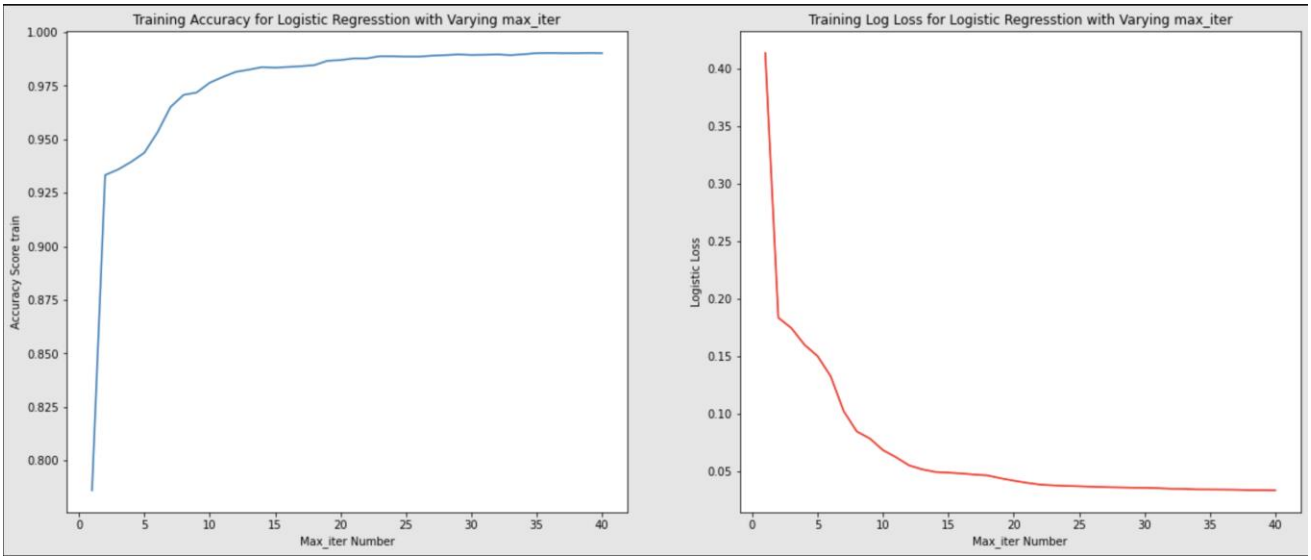
Project 1

Part One: Logistic Regression for Digit Classification

1. The image below shows the accuracy (left) and the logistic loss (right) of the *data_digits_8_vs_9_noisy* dataset which is comprised of 28x28 pixel handwritten images of 8's or 9's which are represented as 0 (if 8) and 1 (if 9). A logistic regression model with all default parameters except *max_iter* was built to predict outcomes using this dataset. The model was tested with a range of max_iter parameters from 1 to 40 in order to explore what happens when we limit the iterations allowed for the solver to converge on its solution. The *data_digits_8_vs_9_noisy* dataset was split into training, and test data. The analysis in Figure 1 was conducted on the training data.



Max_iter Number	Accuracy Score train	Accuracy Score test	Logistic Loss
36	0.990339	0.955119	0.033711
40	0.990254	0.956631	0.033080

Figure 1: Plots of accuracy (left) and logistic loss (right) of a logistic regression model fitted to the data_digits_8_vs_9_noisy dataset with various max_iter values ranging from 1 to 40. Below the graphs are the values for the highest accuracy score and lowest log loss.

Logistic loss (log loss) measures the performance of a binary classification model whose output is a probability between 0 and 1. One goal of a machine learning model is to minimize the log loss value. The value of log loss can be anything from zero to infinity. The lower the log loss the higher the predictive probability-power of your model. The training accuracy score is found by calculating correct_predictions / total_predictions in the training dataset. For the best model, we want to maximize the accuracy score our model produces with 1 being the best possible accuracy for our model. Because max_iter controls how many steps the model takes in the gradient descent before giving up, a higher max_iter is needed if the model does not converge at lower max_iter numbers (as was the case in part 1 problem 1 of this project).

We can see from Figure 1 that as the max_iter number increases the Logistic loss approaches 0 and the Accuracy score approaches 1. In order to maximize model potential, we must minimize the log loss while maximizing accuracy. For this model we get this with a higher max_iter number. By looking at the tabular data we can see the MAX for the training accuracy happens at *max_iter = 36* while the min for the log loss happens around *max_iter = 40*. When choosing the max_iter that we want we will have to decide if we want to prioritize maximizing the accuracy score or minimizing the log loss. In this case the difference between the accuracy scores and log loss scores from max_iter 36 vs 40 is miniscule with only a 0.01% difference in accuracy and a 0.7% difference in log loss.

2. The figure below is of the weights associated with the pixel 000 from the data_digits_8_vs_9_noisy dataset for each max_iter value in the range 1 to 40.

