Allen Lau
Project 2: MNIST Dataset and Regression

I. Project Description

Computer vision, defined as the process of training computers to interpret and understand digital images, has an increasingly important role in society. Applying computer vision in areas, like healthcare and automation, allows us to extract information and insight from images that would otherwise be difficult for humans to decipher. The MNIST dataset, consisting of 60,000 28x28 pixel grayscale images of handwritten single digit numbers from 0 to 9, serves as a first steppingstone in exploring computer vision and implementing classification algorithms.

This report will outline the observations, analyses, and results of implementing a least-squares binary classifier and logistic regression model to classify the digits of the MNIST dataset. In addition, Cook's Distance will be used to remove outliers of the training dataset to understand the impact of outliers on classification. Lastly, backward selection will be used to select statistically significant features for the mode.

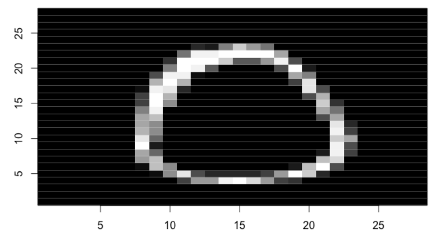
II. Ordinary Least Squares Binary Classifier for digit k vs not k

The ordinary least squares (OLS) method is a linear regression technique that aims to minimize the sum of squared residuals between the actual and predicted values. Using this technique, we can create a binary classifier to determine if an image is digit k or not digit k.

Below is a summary of the procedure taken to create an OLS binary classifier, where the complete R code is in Appendix A:

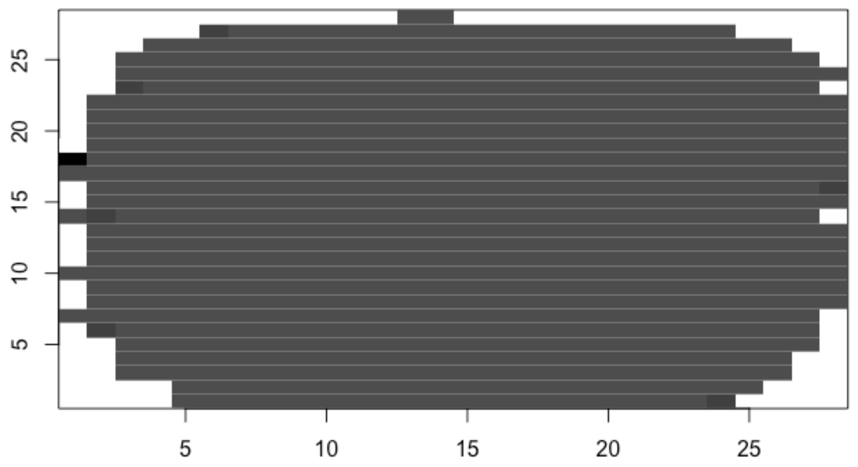
1. Load the digit and label files into dataframes, where each row is a flattened array of the 28x28 image. The dimension of each row is 1x784. These 784 columns are our model features. There are a total of 60,000 rows or images in the entire dataset.
2. Remove model features (columns), where all rows contained zeros. This reduces the dimensionality of the dataset, while addressing potential collinearity considerations within the modeling.
3. Choose 0 as digit k and find indices for k and not k in the digit and label files, which will be used for splitting the dataset for modeling.
4. Split the dataset into one for training and one for testing the model, such that each datapoint has a 50% chance of being randomly sampled.
5. Label images that are digit k as 1, and images that are not digit k as -1.
6. Perform OLS modeling on the training dataset.
7. Visualize model coefficients, calculate error rates of the model on the train and test datasets, create confusion matrices for train and test dataset results.
8. Plot Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage plots.

Below is an example output of digit k = 0, where 0 is classified as 1 and all other digits are -1.



The lm() function is used to create an OLS model, where our explanatory variables are the pixels of the images. The output of the model will be a number that can be negative or positive. We base the classification of the image on this fact, where positive outputs are classified as digit k, and negative outputs are classified as not digit k. The resultant model coefficients can be visualized as a 28x28 image, as seen below. We can see that areas, where digits have not been written on in the 28x28 matrix, are seen as white. This indicates that the model coefficient for these specific pixels is 0 or NA; therefore, they are not used in the calculation of the classification.

OLS Model Coefficient (β) as Image for k = 0; Intercept (β_0) = -0.6809402

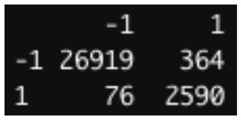


Using the predict() function, we can use the resultant model to classify an input of images. This is done with both the train and test datasets. The classification error rate, which is the number of incorrectly classified images divided by the total number of images, is calculated for both the train and test datasets:

Classification Error Rate for OLS Model k = 0	
Error Rate on Train Dataset	0.01469164
Error Rate on Test Dataset	0.01733719

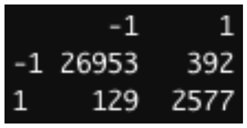
We can also express the performance of the classification resulted from the model in confusion matrices for both train and test:

Confusion Matrix for Train Dataset



```
      -1     1
-1 26919   364
1     76  2590
```

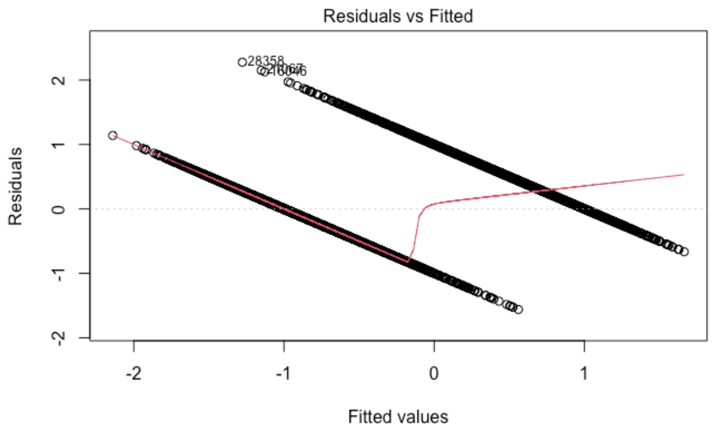
Confusion Matrix for Test Dataset



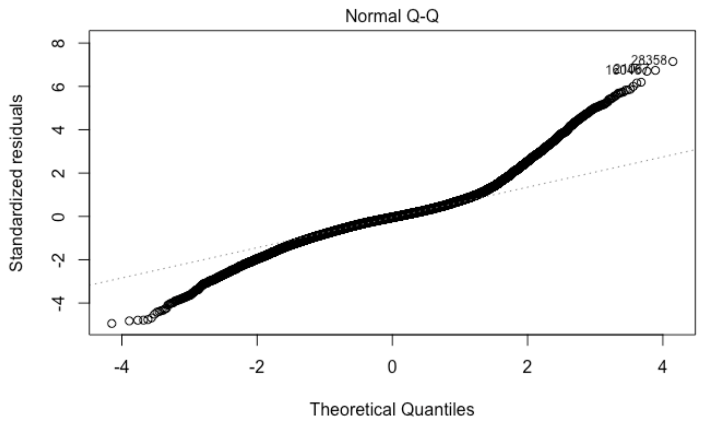
```
      -1     1
-1 26953   392
1    129  2577
```

It is observed that the error rate on the train dataset is lower than test, which is expected since the model was fit on the train dataset. In contrast, the test dataset contains new images never seen by the model. This information is reinforced by the confusion matrices. It is seen that the OLS model does a good job at classifying the MNIST digits, in terms of error rates, with only 1.5% - 1.7% of classifications being erroneous. However, we can also use plot visualizations to determine if OLS is a good fit for this dataset.

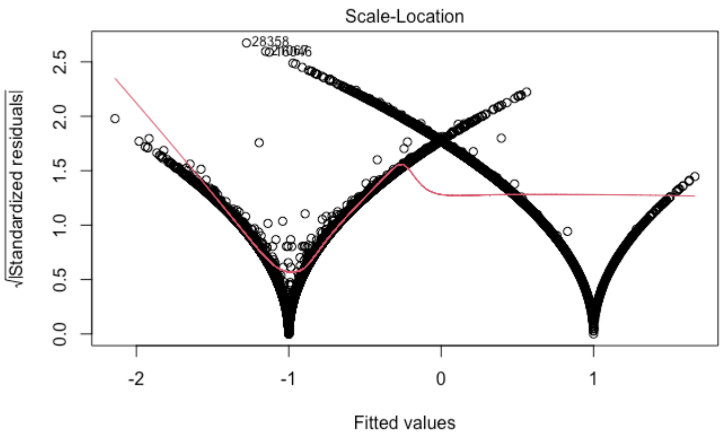
The Residuals vs Fitted plot is used to determine whether the linearity and homoscedasticity assumptions for linear modeling are true. For a good fit, the plot should indicate no correlation or patterns between residuals and fitted values. In other words, we should expect the residuals to be randomly scattered about the dotted line at 0, with no non-linear trends or non-constant variance. For the Residuals vs Fitted Plot below, we do not see the residuals being randomly scattered, with no non-linear trends or non-constant variance.



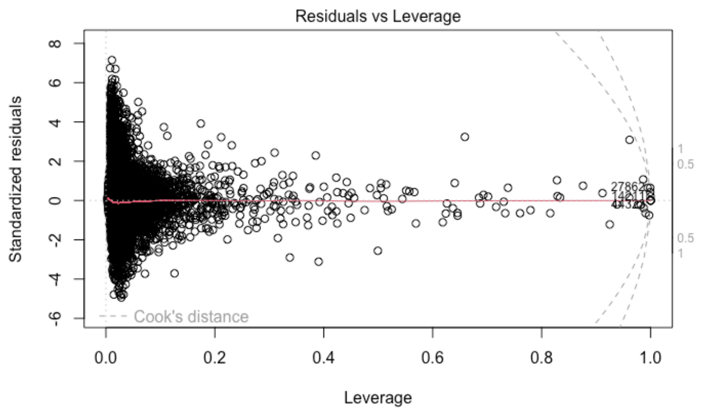
The Normal Q-Q plot is used to determine if data follows the normal distribution. For a good fitting linear model, we should expect that the standardized residuals are normally distributed. On the Normal Q-Q plot, the standardized residuals would follow the dotted line closely with no major deviations from that line near the ends. For the plot below, we see that the points do deviate from the dotted line at the ends.



The Scale-Location plot is used to determine if homoscedasticity assumption of linear modeling is satisfied. For a good fitting model, we would expect that the spread of the residuals is roughly equal at all fitted values. In other words, the plotted residuals should not follow a clear pattern, with the red line roughly being horizontal. In the plot below, we see that the points do follow a pattern and the red line is not approximately horizontal.



The Residuals vs Leverage plot is used to identify influential data points in the model, where leverage is defined as the measure of how much the coefficients of the model would change if a particular data point was removed from the dataset. Therefore, this plot can be used to identify potential outliers. In addition, influential observations may indicate that a linear model may not be a good fit. From the plot below, it is observed that there are a handful of images that touches the dashed Cook's distance line, which indicates that they may be outliers. This will be explored in later sections of this report.



Based on the plots above, we can conclude that an OLS linear model is not a good fit for the MNIST dataset; however, it does reasonably well at classifying k vs not k correctly. Additionally, from the above findings, there are improvements that can be made to the modeling, such as removing outliers or exploring other non-linear models. Improvements in pre-processing can be achieved as well to ensure the best results, like ensuring that the images are scaled and centered consistently or performing additional dimensionality reduction using techniques like PCA.

III. Ordinary Least Squares Binary Classifier for digit vs digit

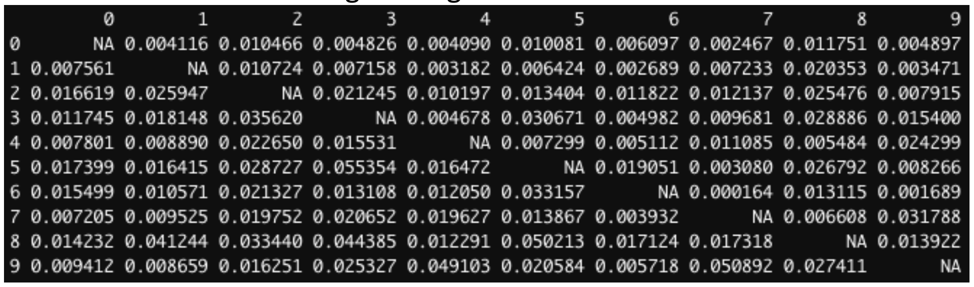
Performing OLS modeling on a specific digit vs another digit can provide insights on which digits are easier to classify when compared to another. The procedure taken to produce the digit vs digit analysis is like section II, with the below differences. The complete R code is in Appendix B.

