COMP 135: Project 01

April 4, 2021

Introduction:

This project involves exploring two different datasets and finalize a model which gives best results. The best results will be determined by the model that can give me least error rate on the test set and increase accuracy (AUROC). In Part 1 of the project, I will evaluate two digits [8, 9] pixel values '0' as 8-digits and '1' as 9-digits, to come with an output that gives me the best result via Logistic Regression modelling. In Part 2 of the project, I will analyze a dataset consisting of images of sneakers and sandals on which to test various models.

Part One: Logistic Regression for Digit Classification

Here we have given Input & Output Training datasets and Input & Output Testing datasets in regards to 8-digits vs 9-digits pixel images. The training dataset encompasses 784 feature values on 11,800 data points on a 28x28 grid. Similarly, the test dataset also has 784 feature values with 1983 data points. I have performed few Logistic Regression model to determine the resulting classification with the two digits.

1.1 Comparison of 'Accuracy Score' & 'Log Loss' with different iterations on Training data

In this section, I ran Logistic Regression model on training data with default values. The goal of this exercise is to determine if limiting the number of iterations of this model has an effect on the accuracy of the model. We plotted accuracy score and logistics loss against the max_iter field in 'Figure 1' and 'Figure 2'.

Figure 1: Accuracy Score vs. # of Iterations

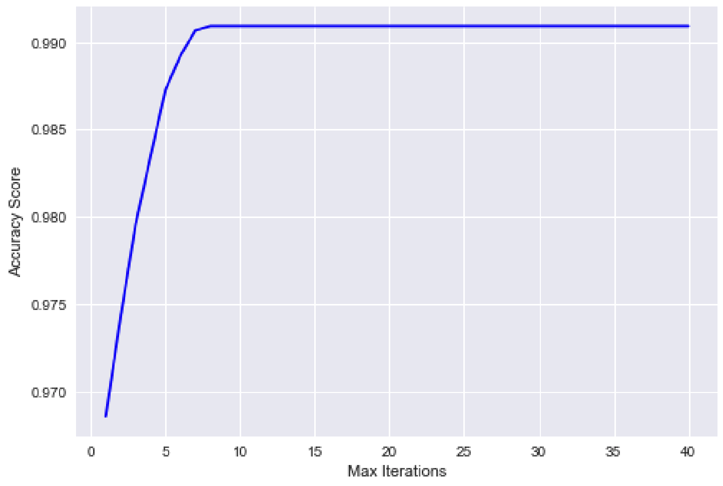
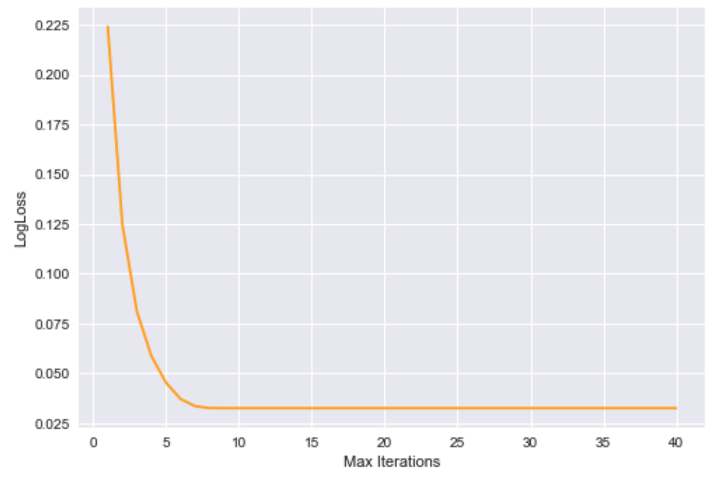


