Project 1

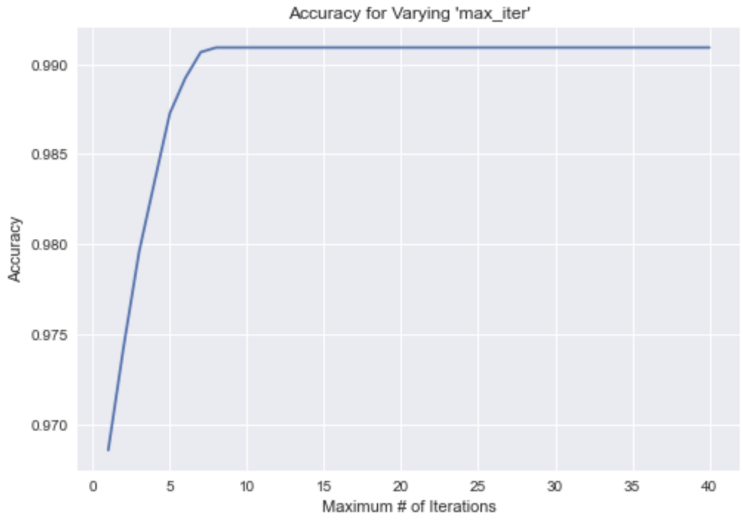
Part One: Logistic Regression for Digit Classification

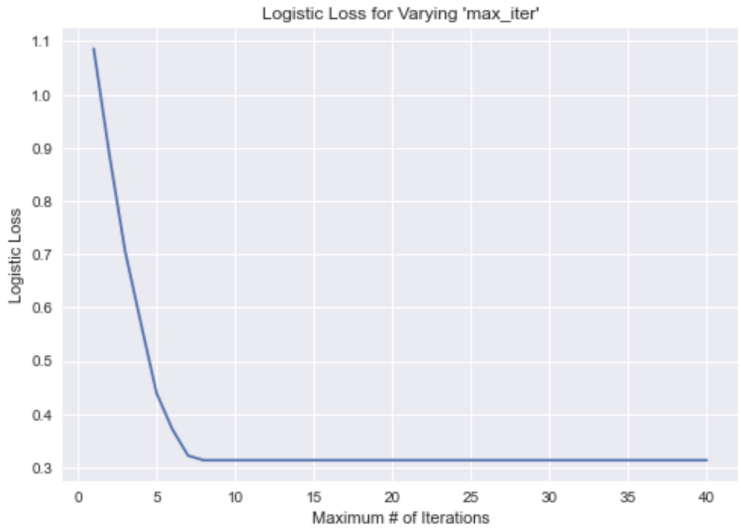
Objective:

Given a set of grayscale image data with images being hand drawn as an 8 or 9, generate a logistic regression model for binary classification of images as a 0 (8) or 1 (9).

1.1

Using sklearn's logistic regression model with the liblinear solver, the max iteration parameter was tested on values 1 to 40 to observe what happens when the model is limited to the number of iterations allowed to converge on its solution. For each max iteration value, the accuracy and logistic loss of the model was measured on the training data. The results were then plotted on two separate graphs as seen below:



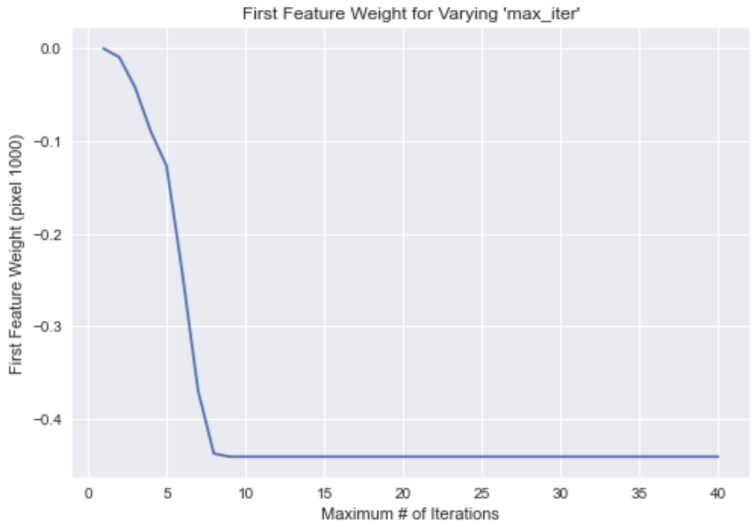


Results:

Increasing the maximum number of iterations for the logistic regression model to converge on its solution resulted in increased accuracy and decreased logistic loss until 8 iterations were reached. Beyond this point the accuracy and logistic loss of the model does not seem to show any improvement, indicating that the model only needs 8 iterations to converge on its solution. It is also interesting to note that both accuracy and logistic loss level off at the same number of iterations. However, these results make sense since accuracy measures the number of correct predictions and logistic loss measures how close the prediction probability is to the actual value, therefore, we can expect that higher accuracy would normally result in lower logistic loss.