1. Create a function to perform the identification of digit 1 and digit 2, dataset train/test splitting, modeling/predictions, error rate calculations, and visualization of model coefficients.
    a. Label digit 1 as 1 and label digit 2 as -1.
2. Use loops to call the function over all pairs of digits to generate an error rate matrix.
3. Display coefficients (β) and intercepts (β_0) for the pairs with lowest and highest error rates.

With the model input being only images of digit 1 and 2, labeled as 1 and -1 respectively, the model output will be a positive or negative number. We classify positive outputs as digit 1 and negative outputs as digit 2.

The error rates for both the train and test datasets of a specific digit versus another digit is shown below:

OLS Digit vs Digit Error Rate Matrix



```
         0        1        2        3        4        5        6        7        8        9
0       NA 0.004116 0.010466 0.004826 0.004090 0.010081 0.006097 0.002467 0.011751 0.004897
1 0.007561       NA 0.010724 0.007158 0.003182 0.006424 0.002689 0.007233 0.020353 0.003471
2 0.016619 0.025947       NA 0.021245 0.010197 0.013404 0.011822 0.012137 0.025476 0.007915
3 0.011745 0.018148 0.035620       NA 0.004678 0.030671 0.004982 0.009681 0.028886 0.015400
4 0.007801 0.008890 0.022650 0.015531       NA 0.007299 0.005112 0.011085 0.005484 0.024299
5 0.017399 0.016415 0.028727 0.055354 0.016472       NA 0.019051 0.003080 0.026792 0.008266
6 0.015499 0.010571 0.021327 0.013108 0.012050 0.033157       NA 0.000164 0.013115 0.001689
7 0.007205 0.009525 0.019752 0.020652 0.019627 0.013867 0.003932       NA 0.006608 0.031788
8 0.014232 0.041244 0.033440 0.044385 0.012291 0.050213 0.017124 0.017318       NA 0.013922
9 0.009412 0.008659 0.016251 0.025327 0.049103 0.020584 0.005718 0.050892 0.027411       NA
```

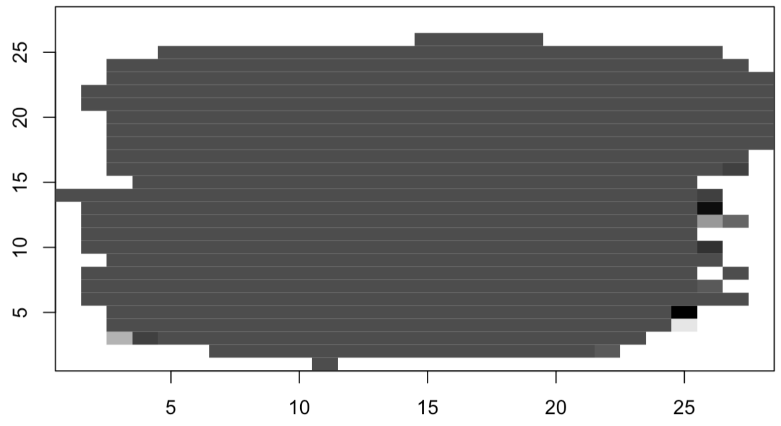
Upper Triangle = Error Rates for Train Dataset; Lower Triangle = Error Rates for Test Dataset

A similar observation made in Part 1 is seen in the digit vs digit modeling error rates, in that the error rate for the train dataset is consistently lower than its corresponding error rate for the test dataset. In addition, we can compare how well the model is able to classify each digit pair, where good model performance is expected for digits with little overlap in their shapes and bad model performance is expected for digits with many overlaps in their shapes. The highest and lowest test error rates are shown below:

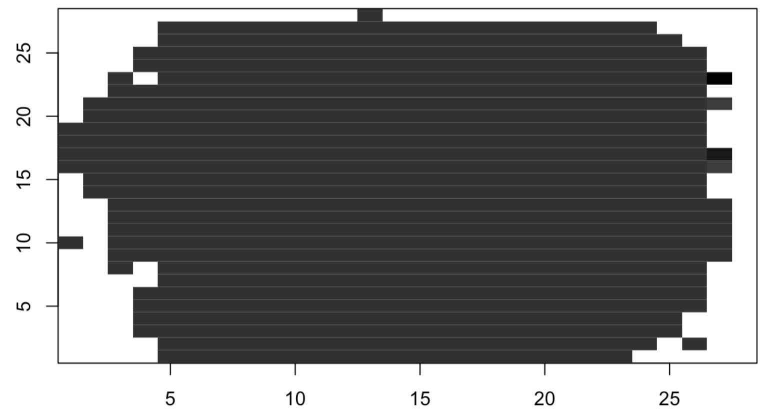
Error Rates for Digit vs Digit on Test Data	
Highest (3 and 5)	0.055354
Lowest (6 and 7)	0.003932

The highest error rate for the OLS method is between 3 and 5. This could be explained by the similarities in the bottom half of the digits and the overlap of the top. The lowest error rate is between 6 and 7, since there are very few portions of the digits that would overlap. Overlapping portions of the digits are important in this analysis since this affects which coefficients in the model are likely to be included in the element-by-element multiplication and summation calculation for the classification. The coefficients for both models are shown below:

OLS Model Coefficient (β) as Image for 3 vs 5; Intercept (β_0) = 0.3776234



OLS Model Coefficient (β) as Image for 6 vs 7; Intercept (β_0) = -0.2175033



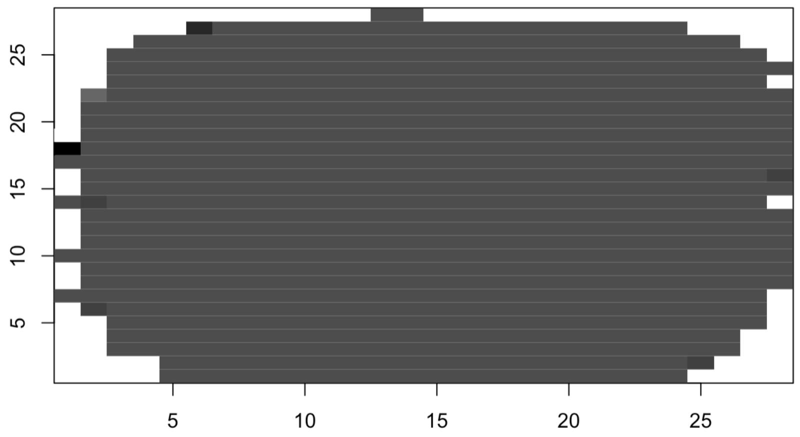
IV. Logistic Regression Classifier for digit k vs not k

Logistic regression is a classification technique that outputs the probability of a binary outcome taking place. As a result, we can both classify an input and report on the probability of the outcome occurring. For logistic modeling for digit k vs not k, a similar process to the one outlined in section II is used. The main differences are as follows, with the complete R code in Appendix C:

1. Model using function, glm() with family = “binomial”.
2. Label digit k as 1 and digits not k as 0

The output of the model will be a number between 0 and 1, where values greater than 0.5 will be classified as 1 or digit k and values less than 0.5 will be classified as 0 or not digit k. This difference in labeling is due to the constraints of the logit function used for modeling. Similar to the OLS model, we can output the coefficients of the model, error rates, confusion matrices, and Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage plots.

Logistic Model Coefficient (β) as Image for k = 0; Intercept (β_0) = -1.535707e15

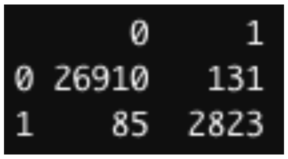


The general shape of the logistic model coefficient, visualized as a 28x28 image, is the same when compared to the OLS model coefficient in section II. However, there are differences in the specific model coefficients and intercept values. This should be expected due to the differences in how the models are generated. The classification error rate for both the train and test datasets are calculated and shown below:

Classification Error Rate for Logistic Model k = 0	
Error Rate on Train Dataset	0.007212261
Error Rate on Test Dataset	0.01717081

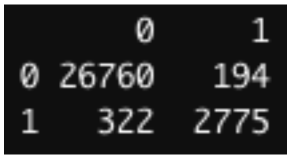
We observe that both error rates for the logistic model are smaller than the corresponding error rates for the OLS model. The below confusion matrices reinforce this fact, when compared to the results in section II.

Confusion Matrix for Train Dataset



```
      0     1
0 26910   131
1    85  2823
```

Confusion Matrix for Test Dataset

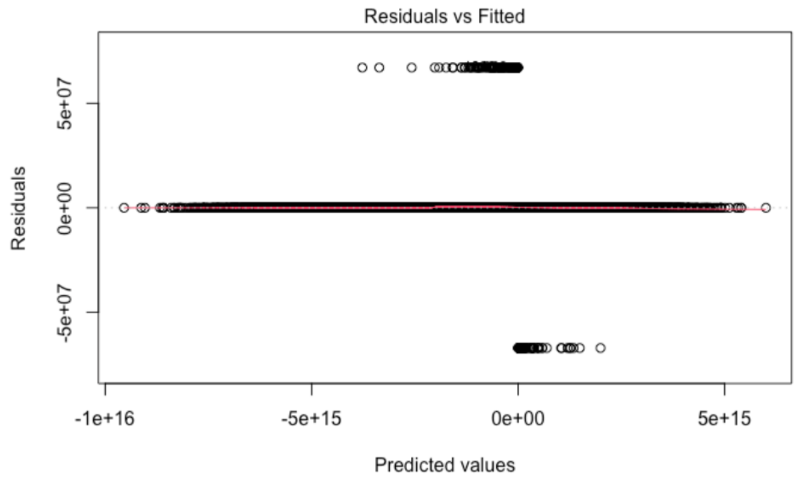


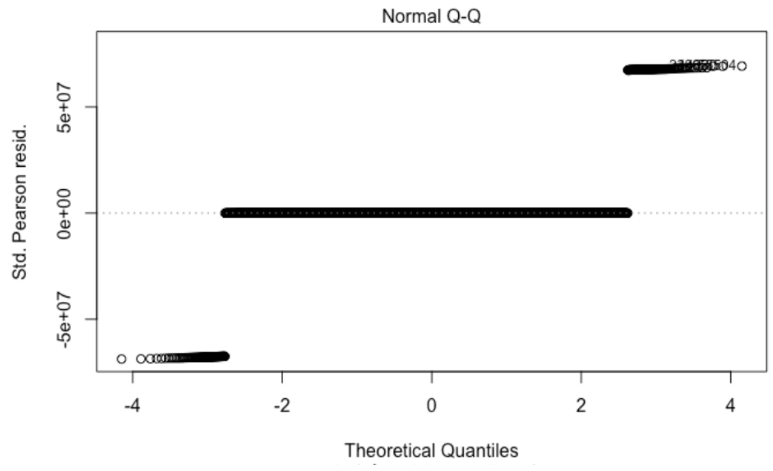
```
      0     1
0 26760   194
1   322  2775
```

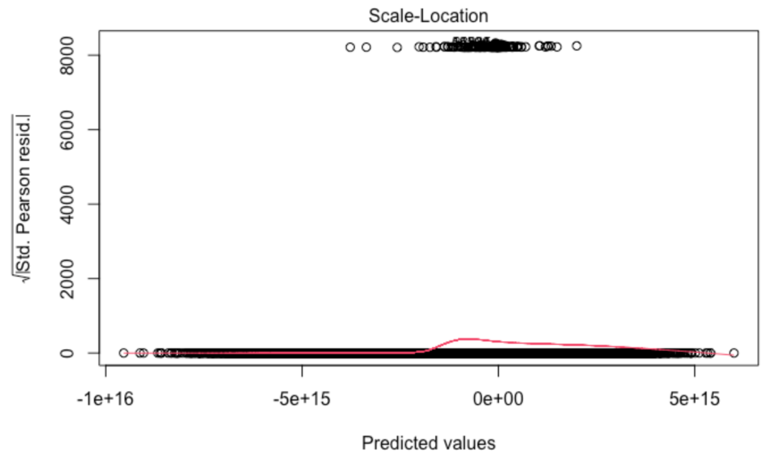
With these results, we can conclude that for digit k vs not k, with k = 0 as an example, logistic regression performs better than OLS at classifying k vs not k correctly.