Figure 2: Log Loss vs. # of Iterations

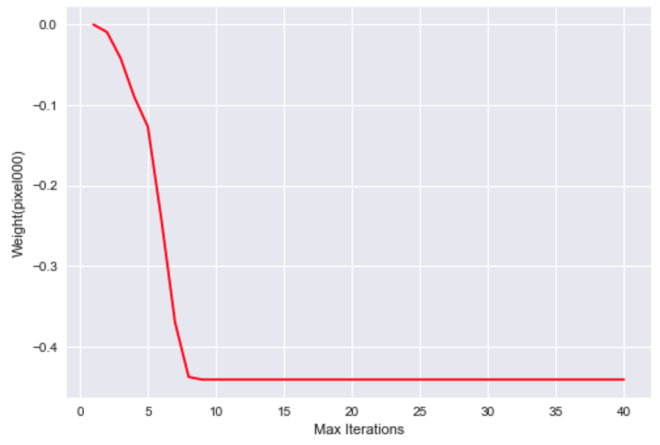


Discussion: A quick glance over the two plots will tell us that in 'Figure 1', the curve exponentially increases as the number of iterations increases, whereas in 'Figure 2' the curve decreases exponentially as the number of iterations increases. The above two plots also have a flat curve at around 7 or 8 iterations point, indicating the point at which both the accuracy score and log loss become stable. This stable point is useful in determining the optimal iterations which can be used to achieve an acceptable model accuracy.

In 'Figure 1', the accuracy starts with 0.970 (97%) at 1 iteration and exponentially increases at 7 or 8 iterations point to 0.990 (99%) accuracy. After iteration 8, the curve flattens indicating the accuracy does not change despite an increase in number of iterations. Thus, there is no need to run the model on the training data after this point, but we should also try to minimize the log loss of this model. In 'Figure 2', the log loss starts with 0.225 (22.5%) at 1 iteration and dramatically decreases at 7 or 8 iterations point to the value of 0.025 (2.25%). Stopping early at 7 or 8 iterations point, compared to 40 iterations, could help us avoid overfitting the training data while still achieving a minimum in log loss and a maximum in accuracy score.

1.2 Plot of Weight on pixel000 via various Iterations on the Training Data

Figure 3: Weight (pixel000) vs. # of Iterations



Discussion: In this section, I used coef_attributes function on all available weights of 784 feature values. The weight for first feature of our training data starts at [0] against the iterations i=[1,2,…,40]. In 'Figure 3', the weight changes as I increase the number of iterations. Similar to 'Figure 1' and 'Figure 2' of Accuracy Score and Log Loss respectively, I achieve a stable curve at 7 or 8 iterations with a final weight of approximately -0.45. This helped me conclude that the convergence described above in previous section with a highest accuracy score and minimum loss at a 7 or 8 iterations point.

1.3 Find the Penalty Strength 'C' and Least-Loss Model

Penalty Strength C value: 0.0316
Minimum log-loss at best C-value: 0.0897
Accuracy score at best C-value: 0.9808

```
Predicted    0    1
True
0          942   32
1           33  976
```

Discussion: In this section, I used inverse penalty C defined by np.logspace(-9, 6, 31). Previously used Logistic Regression model was used on the test data this time around to conclude least loss value of 0.0897 (8.97%) with an accuracy score value of 0.9808 (98.08%). From the confusion matrix above, it tells us that the best model is where True Positive is 976, True Negative is 942, False Positive is 32, and False Negative is 33.