1.2

For each logistic regression model generated, the first feature weight was also recorded. This feature weight was then plotted against each model generated:

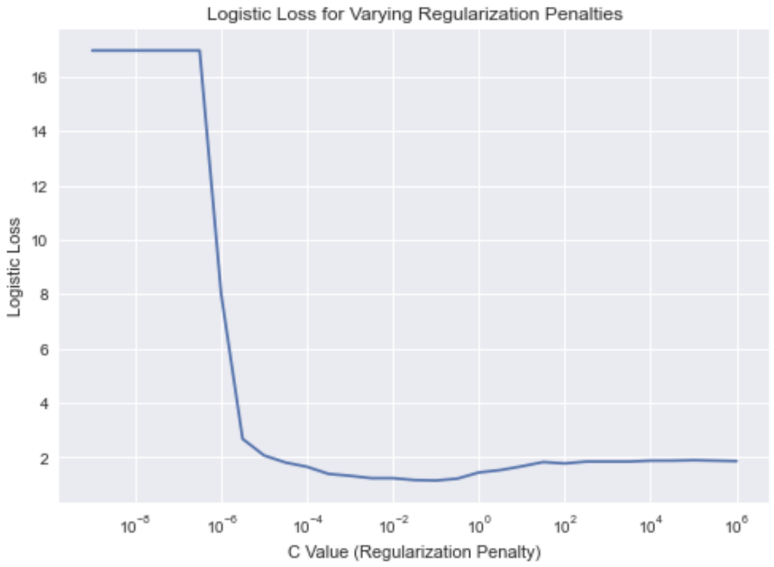


Results:

The resulting graph above shows that as the maximum number of iterations allowed for the solver increases, the first feature weight decreases until about 8 iterations. This is the same number of iterations where the accuracy and logistic loss also level off. This result reinforces the previous statement that the solver only needs 8 iterations to converge on its solution. That is, after 8 iterations, the model has maximized accuracy, minimized logistic loss, and found an appropriate first feature weight corresponding to pixel 1000.

1.3

Next, various regularization penalties were explored for the logistic regression model. In logarithmic space, an evenly spaced range of values from -9 to 6 were tested. For each value of regularization penalty, a model was generated on the training data and logistic loss was calculated on the testing data. The regularization penalty resulting in the least logistic loss was then recorded and the accuracy of that model was also scored on the testing data:



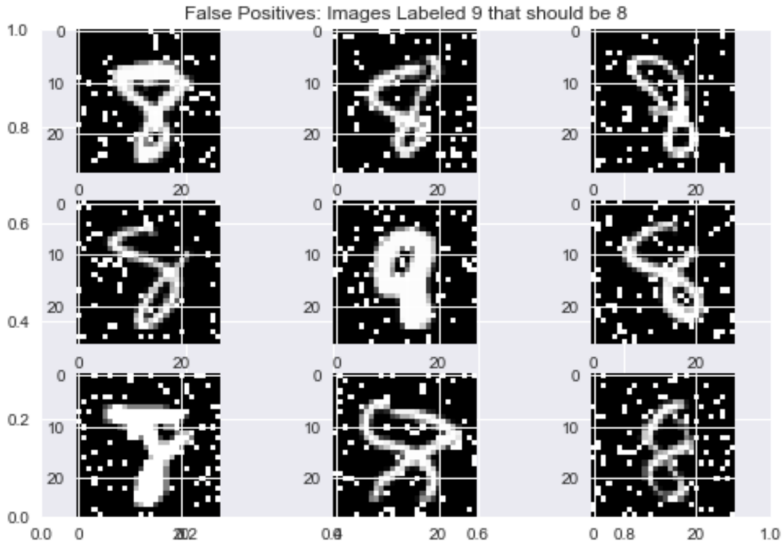
Results:

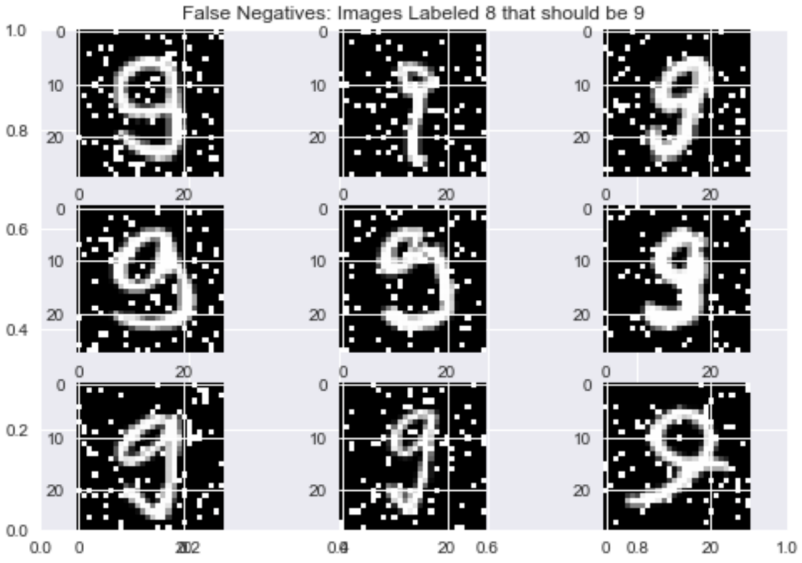
Minimum logistic loss of 1.1496 was achieved with regularization penalty 0.1, this model scored an accuracy of 0.9667 on the testing data. Important to note, the c value represents an inverse penalty, this means smaller values specify stronger regularization. The confusion matrix of this model was also generated:

Predicted	0	1
True		
0	942	32
1	34	975

1.4

In order to further analyze the mistakes being made by the model. The 9 instances of false positives and 9 instances of false negatives were collected and plotted to visualize why the model might be incorrectly classifying these models:



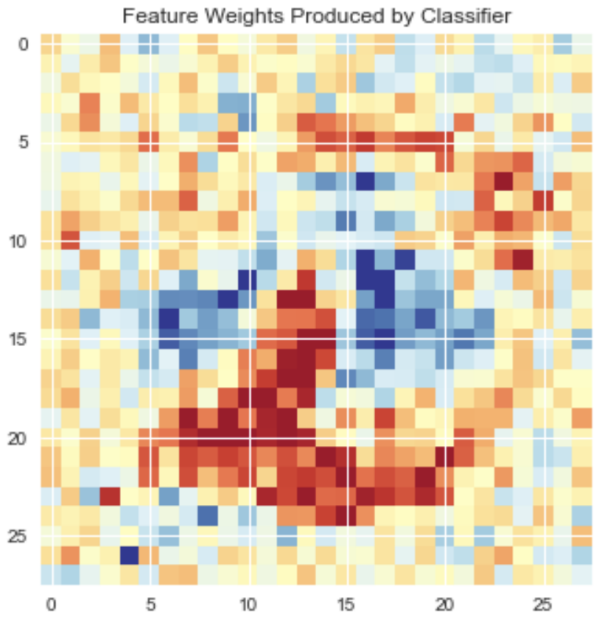


Results:

For the false positives, the eights appear to have pretty tight bottom loops that make them appear to be more of a line than a circle causing the model to classify them as a nine. For the false negatives, the bottom part of the nine seems to curve up quite a bit, almost appearing to close the circle, which I'm assuming is why the model is classifying them as eights. Other than that, the images are pretty 8s and 9s. Another issue that may be leading to incorrect classification, could be from the exterior noise. These extraneous pixels / features may be incorporated to have classification value when they are just noise. Thus, I believe removing these white pixels may increase the accuracy of the model.

1.5

To analyze the final weights produced by the classifier, the feature weight coefficients were reshaped to a 28 x 28 pixel matrix corresponding to the pixels of the original images. The results were plotted on a red, yellow, blue colormap:



Results:

Based on the colormap image it appears as if the red pixels correspond to an 8, meaning they have negative weights. This is because the red is highly concentrated in the lower left of the image, which is exactly where an 8 closes the circle where a 9 is kept open. This would mean that the blue weights correspond to a 9, meaning they have positive weights. This again, also makes sense since the blue is concentrated in a top semicircle which is where a nine generates most of its image. In order to check these predictions I printed key indices of red and blue pixels which confirmed my hypothesis.