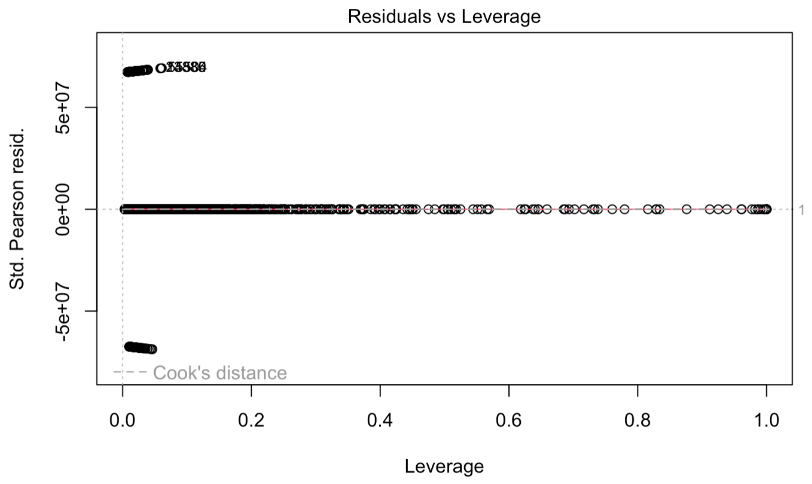
Going further, the Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage plots are used to determine if logistic regression is a good fit for the MNIST dataset.

Looking at the plots below, we see that the scale of the y-axis is consistently large, indicating that the residuals for a portion of our dataset is large. These values should be evaluated as potential outliers, using techniques like Cook's distance. Additionally, this could signal collinearity in the model since the linear dependence of the features could cause the magnitude of the response variable to increase significantly with slight changes to the linearly dependent variable. Cook's distance will be explored in section VII.









V. Logistic Regression Classifier for digit vs digit

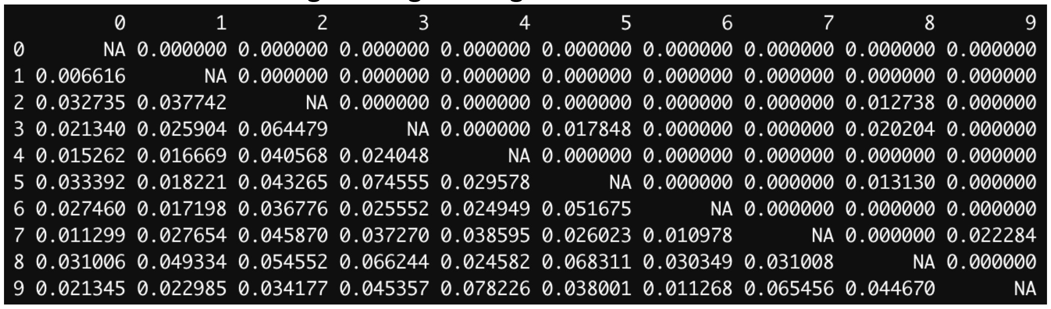
Performing logistic regression modeling on a specific digit vs another digit is performed with a similar procedure to section III, with the below differences. The complete R code is in Appendix D:

1. Create a function to perform the identification of digit 1 and digit 2, dataset train/test splitting, modeling/predictions, error rate calculations, and visualization of model coefficients.
    a. Label digit 1 as 1 and label digit 2 as 0.
2. Use loops to call the function over all pairs of digits to generate an error rate matrix.
3. Display coefficients (β) and intercepts (β_0) for the pairs with lowest and highest error rates.

With the model input being only images of digit 1 and 2, labeled as 1 and 0 respectively, the model output will be between 0 and 1. We classify outputs greater than 0.5 as digit 1 and outputs less than 0.5 as digit 2.

The error rates for both the train and test datasets of a specific digit versus another digit is shown below:

Logistic Digit vs Digit Error Rate Matrix



```
         0        1        2        3        4        5        6        7        8        9
0       NA 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
1 0.006616       NA 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
2 0.032735 0.037742       NA 0.000000 0.000000 0.000000 0.000000 0.000000 0.012738 0.000000
3 0.021340 0.025904 0.064479       NA 0.000000 0.017848 0.000000 0.000000 0.020204 0.000000
4 0.015262 0.016669 0.040568 0.024048       NA 0.000000 0.000000 0.000000 0.000000 0.000000
5 0.033392 0.018221 0.043265 0.074555 0.029578       NA 0.000000 0.000000 0.013130 0.000000
6 0.027460 0.017198 0.036776 0.025552 0.024949 0.051675       NA 0.000000 0.000000 0.000000
7 0.011299 0.027654 0.045870 0.037270 0.038595 0.026023 0.010978       NA 0.000000 0.022284
8 0.031006 0.049334 0.054552 0.066244 0.024582 0.068311 0.030349 0.031008       NA 0.000000
9 0.021345 0.022985 0.034177 0.045357 0.078226 0.038001 0.011268 0.065456 0.044670       NA
```

Upper Triangle = Error Rates for Train Dataset; Lower Triangle = Error Rates for Test Dataset

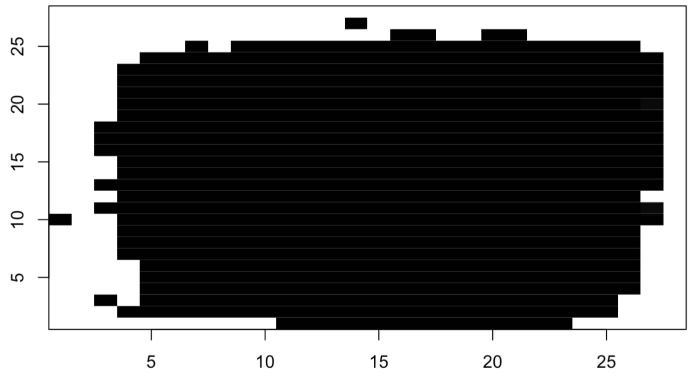
A major difference seen in the logistic error rate matrix is the presence of error rates for the train dataset being zero. In addition, the error rates for the test data are slightly higher than those of the OLS test dataset error rates. This can be attributed to the logistic model being overfitted to the training data. When a model is overfitted, it performs very well on the training data but does not generalize well. Since the test dataset contains new images, never seen by the model, and the model is overfitted to the training images, it does not perform as well in classification, when compared to the OLS model. Potential dimensionality reduction, like PCA, should be explored to reduce the number of features being trained on, thus reducing the likelihood of the model overfitting. In addition, reducing features will address multicollinearity, which occurs when model features are linearly dependent on another feature. This phenomenon can cause the model to overfit.

The highest and lowest test error rates are shown below:

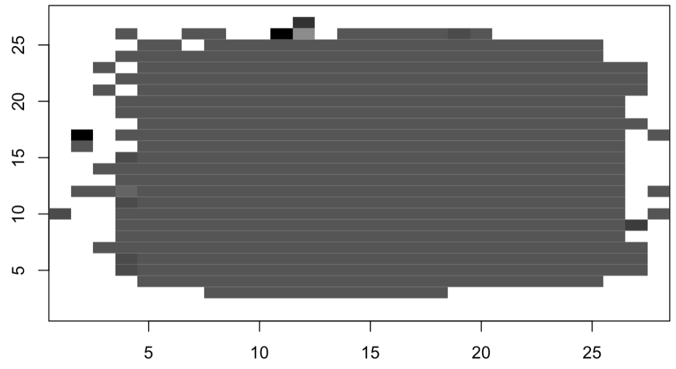
Error Rates for Digit vs Digit on Test Data	
Highest (4 and 9)	0.078226
Lowest (0 and 1)	0.006616

The highest error rate for the logistic model is between 4 and 9, rather than 3 and 5 for the OLS model. The lowest error rate for logistic is between 0 and 1, rather than 6 and 6 for OLS. This is explained by the differences in model coefficients, as well as the general differences between logistic and OLS modeling. However, we still see that higher error rates result from digits with more overlapping sections, and lower rates from digits with less overlapping sections. The coefficients for both models are shown below:

Logistic Model Coefficient (β) as Image for 4 vs 9; Intercept (β_0) = -4.373008e14



Logistic Model Coefficient (β) as Image for 0 vs 1; Intercept (β_0) = 5.706332



VI. Cook's Distance and Removing Outliers for the OLS Binary Classifier

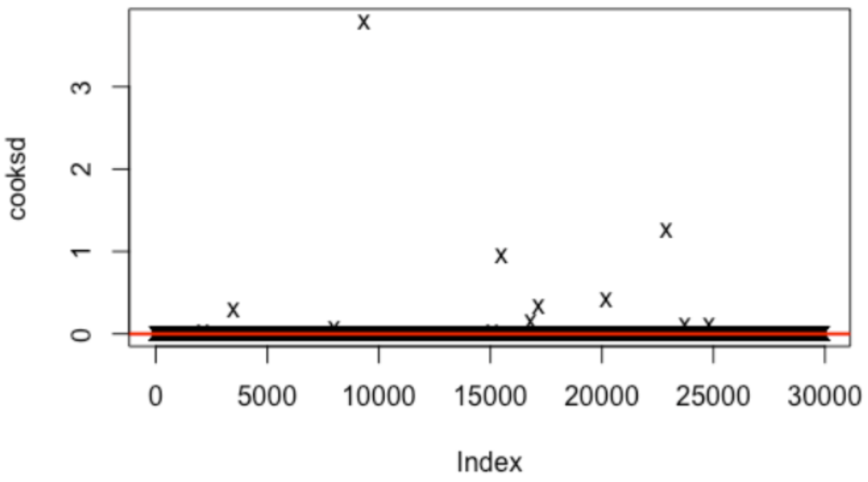
Cook's distance is used to find influential datapoints for regression and classification modeling, where the identified observations should be checked for validity and be potentially classified as outliers. For the purposes of investigating how removing datapoints with high influence affects modeling, images with a Cook's distance greater than 4 divided by the sample size will be used.

For k vs not k, the procedure for calculating Cook's distance, removing outliers, and performing modeling is summarized below, with the complete R code in Appendix E:

1. Use the cooks.distance() function on the k vs not k model to calculate Cook's distance
2. Visualize Cook's distance, with a plotted Cook's distance mean line
3. Identify the images with the highest Cook's distance, and plot
4. Remove the images with a Cook's distance greater than 4 / sample size from the training dataset
5. Visualize model coefficients, calculate error rates, and confusion matrices

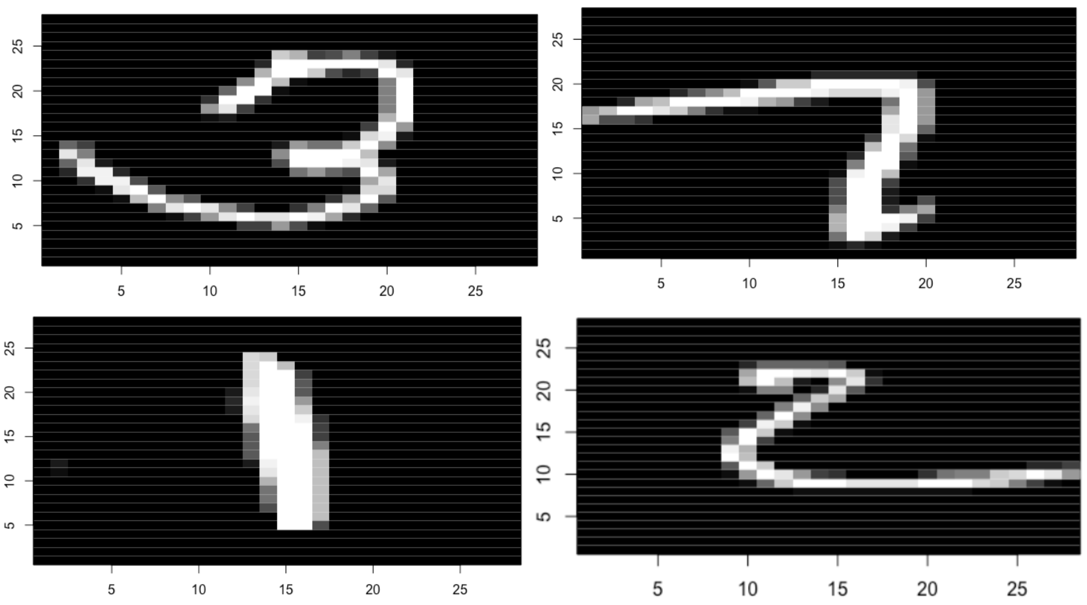
The resultant Cook's distance calculation for the k vs not k OLS model, where k = 0, is visualized in the below plot:

