. For the denominator, you would calculate the difference between the input and input mean again, except this time that value would be squared. Finally, to get the slope, divide the numerator by the denominator. To find the intercept, b, all that is needed is to subtract the output by the product of the slope and input mean. Finally, the function will return the slope(m) and the intercept(b) when it is called. To train this function, I fitted the model to my training set. Next, I made predictions using the validation set to test the accuracy of my model. To make comparisons, I printed out the actual vs predicted values. However, that is not enough to test the accuracy. Thus, I went ahead and made functions for various performance metrics

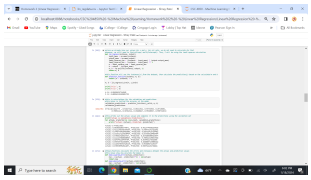

Figure 5: Calling the least squares regression function

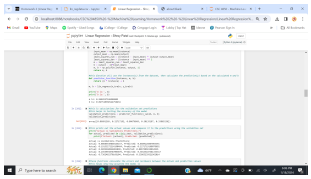

Figure 6: Calling the least squares regression function

2.2 Performance Functions

For the first performance metric, I made a function that calculates the mean squared error between the actual and predicted value. To calculate this, the function would take actual value and subtract it by the predicted value. Then, this value would be squared and divided by the length of the actual

dataset(will be the validation and test sets). This metric helps in ensuring that there isn't a large margin of error between the original data points and the regression line. A lower error score indicates that the model is a good fit to the data. Mean absolute error is calculated the same way as mean squared error, except the difference between the actual and prediction isn't squared. Mean absolute error is just the average of the difference between the actual values and predictions. Thus, this metrics asses the effectiveness of the regression model in a more direct way. Next is the R-squared metric, or coefficient of determination. This measures the goodness of fit, where values closer to 1 indicate better fitting. Finally, the root mean squared error tells us how closely concentrated the original data points are around the regression line.

2.3 Metrics and Test Set

After making the performance metric functions, it was time to use them and compare the validation set values vs the predictions. The values returned for mean squared error, mean absolute error, and root mean squared error were all very close to 0. This indicates that the model is a good fit. For the r-squared value, it was very close to 1 suggesting that the relationship between the variables(x and y) are strong. Next, I made predictions for the test set. The performance metrics returned for these predictions were higher in the error metrics and lower for the r-squared metric. This suggests that the model performed less well on the test set than the validation set. However, the model still had good metric values. This indicates that the model is still a good fit for the test set, even though there was a degradation in performance. Finally, I made a scatter-plot with the regression line to show the relationship between the original points and predictions.

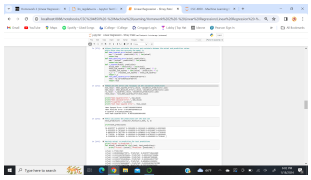

Figure 7: Metric calculations for validation on 60-10-30

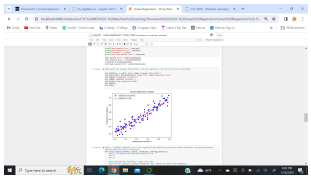

Figure 8: 60-10-30 test set predictions and scatter-plot

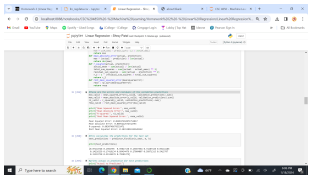

Figure 9: Metric calculations for 80-5-15 valid

3 The Second regression function: Gradient Descent

3.1 Making the Gradient Descent Function

To start off in making this function, I initialized the slope and intercepts to 0. In addition, I made a list for both the slope and intercept to store their gradient values after each iteration. Next, I made a

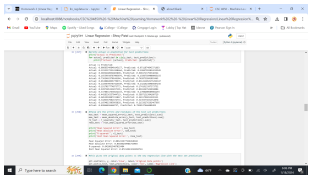

Figure 10: Metric calculations for 80-5-15 test set

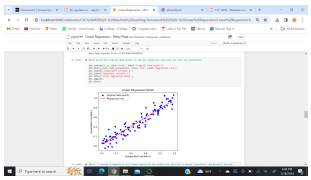

Figure 11: Scatter-plot for 80-5-15

for loop that would go through a chosen number of iterations. Each time the loop is ran through, the prediction is first calculated. Then, the error between the prediction and original value is calculated. Then, the parameters are updated by using the derivatives of the mean squared error. Then, the slope and intercept are updated by subtracting by the product of the learning rate and the updated parameters of the derivatives. Finally, these updated slopes and intercepts are stored into the gradient lists, and also each update prints out the gradient values. The next step is to go through and run through the functions just like I did for the Least Squares Regression Function.