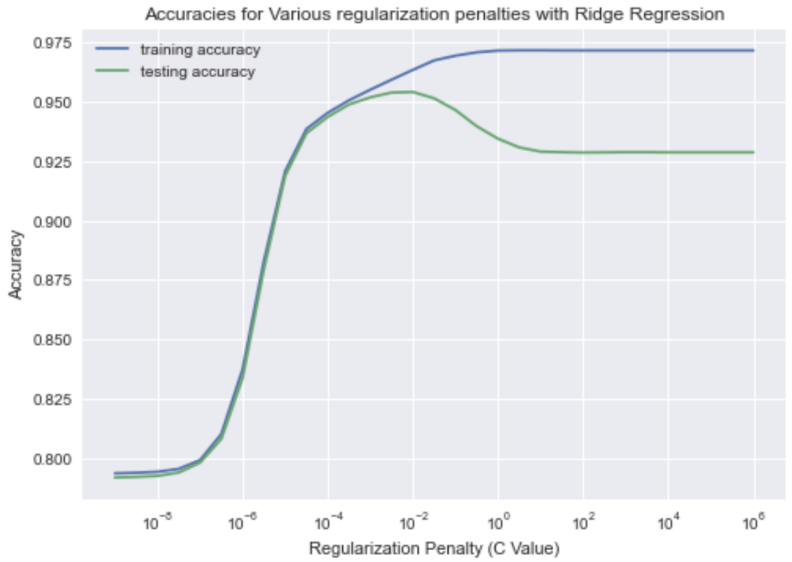
Part Two: Trousers vs Dresses

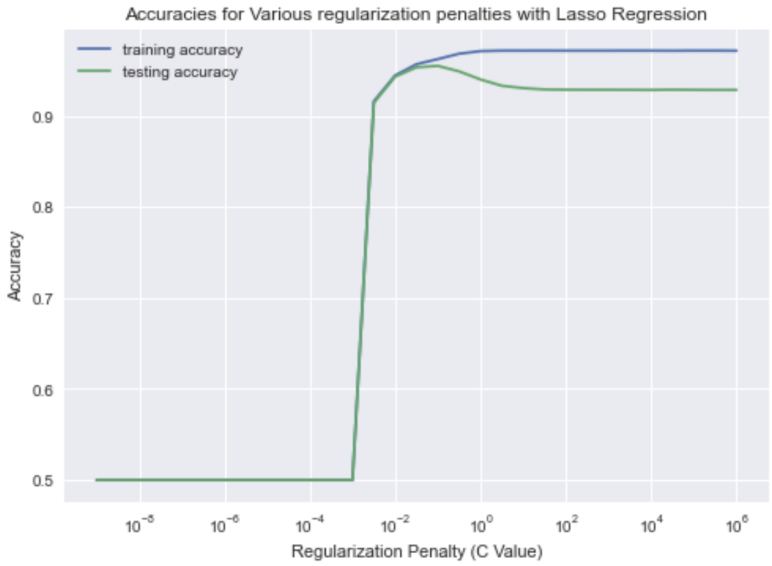
Objective:

Given a set of grayscale image data of trousers and dresses generate a logistic regression model for binary classification of images as a 0 (dress) or 1 (trousers).

2.1

For the first step in generating my logistic regression model I tested various regularization penalties on sklearn's logistic regression model with the liblinear solver. In logarithmic space, an evenly spaced range of values from -9 to 6 were tested. Since no testing output data was provided, for each value of regularization penalty, the model was 5-fold-cross-validated on the training data and the average accuracy was recorded. The model with highest accuracy was then determined and the corresponding regularization penalty was recorded. This entire process was conducted two times, first using L2 Regularization (Ridge Regression), then again using L1 Regularization (Lasso Regression):