Cook's Distance k vs not k OLS Model, k = 0



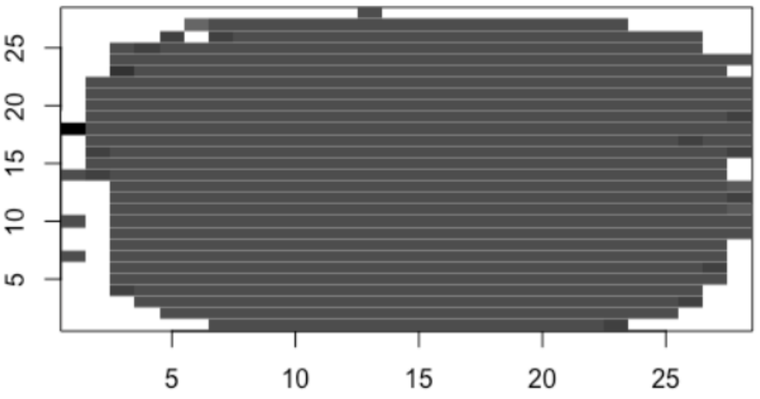
The red line marks the mean Cook's distance among all datapoints. We see that the majority of the datapoints fall on or are very close to the mean distance; however, there are a handful of points far away. Counting all points with a distance greater than 4 divided by the sample size, we get a total of 1902 images that will be removed. The top four images with high Cook's distance for this model are visualized below:

Digits with High Cook's Distance for OLS Model k vs Not k, k = 5



We can see that these digits have high influence on our model due to the abnormal lengths of the strokes or the overall width of the digit. When these digits are removed from the model training, it is expected that the model coefficients will change. This is seen in the below visualization of the new model coefficients, when the outliers are removed:

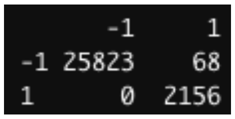
OLS Model (Outliers Removed) Coefficient (β) as Image for k = 0; Intercept (β_0) = -0.7419462



We see that the overall shape and shading of certain pixels has changed with the outliers removed from the training dataset. Additionally, the intercept (β_0) changed from -0.6809402 to -0.7419462. The classification error rates and confusion matrices are calculated below:

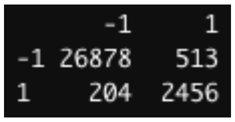
Classification Error Rate for OLS Model k = 0 (Outliers Removed)	
Error Rate on Train Dataset	0.002424502
Error Rate on Test Dataset	0.02385944

Confusion Matrix for Train Dataset



```
          -1     1
-1 25823    68
1      0  2156
```

Confusion Matrix for Test Dataset

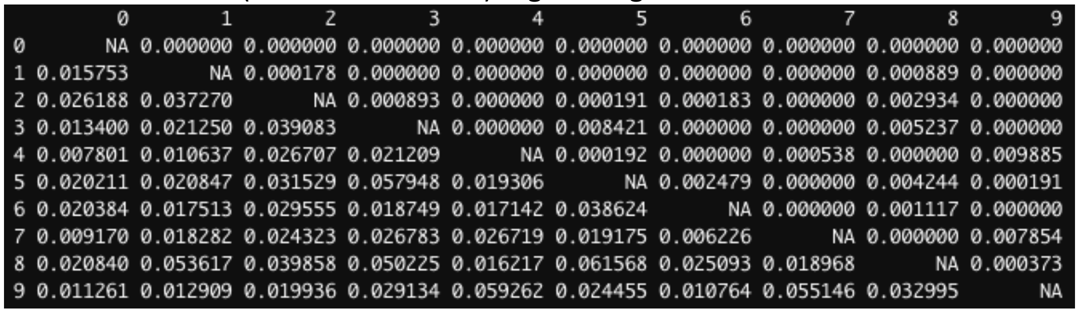


```
         -1     1
-1 26878   513
1    204  2456
```

It is observed that the error rate for the OLS model with outliers removed decreased a lot for the train dataset, but stayed relatively constant for the test dataset, when compared to the OLS model with outliers. It should be expected that the error rate will decrease, due to the model not training on digit images that are not normally written in an unusual fashion. However, since outliers are likely present in the test dataset and considerations like digits being slightly off centered are present, the performance increase of our model with the test data is not seen.

Following a similar procedure for removing outliers for OLS modeling of digit vs digit, the below error rate matrix is generated. The complete R Code is in Appendix F.

OLS (Outliers Removed) Digit vs Digit Error Rate Matrix



```
         0        1        2        3        4        5        6        7        8        9
0       NA 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
1 0.015753       NA 0.000178 0.000000 0.000000 0.000000 0.000000 0.000000 0.000889 0.000000
2 0.026188 0.037270       NA 0.000893 0.000000 0.000191 0.000183 0.000000 0.002934 0.000000
3 0.013400 0.021250 0.039083       NA 0.000000 0.008421 0.000000 0.000000 0.005237 0.000000
4 0.007801 0.010637 0.026707 0.021209       NA 0.000192 0.000000 0.000538 0.000000 0.009885
5 0.020211 0.020847 0.031529 0.057948 0.019306       NA 0.002479 0.000000 0.004244 0.000191
6 0.020384 0.017513 0.029555 0.018749 0.017142 0.038624       NA 0.000000 0.001117 0.000000
7 0.009170 0.018282 0.024323 0.026783 0.026719 0.019175 0.006226       NA 0.000000 0.007854
8 0.020840 0.053617 0.039858 0.050225 0.016217 0.061568 0.025093 0.018968       NA 0.000373
9 0.011261 0.012909 0.019936 0.029134 0.059262 0.024455 0.010764 0.055146 0.032995       NA
```

We again see that the model may be overfitted with the outliers removed, given the many zero error rates for the train dataset. A potential reason for this is due to multicollinearity, that may still exist even though columns with zero were removed as part of the pre-processing procedure. As a result, feature feature reduction should be explored. Theoretically, we would expect that both the train and test error rates should decrease, with outliers being removed from the model training. This overfitting results in slightly higher train error rates when compared to the OLS model with outliers remaining in the input.

The highest and lowest test error rates are shown below, where the lowest error rate is still between digits 6 and 7. However, the highest error rate changed from 3 and 5 to 5 and 8. This difference is explained by the changes in model coefficients and intercepts due to the removal of high influence data points.

Error Rates for Digit vs Digit on Test Data (Outliers Removed)	
Highest (5 and 8)	0.061568
Lowest (6 and 7)	0.006226

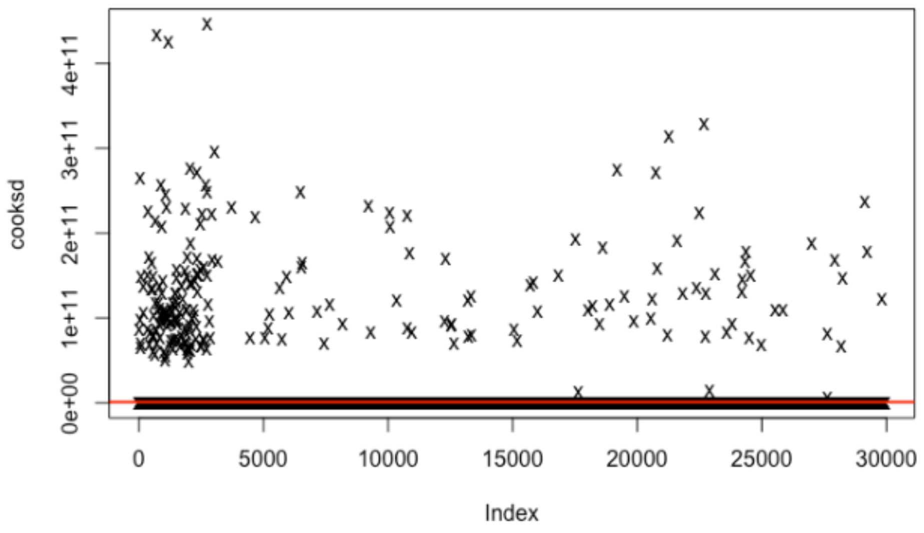
VII. Cook's Distance and Removing Outliers for the Logistic Regression Classifier

Cooks distance is similarly calculated for the logistic regression model for k vs not k, with k = 0. The process for is the same as outlined in section VI, but with the following differences. The complete R code is in Appendix G.

1. Use the cooks.distance() function on the k vs not k model to calculate Cook's distance
2. Label digit k as 1 and not digit k as 0

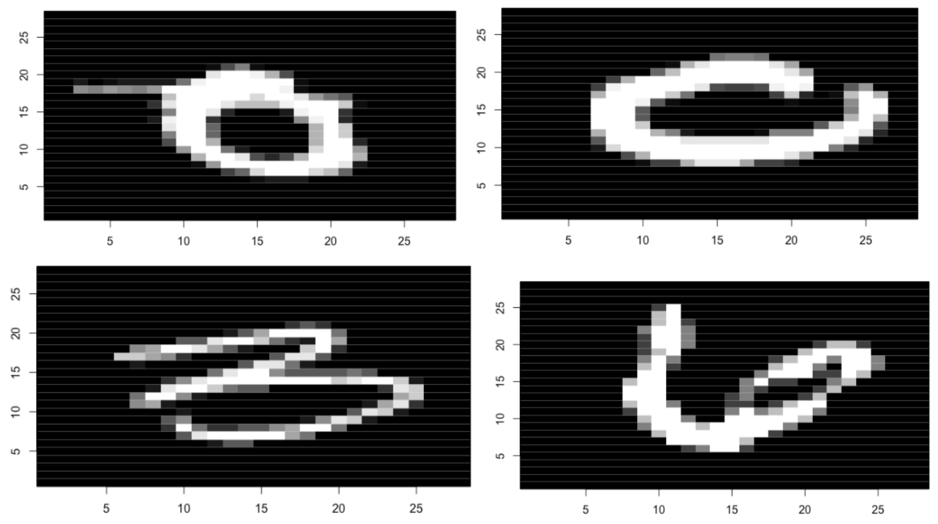
The resultant Cook's distance calculation for the k vs not k Logistic model, where k = 0, is visualized in the below plot:

Cook's Distance k vs not k Logistic Model, k = 0



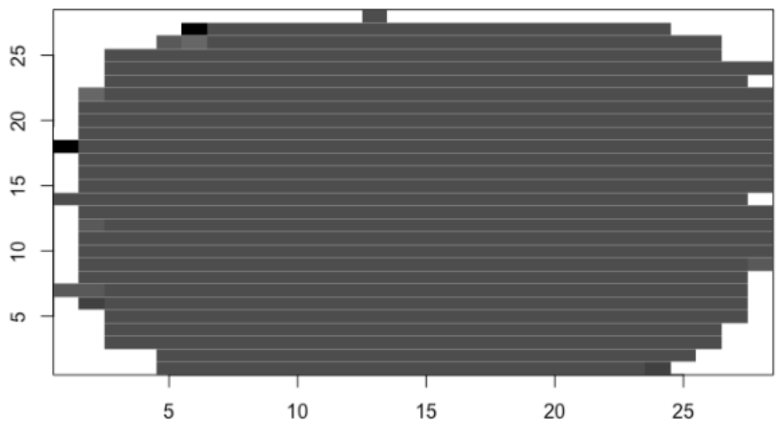
We see that there is a larger number of data points that are not clustered around the mean Cook's distance, as seen as the red horizontal line, than when compared to the OLS model. This indicates that these points should be evaluated as potential outliers. The four images with the largest Cook's distance are visualized below:

Digits with High Cook's Distance for Logistic Model k vs Not k, k = 5



Again, we visualize the model coefficient and intercept, and see differences due to the removal of the high influence points:

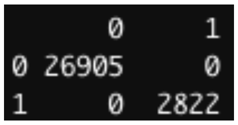
Logistics Model (Outliers Removed) Coefficient (β) as Image for k = 0; Intercept (β_0) = -56.13371



The classification error rates and confusion matrices are calculated below:

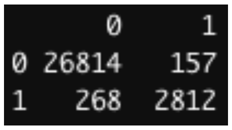
Classification Error Rate for OLS Model k = 0 (Outliers Removed)	
Error Rate on Train Dataset	0
Error Rate on Test Dataset	0.01414262

Confusion Matrix for Train Dataset



```
       0     1
0 26905     0
1     0  2822
```

Confusion Matrix for Test Dataset

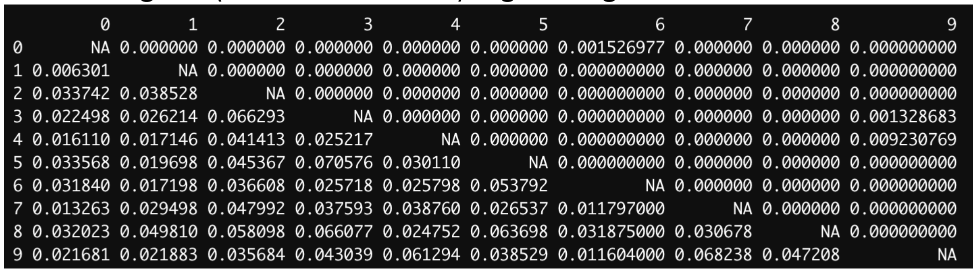


```
       0     1
0 26814   157
1   268  2812
```

We observe that the error rate for the train dataset has now decreased to zero, when compared to 0.0072 for the logistic model with outliers. The error rate for train also increases from 0.0171. This suggests potential overfitting for the model; however, it still does a good job of classifying the digits correctly.

The logistic digit vs digit analysis with outliers removed R Code is in Appendix H, and results in the below error rate matrix:

Logistic (Outliers Removed) Digit vs Digit Error Rate Matrix



```
         0        1        2        3        4        5           6        7        8           9
0       NA 0.000000 0.000000 0.000000 0.000000 0.000000 0.001526977 0.000000 0.000000 0.000000000
1 0.006301       NA 0.000000 0.000000 0.000000 0.000000 0.000000000 0.000000 0.000000 0.000000000
2 0.033742 0.038528       NA 0.000000 0.000000 0.000000 0.000000000 0.000000 0.000000 0.000000000
3 0.022498 0.026214 0.066293       NA 0.000000 0.000000 0.000000000 0.000000 0.000000 0.001328683
4 0.016110 0.017146 0.041413 0.025217       NA 0.000000 0.000000000 0.000000 0.000000 0.009230769
5 0.033568 0.019698 0.045367 0.070576 0.030110       NA 0.000000000 0.000000 0.000000 0.000000000
6 0.031840 0.017198 0.036608 0.025718 0.025798 0.053792          NA 0.000000 0.000000 0.000000000
7 0.013263 0.029498 0.047992 0.037593 0.038760 0.026537 0.011797000       NA 0.000000 0.000000000
8 0.032023 0.049810 0.058098 0.066077 0.024752 0.063698 0.031875000 0.030678       NA 0.000000000
9 0.021681 0.021883 0.035684 0.043039 0.061294 0.038529 0.011604000 0.068238 0.047208          NA
```

The results are similar to the logistic model with outliers error rate matrix, where some digit pairs have improved performance and some have decreased model performance. For example, (0,1), (1,9), and (3,9) have slightly improved error rates.

The highest and lowest test error rates are shown below, where the lowest error rate is still between digits 0 and 1. However, the highest error rate changed from 4 and 9 to 3 and 5. This difference is explained by the changes in model coefficients and intercepts due to the removal of high influence data points.

Error Rates for Digit vs Digit on Test Data (Outliers Removed)	
Highest (3 and 5)	0.070576
Lowest (0 and 1)	0.006301

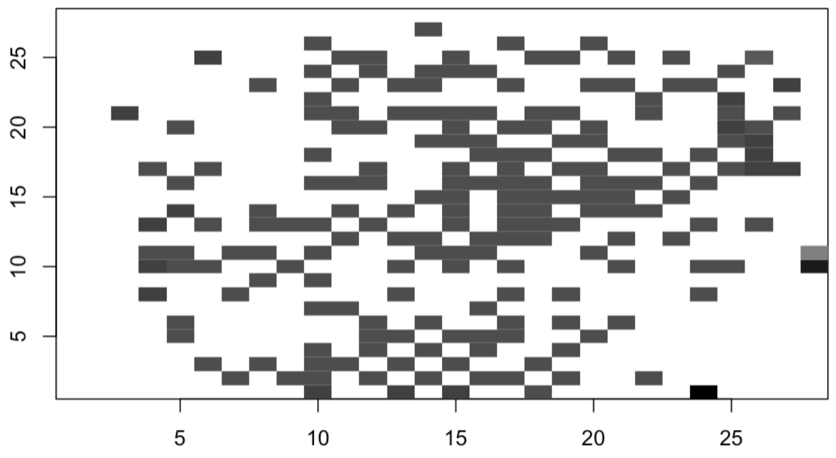
VIII. Backward Selection

Backward selection is a variable selection method that begins with a model that contains all features and then features with the least statistical significance are removed until a pre-defined number of features are left in the model. The least significant feature is determined by identifying the highest p-value among the variables.

This technique is useful for models with potentially large numbers of features, where the goal is to create as simple of a model as possible. This is due to considerations like more complex models requiring more data to train and test with and increased computational power and time. Additionally, models are more likely to be overfitted with higher number of features.

The code for backward selection is in Appendix I, where we use the regsubsets() function, defining “backward” as the selection method. When finding the model with the minimum residual sums squared, we find an RSS = 3040.008, with 202 total features. The below is the visualization of the model coefficients and intercept of the backward selected model:

Backward Selection Method: OLS Model Coefficients (β) as Image for k = 0; Intercept (β_0) = -0.609



We can clearly see a large reduction in the number of features selected for the OLS model for k vs not k, where the 202 most statistically significant features were kept in the model.

IX. Appendix

A. OLS digit k vs not k

```
#Function to load image and label files
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n*nrow*ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow*ncol, byrow = TRUE))
}

load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  L = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  L
}

#Set working directory
setwd("/Users/allenlau/Documents/CCNY/DSEI103_AppliedStatistics/DSEI103_Project2")

#Load images and corresponding labels
train_digits = load_image_file('train-images.idx3-ubyte') #dataframe of flattened images
train_Labels = load_label_file('train-labels.idx1-ubyte')

#Select digit k and re-label the dataset wrt selected digit
k = 0
is_k = which((train_Labels == k) %in% TRUE) #indices corresponding to digit k
not_k = which((train_Labels == k) %in% FALSE) #indices not corresponding to k

#Display i'th instance of selected digit in the training set:
i=104
image(1:28, 1:28, matrix(as.matrix(train_digits[is_k[i],]), nrow=28)[ , 28:1], col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Find columns in digit images where all rows are not 0
ind <- which((colSums(train_digits) > 0) %in% TRUE)
ind

#Drop columns where all rows contain zeros
train_digits <- train_digits[colSums(train_digits) > 0]
dim(train_digits)

#importing library for train/test split
library(caTools)

#set seed
set.seed(1)

#create data frames for k and not k
sample_isk <- train_digits[is_k,]
```

```r
sample_notk <- train_digits[not_k,]

#create train/test split for k
sample <- sample.split(sample_isk, SplitRatio = 0.5)
train_digits_isk <- subset(sample_isk, sample == TRUE)
test_digits_isk <- subset(sample_isk, sample == FALSE)

#create train/test split for not k
sample <- sample.split(sample_notk, SplitRatio = 0.5)
train_digits_notk <- subset(sample_notk, sample == TRUE)
test_digits_notk <- subset(sample_notk, sample == FALSE)

#Count number of data points in is_k and not_k training sets
nrow_isk <- nrow(train_digits_isk)
nrow_notk <- nrow(train_digits_notk)

#Classify 1 as digit k and -1 as not digit k
y_isk <- rep(1, nrow_isk)
y_notk <- rep(-1, nrow_notk)

#Actual response data frame
y <- data.frame(c(y_isk, y_notk))

#Combined is k and not k data frame for modeling
X <- rbind(train_digits_isk, train_digits_notk)
df <- cbind(X,y)

#Ordinary least squares linear modeling
model <- lm(y$c.y_isk..y_notk. ~ ., data = X)

#Coefficients of model
model_coeff <- data.frame(model$coefficients)

#Intercept
beta_0 <- model_coeff[1,]
beta_0

#Model Coefficients
beta <- model_coeff[-1,]
beta

#Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
beta_img = rep(NA, 784)
ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

for (i in 1:length(beta)){
  beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
}

#Visualization of Beta as a 28x28 image
image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Classification error rate and confusion matrices the training data set

#Model predictions on training data set X
model_pred_train <- data.frame(predict(model,X))
```

```
#Train data set pre-processing for error rate calculation
pred_actual_train <- cbind(model_pred_train, y)
pred_actual_train["predicted"] <- pred_actual_train["predict.model..X."]
pred_actual_train$predicted[pred_actual_train$predicted <= 0] <- -1
pred_actual_train$predicted[pred_actual_train$predicted > 0] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_train <- sum(pred_actual_train$c.y_isk..y_notk. != pred_actual_train$predicted) / nrow(pred_actual_train)
error_rate_train

#Classification error rate and confusion matrices the test data set

#Test data set pre-processing for test predictions
X_test <- rbind(test_digits_isk, test_digits_notk)

y_isk_test <- rep(1, nrow(test_digits_isk))
y_notk_test <- rep(-1, nrow(test_digits_notk))
y_test <- data.frame(c(y_isk_test, y_notk_test))

#Model predictions on test data set X
model_pred_test <- data.frame(predict(model,X_test))

#Test data set pre-processing for error rate calculation
pred_actual_test <- cbind(model_pred_test, y_test)
pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test."]
pred_actual_test$predicted[pred_actual_test$predicted <= 0] <- -1
pred_actual_test$predicted[pred_actual_test$predicted > 0] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_test <- sum(pred_actual_test$c.y_isk_test..y_notk_test. != pred_actual_test$predicted) /
nrow(pred_actual_test)
error_rate_test

#Confusion matrices for training data set
outcomes_train <- table(pred_actual_train$predicted, pred_actual_train$c.y_isk..y_notk.)
outcomes_train

#Confusion matrices for test data set
outcomes_test <- table(pred_actual_test$predicted, pred_actual_test$c.y_isk_test..y_notk_test.)
outcomes_test

#Plots for Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage
plot(model, which = c(1,2,3,5))
```

