MATH 6350 – HOMEWORK #2

Brian Le, Basel Najjar, Soo Jong Cho

3rd October 2021

Table of Contents

Scope

This report outlines the application of an automatic classification algorithm to digitized images of typed fonts. Four fonts were utilized in this analysis.

- Vladimir font: Aa Bb Cc 1 2 3
- Georgia font: Aa Bb Cc 1 2 3
- Comic font: Aa Bb Cc 1 2 3
- Nina font: Aa Bb Cc 1 2 3

The data for this analysis was obtained from University of California, Irvine (UCI) Machine Learning Repository. The *R* statistical software package was utilized to complete analysis and classification of the data. The source data for each font contains many digitized images ("cases") of typed characters in the respective font. Each image is 20 x 20 pixels in size and each pixel is quantized using an integer gray level (0 – 255). There are 400 gray levels ("features") for each case, with one gray level describing an individual pixel. In other words, each case is described by 400 features with each feature taking on an integer value between 0 – 255. A gray level of 0 represents pure black and a level of 255 represents pure white. Each font was assigned a class number. The fonts will be referred to by number rather name for the remainder of this report. The number of cases for each font in the raw data set is described below:

- Vladimir – Class CL1 contains 976 cases.
- Georgia – Class CL2 contains 2,450 cases.
- Comic – Class CL3 contains 2,388 cases.
- Nina – Class CL4 contains 2,278 cases.

Preliminary Treatment of Data

Several pre-treatment steps were performed on the data before any analysis took place. First, extraneous descriptors were eliminated. These descriptors do not provide useful information in this analysis and include size, orientation, and font variant. Next, boldface and italicized characters were eliminated. Only normal strength and non-italic characters will be analyzed. The number of cases for each class in the cleaned data set is described below.

- CL1 contains 244 entries (N_1).
- CL2 contains 612 entries (N_2).
- CL3 contains 597 entries (N_3).
- CL4 contains 570 entries (N_4).
- In total, there are 2,023 entries (N).

As previously discussed, there are 400 features describing each entry. Each pixel row and column position correspond to a feature value X_1 through X_{400}.

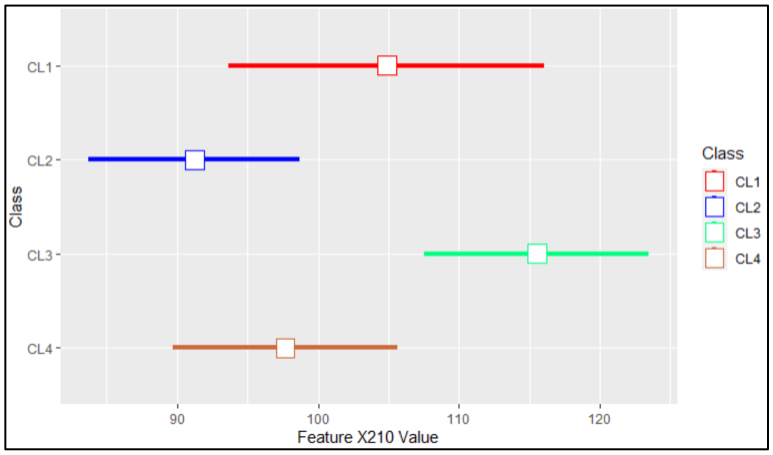
$$\begin{bmatrix} r0, c0 & \cdots & r0, c19 \\ \vdots & \ddots & \vdots \\ r19, c0 & \cdots & r19, c19 \end{bmatrix} \rightarrow \begin{bmatrix} X_1 & \cdots & X_{20} \\ \vdots & \ddots & \vdots \\ X_{381} & \cdots & X_{400} \end{bmatrix}$$

Part 1

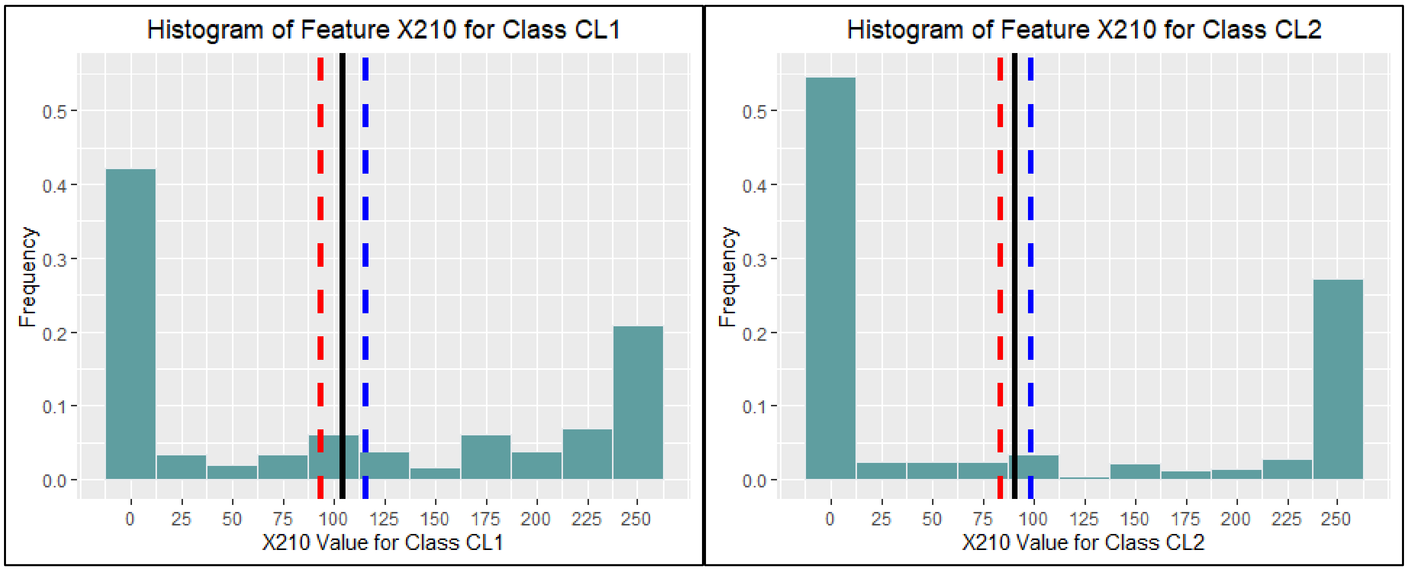
Section 1.1

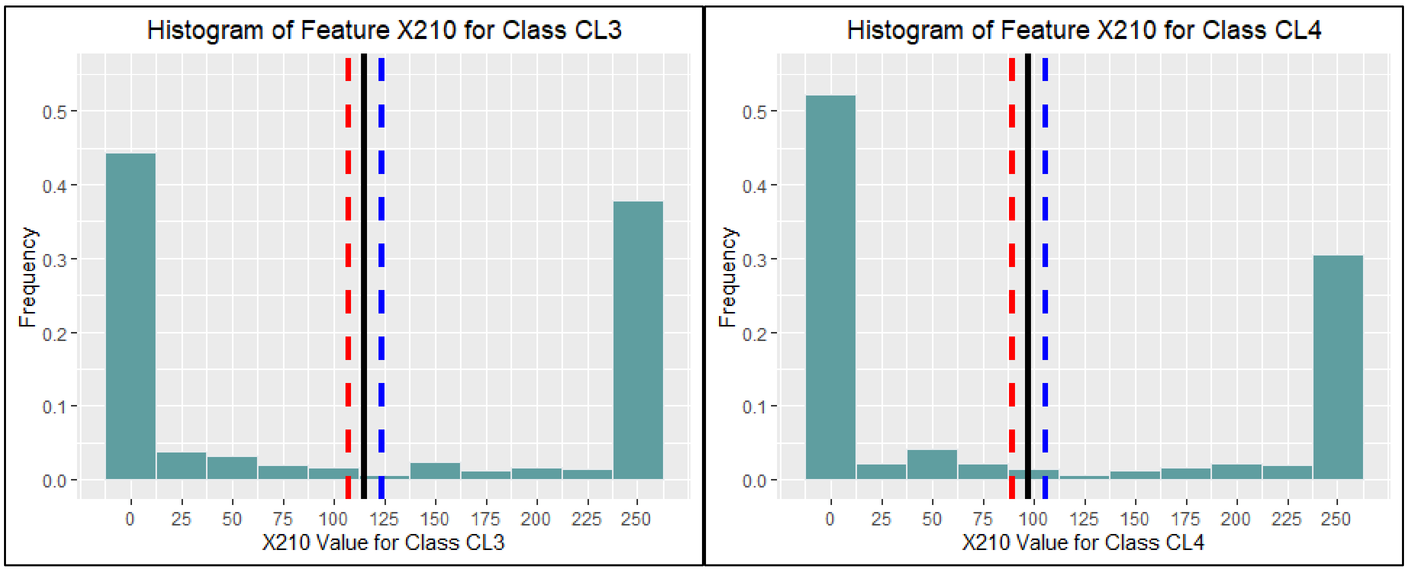
The feature selected for comparison across classes was feature X_{210}. This feature corresponds to the pixel in row 10, column 9 and is near the center of the image. The table and figure below outline the mean of feature X_{210} for each class, along with the 90% confidence interval about the mean. This confidence interval was calculated using the t critical value.

Class	Lower Interval	Mean	Upper Interval
CL1	93.6	105	116.1
CL2	83.7	91	98.7
CL3	107.5	116	123.5
CL4	89.7	98	105.6

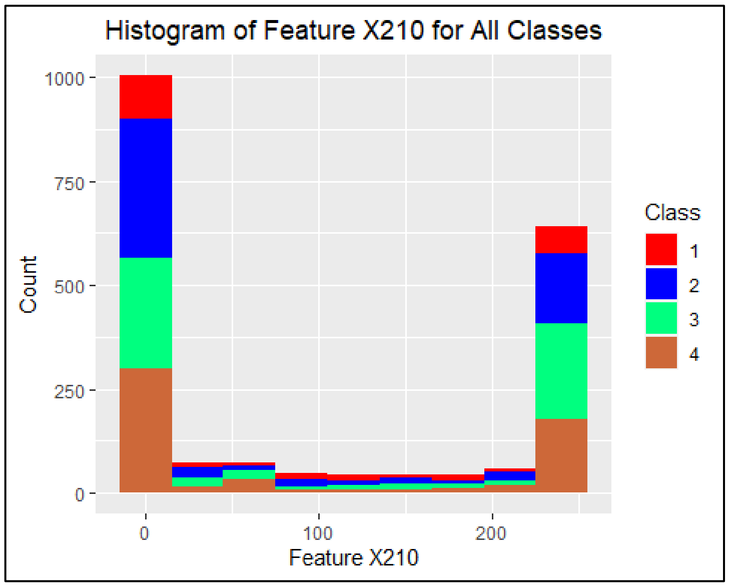


Upon first examination, the mean values of feature X_{210} across the four classes are different. However, the 90% confidence interval suggests significant intersection across the four classes. This intersection can be measured by the width of overlap in interval between two classes. The 90% confidence interval overlaps are as follows: CL1 vs. CL2: 5.1; CL1 vs. CL3: 8.6; CL1 vs. CL4: 12.0; and CL2 vs. CL4: 9.0. The mean value of feature X_{210} in CL3 is the most unique, as CL3 only overlaps CL1. CL1 is the least unique, as it overlaps with all other classes. CL1 also contains the fewest cases; this contributes to the large width of the CL1 confidence interval. Classes CL2 and CL4 share moderate intersection with other classes. It should be noted that the t-test used to calculate these confidence intervals pre-supposes that the data follow a normal distribution. The histograms of feature X_{210} show that the data is not normally distributed in any of the four classes.





The solid black line in each histogram denotes the mean value of feature X_{210} for that class. The dashed red and blue lines denote the lower and upper bounds of the 90% confidence interval, respectively. Comparison of these histograms indicates similar feature behavior across all four classes: there is a concentration of cases at the extremes and a minority of cases near the mean. Each feature X_1 through X_{400} quantify the grayness of a pixel. A feature value of 0 represents pure black, and a feature value of 255 represents pure white. Therefore, it is expected that feature X_{210} primarily takes on values of either nearly white or nearly black. In addition, the analysis of the histograms supports the same conclusion reached by comparing confidence intervals: there is significant intersection of all four classes with respect to feature X_{210}. The stacked histogram below depicts total number of cases (count) from each class with their value of feature X_{210}.



