Homework 3 - Logistic Regression

Shrey Patel

January 25, 2024

1 One-Hot Encoded Version

1.1 Converting Data to One-Hot Encoded Representation

To start off this assignment, I first loaded the dataset into Jupyter Notebook. Next, I plotted the points on a graph to see the relationship between the 'Code' and 'Value' column. Next, I converted the 'Code' column to one-hot encoded representations. Finally, I removed the Code in the column names so that it just contains the numbers of the code.

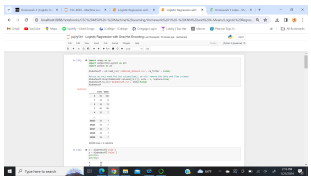

Figure 1: Original Dataset

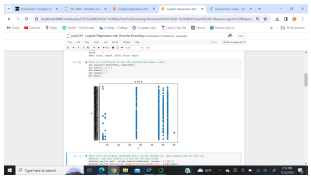

Figure 2: Plot of Original Dataset

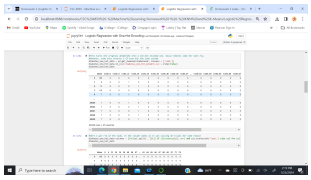

Figure 3: One-Hot Encoded Data

1.2 Training-Validation-Test Sets

After one-hot encoding the data, I split that into training, validation, and test sets. I used a 60-10-30 split, and used the 'Value' column as y. The other 24 one-hot encoded columns are x, or the inputs.

1.3 The Logistic Regression Function

Now, we will build the logisitic regression function, or algorithm. Starting off, I imported the tqdm library just to utilize as a progress bar to monitor the speed of the code running. First, I made a

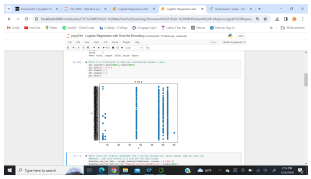

Figure 4: One-Hot Encoded Data Without Code in Column Name

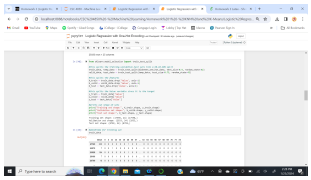

Figure 5: Code for splitting the dataset and size

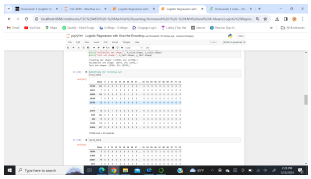

Figure 6: Training Set

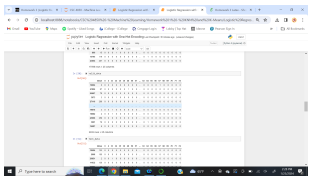

Figure 7: Validation Set

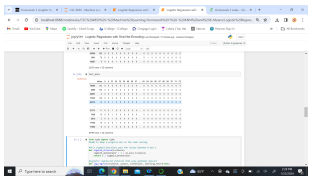

Figure 8: Test Set

sigmoid function. The point of the sigmoid formula is to map the values between a range of 0 and 1. This is useful for the output predictions, as those will be squashed between 0 and 1. The formula for this function is 1 divided by the sum of 1 and e to the power of the negative input. Next, I defined the logistic regression function itself. To start off this function, I first set the weights, m and b, equal to 0. I also made two lists to hold the gradients of those weights. Then, I made a for loop that iterates through how many ever times the function is called for. Through each iteration, predictions are made based off the input values using the sigmoid function. Then, the error is calculated by finding the difference between the predictions and actual values. Then, the gradients are calculates using those errors. This continues until the last iteration where the last weights are set for m and b. m will contain 24 values, 1 for each input column. b will only have 1. Then, I made a predictor function that will calculate new predictions based off the weights that are computed and the inputs. For some reason, my dataset still contained non-numeric values, so I first got rid of those. Then, I called the logistic regression function to fit it to the training set. I ran it with 50 iterations. I then also graphed the gradient values against the iterations, and got a graph that does not seem to make sense. They were all straight lines, but that is probably due to the fact that the iterations and learning rate were low. So, there is a slow growth.