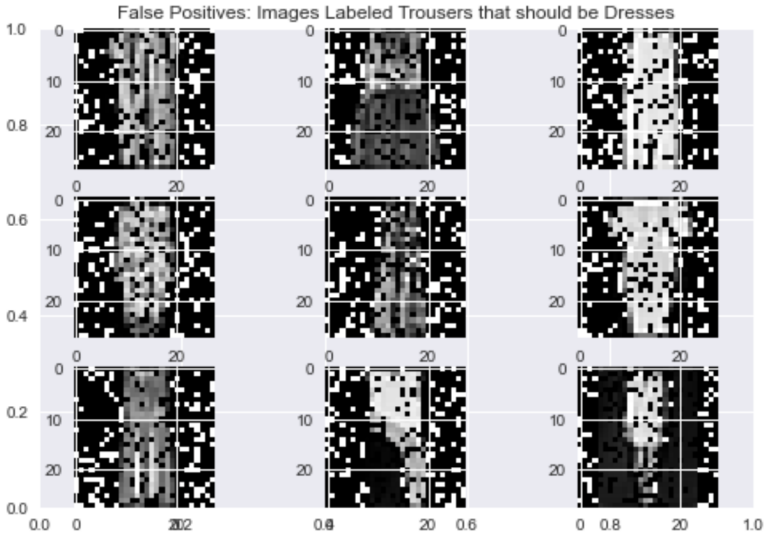
Results:

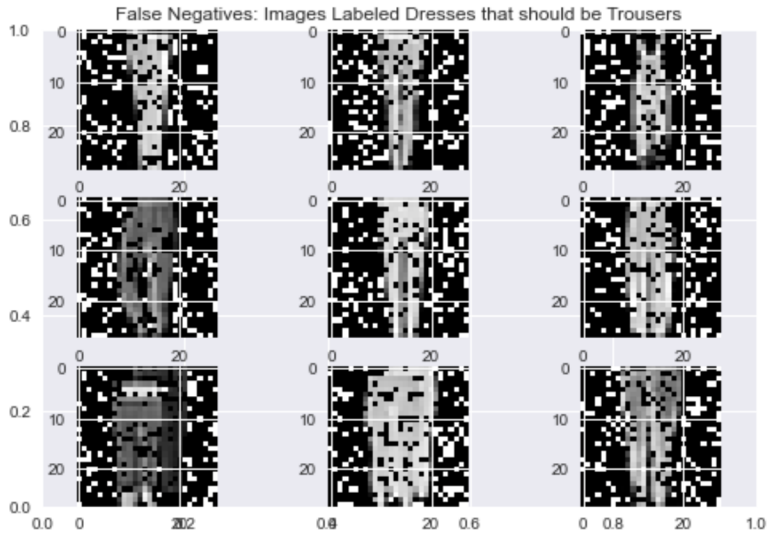
Using Ridge Regression, the best model was generated on an inverse regularization penalty of 0.01 and the cross validated model scored an average accuracy of 0.9541 on the testing data. Using Lasso Regression, the best model was generated on an inverse regularization penalty of 0.1 and the cross validated model scored an average accuracy of 0.9548 on the testing data. Based upon these results, the best model was generated using L1 Regularization / Lasso Regression and an inverse regularization penalty of 0.1. This model was then re-created to generate the following confusion matrix on the training data:

Predicted	0	1
True		
0	5835	165
1	284	5716

2.2

In an attempt to analyze the incorrect classifications, 9 false positives and 9 false negatives were graphed:





Results:

Unlike the digit classification in part 1, these images are not quite as clear. However, once again, there is a lot of external white noise that may be interfering with the model's classification of these images.

2.3

From the conclusion in 2.2, the false negatives and false positives appear to have a lot of external white noise. It is also interesting to note, that these extraneous pixels seem to be very white indicating a grayscale value of 1.0. There are also not many if, if any, of these white 1.0 value pixels in the actual image. Therefore, for the next part of this data exploration I removed these white pixels by iterating through the images and converting all 1.0 grayscale values to 0.0. Once this had been completed I retested the best logistic regression model by cross validating and calculating average accuracy just as had been done before.

Results:

After cleaning up the images, the model now scored an improved accuracy of 0.9721 on the testing data. This was a significant increase from the previous accuracy of 0.9548.

2.4

My next feature engineering attempt was a bit of a shot in the dark. Using the new cleaned up images, I calculated the total sum of the greyscale pictures to generate single feature vectors for each image. This data was run on the model and cross-validated to calculate accuracy.

Results:

To no avail, the new data scored an average testing accuracy of 0.6599.