The Kolmogorov-Smirnov test (KS-test) provides a quantitative method to measure the difference between two distributions. The KS-test returns two values:

1. The maximum vertical distance between two cumulative distribution functions (D-value). The closer the D-value is to zero, the more similar the two distributions are. Within the context of automatic classification, a large D-value indicates good discriminating power of a given feature between two classes.
2. The probability that a case observed in the population is at least as extreme as the cases observed in the sample (p-value). The p-value is not unique to the KS-test and is commonly used in hypothesis testing. Comparing the p-value to a selected significance level, α, we can reject or fail to reject the null hypothesis that the two distributions are not different.

KS-test distances (D-value) of feature X_{210} for each of the four classes is outlined below:

D-Values	CL1	CL2	CL3	CL4
CL1	0	0.13	0.19	0.12
CL2	0.13	0	0.11	0.04
CL3	0.19	0.11	0	0.09
CL4	0.12	0.04	0.09	0

p-Values	CL1	CL2	CL3	CL4
CL1	1	0.007	<0.0005	0.012
CL2	0.007	1	0.001	0.7
CL3	<0.0005	0.001	1	0.019
CL4	0.012	0.7	0.019	1

The D-value represents the maximum vertical distance between the cumulative distribution functions $F_i(X)$ and $F_j(X)$ for given feature variables X_i and X_j. In mathematical notation:

$$D = \max_X \left[F_i(X) - F_j(X)\right]$$

The D-value table diagonal entries represent the distance between identical distributions (CL1 vs. CL1, CL2 vs. CL2, etc.) and are therefore always equal to zero. The largest distribution distance is between CL1 and CL3 (0.19); this indicates that feature X_{210} is a powerful discriminator between the two classes. The smallest distance is between CL2 and CL4 (0.04); this indicates that feature X_{210} is a poor discriminator between the two classes. When a classification algorithm is applied to this data set, the distance values suggest that more CL2/CL4 misclassifications would occur than CL1/CL3 misclassifications. Automatic classification will be discussed in a later section of this report.

The p-value represents the threshold at which the null hypothesis can be rejected or fail to be rejected. The null hypothesis states that the two distributions are the same. The alternative hypothesis states that they are different. If the p-value for a pair of distributions is less than the selected significance level of $\alpha = 0.1$, the null hypothesis can be rejected. In statistical notation:

- H_0: $F_i(X) = F_j(X)$
- H_A: $F_i(X) \neq F_j(X)$
- Reject H_0 if p-value $< \alpha$

The p-values computed from the KS-test indicate similar results as the D-values. The null hypothesis can be rejected for each pair of classes, except for CL2 vs. CL4. For this pair, the null hypothesis must fail to be rejected. In other words, at the selected significance level of $\alpha = 0.1$ (confidence level = 90%), the distributions of feature X_{210} in CL2 and CL4 are not different. With respect to automatic classification, poor discrimination between CL2 and CL4 should be expected. The p-value for the CL1 vs. CL3 pair is very small, below the rounding threshold displayed in the graph. This again indicates that the feature X_{210} distribution is significantly different between CL1 and CL3. There should be comparatively few misclassifications of CL1 as CL3 and vice-versa when an automatic classification algorithm is applied to the data.

Section 1.2

Each of the 400 features across all four classes can be correlated against one another to generate a 400x400 correlation coefficient matrix. The i^{th} x j^{th} entry of this matrix denotes the correlation coefficient between feature i and feature j in the global cleaned data set (N = 2,023). The correlation coefficient describes the "strength" and "direction" of the linear relationship between two features, with -1 describing a perfectly negative linear relationship, 0 describing no linear relationship, and +1 describing a perfectly positive linear relationship. It should be noted that variables whose coefficients are close to 0 may still exhibit a non-linear relationship. The formula for correlation coefficient between two variables CORR(X_i,X_j) is described below.

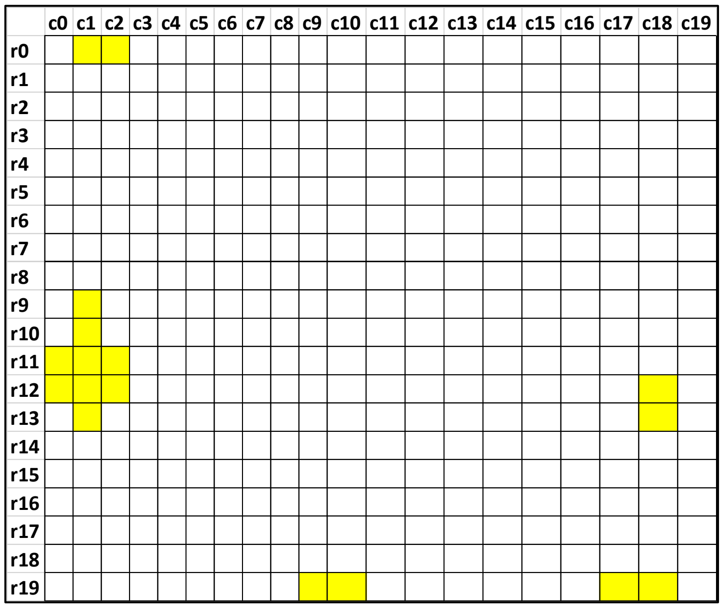
$$CORR(X_i, X_j) = \frac{\sum (x_{k,i} - \bar{x}_i)(x_{k,j} - \bar{x}_j)}{\sqrt{\sum (x_{k,i} - \bar{x}_i)^2 \sum (x_{k,j} - \bar{x}_j)^2}}$$

Where:

- $x_{k,i}$ represents the k^{th} entry of feature X_i.
- $\bar{x}_i$ represents the mean value of feature variable X_i.
- Similar notation is used for the response variable X_j.

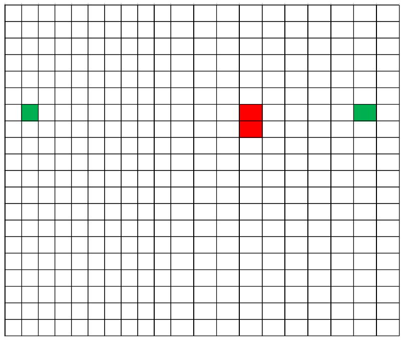
The ten highest absolute values of the correlation coefficient and the pixel positions (row, column) corresponding to each of these pairs is outlined below. It should be noted that true value, not absolute value, is listed in the table. In addition, the figure below outlines the location of each of the top ten correlation coefficients on a 20x20 matrix of pixels. Each cell corresponds to a pixel of the digitized image, and the cells highlighted in yellow correspond to pixel pairs with correlation coefficient values in the top ten.

	Feature X_i (Pixel)	Feature X_j (Pixel)	Correlation Coefficient
1	X_{399} (r19, c18)	X_{398} (r19, c17)	0.943
2	X_{262} (r13, c1)	X_{242} (r12, c1)	0.940
3	X_3 (r0, c2)	X_2 (r0, c1)	0.938
4	X_{391} (r19, c10)	X_{390} (r19, c9)	0.933
5	X_{241} (r12, c0)	X_{221} (r11, c0)	0.932
6	X_{222} (r11, c1)	X_{202} (r10, c1)	0.931
7	X_{243} (r12, c2)	X_{223} (r11, c2)	0.930
8	X_{202} (r10, c1)	X_{182} (r9, c1)	0.929
9	X_{242} (r12, c1)	X_{222} (r11, c1)	0.929
10	X_{279} (r13, c18)	X_{259} (r12, c18)	0.928


c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18 c19
r0 r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15 r16 r17 r18 r19

Features X_{399} and X_{398} have the highest coefficient value in matrix (0.943). This indicates that these two features have a nearly perfectly positive linear relationship. In fact, all features displayed on the table have a nearly perfect positive linear relationship. One interesting observation is that six out of the top ten correlated pairs of features occur in the horizontal left-hand side of the digitized image, near the vertical center. This corresponds to pixel positions approximately in the range of rows 9-13 and columns 0-2. Additionally, all of the top ten correlated feature pairs are adjacent pixels. This is expected, as adjacent pixels in digitized images of typed characters are more likely to show the similar gray levels. To further illustrate this point, compare the correlation coefficient values between the following pairs of pixels:

Pixel Pair Position	Feature X_i (Pixel)	Feature X_j (Pixel)	Correlation Coefficient
Adjacent	X_{134} (r6, c13)	X_{154} (r7, c13)	0.7
Separate	X_{122} (r6, c1)	X_{139} (r6, c18)	0.5



The pixel pair r6, c13 and r7, c13 are adjacent; the correlation coefficient value of this pixel pair is higher than the pixel pair r6, c1 and r6, c18. These pixels are far apart on the digitized image and hence have a much lower correlation coefficient value. The figure above outlines the location of these pixel on a 20x20 matrix of pixels. The cells highlighted in red correspond to the adjacent pair of pixels, and the cells highlighted in green correspond to the separated pixel pair.

Within the context of automatic classification, a feature correlation coefficient value close to -1 or +1 indicates that the features are redundant. If there are many features with high absolute correlation coefficient values, dimensionality reduction should be considered. One commonly used dimensionality reduction method is Principal Component Analysis (PCA). PCA is implemented and discussed in a later section of this report.

Sections 1.3 & 1.4

The feature data for this data set represents pixel gray levels that are quantized between 0 and 255. Although classification can be implemented using the raw data set, it is considered best practice to standardize or normalize the data prior to implementing a classification algorithm such as k-Nearest Neighbors (kNN). Standardizing (centering & rescaling) each feature to a mean value of 0 and a standard deviation of 1 allows the kNN algorithm to compute more a meaningful distance between neighboring points. The following equation describes how an individual feature value is normalized. This formula was applied to each feature value in the data set.

$$v_{j,n} = \frac{x_{j,n} - \bar{X}_J}{S_j}$$

Where:

- j represents a feature vector (X_1 through X_{400})
- n represents a case in the raw data set (1 through 2,023)
- $\bar{X}_j$ represents the mean of feature j
- S_j represents standard deviation of feature j
- $v_{j,n}$ represents the standardized value for feature j case n

Part 2

In Part 2, the k-Nearest Neighbor (kNN) automatic classification algorithm is applied to the standardized data set of digitized font character images. The kNN algorithm is a method of supervised learning that attempts to classify new (test) data based on distances to neighboring points in the training data. The number of neighbors that the classification is based on is the pre-specified k-value meta-parameter; the classification of the majority of points out of k neighbors determines the class of the new (test) point. The algorithm computes the Euclidian distance between a pre-specified (k) number of neighboring points in dimension $\mathbf{R}^p$; in this case p = 400 (number of features). As with the use of any model, there are benefits and drawbacks. The kNN algorithm is fast, simple, and can produce a relatively high percent of correct classifications without the need for tedious model training and parameter-fitting. The algorithm and close variations of the it can be used for both classification and regression tasks, and its applications are broad. One drawback of the kNN algorithm is the complexity introduced by many feature variables. For this data set with 400 features, a dimensionality reduction could be applied to reduce the number of features. The PCA method for dimensionality reduction is discussed in Part 3.

Section 2.1

Prior to implementing the kNN algorithm on the standardized data set (SDATA), the data was partitioned into training and test sets. First, SDATA was separated into four separate data sets, each containing one class (CL1 through CL4). Next, each class data set was randomly partitioned into training and test sets using the "80/20" heuristic. 80% of each class was assigned as the training set and 20% was assigned as the test set. The random partitioning mitigates against sample bias, and the decomposition of SDATA into smaller single-class data sets ensures even representation of all four classes in the training and test sets. Finally, the comparatively small number of CL1 cases relative to the other classes will hamper kNN model performance. This can be rectified by synthesizing additional data points. Several methods exist in the literature and in practice for amplifying minority data sets, including cloning, perturbation, and synthetic minority oversampling (SMOTE). The perturbation method was used to amplify CL1 in this data set. Perturbation generates synthetic data by taking a random sample of existing data points and "perturbing" the data across all features. The degree of perturbation is a randomly generated value between -3% to +3%. CL1 was doubled in both the training and test sets. Despite amplification, the number of cases is smaller than optimal for kNN implementation. Total set size on the order of several thousand cases would provide more concrete results.