3.2 The Results for Gradient Descent

After making the functions, I fit my gradient descent model to the training data, once again. I tried a different number of iterations each time. However, the outcome(the code output, not the values itself) should be the same. It should print out the gradient values for the slope and intercept for each iteration. Then, the final slope and intercept values will be returned. Just like before, I first used the validation set to test the accuracy of the model. In this case, I found the margin of errors to be a little bit higher for this function than the least squares function. However, the big difference here was the r-squared metric. I found it to be much less than the r-squared metrics for the least squares function. This means that the relationship was not as strong between x and y for this function. The same outcome was seen for the test set, the margin of errors were higher and the r-squared metric was lower. However, I noticed why this was occurring. This is because of the number of iterations it was making. With 100 iterations, the relationship was not as strong as seen in the images in figures 14 and 18 of the scatter-plots below. The regression lines did not follow have a strong relationship with the other points. However, as I increased the number of iterations, the regression line showed a much stronger relationship with the data points. This is due to the fact that function or algorithm is minimizing the amount of loss happening during each iteration. Thus, more iterations means less loss. This can be seen with the image in figure 19 below, where it shows the a much more positive and strong relationship between the regression line and data points.

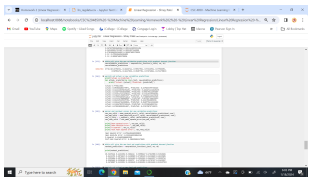

Figure 12: Gradient descent for 60-10-30 validation set

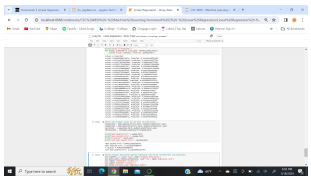

Figure 13: Gradient descent for 60-10-30 test set

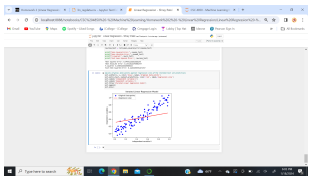

Figure 14: Scatter-plot for 60-10-30 in gradient descent

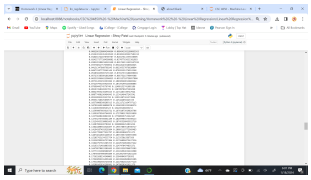

Figure 15: Gradient descent iteration results for 80-5-15

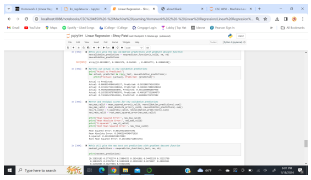

Figure 16: Gradient descent for 80-5-15 validation set

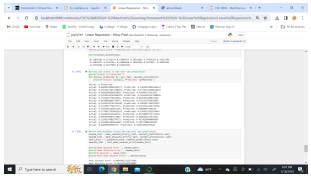

Figure 17: Gradient descent for 80-5-15 test set

Figure 18: Scatter-plot for 80-5-15 in gradient descent

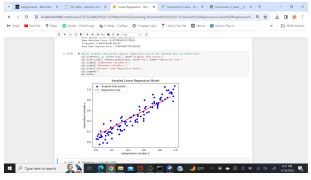

Figure 19: Scatter-plot for 1000 iterations

4 Gradient Values

For the gradient value, I tested out multiple iterations to see the results and compare. For very high iterations, I found that the gradient values would steadily decrease towards 0(the slope of the lines would become flat). This suggests that the model is very close to minimizing the loss function and is near a minimum. For a low number of iterations, the lines would not minimize loss as well as it is not approaching 0 as fast or at all. This makes sense as less iterations means less chances for the gradient values to be updated. In addition, when running the gradient descent function with less iterations, there was higher margins of error and the coefficient of determination was lower. This means the model is not well fit to the data. However, when running more iterations, the margin of error was much lower and coefficient of determination was much higher. However, it has to be noted where that less iterations led to underfitting, an excessive number of iterations could lead to overfitting. Another observation is that the slope line would approach the minimum faster than the intercept line.

Figure 20: Gradient descent convergence for 100 iterations

Figure 21: Gradient descent convergence

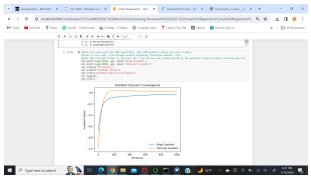

Figure 22: Gradient descent convergence for 1000 iterations

5 Weka results

To compare the accuracy of my gradient descent model, I will compare it to the results of the linear regression algorithm on Weka. From Weka, I got a model of $y = 0.954x + 0.0215$. Due to the nature of the iterations and how it impacted the intercept and slope, I can see that as the number of iterations increased, I got closer to the to slope and intercept values of the Weka. With this in mind, higher iterations also showed regression lines similar to those of Weka's.

References

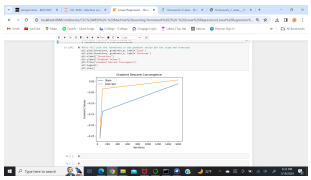

Figure 23: Gradient descent convergence

Figure 24: Linear Regression Model

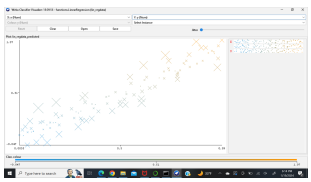

Figure 25: Weka x vs y original dataset

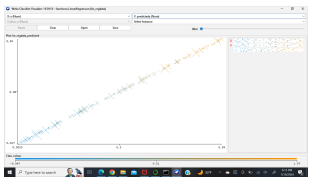

Figure 26: Linear Regression Line