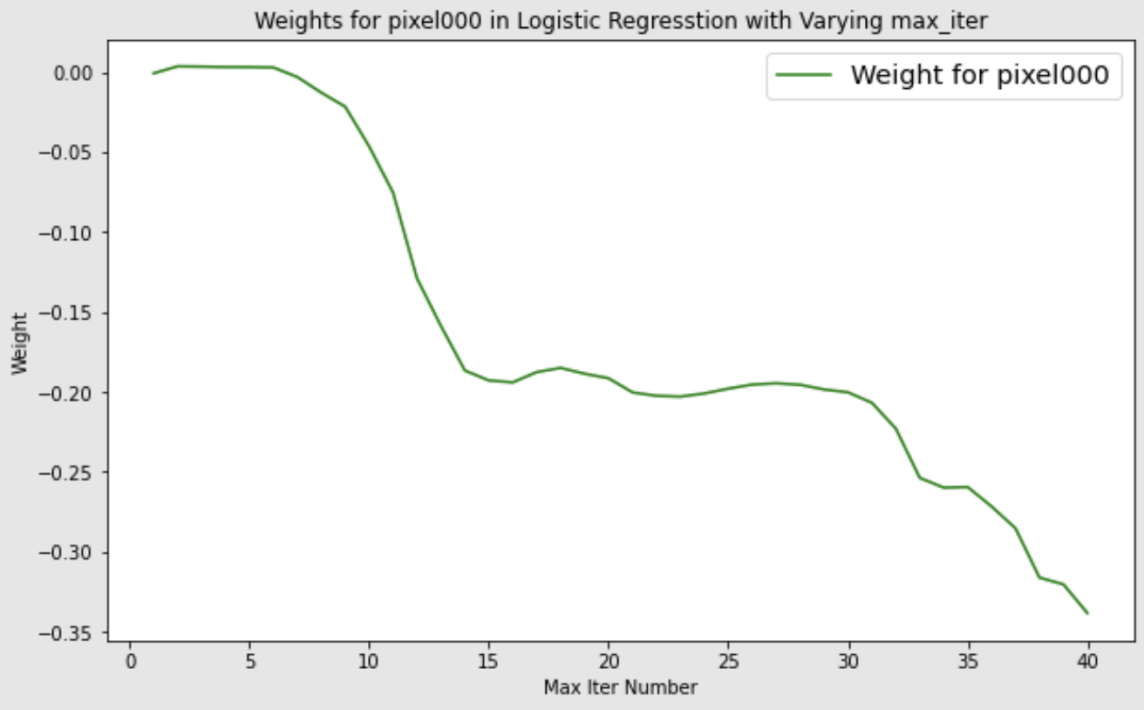


Figure 2: Weights for each max_iter value in the range 1-40 for pixel 000 in the data_digits_8_vs_9_noisy dataset

In Figure 2, we can see the first coefficient/weight values for each model with *i-th* max_iter value. The *coef_* recorded here are non-intercept coefficients and thus tells you how pixel 000 being a 1 and not a 0 (zero being that pixel is black and 1 meaning that pixel is colored white) increases or decreases the odds of the outcome to be a number 9 or a number 8. A greater absolute value of the weight means that this pixel has a larger effect on the odds of the outcome being a 0 or a 1 (8 or 9) and a smaller absolute value means that this pixel does not have much effect on determining the outcome of the handwritten letter in the image. A larger negative value is associated with a greater likely hood of that pixel predicting an 8 and a greater positive value is associated with a greater likely hood of that pixel predicting a 9. We can see from Figure 2 that there is a general trend that as the max_iter number increases the weight associated with pixel 000 decreases. By looking at the tabular data we can see the highest weight score

for pixel 000 is at max_iter = 2 while the lowest weights score for pixel 000 is Max_iter = 40. Max_iter = 1 has the absolute values closest to zero and thus represents the max_iter where pixel 000 has the least influence on predicting the image. Figure 1 above showed us that the smallest log loss was achieved at max_iter 40 which Figure 2 tells us is associated with the greatest absolute value of weights for pixel 000. Meaning at max_iter = 40 pixel 000 is likely a predictor of the image.

3. Figure 3 shows the log loss of the test data for a logistic regression with a regularization parameter C = np.logspace(-9, 6, 31). The log loss is minimized at log loss= 0.090, accuracy = 96.6%, and C = -1.5 in log value (C= 3.162278e-02). The accuracy is maximized at log loss= 0.091, accuracy = 96.7%, and C = -1.0 in log value (C= 1.000000e-01).