Partitioned SDATA	Training Set	Test Set
CL1	195	49
CL2	489	123
CL3	477	120
CL4	456	114
Total	1,617	406

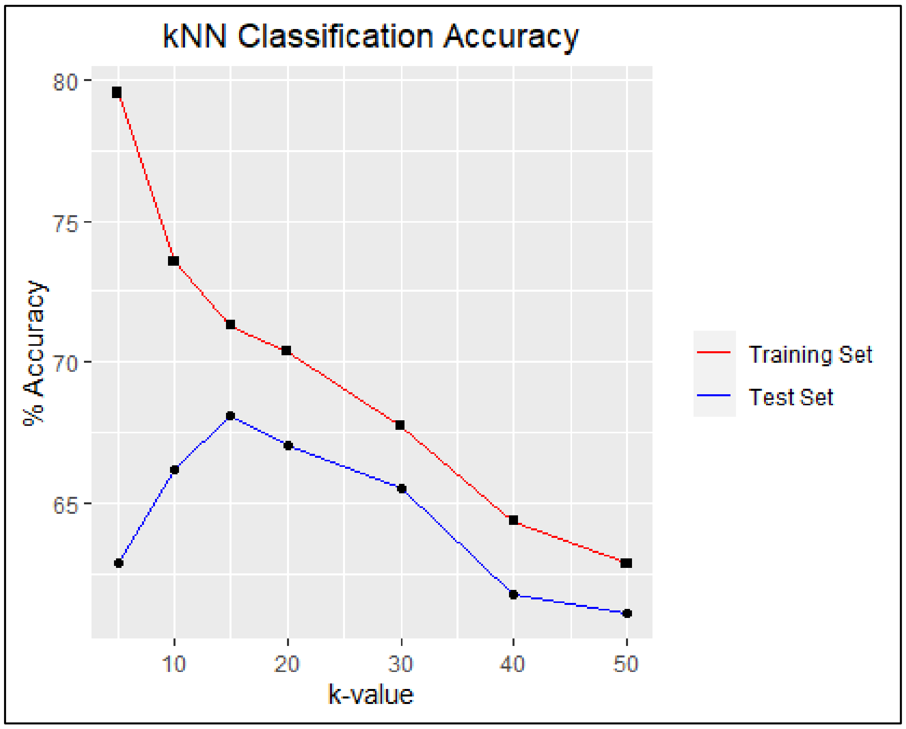
Amplified SDATA	Training Set	Test Set
CL1	390	98
CL2	489	123
CL3	477	120
CL4	456	114
Total	1,812	455

Section 2.2

A range of k-values between 5 and 50 were tested to determine optimum model performance. Performance is measured using percent accuracy:

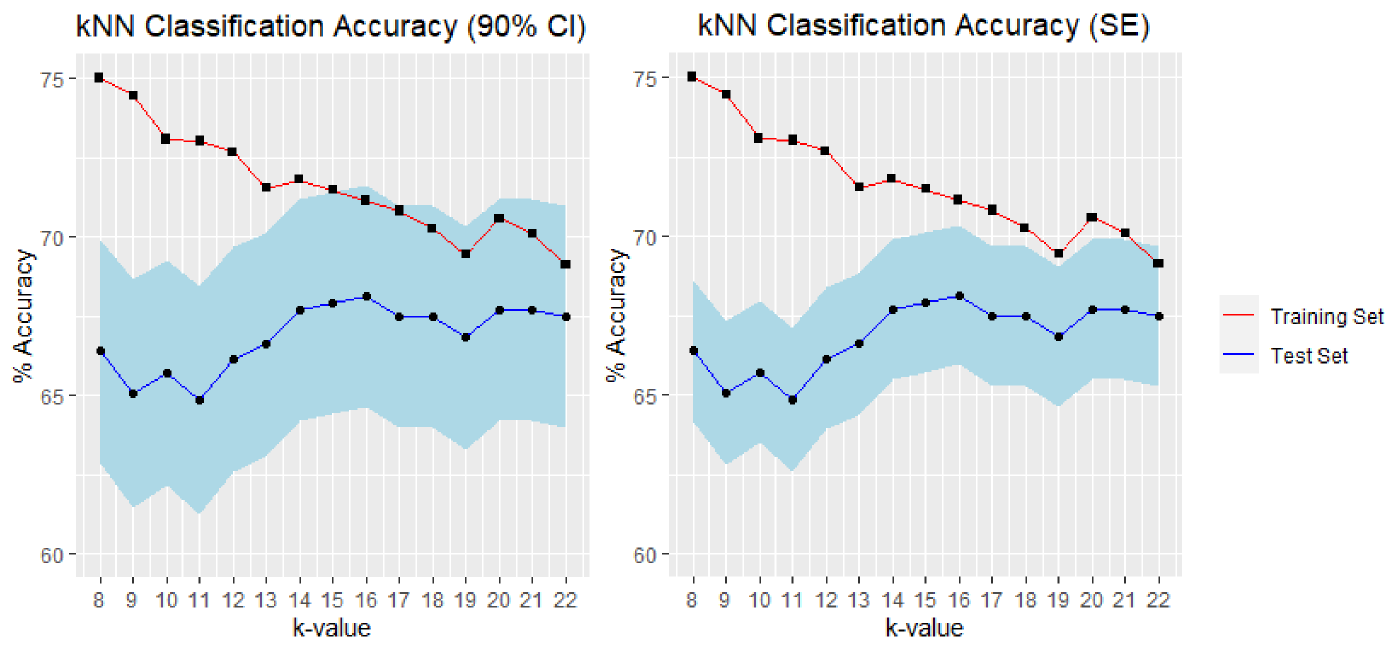
$$\%\ Accuracy = \frac{\#\ of\ Cases\ Classified\ Correctly}{Total\ \#\ of\ Cases\ in\ Class}$$

Using the amplified training and test sets, k-values between 10 and 20 offer the highest accuracy. In selecting k, it is worth noting that small k-values (k <10) tend to result in model "overfit". This means that the model performance is good when applied to the training set, but poor when applied to the test set or a new un-tested data set. The highest accuracy for the training set was observed at the smallest value of k (k=5). This k-value, however, offered the worst performance for the test set with an accuracy of 63%. Conversely, large k-values (k > 20) should also be avoided as the model is forced to classify based on a pool of training data that is too large. The results of kNN implementation show overfit at small k-values, peak accuracy for k-values between 10 and 20, and a gradual decline in performance for k-values greater than 20. Seven k-values were tested iteratively for both the training and test sets; 14 k-values were tested overall, and the computation time was 25 seconds (approx. 1.8 sec/iteration)



Section 2.3

Iterative testing was completed for k-values between 8 and 22 (slightly beyond the 10-20 optimum range) to determine a "best" k-value (k*) for this data set. Due to the size of the overall data set, particularly the small number of cases in the test set, generalizability of these results is limited.



Based on these results, k* = 15.

- Train performance for k*: 71.5%.
- Test performance for k*: 67.9%
    - Test performance 90% confidence interval: 64.4% - 71.4%
    - Test performance standard error bounds: 65.7% - 70.1%

Significant differences in model accuracy are observed between the training and test data sets. The best performance overall was given by k = 15, with 71.8% test set accuracy and 69% training set accuracy. The best performance for the training set was given by k = 8 with 75% accuracy; this value of k gave poor performance for the test set with 66.4% accuracy. As expected, the best k-value for the training data does not always give the best performance when applied to the test set. The computation time for this kNN implementation was 50 seconds (approx. 1.7 sec/iteration).

Section 2.4

Confusion matrices for k* = 15 are described below for both the training and test sets. The classes on the top horizontal are the predicted classes, the classes on the left vertical are actual classes. Diagonal entries are correctly classified cases.

Training Set	CL1	CL2	CL3	CL4
CL1	81%	8%	5%	5%
CL2	3%	69%	11%	16%
CL3	3%	9%	76%	11%
CL4	3%	11%	24%	62%

Test Set	CL1	CL2	CL3	CL4
CL1	82%	5%	7%	7%
CL2	1%	67%	8%	23%
CL3	6%	9%	72%	13%
CL4	3%	12%	32%	52%

Example (training set): 69% of CL2 cases were correctly classified as CL2. 3% of CL2 cases were incorrectly classified as CL1.

For the testing set, the 90% confidence interval for correct classifications is as follows:

- CL1: 77% – 87%
- CL2: 64% – 70%
- CL3: 69% – 75%
- CL4: 49% – 55%

Class CL1 was correctly classified most frequently in both the training and test sets (82% and 81% correct classifications, respectively). CL4 had the most misclassifications of any class. Only 62% of CL4 cases were correctly classified as CL4 in training data, and similarly only 52% were correctly classified in the test data. CL4 classification accuracy could be less than half based on the 90% confidence interval. As predicted KS-test D-values and p-values, CL2/CL4 misclassification errors were significant. In fact, test set results show that 23% of true CL2 cases were misclassified as CL4. Interestingly, CL4 was most commonly mis-classified as CL3 in both the training and the test sets (24% and 32%, respectively).

Section 2.5

In this section, distances were manually computed between misclassified points and the 15 nearest neighbors. Samples of misclassified points were taken from test set accuracy data at k* = 15.

True CL2 Predicted as CL1

A CL2 point that was misclassified as CL1 was randomly sampled from the test set results; the selected point was case #256. Using k* = 15, the 15 nearest points within the training data to case #256 are shown in the table below.

Point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Distance	8.54	16.69	16.78	18.61	19.59	19.69	20.12	20.47	20.58	20.59	20.70	20.80	20.83	20.90	20.93
Class	4	1	1	2	4	3	3	4	1	2	1	3	1	1	4

Class	Frequency
CL1	6
CL2	2
CL3	3
CL4	4

The distance calculations show that among the 15 nearest neighbors, 6 are CL1, 2 are CL2, 3 are CL3, and 4 are CL4. The majority vote for these 15 neighbors results in a classification of CL1; interestingly, the fewest number of neighbors belong to CL2 (the true class of case #256). One possible reason for the misclassification is due to the noise of data surrounding this case. All four classes are present among the 15 nearest neighbors. Observing the confusion matrix in Section 2.4, CL2 (Actual) to CL1 (Predicted) errors are rare with only 1% (5 points) exhbiting this misclassification. Therefore, it is reasonable to conclude that case #256 was misclassified due to noise surrounding the data.

True CL2 Predicted as CL3

A CL2 point that was misclassified as CL3 was randomly sampled from the test set results; the selected point was case #779. Using k* = 15, the 15 nearest points within the training data to case #779 are shown in the table below.

Point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Distance	6.34	13.62	14.73	16.69	17.37	17.37	17.37	17.71	17.75	18.11	18.39	18.77	18.79	19.38	19.45	19.46
Class	2	4	3	3	3	3	2	2	2	3	2	2	3	2	3	2

The distance calculations show that among the 15 nearest neighbors, 0 are CL1, 7 are CL2, 7 are CL3, and 1 is CL4. There is a tie in votes between CL2 (Actual) and CL3 (Predicted); to break a tie, the algorithm selects randomly. Observing the confusion matrix in Section 2.4, CL2 (Actual) to CL3 (Predicted) errors are moderate with 8% (8 points) exhibiting this misclassification. Below is a table of the 14, 15, and 16 nearest points and distances to the points of each class. Case #779 is only misclassified as a CL3 when k = 15. When the k-value is 14 or 16, case #779 is correctly classified as CL2.

Class	Frequency (K = 14)	Frequency (K = 15)	Frequency (K = 16)
CL1	0	0	0
CL2	7	7	8
CL3	6	7	7
CL4	1	1	1

True CL2 Predicted as CL4

A CL2 point that was misclassified as CL4 was randomly sampled from the test set results; the selected point was case #773. Using k* = 15, the 15 nearest points within the training data to case #773 are shown in the table below.

Point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Distance	4.90	5.60	5.62	8.21	8.21	8.21	9.47	10.48	10.95	10.95	10.95	11.73	11.73	11.77	12.24
Class	4	1	1	4	4	2	3	4	2	4	2	3	3	4	2

The distance calculations show that among the 15 nearest neighbors, 2 are CL1, 4 are CL2, 3 are CL3, and 6 are CL4. There is representation from every class within the 15 nearest neighbors, Observing the confusion matrix in Section 2.4, CL2 (Actual) to CL3 (Predicted) errors are significant with 23% (15 points) exhibiting this misclassification. As previously discussed, CL2 misclassifications as CL4 are the most common errors generated by this model. Case #779 is only misclassified as a CL3 when k = 15. When the k-value is increased to 16, 17, 18, or 19, case #773 is correctly classified as CL2.