1.4 Mistakes of the best model on the Test Data - Plots of False Positives & False Negatives

Figure 4: False Positives consisting of digits-9 by the model

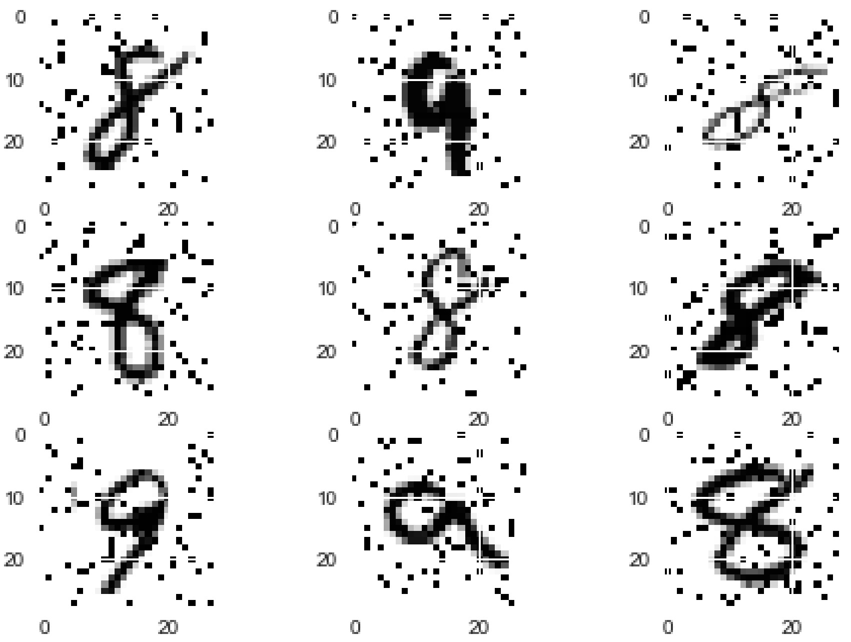
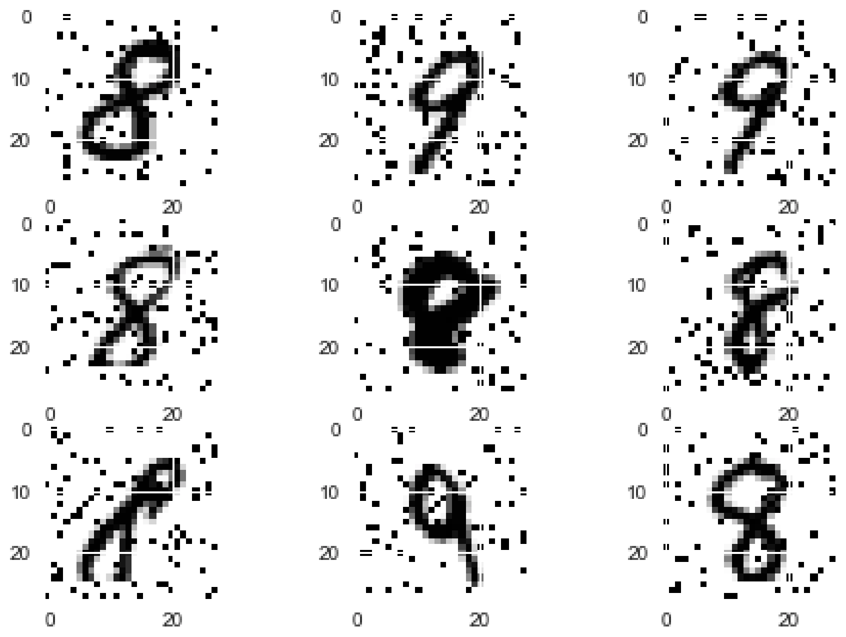


Figure 5: False Negatives consisting of digits-8 by the model



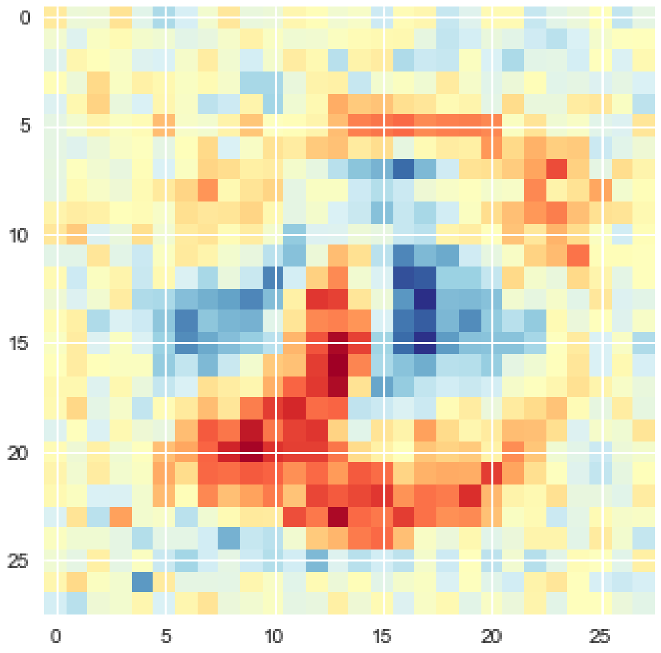
Discussion: The above two plots represent our mistakes in terms of false positives and false negatives, by utilizing our best model with values 'C', the least loss function, and the accuracy score. At a glance, view of above plots, there are many instances where the model mistakenly classifies 9-digits and 8-digits. In 'Figure 4', the model is successful only six out of nine times, where the ideal case was to achieve an 8-digit image. In 'Figure 5', the model results in only three 9-digit images instead of all nine being the ideal case of a 9-digit image.

The model utilized default parameter values, where the threshold to check for 8-digit or 9-digit was 0.5 in sklearn. By looking at the mistakes above, there is a need to use better threshold to distinguish between the two digits. As a best practice, there should be at least three different threshold attempts to check for accuracy score in the confusion metrics. There is also a possibility that the training data input features have discrepancy when applied to the test data. Another possibility of confusion between the two digits is that where the lower part of 9-digit character can be easily transformed to 8-digit. Therefore, on the wrong images in above plots, one can easily see the connection between the two digits on the lower part of their shape whereas the top part is clearly distinguishable.