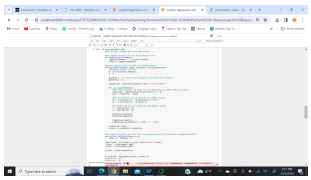

Figure 9: Code for logistic regression

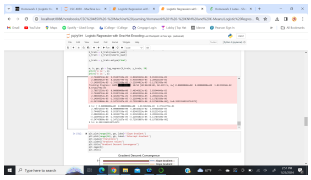

Figure 10: Outputs for m and b

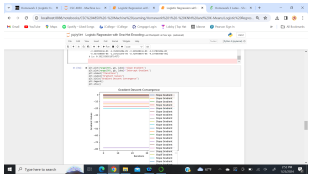

Figure 11: Gradient vs iteration plot for 50 iterations

1.4 Making Predictions

First off, I started making predictions using the validation set. After fitting the model to the training set, I had to ensure the model works using the validation set. Thus, I made predictions, and then removed all the values that didn't have one hot encoded 1s in that row. Then, I used sigmoid calculations to sqaush those predictions between 0 and 1. I also made sigmoid calculates for the test dataset which I am comparing the predictions too. After that, I printed out the actual values next to the predicted values for comparision. Then, I also printed out the sigmoid probabilities for the actual values next to the predicted values. Next, I did the same thing for the test set. Except, I also applied a threshold for the sigmoid probabilities of 0.5. If greater than 0.5, it is a positive class(1), if lower then it is a negative class(0). I noticed that all of my predictions returned a positive class unlike the actual values. This could be a sign that model has a bias towards the positive class. Another factor I noticed

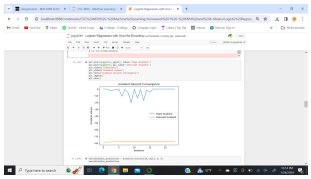

Figure 12: Gradient vs iteration plot for 24 iterations

was that the lower learning rates gave lower sigmoid probabilities. As I increased the learning rate, the probabilities became closer to 1. Lastly, as the number of iterations grew, the probability increased as well. This makes sense as more iterations mean more growth, and higher learning rates allows for more drastic updates to the gradients. Finally, I made a graph to display the logistic regression. However, it did not come out looking right. This is most likely due to how small the probabilities are.

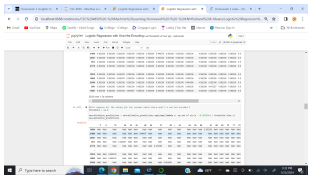

Figure 13: Validation Predictions Data frame

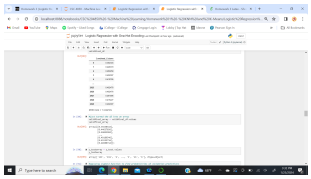

Figure 14: Validation Predictions Data frame 2

1.5 Metrics

Finally I made calculations for accuracy, precision, recall, and f1 score. For these I got 0 across the board. For accuracy, this is due to the fact that none of my predictions were equal to the actual values, no matter the learning rate and number of iterations. With that in mind, that means I have no true positive values, which is why I got 0 for precision and recall as well. For f1 score, I could not output a value due to the fact that precision and recall are both 0. Since it is not possible to divide by 0, I could not get an f1 score.

2 Ngrams Version

2.1 Converting Data to Ngram Representation

For this part of the assignment, I did the same as the one-hot encoded. Except, instead of converting the data into one-hot encoded versions, I used ngram representation. Following that, I kept everything the same from the one-hot encoded version above. The code for the training-validation-test splits, and the functions will remain the same.

2.2 Running Logistic Regression on Ngram Dataset

For the figures below, I will be presenting the code where I had an ngram range of (2,4), a learning rate of 0.00001, and 50 iterations. After fitting the model to the training set, I made predictions using the validation set using the weights and bias from m and b. Then, I used the sigmoid formula to squash