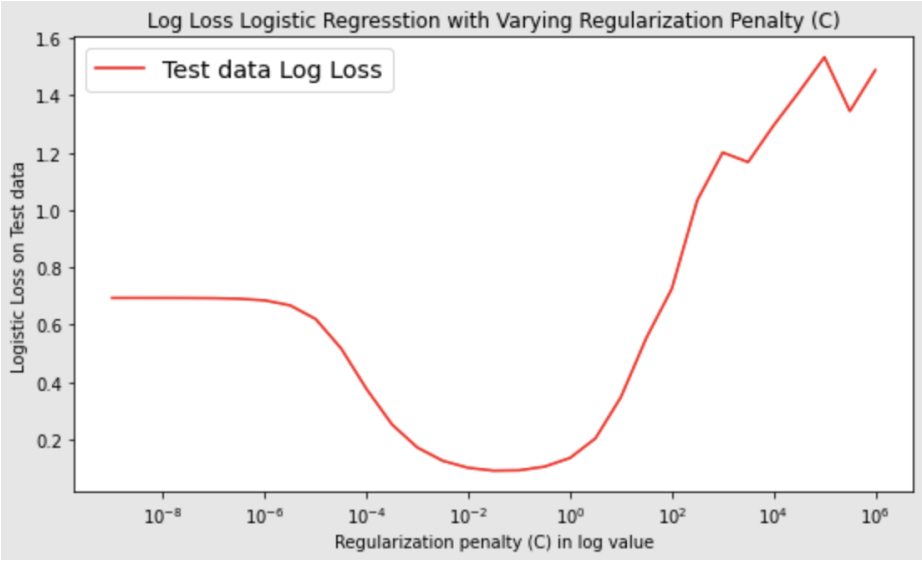
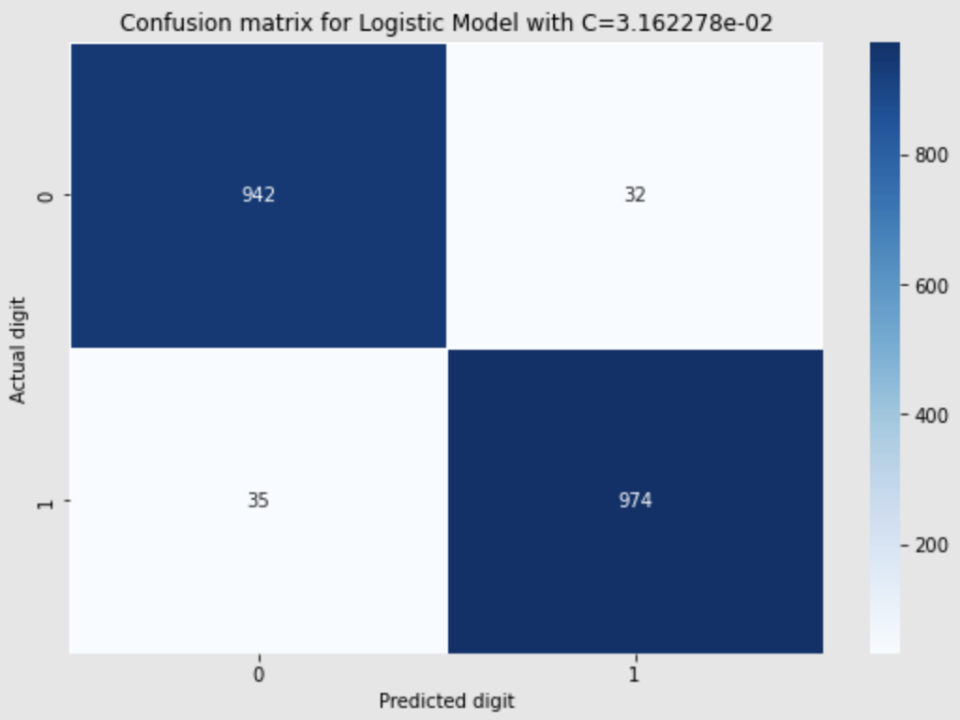


Figure 3: Logistic regression model fitted to data_digits_8_vs_9_noisy dataset with the regularization parameter C ranging from 10^-9 to 10^6 in log value (with 31 total values of C).




```
The acuracy of the model = TP+TN/(TP+TN+FP+FN) =  0.924
The Missclassification = 1-Accuracy =  0.076
Sensitivity or True Positive Rate = TP/(TP+FN) =  0.927
Specificity or True Negative Rate = TN/(TN+FP) =  0.922
              precision    recall  f1-score   support

         0.0       0.96      0.95      0.96       974
         1.0       0.96      0.96      0.96      1009

    accuracy                           0.96      1983
   macro avg       0.96      0.96      0.96      1983
weighted avg       0.96      0.96      0.96      1983
```

Figure 4: Confusion Matrix and basic stats of logistic model of data_digits_8_vs_9_noisy dataset at C = -1.5 in log value (C= 3.162278e-02)

4. Visualizing the images where the model did not correctly identify them.

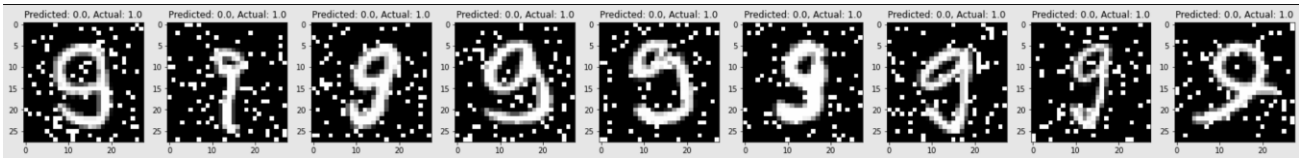


Figure 5: Nine false negatives produced by the model. These images were of a 9 and an 8 was predicted by the model.

Many of the mistakes seen in the false negatives can be seen to be written 9's that have a very loopy tail at the bottom that resembles but is not exactly the bottom half of an 8.

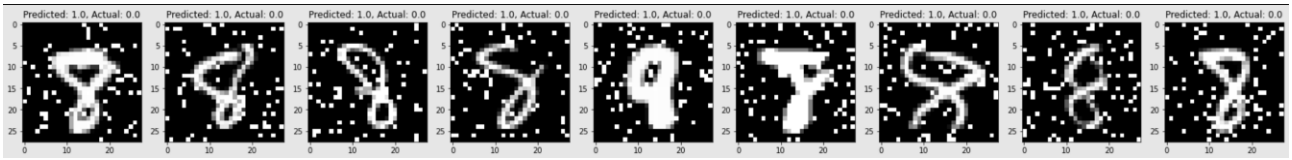


Figure 6: Nine false positives produced by the model. These images were of an 8 and a 9 was predicted by the model.

Many of the false positive images seen above show 8's that have a bigger top loop than bottom loop almost resembling a thick line on the bottom half of a 9.

The model appears to be susceptible to irregular writing and positioning of numbers 8 and 9 which leads to image missclassifications.