1.5 Examining Final Weights by the model

Figure 6: Plot of all final Weight Coefficients

```
<matplotlib.image.AxesImage at 0x7fa9bf183880>
```



Discussion: The above image is using all final weights produced by the classifier model. These final weights are a results of our best Logistic Regression classifier model that had array of 784 with 1-dimensional view. Then, there was a transformation of 28x28 plot with 8-digit having negative weights in red and 9-digit having positive weights in blue. I used vmin= -0.05 for negative weights that are in red and vmax=+0.05 for positive weights that are in blue; everything in between closer to 0 weight is yellow color and results is in 'Figure 6'. The higher weight points are in red and blue pixels that should be analyzed further. The pixels in red are related to 8-digit due to the circle or round shape in the middle of the numerical character and blue pixels at the top in 'Figure 6' represent 9-digit due to the round nature of the numerical character. Looking at the bottom left corner in red, one can tell that this area of pixels has a negative effect when trying to calculate 9-digit since this digit won't have the corner bottom compare to 8-digit character. Therefore, this model using sklearn does a good job identifying 8- vs. 9- digits.

Part Two: Sneakers versus Sandals

In this section of the project, I was free to utilize any model within Logistic Regression space using sklearn. The goal of this exercise is to best classify sandals and sneakers with the help of Gradescope Leaderboard, while minimizing error rate and highest AUROC on the test dataset. Here, I will build different types of models in accordance with various parameters or feature transformation on 12,000 training data points with 784 input features. As a result, I have created 4 different test scenarios with their performance listing.

2.1 Load the Dataset

First, I will read the two training and one testing data from csv files. Afterwards, I can transform the training data and visualize it on a plot to identify the two different shapes. Using imshow function in matplotlib, I transformed the data from 784 pixels onto a 28x28 matrix. I have attached four images, two associated with sneakers [0] in gray (gist_yarg) and color (seismic) formats, and two associated with sandal [1] also in gray (gist_yarg) and color (seismic) formats. The training data was (12000, 784) and the test prediction data was (2000, 784).