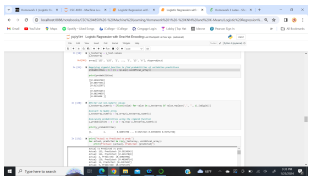

Figure 15: Sigmoid Probability Calculations

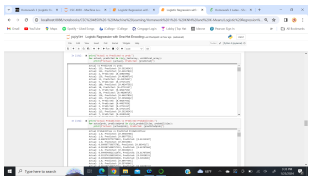

Figure 16: Comparison of Validation Values

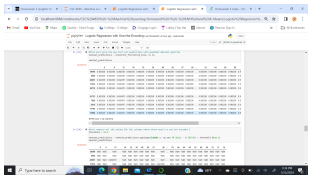

Figure 17: Test Set Predictions

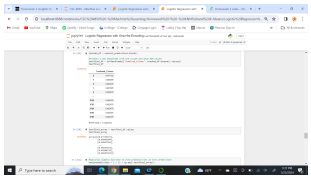

Figure 18: Test Set Predictions Dataframe

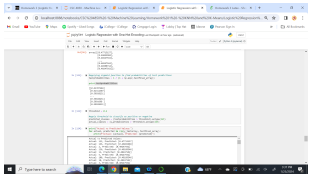

Figure 19: Probabilities and Threshold for Test Set

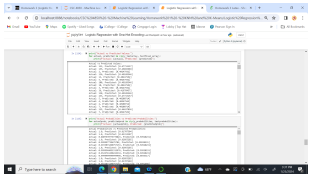

Figure 20: Comparison of Test Values

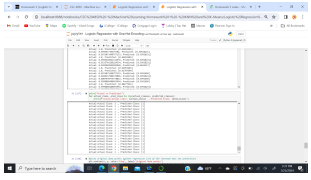

Figure 21: Comparison of Probability Values for Test Set

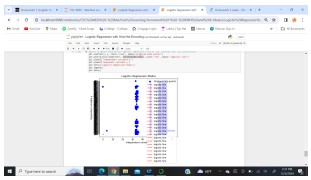

Figure 22: Graph of Logistic Regression

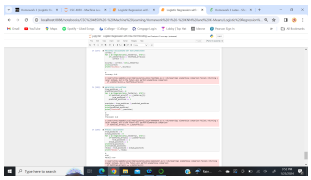

Figure 23: Metric Calculations

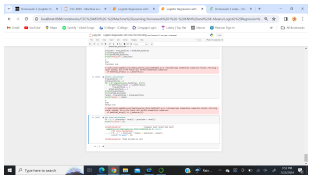

Figure 24: Metric Calculations

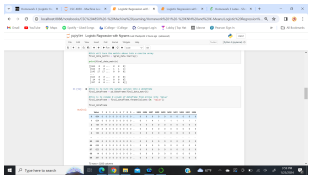

Figure 25: Ngram Dataframe

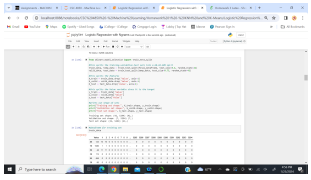

Figure 26: Splitting Ngram Dataset

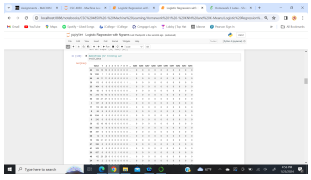

Figure 27: Training set

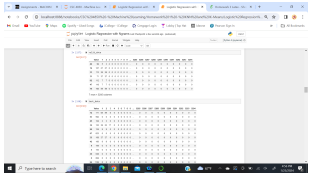

Figure 28: Validation and Test sets

those predictions and also the actual values between 0 and 1. Surprisingly, I saw that all of the actual values returned a value of 1. For my predicted values, a majority were 1 or around it. There were other values that were lower but still above a threshold of 0.5. Next, I found predictions for the test set. I found that the sigmoid probabilities here were also 1 or near 1. The other values were also above 0.5. Thus, when I checked for classes with a threshold above 0.5, it was no suprise that all the actual values and test predictions values were a part of the positive class. Now, for differences between ngram and one-hot encoded. For ngrams, I saw that learning rate and number of iterations had the same impact. As those increased, so did the predictions values. In addition, the probabilities approached 1. However, the difference here is that as the ngram range increased, the predictions also became larger. However, the sigmoid probabilities varied heavily. Although, all were above the threshold of 0.5, there clearly was a drop off in probability rate for about half of the predictions.

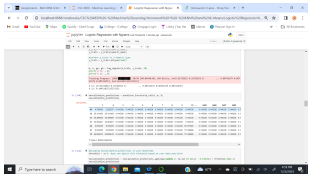

Figure 29: Validation Predictions

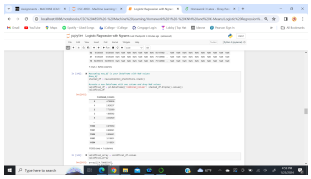

Figure 30: Validation Predictions

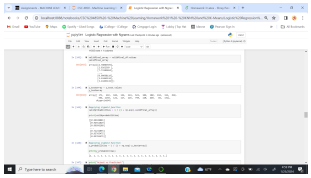

Figure 31: Sigmoid Calculations for Validation

2.3 Metrics

Due to the nature of the logistic regression model that was implemented, I was given no difference here with the metric results. I got the same results as the one-hot encoded data. Once again, this is due to the fact that none of the predictions were equal to the actual values. Thus, there were no true positives.

3 Weka Results

Finally, for the Weka results. After implementing the dataset for 100 instances with logistic regression, weka returned an accuracy of 37.5 percent. That is quite low, but more accurate than the model that I made which returned a 0 percent. In addition, the precision and recall values were quite small, which is another indicator for a poor performance. Classes for codes 33,34, and 62 returned many false positives. Finally, there were high error rates across the board. I believe this is the case due to the nature of the dataset itself. Going back to the plot of the original dataset, it is obvious there isn't a strong linear relationship. This makes sense as logistic regression is built upon by linear relationships and models with the inputs and parameters.