2.5

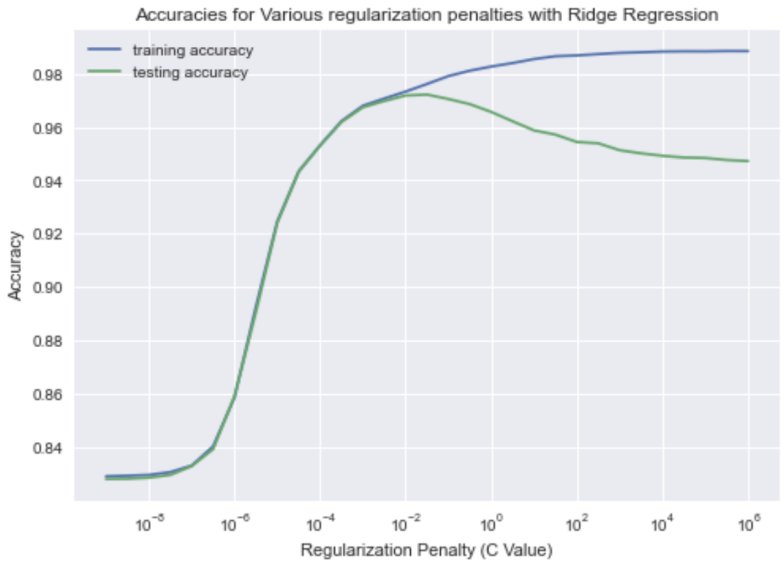
Next I attempted to double the size of my training data by flipping each of the cleaned up images horizontally and adding this new data to the current training data. The doubled data set was ran on the model and 5-fold-cross-validated:

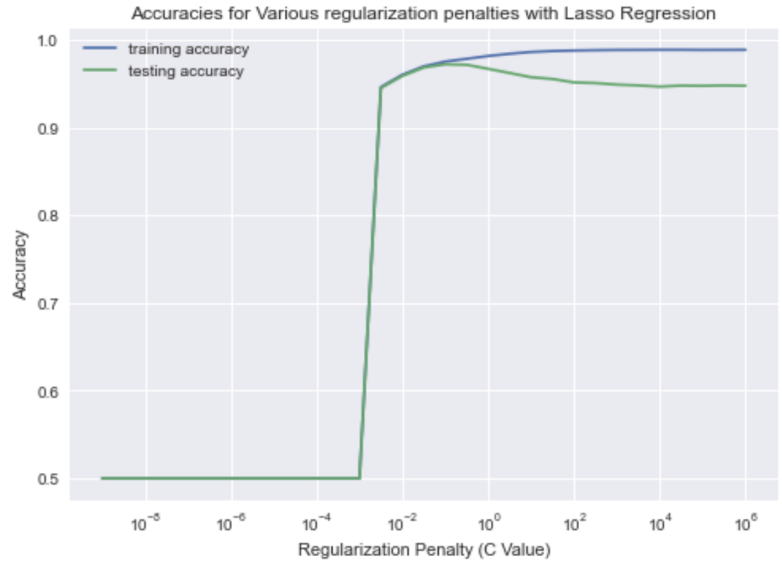
Results:

The model scored an average accuracy of 0.9364, resulting in decreased accuracy. This result was quite surprising as trousers and dresses are both quite symmetrical, so you would expect twice the training data to increase accuracy not decrease accuracy.

2.6

After a couple failed feature engineering attempts, I decided to retrace my steps and repeat the model generation process from 2.1 on the new clean data. That is, I tested the same range of regularization penalties for both Ridge and Lasso Regression:





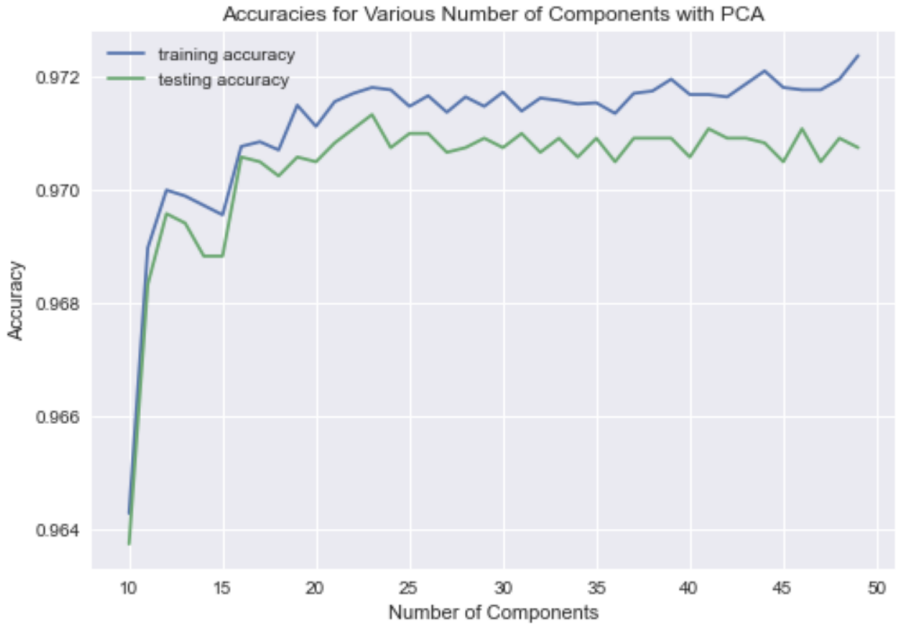
Results:

For L2 Regularization (Ridge Regression) the updated cross-validated testing accuracy was 0.9722 with an inverse regularization penalty of 0.0316. Which was the highest accuracy yet! For L1 Regularization (Lasso Regression) the updated cross-validated testing accuracy was 0.9721 with an inverse regularization penalty of 0.1 (the same as before). Here is the updated confusion matrix on the training data:

Predicted	0	1
True		
0	5915	85
1	201	5799

2.7

Upon some online research for image classification techniques I found a technique called Principal Component Analysis (PCA). PCA is a linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. Essentially, it reduces the number of features in each vector by generating a smaller set of "summary features". Using sklearn's implementation of Principal Component Analysis, I transformed each feature vector on a range of componentes from 10 to 50. The new model was then fit to the new training data and the accuracy was calculated using cross validation:



Results:

The best testing accuracy recorded was 0.9713 using 23 components. However, this was still slightly lower than the best accuracy of 0.9722.

2.8

My last feature engineering attempt involved creating high contrast images of the data essentially rounding all values. That is, for each image vector I converted all values less than 0.5 to 0.0 and values greater than or equal to 0.5 to 1.0. The new data set was then 5-fold cross validated:

Results:

The model scored a testing accuracy of 0.9543, which was quite lower then the best accuracy.

Conclusion:

After testing several different logistic regression models on several different feature engineered data sets, the best model was determined to use L2 Regularization, 0.0316 regularization strength, and the liblinear solver. The best data set was the one that only removed the white external noise pixels. This model paired with this data set scored an accuracy of 0.973 (0.027 error rate) on the leaderboard testing data and had an AUROC of 0.991347. The tables below summarizes the results of the various models tested on the feature engineered data sets along with their non cross validated training accuracy and leaderboard testing accuracy. One last thing that is important to note, all feature engineering performed after the noise removal was performed on the noise removed data set.

Model 1:
Logistic Regression, Penalty: L2, Regularization Strength: 0.01, Solver: 'liblinear'

	Original Data (2.1)	Noise Removed (2.3)	Single Feature (2.4)	Doubled Data (2.5)	PCA (2.7)	Rounded Data (2.8)
Training Accuracy	0.9625	0.973	0.643	0.948	0.971	0.959
Testing Accuracy	0.946	0.967	0.644	0.940	0.967	0.955

Model 2:
Logistic Regression, Penalty: L1, Regularization Strength: 0.1, Solver: 'liblinear'

	Original Data (2.1)	Noise Removed (2.3)	Single Feature (2.4)	Doubled Data (2.5)	PCA (2.7)	Rounded Data (2.8)
Training Accuracy	0.9625	0.975	0.661	0.950	0.972	0.961
Testing Accuracy	0.947	0.971	0.660	0.930	0.969	0.954

****Model 3 (Best Model)****:
Logistic Regression, Penalty: L2, Regularization Strength: 0.0316, Solver: 'liblinear'

	Original Data (2.1)	Noise Removed (2.3)	Single Feature (2.4)	Doubled Data (2.5)	PCA (2.7)	Rounded Data (2.8)
Training Accuracy	0.966	0.976	0.656	0.951	0.972	0.963
Testing Accuracy	0.946	**0.973	0.655	0.940	0.969	0.955

**Best Testing Accuracy