Figure 1a: Plot of sneaker [0] with gray scale

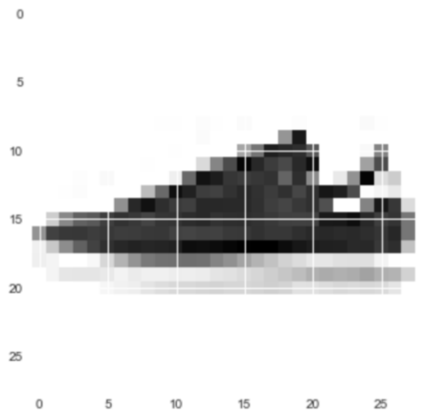


Figure 1b: Plot of sneaker [0] with color scale

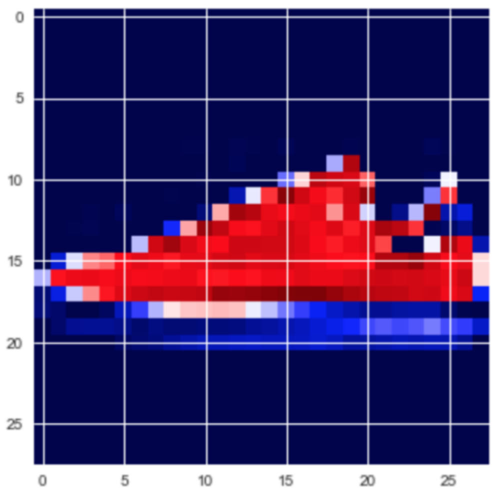


Figure 2a: Plot of sandal [0] with gray scale

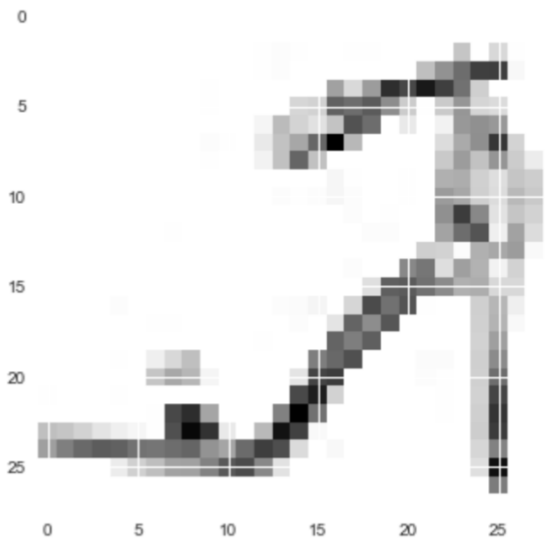
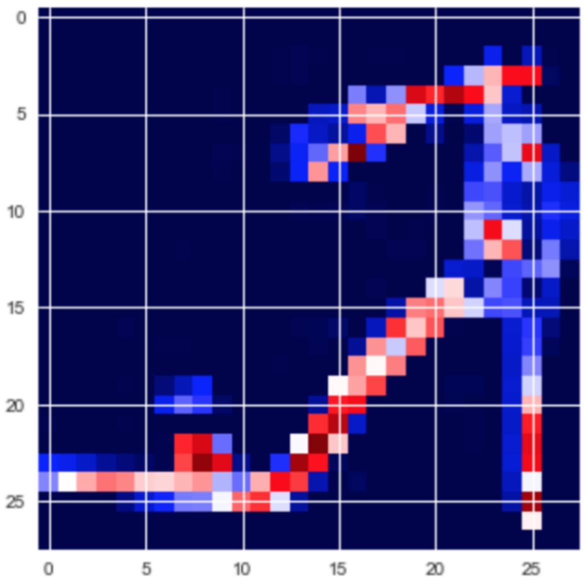


Figure 2b: Plot of sandal [0] with color scale



2.2 Logistic Regression - the Baseline Model without any parameters

First, I used sklearn's Logistic Regression model to apply our training data using only default parameters. After applying the model to test dataset and uploading to Leaderboard, I get an error of 4.25% with accuracy score of 95.75%. These metrics are really good for first time around, but I wanted to try to make these metrics better via adding parameters or applying features transformation.

Error Rate= 4.249%
AUROC= 95.749%

Plan of action: I have four types of model I would like to explore on my training dataset and get better results on my test dataset to achieve least error rate and highest AUROC.

1) Data Augmentation Horizontally
2) Feature Splitting: 1/3 Test & 2/3 Train
3) K-Fold Cross Validation
4) Best Penalty Strength Value C

2.2.1 MODEL ONE: Exploring Data Augmentation Horizontally

On this test of data transformation for training dataset, I performed a horizontal flip to double the training data size without needing any new data. Thus, my original dataset of [12000, 784] resulted in [24000, 784] by defining add_horizontal_flip_input and using np.flipud. Now that I have twice the size of the data, I can run the same Logistic Regression model as 2.2 without any need for more data add-ons and features transformation. I uploaded my result of yproba1_test.txt to the Leaderboard and it performed worse than section 2.2 when it comes to error rate but better in AUROC. There is a possibility that twice as much as data of same original inputs didn't help in this case. Also, there is possibility of overfitting because of increased data on a similar values from our original baseline data.

Error Rate= 5.3%
AUROC= 98.6632%

2.2.2 MODEL TWO: Exploring Feature Splitting: 1/3 Test & 2/3 Training

On this test, I divided up the initial training dataset into two further breakdowns, 1/3 of test and 2/3 of training sets. The 2/3 of the set was used to run Logistic Regression as a training set and the rest 1/3 was used to run as a test set. I wanted to explore this avenue to see if the original training set had any biases for overfitting or if there was a way to reduce overfitting. I utilized selection.train_test_split function in the sklearn. This had a good performance compared to past sections (2.2 & 2.2.1) with reduced error rate and increased AUROC.

Error Rate= 4%
AUROC= 99.2857%

2.2.3 MODEL THREE: Exploring K-Fold Cross Validation (K-10)

For this test, I utilized k-fold cross validation to see what happens when the model is run on the training data. My model had model_selection.KFold with n_splits=10. However, the results were worse when it comes to error rate and slight better in AUROC, than the last section of Feature Splitting (2.2.2).