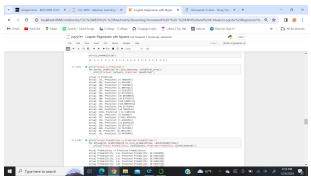

Figure 32: Actual vs Predicted Values

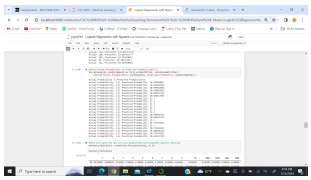

Figure 33: Actual vs Predicted Probabilities

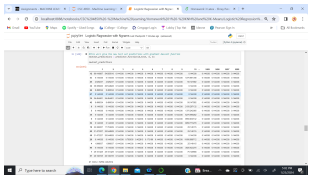

Figure 34: Test Set Predictions

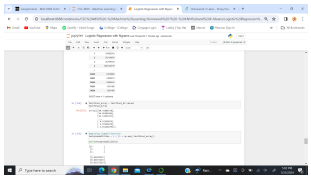

Figure 35: Sigmoid Calculations for Test set

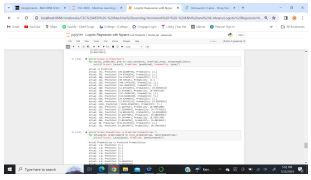

Figure 36: Actual vs Predicted Values for Test set

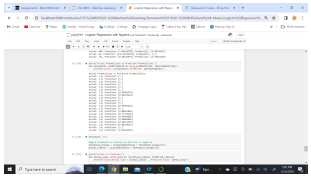

Figure 37: Actual vs Predicted Probabilities for Test set

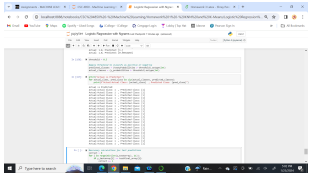

Figure 38: Actual vs Predicted Probabilities Classes

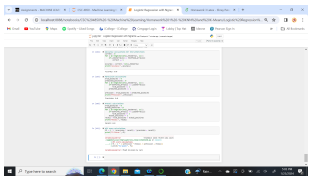

Figure 39: Metric Calculations for Ngrams

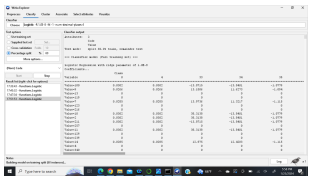

Figure 40: Weka Classes

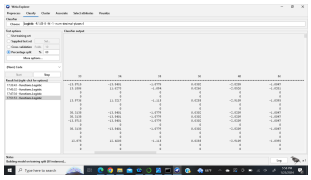

Figure 41: Weka Classes 2

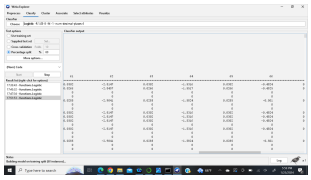

Figure 42: Weka Classes 3

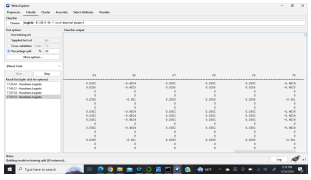

Figure 43: Weka Classes 4

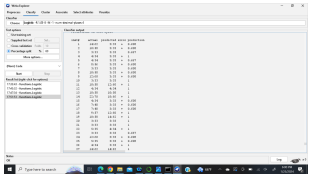

Figure 44: Error and Predictions

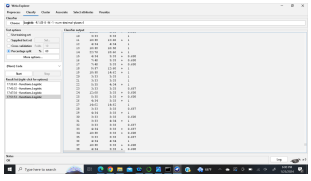

Figure 45: Error and Predictions 2

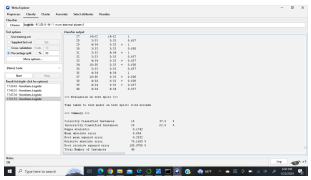

Figure 46: Metric Summary

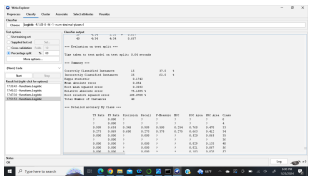

Figure 47: Metric summary and Accuracy

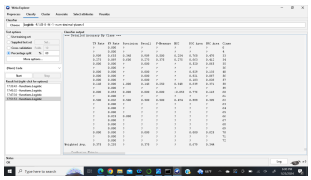

Figure 48: Accuracy

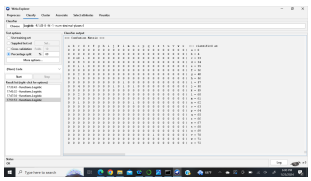

Figure 49: Confusion Matrix Weka

References