Point	16	17	18	19
Distance	13.37	13.37	13.51	13.56
Class	2	2	2	2

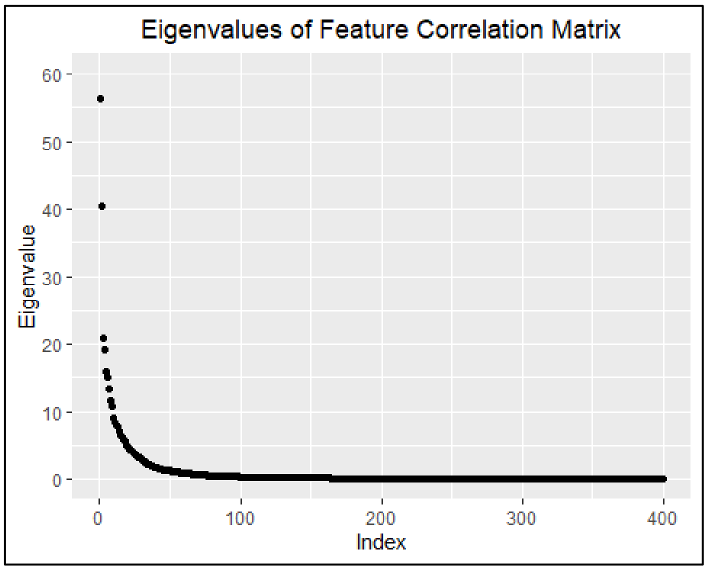
Class	Frequency
CL1	2
CL2	4
CL3	3
CL4	6

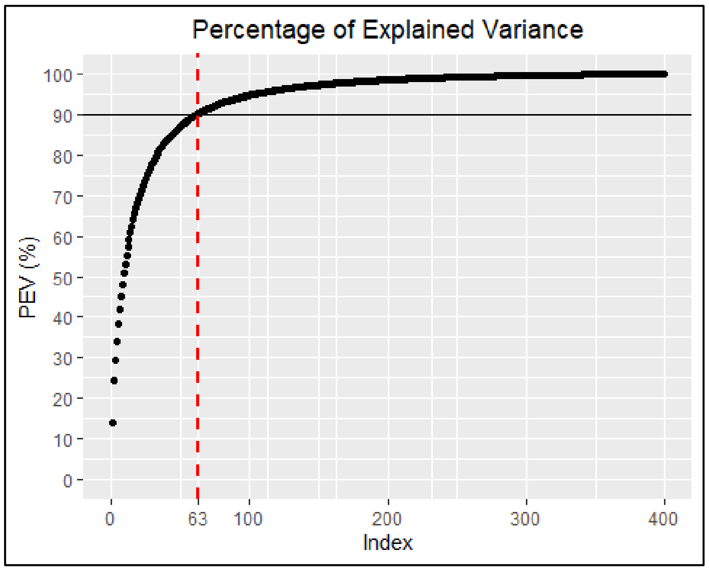
Part 3

In Part 3, Principal Component Analysis (PCA) is applied to this data set. PCA is a commonly used dimensionality reduction method that reduces the number of feature variables of a data set while preserving much of the explained variance. This is accomplished by computing and analyzing the eigenvectors and eigenvalues of the 400x400 feature correlation matrix discussed in Section 1.2. Implementation of PCA is discussed in detail, but the underlying mathematical theory is beyond the scope of this report.

Sections 3.1, 3.2, & 3.3

The left-hand figure below plots the eigenvalues of the 400x400 feature correlation matrix vs. the index number (1 through 400). The curve is non-linear; the slope is steeply decreasing from index position 1 through 50. Around index position 50, the slope flattens, and the curve very gradually decreases to zero as the index position approaches 400. The right-hand figure plots the cumulative sum of the eigenvalues, divided by 400 vs. index number (1 through 400). The cumulative sum of eigenvalues represents percentage of explained variance (PEV); this is essentially the percentage of variance in the data that can be explained by the i^{th} eigenvalue.

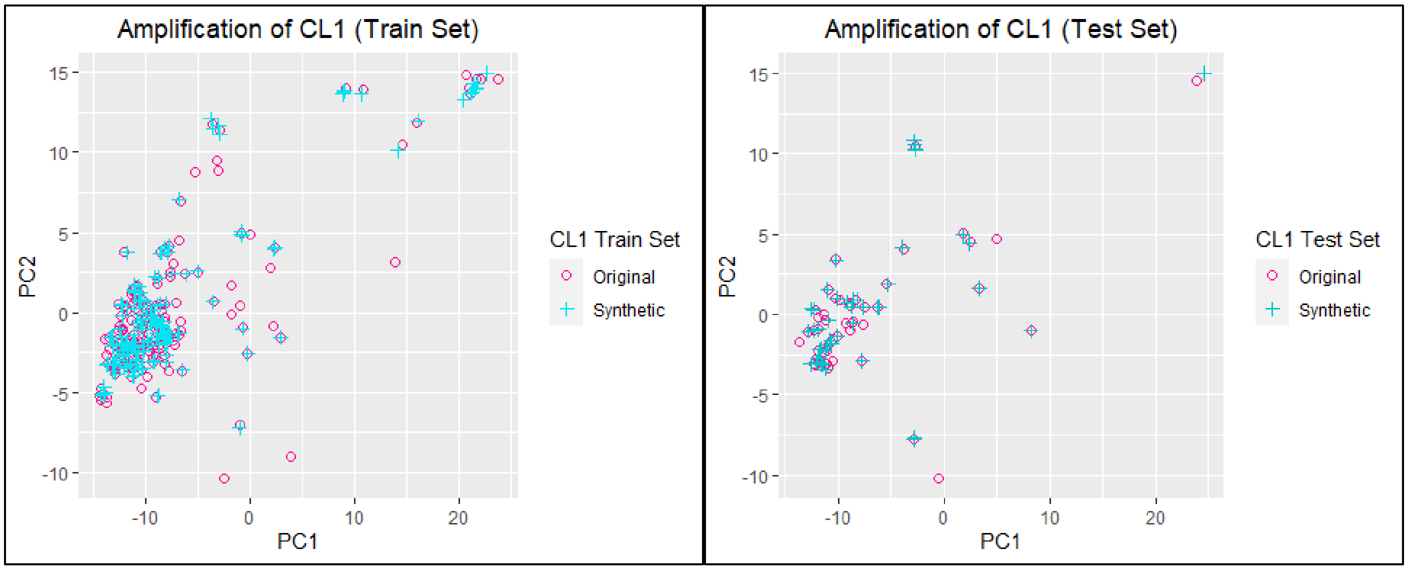




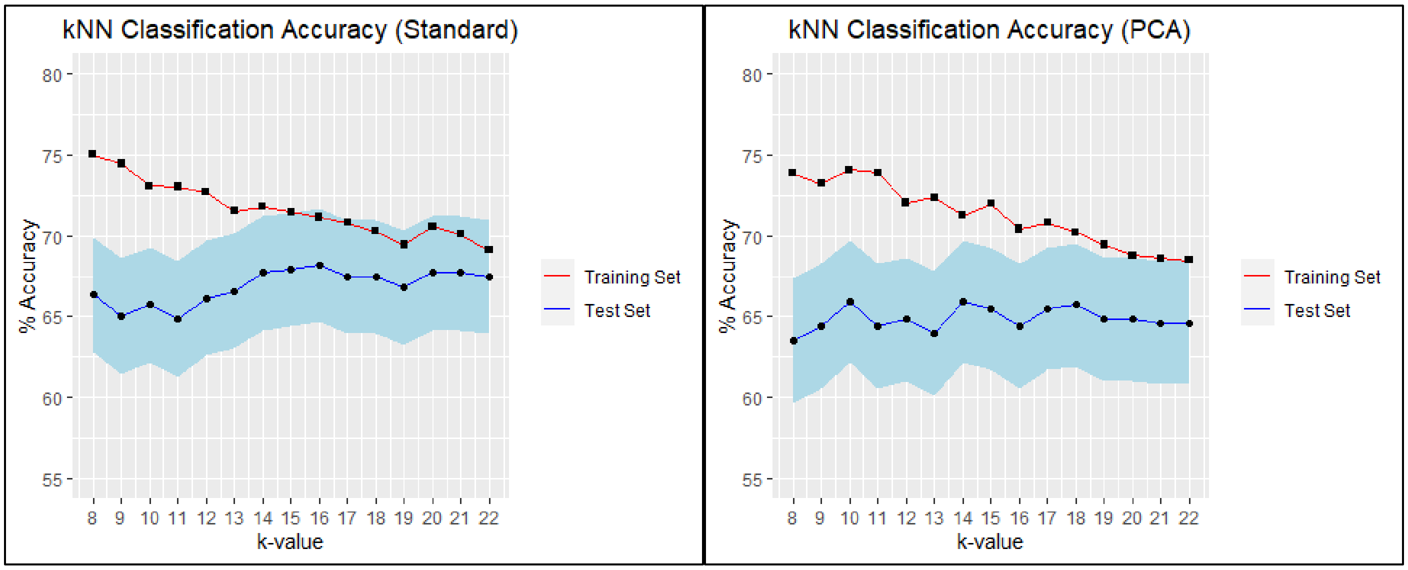
For this data set, 90% of the variance can be explained by the first 63 eigenvalues (r = 63). Each eigenvalue has a corresponding eigenvector (column vector of length 400). The eigenvectors represent principal components of the data set. As such, each of the first 63 eigenvectors can be multiplied by each row of the standardized data set (SDATA) to produce a new smaller data set (ZDATA). The ZDATA set of 63 principal components can replace the SDATA set with the original 400 features while capturing 90% of the variance in the data. The full 400 x 400 matrix of eigenvectors is provided as an attachment to this report.

Section 3.4

In an identical fashion as Section 2.1, the ZDATA set was randomly partitioned into training and test sets. Again, the comparatively small number class CL1 was amplified by perturbation. The amplified points in both the training and test sets are displayed in the figures below. The points are plotted against the first two principal components (PC1 and PC2). These plots provide a visual method of depicting the perturbation in 2-dimensions; it should be noted that PC1 and PC2 only explain 24% of the variance in the data.



In a similar fashion as Section 2.3, k-values were iteratively tested over the initially identified optimum range of 8 - 22. The results of kNN using the ZDATA set is compared side-by-side as the previously displayed results of the SDATA set. Performing kNN using the reduced ZDATA set results in slightly lower classification performance. The classification performance (% accuracy) is displayed in the table below for both SDATA and ZDATA.



k* = 15	SDATA	ZDATA
Training Accuracy	71.5%	72.0%
Test Accuracy	67.9%	65.5%
Test 90% CI	64.4% - 71.4%	61.7% - 69.3 %

Training set classification accuracy was similar for both data sets. Test accuracy, however, was 2.4% lower for the reduced ZDATA set. Though most of the explained variance is preserved, 10% of data variance is lost in the ZDATA set.

Below are confusion matrices for k* = 15. The left-hand confusion matrix represents accuracy values from the original SDATA set, while the right-hand matrix represents accuracy values for the reduced ZDATA set. Once again, the classes on the top horizontal are the predicted classes, the classes on the left vertical are actual classes. Diagonal entries are correctly classified cases.

Test Set (SDATA)	CL1	CL2	CL3	CL4
CL1	82%	5%	7%	7%
CL2	1%	67%	8%	23%
CL3	6%	9%	72%	13%
CL4	3%	12%	32%	52%

Test Set (ZDATA	CL1	CL2	CL3	CL4
CL1	82%	9%	7%	2%
CL2	5%	64%	10%	22%
CL3	7%	7%	63%	23%
CL4	2%	12%	33%	53%

Example (SDATA): 67% of CL2 cases were correctly classified as CL2. 1% of CL2 cases were incorrectly classified as CL1.