B. OLS digit vs digit

```
lmDigits <- function(digit1, digit2, ind = NULL, betaimg = FALSE){
  set.seed(1)

  #Select digit k and re-label the data set wrt selected digit
  is_digit1 = which((train_Labels == digit1) %in% TRUE) #indices corresponding to digit1
  is_digit2 = which((train_Labels == digit2) %in% TRUE) #indices corresponding to digit2

  #create data frames for digit1 and digit2
  sample_isdigit1 <- train_digits[is_digit1,]
  sample_isdigit2 <- train_digits[is_digit2,]
```

```
 #create train/test split for digit1
 sample <- sample.split(sample_isdigit1, SplitRatio = 0.5)
 train_digits1 <- subset(sample_isdigit1, sample == TRUE)
 test_digits1<- subset(sample_isdigit1, sample == FALSE)

 #create train/test split for digit2
 sample <- sample.split(sample_isdigit2, SplitRatio = 0.5)
 train_digits2 <- subset(sample_isdigit2, sample == TRUE)
 test_digits2 <- subset(sample_isdigit2, sample == FALSE)

 #Classify 1 as digit 1 and -1 as digit 2
 y_isdigit1 <- rep(1, nrow(train_digits1))
 y_isdigit2 <- rep(-1, nrow(train_digits2))

 #Actual response data frame
 y <- data.frame(c(y_isdigit1, y_isdigit2))

 #Combined is k and not k data frame for modeling
 X <- rbind(train_digits1, train_digits2)

 #Ordinary least squares linear modeling
 model <- lm(unlist(y) ~ ., data = X)

 #Classification error rate and confusion matrices the training data set
 #Model predictions on training data set X
 model_pred_train <- data.frame(suppressWarnings(predict(model,X))) #Suppress warnings for outputs
 colnames(model_pred_train)[1] <- "predict.model..X."

 #Train data set pre-processing for error rate calculation
 pred_actual_train <- cbind(model_pred_train, y)
 pred_actual_train["predicted"] <- pred_actual_train["predict.model..X."]
 pred_actual_train$predicted[pred_actual_train$predicted <= 0] <- -1
 pred_actual_train$predicted[pred_actual_train$predicted > 0] <- 1

 #Classification error rate = incorrectly classified / total number of objects
 error_rate_train <- round(sum(pred_actual_train$c.y_isdigit1..y_isdigit2. != pred_actual_train$predicted) /
nrow(pred_actual_train), digits = 6)
 #cat("\n","Error Rate (Train): ", error_rate_train, "\n")

 #Classification error rate and confusion matrices the test data set
 #Test data set pre-processing for test predictions
 X_test <- rbind(test_digits1, test_digits2)

 y_digit1_test <- rep(1, nrow(test_digits1))
 y_digit2_test <- rep(-1, nrow(test_digits2))
 y_test <- data.frame(c(y_digit1_test, y_digit2_test))

 #Model predictions on test data set X
 model_pred_test <- data.frame(suppressWarnings(predict(model,X_test))) #Suppress warnings for outputs
 colnames(model_pred_test)[1] <- "predict.model..X_test."

 #Test data set pre-processing for error rate calculation
 pred_actual_test <- cbind(model_pred_test, y_test)
 pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test."]
 pred_actual_test$predicted[pred_actual_test$predicted <= 0] <- -1
 pred_actual_test$predicted[pred_actual_test$predicted > 0] <- 1
```

```
  #Classification error rate = incorrectly classified / total number of objects
  error_rate_test <- round(sum(pred_actual_test$c.y_digit1_test..y_digit2_test. != pred_actual_test$predicted) /
nrow(pred_actual_test), digits = 6)
  #cat("\n","Error Rate (Test): ", error_rate_test, "\n")

  #Visualization of Beta as a 28x28 image
  if(betaimg){
    #Model Coefficients
    beta <- data.frame(model$coefficients)[-1,]
    beta_0 <- data.frame(model$coefficients)[1,]

    #Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
    beta_img = rep(NA, 784)
    ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

    for (i in 1:length(beta)){
      beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
    }

    image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
    return(beta_0)
  }


  return(list(error_rate_train, error_rate_test))
  #plot(model, which = c(1,2,3,5))
}

#Create empty matrix, and label rows and columns as digits
error_rate_matrix <- matrix(nrow = 10, ncol = 10)
matrix(error_rate_matrix)
rownames(error_rate_matrix) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
colnames(error_rate_matrix) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

#Compute error rates for all pairs of digits and populate matrix
#Matrix contains train error rates in upper triangle and test error rates in lower triangle
for (digit1 in 0:9){
  for (digit2 in 0:9){
    if (digit2>digit1){
      results = lmDigits(digit1, digit2)
      error_rate_matrix[digit1+1,digit2+1] = results[[1]]
      error_rate_matrix[digit2+1,digit1+1] = results[[2]]
    }
  }
}

#Display Error Rate Matrix
#Upper Triangle = Train Error Rates
#Lower Triangle = Test Error Rates
error_rate_matrix

#Create test_error_rate_matrix for finding min/max error rates
test_error_rate_matrix <- error_rate_matrix
test_error_rate_matrix[upper.tri(test_error_rate_matrix)] <- NA

#Find max error rate
max_error <- max(apply(test_error_rate_matrix[2:10,], 1, max, na.rm = TRUE))
```

```
max_error_ind <- which(test_error_rate_matrix == max_error, arr.ind = TRUE)
cat("Highest Error Rate is", max_error, "for digits", max_error_ind[1] - 1, "and", max_error_ind[2] - 1)

#Find min error rate
min_error <- min(apply(test_error_rate_matrix[2:10,], 1, min, na.rm = TRUE))
min_error_ind <- which(test_error_rate_matrix == min_error, arr.ind = TRUE)
cat("Lowest Error Rate is", min_error, "for digits", min_error_ind[1] - 1, "and", min_error_ind[2] - 1)

#Beta for max error rate model
lmDigits(5,3,ind,betaimg = TRUE)

#Beta for min error rate model
lmDigits(6,7,ind,betaimg = TRUE)
```

C. Logistic digit k vs not k

```
#Function to load image and label files
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n*nrow*ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow*ncol, byrow = TRUE))
}

load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  L = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  L
}

#Set working directory
setwd("/Users/allenlau/Documents/CCNY/DSEI103_AppliedStatistics/DSEI103_Project2")

#Load images and corresponding labels
train_digits = load_image_file('train-images.idx3-ubyte') #dataframe of flattened images
train_Labels = load_label_file('train-labels.idx1-ubyte')

#Select digit k and re-label the dataset wrt selected digit
k = 0
is_k = which((train_Labels == k) %in% TRUE) #indices corresponding to digit k
not_k = which((train_Labels == k) %in% FALSE) #indices not corresponding to k

#Display i'th instance of selected digit in the training set:
i=104
image(1:28, 1:28, matrix(as.matrix(train_digits[is_k[i],]), nrow=28)[ , 28:1],
    col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Find columns in digit images where all rows are not 0
ind <- which((colSums(train_digits) > 0) %in% TRUE)
ind
```

```
#Drop columns where all rows contain zeros
train_digits <- train_digits[colSums(train_digits) > 0]
dim(train_digits)

#importing libary for train/test split
library(caTools)

#set seed
set.seed(1)

#create data frames for k and not k
sample_isk <- train_digits[is_k,]
sample_notk <- train_digits[not_k,]

#create train/test split for k
sample <- sample.split(sample_isk, SplitRatio = 0.5)
train_digits_isk <- subset(sample_isk, sample == TRUE)
test_digits_isk <- subset(sample_isk, sample == FALSE)

#create train/test split for not k
sample <- sample.split(sample_notk, SplitRatio = 0.5)
train_digits_notk <- subset(sample_notk, sample == TRUE)
test_digits_notk <- subset(sample_notk, sample == FALSE)

#Count number of data points in is_k and not_k training sets
nrow_isk <- nrow(train_digits_isk)
nrow_notk <- nrow(train_digits_notk)

#Classify 1 as digit k and 0 as not digit k
y_isk <- rep(1, nrow_isk)
y_notk <- rep(0, nrow_notk)

#Actual response data frame
y <- data.frame(c(y_isk, y_notk))

#Combined is k and not k data frame for modeling
X <- rbind(train_digits_isk, train_digits_notk)

#Ordinary least squares linear modeling
model <- glm(y$c.y_isk..y_notk. ~ ., data = X, family = "binomial")

#Coefficients of model
model_coeff <- data.frame(model$coefficients)

#Intercept
beta_0 <- model_coeff[1,]
beta_0

#Model Coefficients
beta <- model_coeff[-1,]
beta

#Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
beta_img = rep(NA, 784)
ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

for (i in 1:length(beta)){
  beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
}

#Visualization of Beta as a 28x28 image
image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
   col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Classification error rate and confusion matrices the training data set

#Model predictions on training data set X
model_pred_train <- data.frame(predict(model,X, type = "response"))

#Train data set pre-processing for error rate calculation
pred_actual_train <- cbind(model_pred_train, y)
pred_actual_train["predicted"] <- pred_actual_train["predict.model..X..type....response.."]
pred_actual_train$predicted[pred_actual_train$predicted < .5] <- 0
pred_actual_train$predicted[pred_actual_train$predicted > .5] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_train <- sum(pred_actual_train$c.y_isk..y_notk. != pred_actual_train$predicted) / nrow(pred_actual_train)
error_rate_train

#Classification error rate and confusion matrices the test data set

#Test data set pre-processing for test predictions
X_test <- rbind(test_digits_isk, test_digits_notk)

y_isk_test <- rep(1, nrow(test_digits_isk))
y_notk_test <- rep(0, nrow(test_digits_notk))
y_test <- data.frame(c(y_isk_test, y_notk_test))

#Model predictions on test data set X
model_pred_test <- data.frame(predict(model,X_test,type="response"))

#Test data set pre-processing for error rate calculation
pred_actual_test <- cbind(model_pred_test, y_test)
pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test..type....response.."]
pred_actual_test$predicted[pred_actual_test$predicted < .5] <- 0
pred_actual_test$predicted[pred_actual_test$predicted > .5] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_test <- sum(pred_actual_test$c.y_isk_test..y_notk_test. != pred_actual_test$predicted) /
nrow(pred_actual_test)
error_rate_test

#Confusion matrices for training data set
outcomes_train <- table(pred_actual_train$predicted, pred_actual_train$c.y_isk..y_notk.)
outcomes_train

#Confusion matrices for test data set
outcomes_test <- table(pred_actual_test$predicted, pred_actual_test$c.y_isk_test..y_notk_test.)
outcomes_test

#Plots for Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage
plot(model, which = c(1,2,3,5))
```