5. Visualizing pixel weights

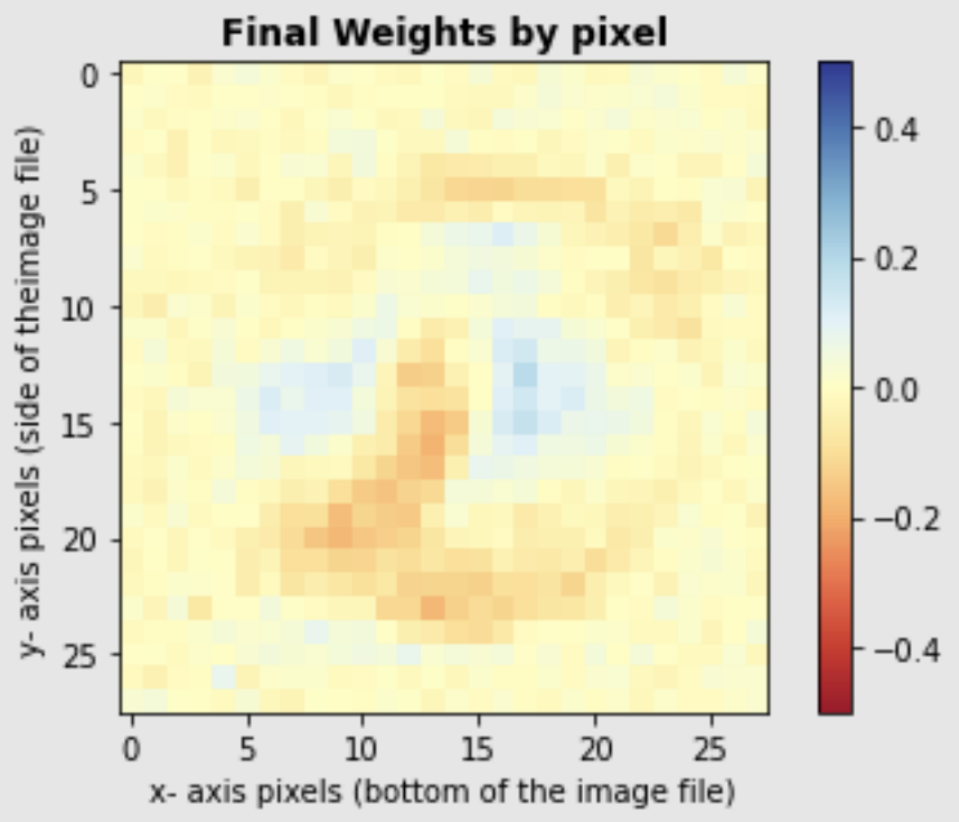


Figure 7: 28x28 pixel visualization of pixel weights in the model. Pixels corresponding to 8 are negative (red weight) while pixels corresponding to 9 are positive (blue weights).

The figure above shows the weights associated with each pixel of the image inputs. We can see above the positive weights in blue and the negative weights in red with the zero (or very small weights) a more

yellow color. Thus, the yellow areas of the figure above indicate areas that do not determine or are not exclusive to the 8 or 9 or tend to not appear in the image and thus these pixels have little to no effect on what the image is or what the prediction for what the image is. We can see a strong negative weight presence (and thus a good predictor of the image being of an 8) in the 5-15 range along the x-axis and the 15-25 range along the y-axis. This intuitively makes sense because it is exactly the area where an 8 would be filled out but would be black space if a 9 was drawn. Figure 7 also suggests that in this data set, 9's tend to be drawn with the top part starting lower than where the top part of the 8 is drawn. as we can see a blue downward curve right below a red one.

Part Two: Trousers v. Dresses

All figures on last page

The first step to building a cogent machine learning model is to familiarize yourself with the dataset. I began by loading in the dataset data_trouser_dress comprised of 28x28 pixel images of various trousers or non-trousers. The output variables were 1 if the image was of trousers and 0 if the image was not of trousers. In order to better visualize the training data, I turned the dataset into a dataframe (see figure 8). The dataframe is made up of 12000 rows and 785 columns. The datatype is integers that range from 0 to 1 with no null values. Next, I visualized the output values in the test data (all the y_test values) to see the distribution of 1 and 0 in the training data (see figure 9). The bar graph in Figure 9 shows us that our training data has an equal number of trouser and non-trouser images. This means that our model is not unbalanced and should not suffer from bias towards false positive or false negative while obtaining a relatively high accuracy score due to an uneven distribution of outcomes/training in outcomes. The distribution of pixel values in the dataset can be seen in Figure 10 which shows the majority of values landing at or near zero.

Some basic geometric transformations were performed on the dataset, but they resulted in slow models that took multiple hours to load. One of the biggest limitations in this analysis was computational power and the inability to load model results when transformations that increased input data size were performed. These transformations were in this case saved until the end, after hyperparameter tuning, which is not ideal but was necessary given the low computational power/speed.

Next, I took a look at what the accuracy score and log loss would look like for various test train splits and plotted the results in Figure 11. Based on these results, I decided to take a **20% test 80% train** split for this model. After this I looked at the ideal K-fold cross validation (within the limits of my laptop's computational power). The accuracy and log loss were plotted for the logistic regression model with a 20-80 test-train split and a k-fold cross validation ranging from 2-11 as seen in Figure 12. The accuracy remained constant for all tested k values while the log loss only increased slightly at 4-fold and after 8-fold. In order to minimize the time needed to run models the lowest k-fold value with a high accuracy and low log loss (**k=2**) was chosen.