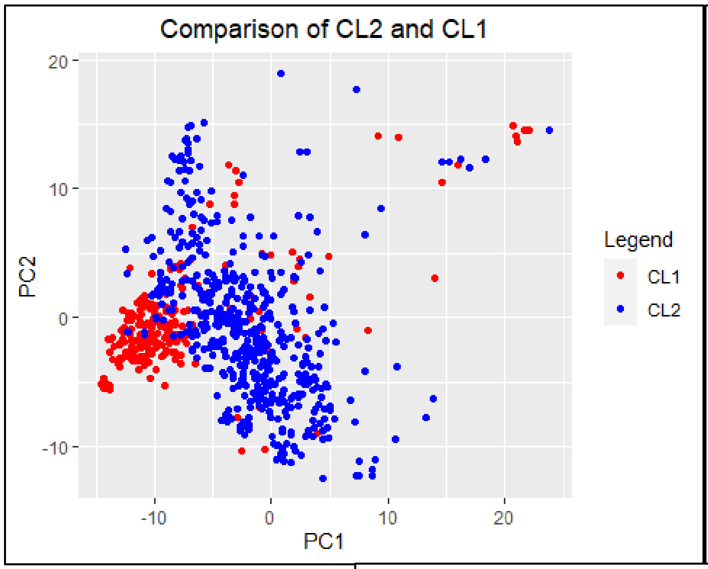
The percentage of correct classifications (diagonal entries) is lower in the reduced ZDATA set compared to the original SDATA set. This is expected, as ZDATA only captures 90% of the variance in the model. The comparative misclassifications between SDATA and ZDATA are nearly identical as well. CL1 was again correctly classified most frequently in the ZDATA set. CL4 again had the most misclassifications of any class. Perhaps the most significant difference between applying kNN to the SDATA set vs. the ZDATA set is computing time. As previously stated, iterative k-value testing on the SDATA set took 50 seconds (1.7 sec/iteration). The identical iterative operation on the ZDATA set took 5 seconds (0.17 sec/iteration, 10% of SDATA). The power of PCA is apparent; PCA significantly reduces the computational burden of automatic classification tasks on large data sets with minimal sacrifice of classification accuracy.

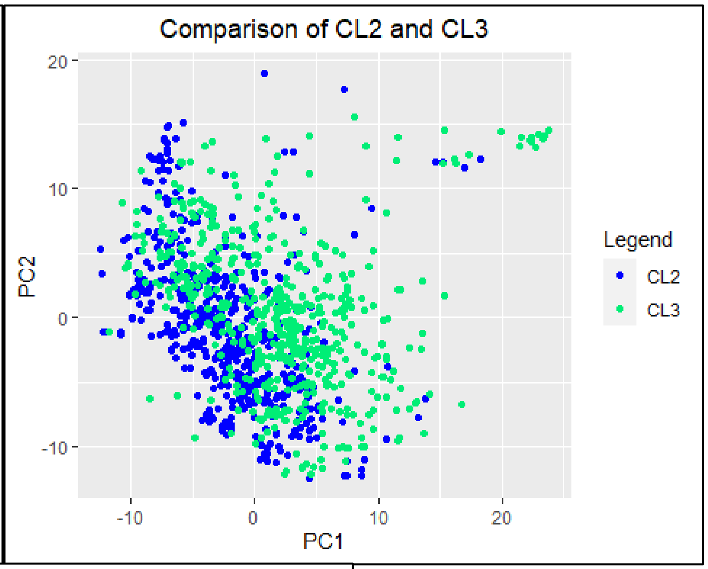
Section 3.5

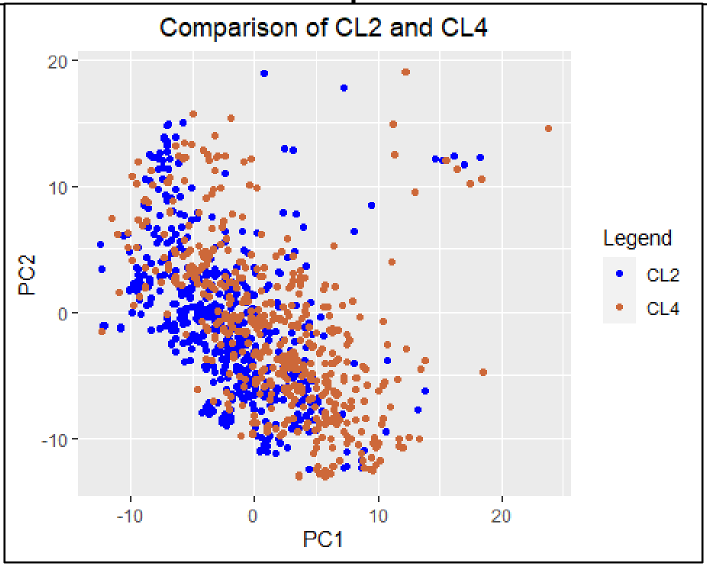
The following figures display planar (2-dimensional) projections of each class using the first two principal components (PC1 and PC2). As previously stated, PC1 and PC2 explain 24% of the variance in the data. The purpose of this comparison is to determine graphically if the classes are well separated with respect to the first two principal components. It should be noted that only the original data for CL1 is represented, no synthetic data is displayed.

Each class is color-coded as follows:

- CL1 – Red
- CL2 – Blue
- CL3 – Green
- CL4 – Brown





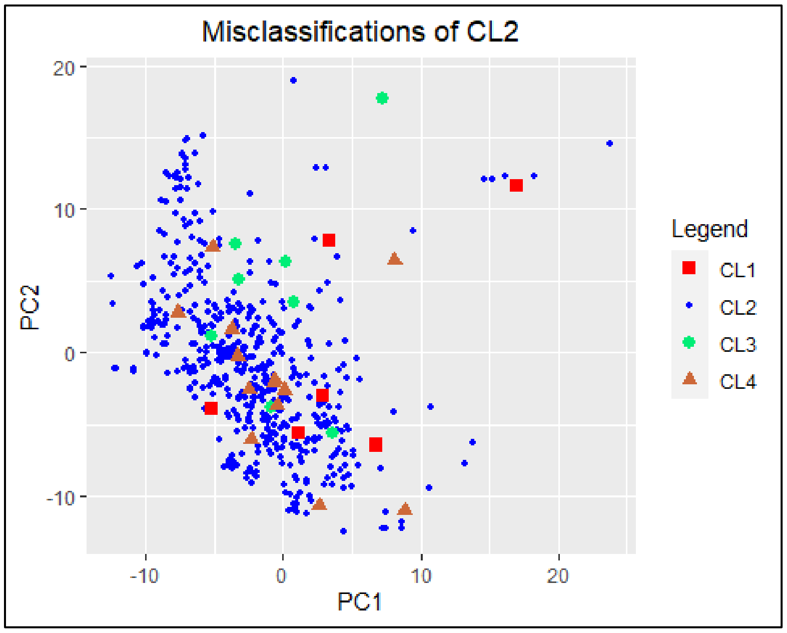


First, CL2 and CL1 appear to be reasonably well-separated and relatively few misclassification errors would be expected between these two classes. From the confusion matrix, 5% of true CL2 points were misclassified as CL1, and 9% of true CL1 points we misclassified as CL2. Next, CL2 and CL3 are not well separated. As a result, more misclassifications between CL2/CL3 should be expected. From the confusion matrix, 10% of true CL2 points were misclassified as CL3, and 7% of true CL3 points we misclassified as CL2. Lastly, CL2 and CL4 are not well separated either; 22% of true CL2 points were misclassified as CL4, and 12% of true CL4 points we misclassified as CL2.

In addition, comparison of the apparent "slope" of the clusters provides some insight into the differences between the classes. This analysis is purely visual and therefore is subject to biases of interpretation. Points in CL2, CL3, and CL4 appear to trend in the negative direction with respect to PC1 and PC2, while CL1 shows a slight positive trend. This could partially explain why CL1 has consistently maintained the best classification accuracy among the four classes.

Section 3.6

In the figure below, CL2 misclassifications are plotted on the same planar projection of PC1 and PC2 as Section 3.5. In addition, the color coding of classes is identical. similar.



First, true CL2 points misclassified as CL1 appear to be near the edges of the cluster of CL2 points. Next, true CL2 points misclassified as CL3 appear to also be near the edges of the CL2 cluster but this is less distinguishable than CL1. Finally, CL2 points misclassified as CL4 do not follow any discernible pattern. CL2/CL4 misclassifications have consistently been the most frequent errors made by the kNN model in all implementations within this report. As has been previously discussed, the feature distributions of CL2 and CL4 are very similar.

Conclusion

In summary, four datasets containing digitized images of typed font characters were analyzed in this report. These font data sets were cleaned, and the k-Nearest Neighbors (kNN) algorithm was applied as an automatic classifier. The cleaned data set consisted of 2,023 cases with 400 features describing each case. The data was standardized a perturbation method was applied to amplify the minority class (CL1). Additionally, Principal Component Analysis (PCA) was performed on the data to reduce the dimensionality of the data set from 400 to 63 while preserving 90% of the explained variance in the data. This dimensionality reduction greatly decreased computation time; in fact, computation time for an identical kNN implementation was 90% faster for the reduced data set. In testing kNN, an optimal k-value was found to be k* = 15. Similar accuracy was observed in both the standardized original data and the reduced PCA data at this value of k. The application of kNN was fast and straightforward. The model accuracy was reasonable considering there was no training or parameter-fitting outside of determining the k-value itself.

Appendix

Appendix A – PC Specifications

Computation time will vary depending on the hardware and software specifications of the computer that calculations are performed on. Computation times listed in this report are from a PC with the following technical specifications:

Operating System	Windows 10 Home Edition Version 21H1 64-bit OS
Processor	Intel Core i5-7400 CPU @ 3.00GHz x64-based processor
Installed RAM	32.0 GB
R-Studio	Version 1.4.1717 – Juliet Rose

Appendix B - R Code

```r
#Setting up knitr, installing packages
knitr::opts_chunk$set(echo = TRUE)
#install.packages("ggplot2")
#install.packages("kableExtra")
#install.packages("tidyverse")
#install.packages("lemon")
library(ggplot2)
library(kableExtra)
library(tidyverse)
library(data.table)
library(rsample)
library(class)
library(reshape2)
library(cowplot)
library(dplyr)
library(lemon)
```