D. Logistic Digit vs Digit

```
logDigits <- function(digit1, digit2, ind = NULL, betaimg = FALSE){
  set.seed(1)
```

```
 #Select digit k and re-label the data set wrt selected digit
 is_digit1 = which((train_Labels == digit1) %in% TRUE) #indices corresponding to digit1
 is_digit2 = which((train_Labels == digit2) %in% TRUE) #indices corresponding to digit2

 #create data frames for digit1 and digit2
 sample_isdigit1 <- train_digits[is_digit1,]
 sample_isdigit2 <- train_digits[is_digit2,]

 #create train/test split for digit1
 sample <- sample.split(sample_isdigit1, SplitRatio = 0.5)
 train_digits1 <- subset(sample_isdigit1, sample == TRUE)
 test_digits1<- subset(sample_isdigit1, sample == FALSE)

 #create train/test split for digit2
 sample <- sample.split(sample_isdigit2, SplitRatio = 0.5)
 train_digits2 <- subset(sample_isdigit2, sample == TRUE)
 test_digits2 <- subset(sample_isdigit2, sample == FALSE)

 #Classify 1 as digit 1 and -1 as digit 2
 y_isdigit1 <- rep(1, nrow(train_digits1))
 y_isdigit2 <- rep(0, nrow(train_digits2))

 #Actual response data frame
 y <- data.frame(c(y_isdigit1, y_isdigit2))

 #Combined is k and not k data frame for modeling
 X <- rbind(train_digits1, train_digits2)

 #Ordinary least squares linear modeling
 model <- suppressWarnings(glm(y$c.y_isdigit1..y_isdigit2. ~ ., data = X, family = "binomial"))

 #Classification error rate and confusion matrices the training data set
 #Model predictions on training data set X
 model_pred_train <- data.frame(suppressWarnings(predict(model,X,type = "response"))) #Suppress warnings for outputs
 colnames(model_pred_train)[1] <- "predict.model..X..type....response.."

 #Train data set pre-processing for error rate calculation
 pred_actual_train <- cbind(model_pred_train, y)
 pred_actual_train["predicted"] <- pred_actual_train["predict.model..X..type....response.."]
 pred_actual_train$predicted[pred_actual_train$predicted < .5] <- 0
 pred_actual_train$predicted[pred_actual_train$predicted > .5] <- 1

 #Classification error rate = incorrectly classified / total number of objects
 error_rate_train <- round(sum(pred_actual_train$c.y_isdigit1..y_isdigit2. != pred_actual_train$predicted) /
nrow(pred_actual_train), digits = 6)
 #cat("\n","Error Rate (Train): ", error_rate_train, "\n")

 #Classification error rate and confusion matrices the test data set
 #Test data set pre-processing for test predictions
 X_test <- rbind(test_digits1, test_digits2)

 y_digit1_test <- rep(1, nrow(test_digits1))
 y_digit2_test <- rep(0, nrow(test_digits2))
 y_test <- data.frame(c(y_digit1_test, y_digit2_test))

 #Model predictions on test data set X
```

```
  model_pred_test <- data.frame(suppressWarnings(predict(model,X_test, type = "response"))) #Suppress warnings for
outputs
  colnames(model_pred_test)[1] <- "predict.model..X_test..type....response.."

  #Test data set pre-processing for error rate calculation
  pred_actual_test <- cbind(model_pred_test, y_test)
  pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test..type....response.."]
  pred_actual_test$predicted[pred_actual_test$predicted < .5] <- 0
  pred_actual_test$predicted[pred_actual_test$predicted > .5] <- 1

  #Classification error rate = incorrectly classified / total number of objects
  error_rate_test <- round(sum(pred_actual_test$c.y_digit1_test..y_digit2_test. != pred_actual_test$predicted) /
nrow(pred_actual_test), digits = 6)
  #cat("\n","Error Rate (Test): ", error_rate_test, "\n")

  #Visualization of Beta as a 28x28 image
  if(betaimg){
    #Model Coefficients
    beta <- data.frame(model$coefficients)[-1,]
    beta_0 <- data.frame(model$coefficients)[1,]

    #Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
    beta_img = rep(NA, 784)
    ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

    for (i in 1:length(beta)){
      beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
    }

    image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
    return(beta_0)
  }

  return(list(error_rate_train, error_rate_test))
  #plot(model, which = c(1,2,3,5))
}

#Create empty matrix, and label rows and columns as digits
error_rate_matrix <- matrix(nrow = 10, ncol = 10)
matrix(error_rate_matrix)
rownames(error_rate_matrix) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
colnames(error_rate_matrix) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

#Compute error rates for all pairs of digits and populate matrix
#Matrix contains train error rates in upper triangle and test error rates in lower triangle
for (digit1 in 0:9){
  for (digit2 in 0:9){
    if (digit2>digit1){
      results = logDigits(digit1, digit2)
      error_rate_matrix[digit1+1,digit2+1] = results[[1]]
      error_rate_matrix[digit2+1,digit1+1] = results[[2]]
    }
  }
}

#Display Error Rate Matrix
#Upper Triangle = Train Error Rates
```

```
#Lower Triangle = Test Error Rates
error_rate_matrix

#Create test_error_rate_matrix for finding min/max error rates
test_error_rate_matrix <- error_rate_matrix
test_error_rate_matrix[upper.tri(test_error_rate_matrix)] <- NA

#Find max error rate
max_error <- max(apply(test_error_rate_matrix[2:10,], 1, max, na.rm = TRUE))
max_error_ind <- which(test_error_rate_matrix == max_error, arr.ind = TRUE)
cat("Highest Error Rate is", max_error, "for digits", max_error_ind[1] - 1, "and", max_error_ind[2] - 1)

#Find min error rate
min_error <- min(apply(test_error_rate_matrix[2:10,], 1, min, na.rm = TRUE))
min_error_ind <- which(test_error_rate_matrix == min_error, arr.ind = TRUE)
cat("Lowest Error Rate is", min_error, "for digits", min_error_ind[1] - 1, "and", min_error_ind[2] - 1)

#Beta for max error rate model
logDigits(9,4,ind,betaimg = TRUE)

#Beta for min error rate model
logDigits(1,0,ind,betaimg = TRUE)
```

E. Cook's Distance and Removing Outliers for the OLS Binary Classifier k vs not k

```
#Calculate Cook's Distance
cooksd <- cooks.distance(model)
cooksd <- na.omit(cooksd)

#Plot Cook's Distance, with Cook's Distance Mean plotted in Red
sample_size <- nrow(X)
plot(cooksd, pch = "x", cex = 1, main = "Cook's Distance")
abline(h = mean(cooksd), col = "red", lwd = 2)

#Number of Outliers based on Cook's Distance > 4 / sample size
sum(cooksd>(4/sample_size))

#Visualize Outliers
outliers <- data.frame(cooksd[cooksd>(4/sample_size)])
outliers <- cbind(index = rownames(outliers), outliers)
outliers <- outliers[order(outliers[,2], decreasing = TRUE),]


#Visualize top 5 most influential data points
train_digits_outliers <- load_image_file('train-images.idx3-ubyte')
for (i in 1:5){
  image(1:28, 1:28, matrix(as.matrix(train_digits_outliers[row.names(train_digits_outliers) == outliers$index[i],]), nrow=28)[ ,
28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
}

#Remove identified outliers from train
reduced_train_digits <- df[!(row.names(df) %in% outliers$index),]

#Ordinary least squares linear modeling
model <- lm(c.y_isk..y_notk. ~ ., data = reduced_train_digits)

#Coefficients of model
model_coeff <- data.frame(model$coefficients)
```

```
#Intercept
beta_0 <- model_coeff[1,]
beta_0

#Model Coefficients
beta <- model_coeff[-1,]
beta

#Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
beta_img = rep(NA, 784)
ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

for (i in 1:length(beta)){
  beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
}

#Visualization of Beta as a 28x28 image
image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Classification error rate and confusion matrices the training data set

#Model predictions on training data set X
model_pred_train <- data.frame(predict(model,reduced_train_digits[,1:(length(reduced_train_digits)-1)]))

#Train data set pre-processing for error rate calculation
pred_actual_train <- cbind(model_pred_train,reduced_train_digits[,length(reduced_train_digits)])
colnames(pred_actual_train)[1] <- "predicted"
colnames(pred_actual_train)[2] <- "label"
pred_actual_train$predicted[pred_actual_train$predicted < 0] <- -1
pred_actual_train$predicted[pred_actual_train$predicted > 0] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_train <- sum(pred_actual_train$label != pred_actual_train$predicted) / nrow(pred_actual_train)
error_rate_train

#Classification error rate and confusion matrices the test data set

#Model predictions on test data set X
model_pred_test <- data.frame(predict(model,X_test))

#Test data set pre-processing for error rate calculation
pred_actual_test <- cbind(model_pred_test, y_test)
pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test."]
pred_actual_test$predicted[pred_actual_test$predicted < 0] <- -1
pred_actual_test$predicted[pred_actual_test$predicted > 0] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_test <- sum(pred_actual_test$c.y_isk_test..y_notk_test. != pred_actual_test$predicted) / nrow(pred_actual_test)
error_rate_test

#Confusion matrices for training data set
outcomes_train <- table(pred_actual_train$predicted, pred_actual_train$label)
outcomes_train

#Confusion matrices for test data set
outcomes_test <- table(pred_actual_test$predicted, pred_actual_test$c.y_isk_test..y_notk_test.)
```

```
outcomes_test
```

F. Cook's Distance and Removing Outliers for the OLS Binary Classifier digit vs digit

```
lmDigitsCooksD <- function(digit1, digit2, ind = NULL, betaimg = FALSE){
  set.seed(1)

  #Select digit k and re-label the data set wrt selected digit
  is_digit1 = which((train_Labels == digit1) %in% TRUE) #indices corresponding to digit1
  is_digit2 = which((train_Labels == digit2) %in% TRUE) #indices corresponding to digit2

  #create data frames for digit1 and digit2
  sample_isdigit1 <- train_digits[is_digit1,]
  sample_isdigit2 <- train_digits[is_digit2,]

  #create train/test split for digit1
  sample <- sample.split(sample_isdigit1, SplitRatio = 0.5)
  train_digits1 <- subset(sample_isdigit1, sample == TRUE)
  test_digits1<- subset(sample_isdigit1, sample == FALSE)

  #create train/test split for digit2
  sample <- sample.split(sample_isdigit2, SplitRatio = 0.5)
  train_digits2 <- subset(sample_isdigit2, sample == TRUE)
  test_digits2 <- subset(sample_isdigit2, sample == FALSE)

  #Classify 1 as digit 1 and -1 as digit 2
  y_isdigit1 <- rep(1, nrow(train_digits1))
  y_isdigit2 <- rep(-1, nrow(train_digits2))

  #Actual response data frame
  y <- data.frame(c(y_isdigit1, y_isdigit2))

  #Combined is k and not k data frame for modeling
  X <- rbind(train_digits1, train_digits2)
  df <- cbind(X,y)

  #Ordinary least squares linear modeling
  model <- lm(unlist(y) ~ ., data = X)

  #Calculate Cook's Distance
  cooksd <- cooks.distance(model)
  cooksd <- na.omit(cooksd)
  sample_size <- nrow(X)

  #Outliers
  outliers <- data.frame(cooksd[cooksd>(4/sample_size)])
  outliers <- cbind(index = rownames(outliers), outliers)
  outliers <- outliers[order(outliers[,2], decreasing = TRUE),]

  #Remove identified outliers from train
  reduced_train_digits <- df[!(row.names(df) %in% outliers$index),]

  #Ordinary least squares linear modeling
  model <- lm(c.y_isdigit1..y_isdigit2. ~ ., data = reduced_train_digits)

  #Classification error rate and confusion matrices the training data set
  #Model predictions on training data set X
  model_pred_train <- data.frame(predict(model,reduced_train_digits[,1:(length(reduced_train_digits)-1)]))
  #Train data set pre-processing for error rate calculation
  pred_actual_train <- cbind(model_pred_train,reduced_train_digits[,length(reduced_train_digits)])
  colnames(pred_actual_train)[1] <- "predicted"
  colnames(pred_actual_train)[2] <- "label"
  pred_actual_train$predicted[pred_actual_train$predicted < 0] <- -1
  pred_actual_train$predicted[pred_actual_train$predicted > 0] <- 1

  #Classification error rate = incorrectly classified / total number of objects
  error_rate_train <- round(sum(pred_actual_train$label != pred_actual_train$predicted) / nrow(pred_actual_train), digits =
6)

  #Classification error rate and confusion matrices the test data set
  #Test data set pre-processing for test predictions
  X_test <- rbind(test_digits1, test_digits2)

  y_digit1_test <- rep(1, nrow(test_digits1))
  y_digit2_test <- rep(-1, nrow(test_digits2))
  y_test <- data.frame(c(y_digit1_test, y_digit2_test))

  #Model predictions on test data set X
  model_pred_test <- data.frame(suppressWarnings(predict(model,X_test))) #Suppress warnings for outputs
  colnames(model_pred_test)[1] <- "predict.model..X_test."

  #Test data set pre-processing for error rate calculation
  pred_actual_test <- cbind(model_pred_test, y_test)
  pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test."]
  pred_actual_test$predicted[pred_actual_test$predicted < 0] <- -1
  pred_actual_test$predicted[pred_actual_test$predicted > 0] <- 1

  #Classification error rate = incorrectly classified / total number of objects
  error_rate_test <- round(sum(pred_actual_test$c.y_digit1_test..y_digit2_test. != pred_actual_test$predicted) /
nrow(pred_actual_test), digits = 6)
  #cat("\n","Error Rate (Test): ", error_rate_test, "\n")

  #Visualization of Beta as a 28x28 image
  if(betaimg){
    #Model Coefficients
    beta <- data.frame(model$coefficients)[-1,]
    beta_0 <- data.frame(model$coefficients)[1,]

    #Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
    beta_img = rep(NA, 784)
    ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

    for (i in 1:length(beta)){
      beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
    }

    image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
    return(beta_0)
  }


  return(list(error_rate_train, error_rate_test))
  #plot(model, which = c(1,2,3,5))
}
```

```
#Create empty matrix, and label rows and columns as digits
error_rate_matrix_cooks <- matrix(nrow = 10, ncol = 10)
matrix(error_rate_matrix_cooks)
rownames(error_rate_matrix_cooks) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
colnames(error_rate_matrix_cooks) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

#Compute error rates for all pairs of digits and populate matrix
#Matrix contains train error rates in upper triangle and test error rates in lower triangle
for (digit1 in 0:9){
  for (digit2 in 0:9){
    if (digit2>digit1){
      results = lmDigitsCooksD(digit1, digit2)
      error_rate_matrix_cooks[digit1+1,digit2+1] = results[[1]]
      error_rate_matrix_cooks[digit2+1,digit1+1] = results[[2]]
    }
  }
}

#Display Error Rate Matrix
#Upper Triangle = Train Error Rates
#Lower Triangle = Test Error Rates
error_rate_matrix_cooks

#Create test_error_rate_matrix for finding min/max error rates
test_error_rate_matrix_cooks <- error_rate_matrix_cooks
test_error_rate_matrix_cooks[upper.tri(test_error_rate_matrix_cooks)] <- NA

#Find max error rate
max_error <- max(apply(test_error_rate_matrix_cooks[2:10,], 1, max, na.rm = TRUE))
max_error_ind <- which(test_error_rate_matrix_cooks == max_error, arr.ind = TRUE)
cat("Highest Error Rate is", max_error, "for digits", max_error_ind[1] - 1, "and", max_error_ind[2] - 1)

#Find min error rate
min_error <- min(apply(test_error_rate_matrix_cooks[2:10,], 1, min, na.rm = TRUE))
min_error_ind <- which(test_error_rate_matrix_cooks == min_error, arr.ind = TRUE)
cat("Lowest Error Rate is", min_error, "for digits", min_error_ind[1] - 1, "and", min_error_ind[2] - 1)
```