The model was then trained with the 80-20 train-test split and a k-fold of 2. The very first model run was a basic logistic regression with no other hyperparameter tuning in order to get a baseline of the model with no further adjustments (see Figure 13 for confusion matrix and basic stats). The accuracy score of this model's testing data prediction was 95% with a true positive rate of 94% and a true negative rate of 97%. The results displayed in Figure 13 display this model's ability to better identify negative values than positive ones with a total error rate of 4.7%.

Next, I explored feature transformation of input features into the established model. Each column in the data frame represents a pixel and each row represents an image. All variables are numeric. By indexing min and max values in the data frame we can see that all the pixels have the same number range of 0 to 1 and thus data scaling such as MinMaxScaler, MaxAbsScaler, and StandardScaler are not necessary on this dataset. MinMaxScaler is not needed as it generally works to make all values scaled from 0 to 1 and this data set only has values in that range. StandardScaler is not ideal since there is not a normal distribution of pixel values in this dataset (as seen in Figure 10).

However, I iterated through the established model with various feature transformations including: RobustScaler, QuantileTransformer, and PowerTransformer. I found Yeo-johnson PowerTransformer to have the highest accuracy score and lowest log loss (see Figure 14). With these changes the accuracy of the model was raised to 96%, with a true positive rate of 94%, a true negative rate of 98%, and an error rate of 4.1%. This feature transformation aided the analysis by working to transform the input variables so that they look more normally distributed and thus result in better accuracy for the logistic regression model since it is easier to make predictions on normalized data. Much of the input data is 0 (black) and thus was skewed with many outliers.

After establishing the feature transformation on input data that worked best for me, I moved on to hyperparameter tuning of the model by utilizing GridSearchCV. The highest accuracy was achieved with the parameters *max_iter=1000000, penalty = 'l2',* and *solver = 'newton-cg*'. However, these changes did not improve upon the accuracy or error rate of the model as it achieved the exact same confusion matrix results as the model without the hyperparameter tuning. While geometric feature transformation was initially attempted it would constantly result in long delays for running models and no outputs generated after several hours.

After finalizing the feature transformation, hyperparameter tuning, k-fold, and test train split for this model I generated ROC curves for both training and test data (see figure 14). The AUROC for the test data was 98.7% while the AUAOC for the training data was 99.1%. As expected, the AUAOC for the training data was higher than the AUAOC for the testing data. This intuitively makes sense as the model was trained with the training data and thus is more susceptible to better fitting (and possibly overfitting) the training data. Figure 16 shows example false positives (predicted the image was of trousers but it was of dresses) and false negatives (predicted image was of a dress but it was actually trousers) produced by my model.

	pixel000	pixel001	pixel002	pixel003	pixel004	pixel005	pixel006	pixel007	pixel008	pixel009	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0522	...	0.0	1.0000	0.0	0.0	0.0	0.0
1	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0000	...	0.0	1.0000	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0000	...	1.0	0.0000	1.0	0.0	0.0	1.0
3	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0080	...	0.0	0.0078	0.0	0.0	1.0	0.0
4	0.0	1.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0000	...	0.0	0.0000	0.0	0.0	0.0	0.0

Figure 8: First five entries of the data_trouser_dress data made into a dataframe.

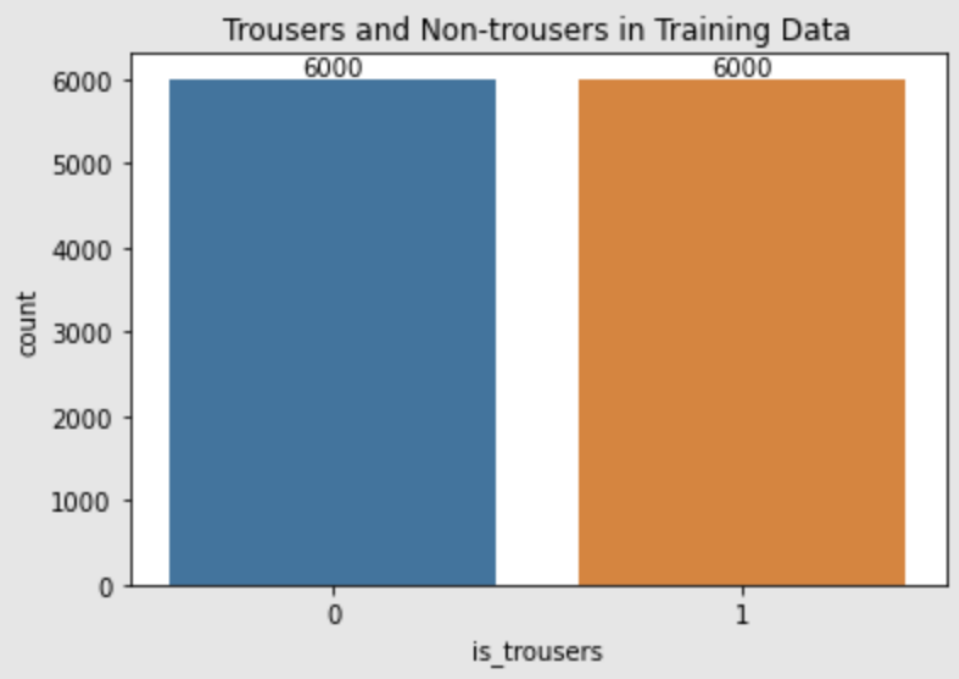


Figure 9: Bar plot distribution of trouser and non-trouser images in the training data