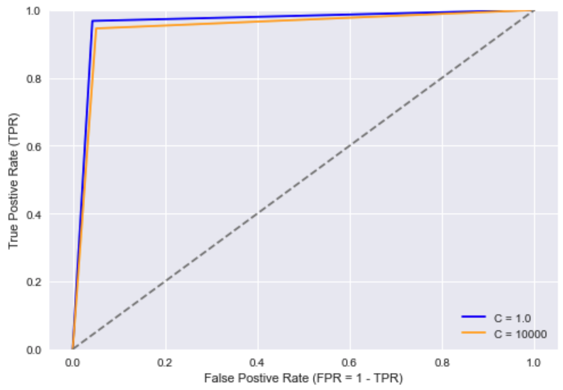
Error Rate= 4.2499%
AUROC= 99.3119%

2.2.4 MODEL FOUR: Exploring Penalty Strength Value C

On this test, I utilized best penalty strength value C to explore if I can reduce our error rate further and/or increase AUROC. I applied C value of 10^5 with grid ranging from 10^{-9} to 10^6. Per the Leaderboard submission, the error rate was increased to 5.55% with AUROC of 94.45%, which are worse than our baseline model in section 2.2.

Error Rate= 5.5499%
AUROC= 94.45%

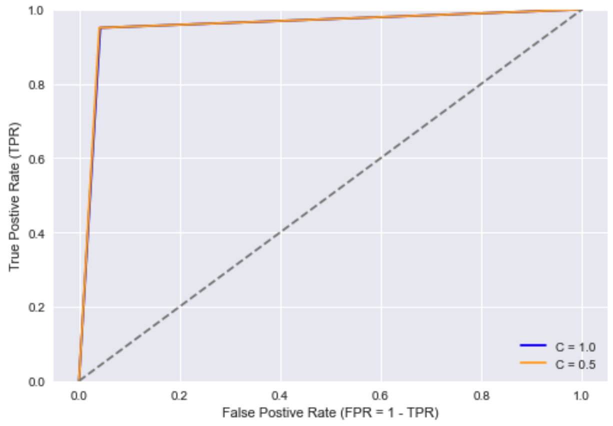
Figure 3: ROC at C=1 vs C=10^5



I plotted both curves, C with original penalty of 1 and C best penalty of 10^5. In 'Figure 3', original penalty has a higher area under the curve, meaning performing better than my best penalty value of C. This could be due to overfitting issue so to solve this issue, I needed to find better C value for my model. After trial and error, I found the correct C value to be 0.5. With the "new" best penalty C value and the original penalty, I saw some improvement in the error rate so far out of all the above sections explained above but not a best AUROC, plotted in the 'Figure 4' for visibility.

Error Rate= 3.80%
AUROC= 96.2%

Figure 4: ROC at C=1 vs C=0.5



2.3 Background:

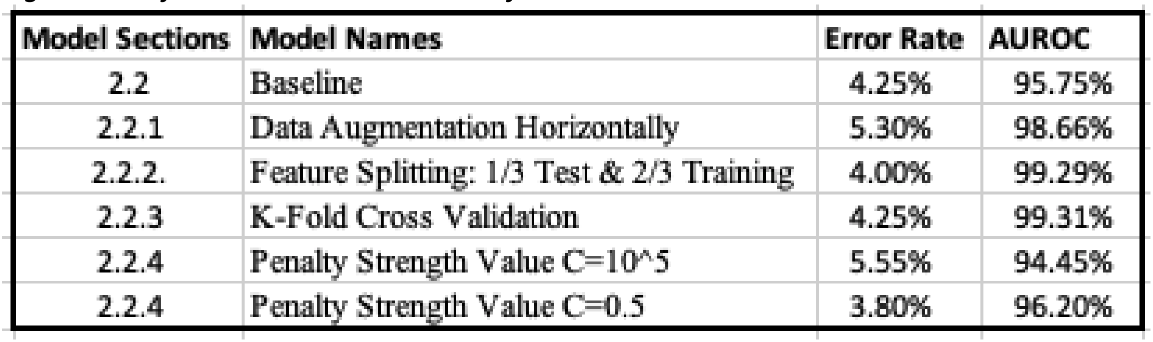
For this project, I had many ideas on what function or model to run on my datasets. Due to time commitment and lack of real practice in running such functions, I had to improvise and run models I knew 100% would work and give me some results. For example, I wanted to utilize SVM, but my research online suggested that images with 784 features may not give me the best results. I also wanted to use hyperparameter as discussed in the lecture on the Logistic Regression but again with little to no experience, I was hesitant to come up with a model of my own that could give me better results.