Scope

```r
#Importing fonts 1, 2, 3, 4
vladimir_raw = read.csv("fonts/VLADIMIR.csv", header = TRUE)
georgia_raw = read.csv("fonts/GEORGIA.csv", header = TRUE)
comic_raw = read.csv("fonts/COMIC.csv", header = TRUE)
nina_raw = read.csv("fonts/NINA.csv", header = TRUE)
#Removing columns containing unneeded information
vladimir = vladimir_raw[c(-2:-3,-6:-12)]
georgia = georgia_raw[c(-2:-3,-6:-12)]
comic = comic_raw[c(-2:-3,-6:-12)]
nina = nina_raw[c(-2:-3,-6:-12)]

#Removing rows that contain missing data
vladimir = vladimir[complete.cases(vladimir),]
georgia = georgia[complete.cases(georgia),]
comic = comic[complete.cases(comic),]
nina = nina[complete.cases(nina),]

#Changing name in 'font' to CL1,...,CL4
vladimir$font = "CL1"
georgia$font = "CL2"
comic$font = "CL3"
nina$font = "CL4"
#Defining "normal" font type as classes CL1,...,CL4
#"normal" means not bold nor italicized
CL1 = subset(vladimir, strength == 0.4 & italic == 0)
CL2 = subset(georgia, strength == 0.4 & italic == 0)
CL3 = subset(comic, strength == 0.4 & italic == 0)
CL4 = subset(nina, strength == 0.4 & italic == 0)

#Removing columns containing 'strength' and 'italic'
CL1 = CL1[-2:-3]
CL2 = CL2[-2:-3]
CL3 = CL3[-2:-3]
CL4 = CL4[-2:-3]

#Renaming the feature columns of CL1,...,CL4 to X1,...,X400
names_list = NULL #initialize an empty list

for (i in 1:400){ #naming 400 columns X1 to X400
 names_list[[i]] = paste("X",i,sep = "")
}
colnames(CL1)[2:401] = c(names_list)
colnames(CL2)[2:401] = c(names_list)
colnames(CL3)[2:401] = c(names_list)
colnames(CL4)[2:401] = c(names_list)

#Defining number of rows
nCL1 = nrow(CL1)
nCL2 = nrow(CL2)
nCL3 = nrow(CL3)
nCL4 = nrow(CL4)

#Regrouping CL1,...,CL4 into one group called "DATA"
DATA = rbind(CL1,CL2,CL3,CL4)
```

```r
nDATA = nrow(DATA) #calculating rows in DATA

colnames(DATA)[1] = c("TRUC") #renaming font to "TRUC"
DATA$TRUC = as.factor(DATA$TRUC) #converting TRUC to factor
#Mean of feature X210 for CL1,...,CL4
meanCL1 = mean(CL1$X210) #mean of feature X210 for CL1
meanCL2 = mean(CL2$X210) #mean of feature X210 for CL2
meanCL3 = mean(CL3$X210) #mean of feature X210 for CL3
meanCL4 = mean(CL4$X210) #mean of feature X210 for CL4
meanDATA = mean(DATA$X210) #mean of feature X210 for all classes

C = .9 #setting confidence level of 90%

#t-test for CL1,...,CL4
#Obtain t critical values for each class
t_CL1 = qt((1+C)/2,nCL1-1) #t critical value CL1
t_CL2 = qt((1+C)/2,nCL2-1) #t critical value CL2
t_CL3 = qt((1+C)/2,nCL3-1) #t critical value CL3
t_CL4 = qt((1+C)/2,nCL4-1) #t critical value CL4

#Confidence intervals for C1,...,CL4
lo_CL1 = meanCL1 - t_CL1*sd(CL1$X210)/sqrt(nCL1) #CI for CL1
hi_CL1 = meanCL1 + t_CL1*sd(CL1$X210)/sqrt(nCL1)

lo_CL2 = meanCL2 - t_CL2*sd(CL2$X210)/sqrt(nCL2) #CI for CL2
hi_CL2 = meanCL2 + t_CL2*sd(CL2$X210)/sqrt(nCL2)

lo_CL3 = meanCL3 - t_CL3*sd(CL3$X210)/sqrt(nCL3) #CI for CL3
hi_CL3 = meanCL3 + t_CL3*sd(CL3$X210)/sqrt(nCL3)

lo_CL4 = meanCL4 - t_CL4*sd(CL4$X210)/sqrt(nCL4) #CI for CL4
hi_CL4 = meanCL4 + t_CL4*sd(CL4$X210)/sqrt(nCL4)

#create a table of the results of CI calculations for each class
conf_table = data.frame(Class = rep(c("CL1", "CL2", "CL3", "CL4")),
                        Lower = rep(round(c(lo_CL1,lo_CL2,lo_CL3,lo_CL4),1)),
                        Mean = rep(round(c(meanCL1,meanCL2,meanCL3,meanCL4)),1),
                        Upper = rep(round(c(hi_CL1,hi_CL2,hi_CL3,hi_CL4),1)))


#get the true class of each row of the column X210
X210_df = as.data.frame(cbind(DATA$TRUC,DATA$X210))
names(X210_df) = c("TRUC","X210") #rename columns
X210_df$TRUC = as.factor(X210_df$TRUC) #convert to TRUC factor

#create a dataframe of the class, their means and CI values
X210_data = data.frame(Class = c("CL4","CL3","CL2","CL1"),
                       Mean = c(meanCL4,meanCL3,meanCL2,meanCL1),
                       Low = c(lo_CL4,lo_CL3,lo_CL2,lo_CL1),
                       High = c(hi_CL4,hi_CL3,hi_CL2,hi_CL1))


#Plot of the CI for X210 in each class
ggplot() +
  geom_pointrange(data = X210_data,mapping = aes(x=Class,y=Mean,ymin=Low,ymax=High,color = Class),
                size=1.5, fill="white", shape=22) +
  labs(title = "Feature X210 - 90% CI for All Classes") +
  xlab("Class")+ ylab("Feature X210 Value")+
  theme(plot.title = element_text(hjust = 0.5)) + coord_flip() +
  scale_color_manual(values = c("red","blue","springgreen","sienna3"))
# Histograms for CL1, CL2, CL3, CL4 with mean and 90% CI
ggplot(CL1,aes(X210)) +
  geom_histogram(aes(y=stat(count)/sum(count)),binwidth = 25, fill = "cadetblue", color="#e9ecef") +
  labs(title = "Histogram of Feature X210 for Class CL1")+ xlab("X210 Value for Class CL1")+ylab("Frequency")+
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(seq(0,max(CL1$X210),25))) +
  scale_y_continuous(breaks=c(seq(0,.6,.1)),limits = c(0,0.55)) +
  geom_vline(aes(xintercept=lo_CL1, color = "Lower"), linetype="dashed", color = "red", size=1.5) +
  geom_vline(aes(xintercept=meanCL1, color = "Mean"), linetype="solid", color = "black", size=1.5) +
  geom_vline(aes(xintercept=hi_CL1,color = "Upper"), linetype="dashed", color = "blue", size=1.5)
ggplot(CL2,aes(X210)) +
  geom_histogram(aes(y=stat(count)/sum(count)),binwidth = 25, fill = "cadetblue", color="#e9ecef") +
  labs(title = "Histogram of Feature X210 for Class CL2")+ xlab("X210 Value for Class CL2")+ylab("Frequency")+
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(seq(0,max(CL2$X210),25))) +
  scale_y_continuous(breaks=c(seq(0,.6,.1)),limits = c(0,0.55)) +
  geom_vline(xintercept = meanCL2, linetype="solid", color = "black", size=1.5) +
  geom_vline(xintercept = lo_CL2, linetype="dashed", color = "red", size=1.5) +
  geom_vline(xintercept = hi_CL2, linetype="dashed", color = "blue", size=1.5)
ggplot(CL3,aes(X210)) +
  geom_histogram(aes(y=stat(count)/sum(count)),binwidth = 25, fill = "cadetblue", color="#e9ecef") +
  labs(title = "Histogram of Feature X210 for Class CL3")+ xlab("X210 Value for Class CL3")+ylab("Frequency")+
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(seq(0,max(CL3$X210),25))) +
  scale_y_continuous(breaks=c(seq(0,.6,.1)),limits = c(0,0.55)) +
  geom_vline(xintercept = meanCL3, linetype="solid", color = "black", size=1.5) +
  geom_vline(xintercept = lo_CL3, linetype="dashed", color = "red", size=1.5) +
  geom_vline(xintercept = hi_CL3, linetype="dashed", color = "blue", size=1.5)
ggplot(CL4,aes(X210)) +
  geom_histogram(aes(y=stat(count)/sum(count)),binwidth = 25, fill = "cadetblue", color="#e9ecef") +
  labs(title = "Histogram of Feature X210 for Class CL4")+ xlab("X210 Value for Class CL4")+ylab("Frequency")+
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_x_continuous(breaks=c(seq(0,max(CL4$X210),25))) +
  scale_y_continuous(breaks=c(seq(0,.6,.1)),limits = c(0,0.55)) +
  geom_vline(xintercept = meanCL4, linetype="solid", color = "black", size=1.5) +
  geom_vline(xintercept = lo_CL4, linetype="dashed", color = "red", size=1.5) +
  geom_vline(xintercept = hi_CL4, linetype="dashed", color = "blue", size=1.5)
#Histogram showing all 4 classes
ggplot() +
  geom_histogram(aes(X210, fill = TRUC),binwidth = 30, data = X210_df) +
  labs(title = "Histogram of Feature X210 for All Classes") +
  ylab("Count") + xlab("Feature X210") +
  theme(plot.title = element_text(hjust = 0.5))  +
  scale_fill_manual(values = c("red","blue","springgreen","sienna3"),name = "Class")


ks_12D = as.numeric(ks.test(CL1$X210, CL2$X210)[1])                                                #KS-test of CL1
and CL2 and the distance between CDFs
## Warning in ks.test(CL1$X210, CL2$X210): p-value will be approximate in the
## presence of ties
ks_13D = as.numeric(ks.test(CL1$X210, CL3$X210)[1])                                                #KS-test of CL1
and CL3 and the distance between CDFs
## Warning in ks.test(CL1$X210, CL3$X210): p-value will be approximate in the
## presence of ties
ks_14D = as.numeric(ks.test(CL1$X210, CL4$X210)[1])                                                #KS-test of CL1
and CL4 and the distance between CDFs
## Warning in ks.test(CL1$X210, CL4$X210): p-value will be approximate in the
## presence of ties
ks_23D = as.numeric(ks.test(CL2$X210, CL3$X210)[1])                                                #KS-test of CL2
and CL3 and the distance between CDFs
## Warning in ks.test(CL2$X210, CL3$X210): p-value will be approximate in the
## presence of ties
ks_24D = as.numeric(ks.test(CL2$X210, CL4$X210)[1])                                                #KS-test of CL2
and CL4 and the distance between CDFs
## Warning in ks.test(CL2$X210, CL4$X210): p-value will be approximate in the
## presence of ties
ks_34D = as.numeric(ks.test(CL3$X210, CL4$X210)[1])                                                #KS-test of CL3
and CL4 and the distance between CDFs
## Warning in ks.test(CL3$X210, CL4$X210): p-value will be approximate in the
## presence of ties
ks_12P = as.numeric(ks.test(CL1$X210, CL2$X210)[2])                                                #KS-test of CL1
and CL2 and how similar they are
## Warning in ks.test(CL1$X210, CL2$X210): p-value will be approximate in the
## presence of ties
ks_13P = as.numeric(ks.test(CL1$X210, CL3$X210)[2])                                                #KS-test of CL1
and CL3 and how similar they are
## Warning in ks.test(CL1$X210, CL3$X210): p-value will be approximate in the
## presence of ties
ks_14P = as.numeric(ks.test(CL1$X210, CL4$X210)[2])                                                #KS-test of CL1
and CL4 and how similar they are
## Warning in ks.test(CL1$X210, CL4$X210): p-value will be approximate in the
## presence of ties
ks_23P = as.numeric(ks.test(CL2$X210, CL3$X210)[2])                                                #KS-test of CL2
and CL3 and how similar they are
## Warning in ks.test(CL2$X210, CL3$X210): p-value will be approximate in the
## presence of ties
ks_24P = as.numeric(ks.test(CL2$X210, CL4$X210)[2])                                                #KS-test of CL2
and CL4 and how similar they are
```

```
## Warning in ks.test(CL2$X210, CL4$X210): p-value will be approximate in the
## presence of ties
ks_34P = as.numeric(ks.test(CL3$X210, CL4$X210)[2])                                      #KS-test of CL3
and CL4 and how similar they are
## Warning in ks.test(CL3$X210, CL4$X210): p-value will be approximate in the
## presence of ties
#create a dataframe of the values produced in the KS-test
d_table = data.frame(Class = rep(c("CL1", "CL2", "CL3", "CL4")),
                        CL1 = rep(round(c(0,ks_12D,ks_13D,ks_14D),2)),
                        CL2 = rep(round(c(ks_12D,0,ks_23D,ks_24D),2)),
                        CL3 = rep(round(c(ks_13D,ks_23D,0,ks_34D),2)),
                        CL4 = rep(round(c(ks_14D,ks_24D,ks_34D,0),2)))
d_table #show results in a table
##   Class  CL1  CL2  CL3  CL4
## 1   CL1 0.00 0.13 0.19 0.12
## 2   CL2 0.13 0.00 0.11 0.04
## 3   CL3 0.19 0.11 0.00 0.09
## 4   CL4 0.12 0.04 0.09 0.00
#create a dataframe of the values produced in the KS-test
p_table = data.frame(Class = rep(c("CL1", "CL2", "CL3", "CL4")),
                        CL1 = rep(round(c(0,ks_12P,ks_13P,ks_14P),3)),
                        CL2 = rep(round(c(ks_12P,0,ks_23P,ks_24P),3)),
                        CL3 = rep(round(c(ks_13P,ks_23P,0,ks_34P),3)),
                        CL4 = rep(round(c(ks_14P,ks_24P,ks_34D,0),3)))
d_table #show results in a table
```