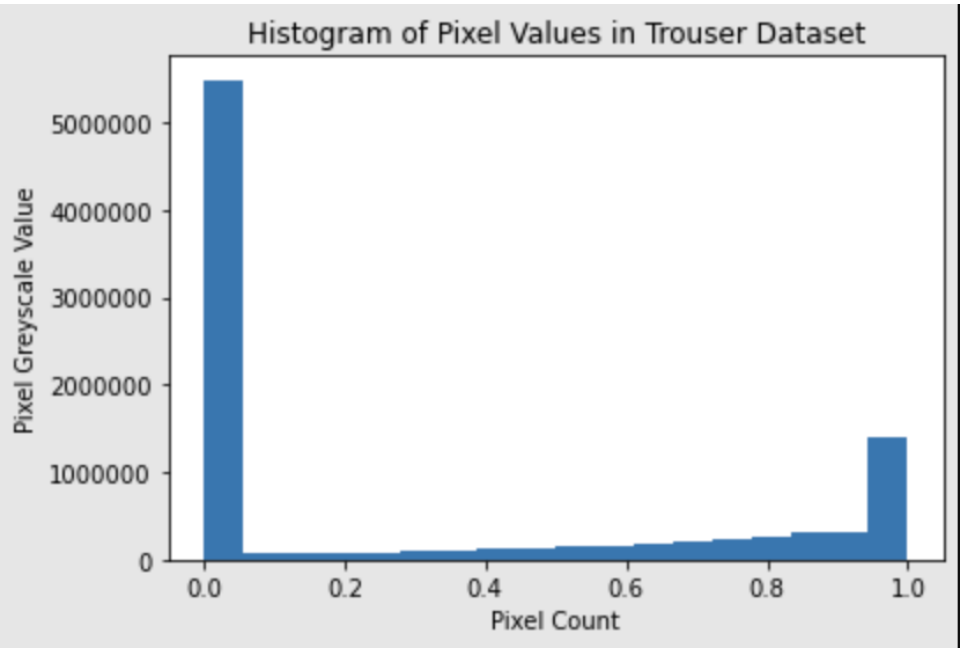


Figure 10: Histogram of pixel values to visualize greyscale value distribution throughout dataset

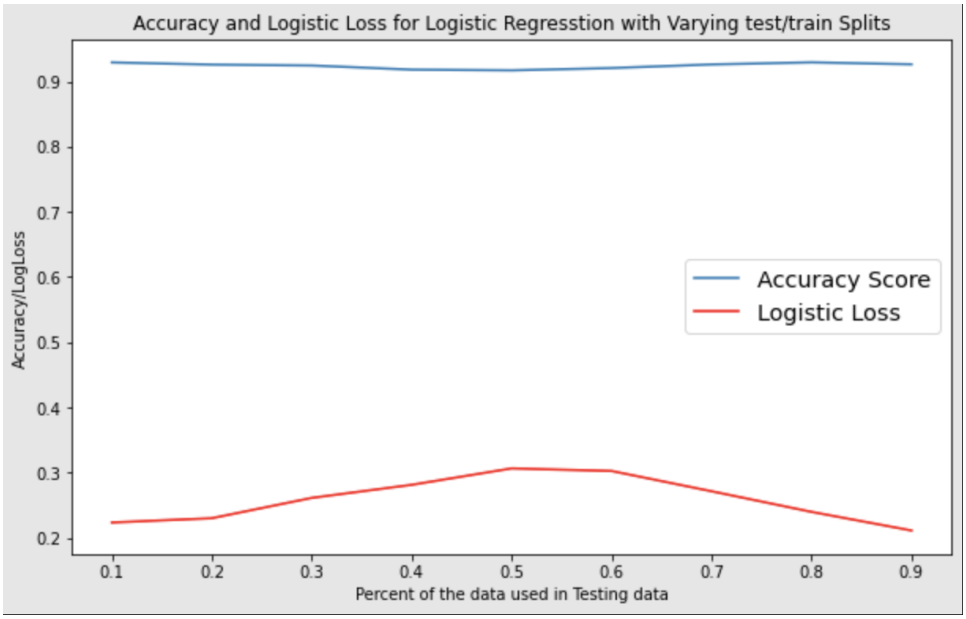


Figure 11: Log Loss and Accuracy for Logistic Regression model iterating over various test train splits