G. Cook’s Distance and Removing Outliers for Logistic Regression k vs not k

```
#Calculate Cook's Distance
cooksd <- cooks.distance(model)
cooksd <- na.omit(cooksd)

#Plot Cook's Distance, with Cook's Distance Mean plotted in Red
sample_size <- nrow(X)
plot(cooksd, pch = "x", cex = 1, main = "Cook's Distance")
abline(h = mean(cooksd), col = "red", lwd = 2)

#Number of Outliers based on Cook's Distance > 4 / sample size
sum(cooksd>(4/sample_size))

#Visualize Outliers
outliers <- data.frame(cooksd[cooksd>(4/sample_size)])
outliers <- cbind(index = rownames(outliers), outliers)
outliers <- outliers[order(outliers[,2], decreasing = TRUE),]


#Visualize top 5 most influential data points
train_digits_outliers <- load_image_file('train-images.idx3-ubyte')
```

```
for (i in 1:5){
  image(1:28, 1:28, matrix(as.matrix(train_digits_outliers[row.names(train_digits_outliers) == outliers$index[i],]), nrow=28)[ ,
28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
}

#Remove identified outliers from train
reduced_train_digits <- df[!(row.names(df) %in% outliers$index),]

#Logistic modeling
model <- glm(c.y_isk..y_notk. ~ ., data = reduced_train_digits, family = "binomial")

#Coefficients of model
model_coeff <- data.frame(model$coefficients)

#Intercept
beta_0 <- model_coeff[1,]
beta_0

#Model Coefficients
beta <- model_coeff[-1,]
beta

#Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
beta_img = rep(NA, 784)
ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

for (i in 1:length(beta)){
  beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
}

#Visualization of Beta as a 28x28 image
image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")

#Classification error rate and confusion matrices the training data set

#Model predictions on training data set X
model_pred_train <- data.frame(predict(model,reduced_train_digits[,1:(length(reduced_train_digits)-1)]))

#Train data set pre-processing for error rate calculation
pred_actual_train <- cbind(model_pred_train,reduced_train_digits[,length(reduced_train_digits)])
colnames(pred_actual_train)[1] <- "predicted"
colnames(pred_actual_train)[2] <- "label"
pred_actual_train$predicted[pred_actual_train$predicted < .5] <- 0
pred_actual_train$predicted[pred_actual_train$predicted > .5] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_train <- sum(pred_actual_train$label != pred_actual_train$predicted) / nrow(pred_actual_train)
error_rate_train

#Classification error rate and confusion matrices the test data set

#Model predictions on test data set X
model_pred_test <- data.frame(predict(model,X_test))

#Test data set pre-processing for error rate calculation
pred_actual_test <- cbind(model_pred_test, y_test)
```

```
pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test."]
pred_actual_test$predicted[pred_actual_test$predicted < .5] <- 0
pred_actual_test$predicted[pred_actual_test$predicted > .5] <- 1

#Classification error rate = incorrectly classified / total number of objects
error_rate_test <- sum(pred_actual_test$c.y_isk_test..y_notk_test. != pred_actual_test$predicted) /
nrow(pred_actual_test)
error_rate_test

#Confusion matrices for training data set
outcomes_train <- table(pred_actual_train$predicted, pred_actual_train$label)
outcomes_train

#Confusion matrices for test data set
outcomes_test <- table(pred_actual_test$predicted, pred_actual_test$c.y_isk_test..y_notk_test.)
outcomes_test
```

H. Cook's Distance and Removing Outliers for Logistic Regression digit vs digit

```
logDigitsCooksD <- function(digit1, digit2, ind = NULL, betaimg = FALSE){
  set.seed(1)

  #Select digit k and re-label the data set wrt selected digit
  is_digit1 = which((train_Labels == digit1) %in% TRUE) #indices corresponding to digit1
  is_digit2 = which((train_Labels == digit2) %in% TRUE) #indices corresponding to digit2

  #create data frames for digit1 and digit2
  sample_isdigit1 <- train_digits[is_digit1,]
  sample_isdigit2 <- train_digits[is_digit2,]

  #create train/test split for digit1
  sample <- sample.split(sample_isdigit1, SplitRatio = 0.5)
  train_digits1 <- subset(sample_isdigit1, sample == TRUE)
  test_digits1<- subset(sample_isdigit1, sample == FALSE)

  #create train/test split for digit2
  sample <- sample.split(sample_isdigit2, SplitRatio = 0.5)
  train_digits2 <- subset(sample_isdigit2, sample == TRUE)
  test_digits2 <- subset(sample_isdigit2, sample == FALSE)

  #Classify 1 as digit 1 and -1 as digit 2
  y_isdigit1 <- rep(1, nrow(train_digits1))
  y_isdigit2 <- rep(0, nrow(train_digits2))

  #Actual response data frame
  y <- data.frame(c(y_isdigit1, y_isdigit2))

  #Combined is k and not k data frame for modeling
  X <- rbind(train_digits1, train_digits2)
  df <- cbind(X,y)

  #Ordinary least squares linear modeling
  model <- suppressWarnings(glm(unlist(y) ~ ., data = X, family = "binomial"))

  #Calculate Cook's Distance
  cooksd <- cooks.distance(model)
  cooksd <- na.omit(cooksd)
  sample_size <- nrow(X)
  #Outliers
  outliers <- data.frame(cooksd[cooksd>(4/sample_size)])
  outliers <- cbind(index = rownames(outliers), outliers)
  outliers <- outliers[order(outliers[,2], decreasing = TRUE),]

  #Remove identified outliers from train
  reduced_train_digits <- df[!(row.names(df) %in% outliers$index),]

  #Ordinary least squares linear modeling
  model <- suppressWarnings(glm(c.y_isdigit1..y_isdigit2. ~ ., data = reduced_train_digits, family = "binomial"))

  #Classification error rate and confusion matrices the training data set

  #Model predictions on training data set X
  model_pred_train <- data.frame(suppressWarnings(predict(model,reduced_train_digits[,1:(length(reduced_train_digits)-
1)])))

  #Train data set pre-processing for error rate calculation
  pred_actual_train <- cbind(model_pred_train,reduced_train_digits[,length(reduced_train_digits)])
  colnames(pred_actual_train)[1] <- "predicted"
  colnames(pred_actual_train)[2] <- "label"
  pred_actual_train$predicted[pred_actual_train$predicted < .5] <- 0
  pred_actual_train$predicted[pred_actual_train$predicted > .5] <- 1

  #Classification error rate = incorrectly classified / total number of objects
  error_rate_train <- sum(pred_actual_train$label != pred_actual_train$predicted) / nrow(pred_actual_train)

    #Classification error rate and confusion matrices the test data set
  #Test data set pre-processing for test predictions
  X_test <- rbind(test_digits1, test_digits2)

  y_digit1_test <- rep(1, nrow(test_digits1))
  y_digit2_test <- rep(0, nrow(test_digits2))
  y_test <- data.frame(c(y_digit1_test, y_digit2_test))

  #Model predictions on test data set X
  model_pred_test <- data.frame(suppressWarnings(predict(model,X_test, type = "response"))) #Suppress warnings for
outputs
  colnames(model_pred_test)[1] <- "predict.model..X_test..type....response.."

  #Test data set pre-processing for error rate calculation
  pred_actual_test <- cbind(model_pred_test, y_test)
  pred_actual_test["predicted"] <- pred_actual_test["predict.model..X_test..type....response.."]
  pred_actual_test$predicted[pred_actual_test$predicted < .5] <- 0
  pred_actual_test$predicted[pred_actual_test$predicted > .5] <- 1

  #Classification error rate = incorrectly classified / total number of objects
  error_rate_test <- round(sum(pred_actual_test$c.y_digit1_test..y_digit2_test. != pred_actual_test$predicted) /
nrow(pred_actual_test), digits = 6)
  #cat("\n","Error Rate (Test): ", error_rate_test, "\n")

  #Visualization of Beta as a 28x28 image
  if(betaimg){
    #Model Coefficients
    beta <- data.frame(model$coefficients)[-1,]
    beta_0 <- data.frame(model$coefficients)[1,]

    #Create beta_img array such that we can still display coefficients as a 28X28 image, after removing zero features
    beta_img = rep(NA, 784)
    ind_betaCoef <- array(c(ind, beta), dim = c(length(beta),2))

    for (i in 1:length(beta)){
      beta_img[ind_betaCoef[i,1]] <- ind_betaCoef[i,2]
    }

    image(1:28, 1:28, matrix(as.matrix(beta_img), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
    return(beta_0)
  }


  return(list(error_rate_train, error_rate_test))
  #plot(model, which = c(1,2,3,5))
}

#Create empty matrix, and label rows and columns as digits
error_rate_matrix_cooks <- matrix(nrow = 10, ncol = 10)
matrix(error_rate_matrix_cooks)
rownames(error_rate_matrix_cooks) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
colnames(error_rate_matrix_cooks) <- c(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

#Compute error rates for all pairs of digits and populate matrix
#Matrix contains train error rates in upper triangle and test error rates in lower triangle
for (digit1 in 0:9){
  for (digit2 in 0:9){
    if (digit2>digit1){
      results = logDigitsCooksD(digit1, digit2)
      error_rate_matrix_cooks[digit1+1,digit2+1] = results[[1]]
      error_rate_matrix_cooks[digit2+1,digit1+1] = results[[2]]
    }
  }
}

#Display Error Rate Matrix
#Upper Triangle = Train Error Rates
#Lower Triangle = Test Error Rates
error_rate_matrix_cooks

#Create test_error_rate_matrix for finding min/max error rates
test_error_rate_matrix_cooks <- error_rate_matrix_cooks
test_error_rate_matrix_cooks[upper.tri(test_error_rate_matrix_cooks)] <- NA

#Find max error rate
max_error <- max(apply(test_error_rate_matrix_cooks[2:10,], 1, max, na.rm = TRUE))
max_error_ind <- which(test_error_rate_matrix_cooks == max_error, arr.ind = TRUE)
cat("Highest Error Rate is", max_error, "for digits", max_error_ind[1] - 1, "and", max_error_ind[2] - 1)

#Find min error rate
min_error <- min(apply(test_error_rate_matrix_cooks[2:10,], 1, min, na.rm = TRUE))
min_error_ind <- which(test_error_rate_matrix_cooks == min_error, arr.ind = TRUE)
cat("Lowest Error Rate is", min_error, "for digits", min_error_ind[1] - 1, "and", min_error_ind[2] - 1)
```

I. Backward Selection

```
#Backward Selection
library(leaps)
backward <- regsubsets(c.y_isk..y_notk. ~., data = df, nbest = 1, method = "backward", really.big = TRUE, nvmax = 200)
```

```
backward_summary <- summary(backward)

#min RSS
min(backward_summary$rss)

#Index of best performing model
which(backward_summary$rss == min(backward_summary$rss))

#Model Coefficients
beta_img_bwd = rep(NA, 784)
bwdmodel <- model$coefficients[backward_summary$which[201,]]
bwdmodel

#Number of Features Selected
length(model$coefficients[backward_summary$which[201,]])

#Convert Model Coefficients to a 28x28 maxtrix
for (i in 2:length(bwdmodel)){
  beta_img_bwd[as.numeric(substring(names(bwdmodel[i]),2))] <- bwdmodel[i]
}

#Visualization of Beta as a 28x28 image
image(1:28, 1:28, matrix(as.matrix(beta_img_bwd), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
```