1.2

```
#DATA contains all 4 classes, filtered for bold and italics

DATA_features = DATA[-1] #Subsetting DATA into features only
corr_matrix_full = cor(DATA_features) #Taking 400x400 correlation matrix

corr_matrix = as.data.frame(as.table(corr_matrix_full)) #Taking 400x400 correlation matrix
corr_matrix = subset(corr_matrix, abs(Freq) !=1) #Removing values = 1 (diagonal entries)

corr_matrix$sign = ifelse(corr_matrix$Freq > 0, 1,-1) #Adding new column to track sign of cor value
corr_matrix$Freq = abs(corr_matrix$Freq) #Converting cor value to abs(cor)
corr_matrix = corr_matrix[!duplicated(corr_matrix$Freq),] #Removing pair-wise entries (duplicate values)

corr_top_10 <- corr_matrix %>% #Taking the top ten abs(cor) values
  arrange(desc(Freq)) %>%
  slice(1:10)

corr_top_10$Freq = corr_top_10$Freq * corr_top_10$sign #multiplying the sign back into the cor value
corr_top_10 = corr_top_10[-4] #removing the column containing sign
names(corr_top_10) = c("Feature 1", "Feature 2", "Correlation Coefficient") #Renaming columns
corr_top_10 #display top ten results
```

1.3 & 1.4

```
normalize = function(x){ #normalize function takes the mean and sd of the column
  return((x - mean(x))/sd(x))
}

#function that will take a given dataframe and apply the normalize function
#to every column in the data set, returning a new dataframe.
standard <- function(df){
  std_df <- df[1]
  parsed <- df[2:dim(df)[2]]
  for (i in 1:dim(parsed)[2]){
    std_df <- cbind(std_df, (as.data.frame(normalize(parsed[,i]))))
  }
  return(std_df)
}
SDATA <- standard(DATA)                                                        #standardize DATA
colnames(SDATA)[2:401] = c(names_list)#rename the columns of the new SDATA dataframe

colnames(DATA)[1] = c("TRUC") #rename the first column in the DATA
colnames(SDATA)[1] = c("TRUC") #rename the first column in the SDATA

SDATA$TRUC = as.factor(SDATA$TRUC) #Create first column as a factor
DATA$TRUC = as.factor(DATA$TRUC)
```

Part 2
2.1
```r
random_subset = function(data){ #function that randomly partitions data into train and test
  CL1_sub = subset(data, TRUC == "CL1")
  CL2_sub = subset(data, TRUC == "CL2")
  CL3_sub = subset(data, TRUC == "CL3")
  CL4_sub = subset(data, TRUC == "CL4")
  #split the data to a training set for CL1 - CL4
  CL1_smp <- initial_split(CL1_sub, prop = .8)
  CL2_smp <- initial_split(CL2_sub, prop = .8)
  CL3_smp <- initial_split(CL3_sub, prop = .8)
  CL4_smp <- initial_split(CL4_sub, prop = .8)
  #recombine into global training and test set
  train_set <- rbind(training(CL1_smp),training(CL2_smp),training(CL3_smp),training(CL4_smp))
  test_set <- rbind(testing(CL1_smp),testing(CL2_smp),testing(CL3_smp),testing(CL4_smp))

  return(list(train_set,test_set))
}

#function takes a standardized feature matrix and perturbes between -3% and +3%
#Amplifies 2x by default
perturb = function(sfdata){
  set.seed(4)
  rownum = c(1:nrow(sfdata))
  sampled_rows = sample(nrow(sfdata),replace = TRUE)
  for (i in sampled_rows) {
    pert_factor = runif(1,.97,1.03) #random perturbation between -3 and +3 percent
    pert_case = sfdata[i,]*pert_factor
    sfdata = rbind(sfdata,pert_case)
  }
  return(sfdata)
}
```

2.2
```r
set.seed(3)
S_split = random_subset(SDATA) #randomly partitioning data into train/test set

train_S = S_split[[1]] #extracting training set

### Amplify class CL1 training set by perturbation
train_S_CL1 = subset(train_S, TRUC == "CL1")[-1]
train_S = subset(train_S, TRUC != "CL1")
train_S_CL1 = perturb(train_S_CL1)
train_S_CL1 = cbind(TRUC = "CL1",train_S_CL1)
train_S = rbind(train_S_CL1,train_S)

### Amplify class CL1 test set by perturbation
test_S = S_split[[2]]
test_S_CL1 = subset(test_S, TRUC == "CL1")[-1]
test_S = subset(test_S, TRUC != "CL1")
test_S_CL1 = perturb(test_S_CL1)
test_S_CL1 = cbind(TRUC = "CL1",test_S_CL1)
test_S = rbind(test_S_CL1,test_S)
set.seed(3) #set a seed for reproducability

k <- c(5,10,15,20,30,40,50) #define k-values to be used
train_perf <- c() #intialize empty vector to store training performance
test_perf <- c() #intialize empty vector to store testing performance

start_time = Sys.time() #log start time

for (i in 1:length(k)){ #iterate through each k-value.
  #Apply knn algorithm to training set
  train_perf[i] <- sum(train_S[,1] == knn(train_S[2:401], train_S[2:401], train_S[,1], k = k[i])) / nrow(train_S[1])
  #Apply knn algorithm to test set
  test_perf[i] <- sum(test_S[,1] == knn(train_S[2:401], test_S[2:401], train_S[,1], k = k[i])) / nrow(test_S[1])
}
end_time = Sys.time() #log end time

accuracy_df <- cbind(train_perf, test_perf, k) #store accuracy and k-values
accuracy_df <- as.data.frame(accuracy_df) #convert to dataframe
computing_duration = end_time - start_time #calculate computation time

print(computing_duration) #display computation time
```

```
## Time difference of 12.28538 secs
#plot accuracy of training and test accuracy vs. k-value
ggplot(accuracy_df, aes(k)) +
  geom_line(aes(y = train_perf*100, color = "Training Set")) +
  geom_point(aes(y = train_perf*100),shape = 15) +
  labs(title = "kNN Classification Accuracy",x="k-value",y="% Accuracy") +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_line(aes(y = test_perf*100, color = "Test Set")) +
  geom_point(aes(y = test_perf*100),shape = 16) +
  scale_color_manual("",
                     breaks = c("Training Set", "Test Set"),
                     values = c("Training Set" = "red", "Test Set" = "blue")
```

2.3

```
set.seed(3) #set seed for reproducibility
k <- c(8:22) #define k-values to be used
train_perf <- c() #intialize empty vector to store training performance
test_perf <- c() #intialize empty vector to store testing performance

start_time = Sys.time() #log start time
for (i in 1:length(k)){ #same operation as section 2.2
  train_perf[i] <- sum(train_S[,1] == knn(train_S[2:401], train_S[2:401], train_S[,1], k = k[i])) / nrow(train_S[1])
  test_perf[i] <- sum(test_S[,1] == knn(train_S[2:401], test_S[2:401], train_S[,1], k = k[i])) / nrow(test_S[1])
}
end_time = Sys.time() #log end time

#calculate standard error
testmargin = sqrt(test_perf*(1-test_perf)/nrow(test_S))
test_t = 1.6 #use t-statistic = 1.6
upper_margin = test_perf+test_t*testmargin #calculate upper margin of 90% CI
lower_margin = test_perf-test_t*testmargin #calculate lower margin of 90% CI

accuracy_df <- cbind(train_perf, test_perf, k) #store results from kNN
accuracy_df <- as.data.frame(accuracy_df) #convert to dataframe

computing_duration = end_time - start_time #calculate computation time
print(computing_duration) #display compuation time
## Time difference of 25.51012 secs
#create a plot of accuracy models for train and test with the CI of the test
ggplot(accuracy_df, aes(k)) +
  geom_ribbon(aes(ymin = upper_margin*100, ymax = lower_margin*100), fill = "lightblue")+
  geom_line(aes(y = train_perf*100, color = "Training Set")) +
  geom_point(aes(y = train_perf*100),shape = 15) +
  geom_line(aes(y = test_perf*100, color = "Test Set")) +
  geom_point(aes(y = test_perf*100),shape = 16) +
  labs(title = "kNN Classification Accuracy (90% CI)",x="k-value",y="% Accuracy") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual("",
                     breaks = c("Training Set", "Test Set"),
                     values = c("Training Set" = "red", "Test Set" = "blue")) +
  scale_x_continuous(breaks = k) + ylim(60,75)
ggplot(accuracy_df, aes(k)) +
  geom_ribbon(aes(ymin = (test_perf - testmargin)*100,
                  ymax = (test_perf + testmargin)*100), fill = "lightblue")+
  geom_line(aes(y = train_perf*100, color = "Training Set")) +
  geom_point(aes(y = train_perf*100),shape = 15) +
  geom_line(aes(y = test_perf*100, color = "Test Set")) +
  geom_point(aes(y = test_perf*100),shape = 16) +
  labs(title = "kNN Classification Accuracy (SE)",x="k-value",y="% Accuracy") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual("",
                     breaks = c("Training Set", "Test Set"),
                     values = c("Training Set" = "red", "Test Set" = "blue")) +
  scale_x_continuous(breaks = k) + ylim(60,75)
```

2.4

```
set.seed(3) #set seed for reproducibility
#get the KNN model for training at k = 15
training_matrix <- knn(train_S[2:401], train_S[2:401], train_S[,1], k = 15)
#assign a table to the model and the predictors.
confusion_table_train <- table(training_matrix,train_S[,1])
#create a prop table that will round the percents of factors based on rows
round(prop.table(confusion_table_train, 1),2)*100
```

```
#get the KNN model for testing at k = 15
testing_matrix <- knn(train_S[2:401], test_S[2:401], train_S[,1], k = 15)
confusion_table_test <- table(testing_matrix,test_S[,1])
round(prop.table(confusion_table_test, 1),2)*100

testmargin_1 = sqrt(test_perf*(1-test_perf)/nrow(test_S$TRUC == CL1))
testmargin_2 = sqrt(test_perf*(1-test_perf)/nrow(test_S$TRUC == CL2))
testmargin_3 = sqrt(test_perf*(1-test_perf)/nrow(test_S$TRUC == CL3))
testmargin_4 = sqrt(test_perf*(1-test_perf)/nrow(test_S$TRUC == CL4))
#function to input any category selected for errors
#finds the selected errors in a dataframe.
error_list <- function(List,predict_list, cat1, cat2){
  Error_table <- cbind(List[1], predict_list, List[2:401])
  Error_table <- Error_table[which(Error_table[1] != Error_table[2] & Error_table[1] == cat1 & Error_table[2] == cat2),]
  return(Error_table)
}

#formula to preform distance from a given point to all other points in a given dataset
K_distance <- function(point, data){
  dist <- c()
  for (i in 1:dim(data)[1]){
    #use dist function to find distance between two points
    distance <- round(dist(rbind(point[3:402], data[i,2:401])),2)
    if (data[i,1] == "CL1"){
      cl <- 1
    }
    else if (data[i,1] == "CL2"){
      cl <- 2
    }
    else if (data[i,1] == "CL3"){
      cl <- 3
    }
    else if(data[i,1] == "CL4"){
      cl <- 4
    }
    point_distance <- cbind(i , distance, cl)
    dist <- rbind(dist, point_distance)
  }
  return(dist)#return a table of distances of that point to all points
}
```

2.5

```
PredC <- testing_matrix  #defining matrix of predicted classes

#combine the test set with the predicted vector but as the second column
Error_table <- cbind(test_S[1], PredC, test_S[2:401])

#using function error list, return a list of errors, return a list of errors, class 2 but predicted class 1
errors21 <- error_list(test_S, PredC, "CL2", "CL1")
#select a random point of error that we may want to use
selected21 <- errors21[sample(nrow(errors21), 1), ]
#find the distance between that point and all the data in the train dataset
KNN_select21 <- K_distance(selected21, train_S)
#sort the distances to obtain the nearest 15 points
KNN_select21 <- KNN_select21[order(KNN_select21[,2]),]
head(KNN_select21,15)
errors23 <- error_list(test_S, PredC, "CL2", "CL3")
#select a random point of error that we may want to use
selected23 <- errors23[sample(nrow(errors23), 1), ]
#find the distance between that point and all the data in the train dataset
KNN_select23 <- K_distance(selected23, train_S)
#sort the distances to obtain the nearest 15 points
KNN_select23 <- KNN_select23[order(KNN_select23[,2]),]
head(KNN_select23,15)
errors24 <- error_list(test_S, PredC, "CL2", "CL4")
selected24 <- errors24[sample(nrow(errors24), 1), ]
KNN_select24 <- K_distance(selected24, train_S)
KNN_select24 <- KNN_select24[order(KNN_select24[,2]),]
head(KNN_select24,15)
```