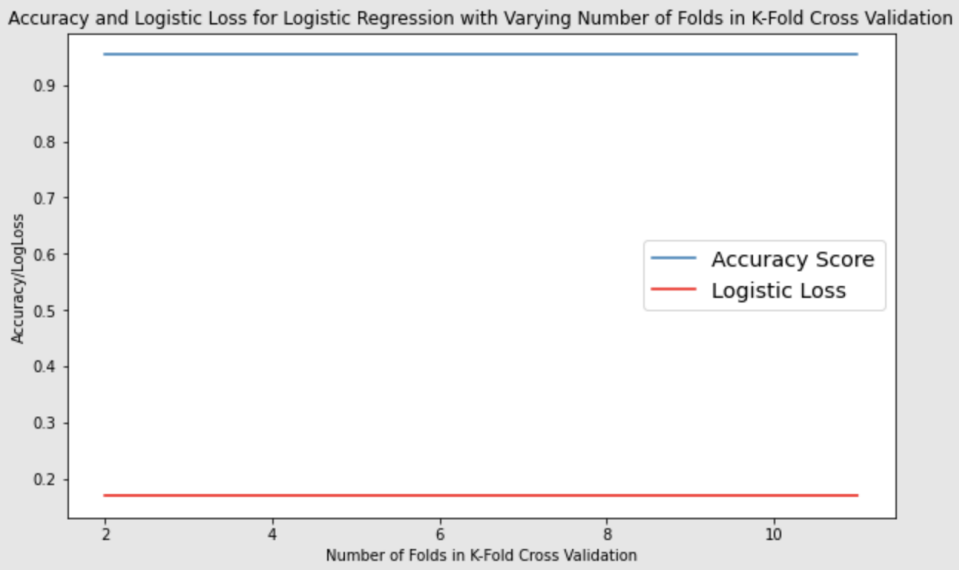
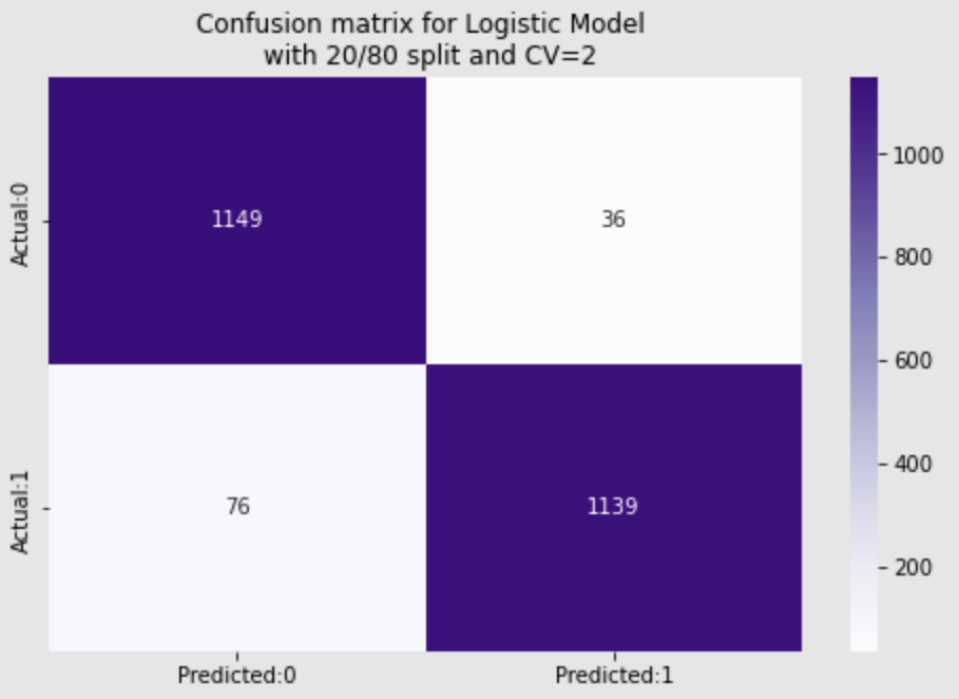


Figure 12: Accuracy and Logistic Loss for Logistic Regression with Varying Number of Folds in K-Fold Cross Validation

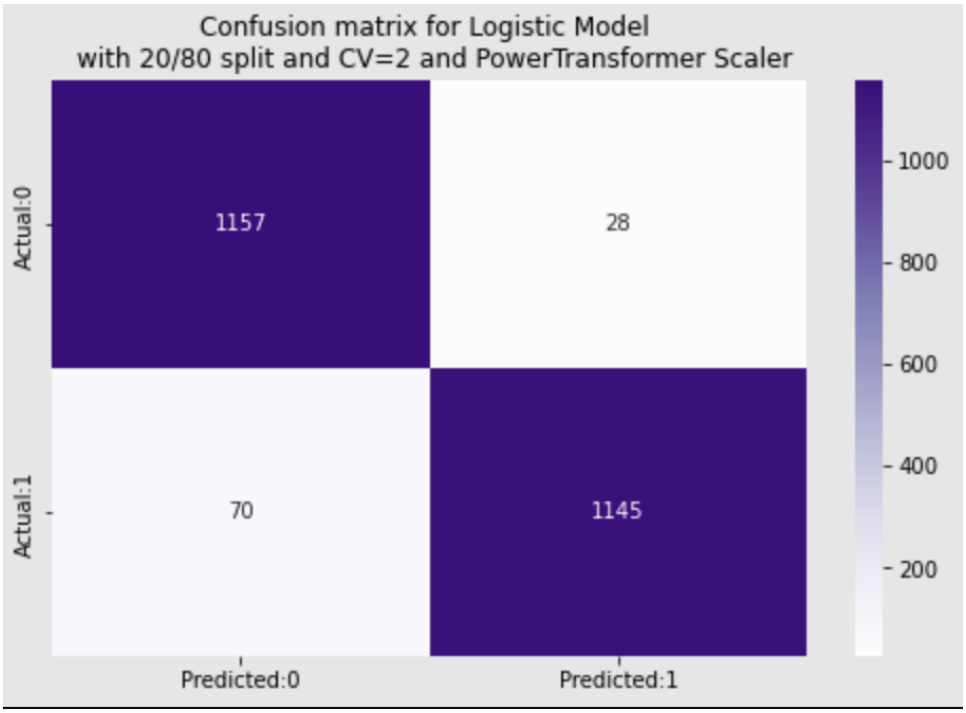


```
The acuracy of the model = TP+TN/(TP+TN+FP+FN) =  0.953
The Missclassification = 1-Accuracy =  0.047
Sensitivity or True Positive Rate = TP/(TP+FN) =  0.937
Specificity or True Negative Rate = TN/(TN+FP) =  0.97
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      1185
           1       0.97      0.94      0.95      1215

    accuracy                           0.95      2400
   macro avg       0.95      0.95      0.95      2400
weighted avg       0.95      0.95      0.95      2400
```