2.4 Conclusion:

For this project, I created four model to see which one can help me with the best model result, least error rate and highest AUROC. I started with a baseline model with no parameters or changes to any features on the training data to see what my error rate and AUROC were on the test dataset, so I can work to perfect such metrics. Next, I added extra datasets via data augmentation horizontally to help me get better results but it only increased AUROC and got me worse error rate. Then, I performed Feature Split theory to divide up my data into 2/3 training and 1/3 test sets. So far, this gave me the best results. I also tried K-Fold Cross Validation that gave me slightly better AUROC than Feature Split but worse error rate. Now, I was stuck with two best results; Feature Split and K-Fold Cross Validation. To break the tie, I performed another test by exploring best penalty strength value C. Here I had two results, first one had the worst performance out of all the models so far since I used a C=1 and my best penalty strength of C=10^5. After playing with some C values, I settled on 0.5 that gave me best ROC curve area and best performance according to the Leaderboard.

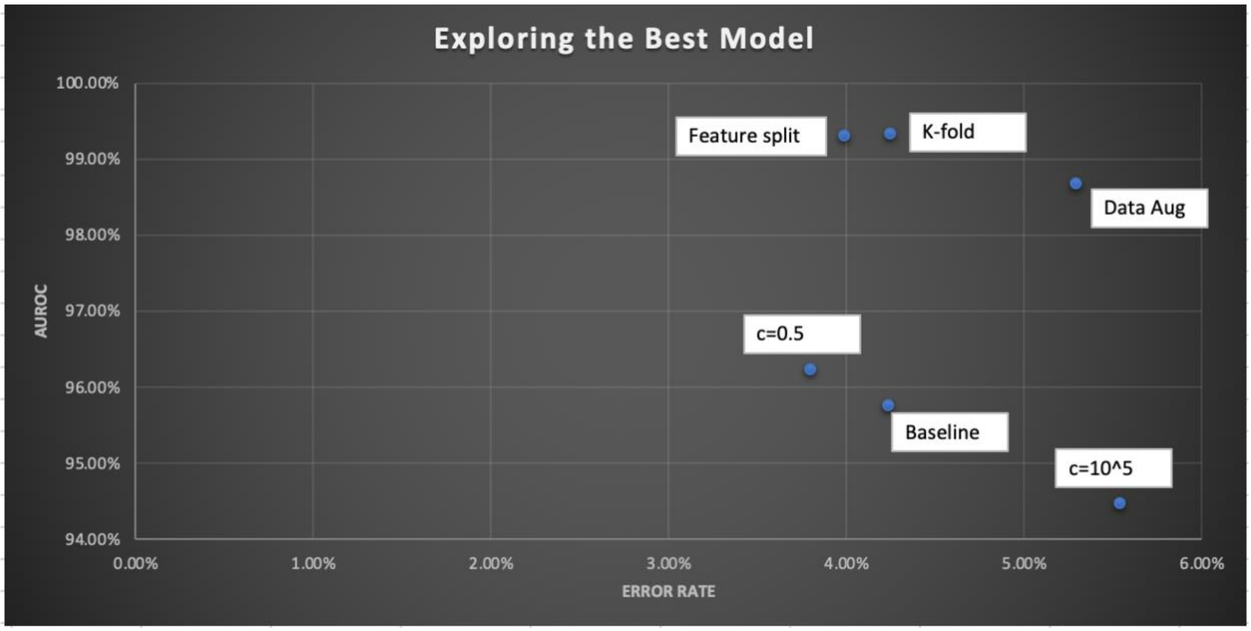
I created a table below and scatterplot graph for better visibility to choose the best model in terms of least error rate and highest AUROC. My best two model would be Penalty Strength Value C=0.5 and Feature Splitting.

Figure 5: All four model results in table format



Model Sections	Model Names	Error Rate	AUROC
2.2	Baseline	4.25%	95.75%
2.2.1	Data Augmentation Horizontally	5.30%	98.66%
2.2.2.	Feature Splitting: 1/3 Test & 2/3 Training	4.00%	99.29%
2.2.3	K-Fold Cross Validation	4.25%	99.31%
2.2.4	Penalty Strength Value C=10^5	5.55%	94.45%
2.2.4	Penalty Strength Value C=0.5	3.80%	96.20%

Figure 6: All four model results in scatterplot format


Exploring the Best Model
AUROC
Feature split
K-fold
Data Aug
c=0.5
Baseline
c=10^5
ERROR RATE