Part 3
3.1
```
#calculating eigenvectors/values of 400x400 correlation matrix
corr_ev = eigen(corr_matrix_full)
L_values = corr_ev$values #storing eigenvalues
W_vectors = corr_ev$vectors #storing eigenvectros
#create 400x400 eigenvector matrix as csv file
write.csv(W_vectors,"C:\\Users\\basel\\OneDrive\\MSDS FA 2021\\6350 - Stat Learn & Data Mining\\HW\\Eigenvectors.csv",
row.names = FALSE)

#plot eigenvalue vs index
qplot(seq_along(L_values),L_values) +
  labs(title = "Eigenvalues of Feature Correlation Matrix",x="Index",y="Eigenvalue") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,60,10),limits = c(0,60))
```

3.2
```
PEV = cumsum(L_values/length(L_values))*100 #taking cumulative sum of eigenvalues divided by 400
PEV_opt = which.min(abs(PEV - 90)) + 1 #finding optimum % explained variance

#plot PEV vs. index
qplot(seq_along(PEV),PEV) +
  labs(title = "Percentage of Explained Variance" , x="Index" , y="PEV (%)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,100,10),limits = c(0,100)) +
  scale_x_continuous(breaks = c(seq(0,400,100),PEV_opt),limits = c(0,400)) +
  geom_vline(xintercept = PEV_opt, color = "red", linetype = "dashed",size = 1) +
  geom_hline(yintercept = 90, color = "black", linetype = "solid",size = .01)
```

3.4
```
set.seed(4) #setting random seed for reproducibility

Z_split = random_subset(ZDATA) #randomly subsetting into train/test splits

train_Z = Z_split[[1]] #defining training split

train_Z_CL1_original = subset(train_Z, TRUC == "CL1") #subset of original CL1 training data
train_Z = subset(train_Z, TRUC != "CL1") #subset of non-CL1 training data

train_Z_CL1 = train_Z_CL1_original[-1] #removing "TRUC" column
train_Z_CL1 = perturb(train_Z_CL1) #perturbing data in CL1 training set
train_Z_CL1 = cbind(TRUC = "CL1",train_Z_CL1) #adding "TRUC" column
train_Z = rbind(train_Z_CL1,train_Z) #re-combining into global training set.


train_Z_CL1_synth = train_Z_CL1[-1:-nrow(train_Z_CL1_original),] #subsetting synthetic data
train_Z_CL1_synth$TRUC = as.factor("Synthetic") #labelling synthetic data
train_Z_CL1_original$TRUC = as.factor("Original") #labelling original data
train_Z_CL1 = rbind(train_Z_CL1_original,train_Z_CL1_synth) #re-combinining

test_Z = Z_split[[2]] #defining test split

test_Z_CL1_original = subset(test_Z, TRUC == "CL1") #subset of original CL1 testing data
test_Z = subset(test_Z, TRUC != "CL1") #subset of non-CL1 testing data

test_Z_CL1 = test_Z_CL1_original[-1] #removing "TRUC" column
test_Z_CL1 = perturb(test_Z_CL1) #perturbing data in CL1 testing set
test_Z_CL1 = cbind(TRUC = "CL1",test_Z_CL1) #adding "TRUC" column
test_Z = rbind(test_Z_CL1,test_Z) #re-combining into global testing set.

test_Z_CL1_synth = test_Z_CL1[-1:-nrow(test_Z_CL1_original),] #subsetting synthetic data
test_Z_CL1_synth$TRUC = as.factor("Synthetic") #labelling synthetic data
test_Z_CL1_original$TRUC = as.factor("Original") #labelling original data
test_Z_CL1 = rbind(test_Z_CL1_original,test_Z_CL1_synth) #re-combinining


ggplot(train_Z_CL1,aes(x = PC1, y =PC2, color = factor(TRUC))) +
  geom_point(aes(shape = factor(TRUC),size = factor(TRUC))) +
  scale_color_manual(values=c("deeppink", "turquoise2"), name = "CL1 Train Set") +
  scale_shape_manual(values=c(1,3), name = "CL1 Train Set") +
  scale_size_manual(values = c(2,2),name="CL1 Train Set") +
  labs(title = "Amplification of CL1 (Train Set)") +
  theme(plot.title = element_text(hjust = 0.5))
```

```r
ggplot(test_Z_CL1,aes(x = PC1, y =PC2, color = factor(TRUC))) +
  geom_point(aes(shape = factor(TRUC),size = factor(TRUC))) +
  scale_color_manual(values=c("deeppink", "turquoise3"), name = "CL1 Test Set") +
  scale_shape_manual(values=c(1,3), name = "CL1 Test Set") +
  scale_size_manual(values = c(2,2),name="CL1 Test Set") +
  labs(title = "Amplification of CL1 (Test Set)") +
  theme(plot.title = element_text(hjust = 0.5))
#PCA
set.seed(3) #set seed for reproducability

k <- c(8:22) #set k-values
train_perf <- c() #initialize train/test vectors
test_perf <- c()

start_time = Sys.time() #log start time
for (i in 1:length(k)){ #apply kNN algorithm
  train_perf[i] <- sum(train_Z[,1] == knn(train_Z[2:dim(train_Z)[2]], train_Z[2:dim(train_Z)[2]], train_Z[,1], k = k[i]))
/ nrow(train_Z[1])
  test_perf[i] <- sum(test_Z[,1] == knn(train_Z[2:dim(train_Z)[2]], test_Z[2:dim(test_Z)[2]], train_Z[,1], k = k[i])) /
nrow(test_Z[1])
}
end_time = Sys.time() #log end time

### similar code as previous kNN implementations (see section 2) ###
accuracy_df_Z <- cbind(train_perf, test_perf, k)
accuracy_df_Z <- as.data.frame(accuracy_df_Z)

testmargin_Z = sqrt(test_perf*(1-test_perf)/length(test_S))

test_t = 1.6

upper_margin_Z = test_perf+test_t*testmargin_Z
lower_margin_Z = test_perf-test_t*testmargin_Z

computing_duration = end_time - start_time
print(computing_duration)
## Time difference of 2.53167 secs
ggplot(accuracy_df, aes(k)) +
  geom_ribbon(aes(ymin = upper_margin*100, ymax = lower_margin*100), fill = "lightblue")+
  geom_line(aes(y = train_perf*100, color = "Training Set")) +
  geom_point(aes(y = train_perf*100),shape = 15) +
  geom_line(aes(y = test_perf*100, color = "Test Set")) +
  geom_point(aes(y = test_perf*100),shape = 16) +
  labs(title = "kNN Classification Accuracy (Standard)",x="k-value",y="% Accuracy") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual("",
                     breaks = c("Training Set", "Test Set"),
                     values = c("Training Set" = "red", "Test Set" = "blue")) +
  scale_x_continuous(breaks = k) +
  ylim(55,80)
ggplot(accuracy_df_Z, aes(k)) +
  geom_ribbon(aes(ymin = upper_margin_Z*100, ymax = lower_margin_Z*100), fill = "lightblue")+
  geom_line(aes(y = train_perf*100, color = "Training Set")) +
  geom_point(aes(y = train_perf*100),shape = 15) +
  geom_line(aes(y = test_perf*100, color = "Test Set")) +
  geom_point(aes(y = test_perf*100),shape = 16) +
  labs(title = "kNN Classification Accuracy (PCA)",x="k-value",y="% Accuracy") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_color_manual("",
                     breaks = c("Training Set", "Test Set"),
                     values = c("Training Set" = "red", "Test Set" = "blue")) +
  scale_x_continuous(breaks = k) +
  ylim(55,80)
#confusion matrix comparison for k = 20 w/ confidence intervals
k <- 15

train_perf <- knn(train_Z[2:dim(train_Z)[2]], train_Z[2:dim(train_Z)[2]], train_Z[,1], k = k)
test_perf <- knn(train_Z[2:dim(train_Z)[2]], test_Z[2:dim(test_Z)[2]], train_Z[,1], k = k)

training_matrix <- train_perf         #get the KNN model for training at k = 20
confusion_table_train <- table(training_matrix,train_S[,1])                    #assign a table to the model and
the predictors.
```

```
round(prop.table(confusion_table_train, 1),2)*100                    #create a prop table that will ouput the percents of
the factors based on the rows and round it
testing_matrix <- test_perf           #get the KNN model for testing at k = 20
confusion_table_test <- table(testing_matrix,test_S[,1])
round(prop.table(confusion_table_test, 1),2)*100
###comments about ZDATA vs SDATA confusion matrix.
```

3.5

```
#CL1 = red
#CL2 = blue
#CL3 = green
#CL4 = brown

CL1_Zsub = subset(ZDATA, TRUC == "CL1")[1:3]
CL2_Zsub = subset(ZDATA, TRUC == "CL2")[1:3]
CL3_Zsub = subset(ZDATA, TRUC == "CL3")[1:3]
CL4_Zsub = subset(ZDATA, TRUC == "CL4")[1:3]

names(CL1_Zsub)[1] = "CLASS"
names(CL2_Zsub)[1] = "CLASS"
names(CL3_Zsub)[1] = "CLASS"
names(CL4_Zsub)[1] = "CLASS"

CL12_Zsub = rbind(CL1_Zsub,CL2_Zsub)
CL13_Zsub = rbind(CL1_Zsub,CL3_Zsub)
CL14_Zsub = rbind(CL1_Zsub,CL4_Zsub)
CL23_Zsub = rbind(CL2_Zsub,CL3_Zsub)
CL24_Zsub = rbind(CL2_Zsub,CL4_Zsub)
CL34_Zsub = rbind(CL3_Zsub,CL4_Zsub)

ggplot(CL12_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +
  scale_color_manual(values=c("red", "blue"),name = "Legend") +
  labs(title = "Comparison of CL2 and CL1") +
  theme(plot.title = element_text(hjust = 0.5))
ggplot(CL23_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +
  scale_color_manual(values=c("blue", "springgreen2"),name = "Legend") +
  labs(title = "Comparison of CL2 and CL3") +
  theme(plot.title = element_text(hjust = 0.5))
ggplot(CL24_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +
  scale_color_manual(values=c("blue", "sienna3"),name = "Legend") +
  labs(title = "Comparison of CL2 and CL4") +
  theme(plot.title = element_text(hjust = 0.5))
```

3.6

```
error_list2 <- function(List,predict_list, cat1, cat2){           #function to input any category selected for errors
  Error_table <- cbind(List[1], predict_list, List[2:64])
  Error_table <- Error_table[which(Error_table[1] != Error_table[2] & Error_table[1] == cat1 & Error_table[2] == cat2),]
  return(Error_table)
}
#combine the test set with the predicted vector but as the second column
Error_table <- cbind(test_Z[1], PredC, test_Z[2:64])

# see if we increase K how much does the total amount of class changes. do we end up with the right class after more ks
are added

errors21_Z <- error_list2(test_Z, PredC, "CL2", "CL1")[2:4]
errors23_Z <- error_list2(test_Z, PredC, "CL2", "CL3")[2:4]
errors24_Z <- error_list2(test_Z, PredC, "CL2", "CL4")[2:4]
names(errors21_Z)[1] = "CLASS"
names(errors23_Z)[1] = "CLASS"
names(errors24_Z)[1] = "CLASS"
#CL1 = red
#CL2 = blue
#CL3 = green
#CL4 = brown
CL2_Zsub_err = rbind(CL2_Zsub,errors21_Z,errors23_Z,errors24_Z)
ggplot(CL2_Zsub_err,aes(x = PC1, y =PC2, color = factor(CLASS))) +
  geom_point(aes(shape = factor(CLASS),color = factor(CLASS),size = factor(CLASS))) +
  scale_color_manual(values=c("red", "blue", "springgreen2", "sienna3"),name = "Legend") +
  scale_shape_manual(values=c(15,20,16,17),name = "Legend")+
  scale_size_manual(values=c(2.5,1.7,2.5,2.5),name = "Legend") +
  labs(title = "Misclassifications of CL2") +
  theme(plot.title = element_text(hjust = 0.5))
```