Figure 13: Basic Logistic Regression Model with 20/80 split and CV=2 and basic stats



```
The acuracy of the model = TP+TN/(TP+TN+FP+FN) =  0.959
The Missclassification = 1-Accuracy =  0.041
Sensitivity or True Positive Rate = TP/(TP+FN) =  0.942
Specificity or True Negative Rate = TN/(TN+FP) =  0.976
Error Rate = (FP+FN)/(TP+TN+FP+FN) =  0.041
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      1185
           1       0.98      0.94      0.96      1215

    accuracy                           0.96      2400
   macro avg       0.96      0.96      0.96      2400
weighted avg       0.96      0.96      0.96      2400
```

Figure 14: Logistic Regression Model with 20/80 split, CV=2, PowerTransformer Scaler and basic stats

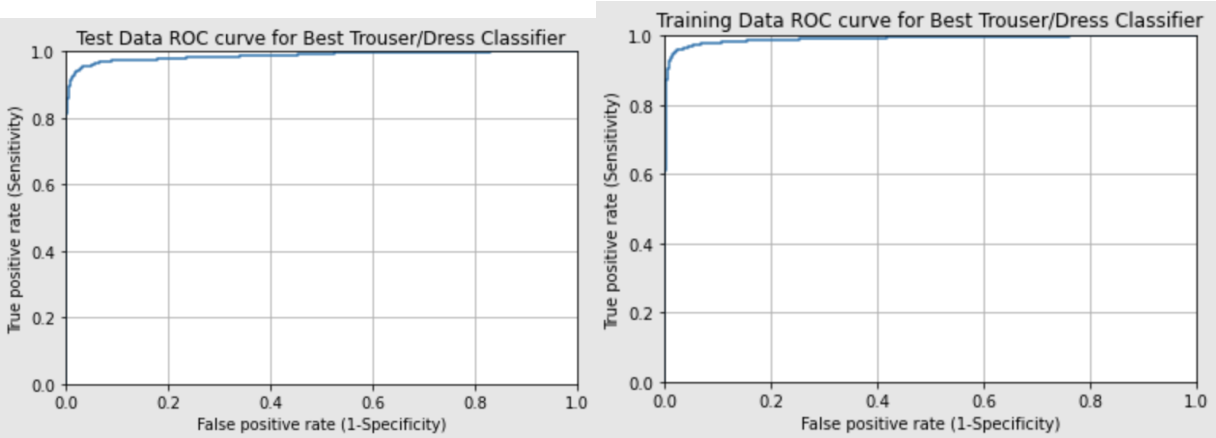


Figure 15: ROC for training (left) and test data (right)

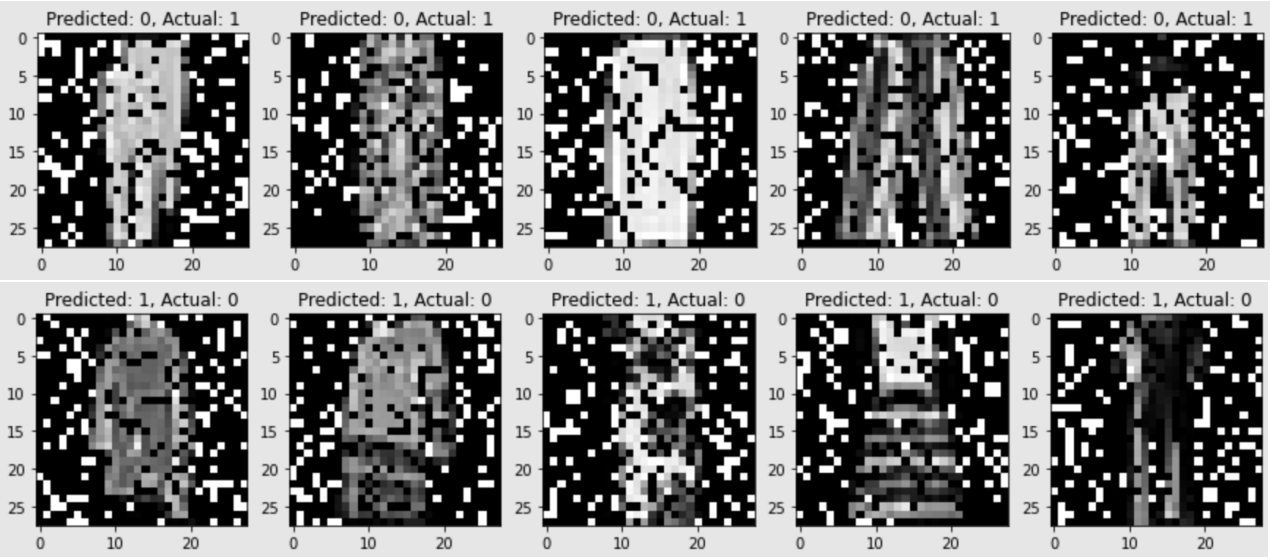


Figure 16: Example images of False negatives (top) and False positives (bottom) predicted by my model