10/4/2021

# MATH 6350

Homework #2

Chia-Hung Chien, Tola Ouk, Tyler Stinson
UNIVERSITY OF HOUSTON

# Table of Contents

## Scope of the Report

This report is going to classify the font by its features which are the gray scale value extracted from a 20 x 20 pixels image. Thus, each case will have 400 features. The interested data is downloaded from "University of California Irvine Repository of Machine Learning Datasets". There are 153 .csv files and each file contain a bunch of cases with some information about the case such as if the font is bolded or presented in italic style, etc. The most important things are the 400 features came from the gray scale value of each pixel. We are going to select 4 out of 153 files. Data will be cleaned and treated to apply the k-Nearest Neighbor (kNN) Automatic Classifier. We would like to know if it is possible to use gray scale value of each pixel to predict the correct font. If so, what is the accuracy of prediction.

Firstly, the data will be split into 20% of test set and 80% of training set randomly. Then, use these two sets to find out the best hyperparameter, $k$, in kNN model and identify the misclassified data and explain them.

Second, Principal Component Analysis method will be introduced in Part 3 to reduce the number of features. This can help to reduce the computation time but keep the prediction accuracy as high as possible.

## Data Cleaning and Treatment

The four font files listed below were chosen for this study:

1. GILL.csv
2. LEELAWADEE.csv
3. ROMAN.csv
4. TECHNIC.csv

The informative features of these datasets are

- font: font tyle of each case
- italics: 1 if the character is italic and 0 otherwise
- strength: 0.4 for normal characters and 0.7 for bold characters
- as well as 400 others that correspond to a specific pixel.

A single case in the font dataset describes a digitized image of a specific character typed in a specific font. Each image has size 20 x 20 = 400 pixels. Each pixel is assigned an integer value between 0 and 255, known as a "gray level".

These 9 columns are discarded: fontVariant, m_label, orientation, m_top, m_left, originalH, originalW, h, w while 3 columns, font, strength, italic, were kept along with the 400 columns named r0c0, r0c1, r0c2, ..., r19c18, r19c19. Any row containing missing numerical data were also discarded.

We then defined GILL as CL1, LEELAWADIE as CL2, ROMAN as CL3, and TECHNIC as CL4. The size of each class and total size of all the classes are shown in Table 1 below.

*Table 1 Selected Files and Size of each class and total*

| | File | Size |
|---|---|---|
| CL1 | GILL.csv | 1459 |
| CL2 | LEELAWADEE.csv | 1378 |
| CL3 | ROMAN.csv | 1194 |
| CL4 | TECHNIC.csv | 1380 |
| Total | | 5411 |

Since each case has about 22 ~ 27% of the total data set, it can be considered as balanced data. Thus, further treatment such as cloning, perturbation, or SMOTE which are used to deal with imbalanced data are not required. These four classes combined make a 5411 x 400 data frame.

# 1 Preliminary Analysis of the Data Set

## 1.1 Statistical Parameter Analysis of Feature X210

Feature X210 is the pixel located at row 10 and column 9. This example should give some insight about the information for the entire data set. Since this data is the gray level image of a font, it can be imaged that most of the values in each feature are close to either 0 (white) or 255 (black). There are very few numbers between 1 and 254.

### *1.1.1 Mean of Feature X210 for each class*

Firstly, the mean of this feature among all the cases is calculated. Under this circumstance, the means can be interpreted as the ratio of 0 and 255. If the mean is higher, more 255 are contained in this feature which means that most of the cases have black color at this pixel and vice versa. From Table 2, it can be preliminarily observed that the means for the four classes are noticeably distinct. The difference in mean will be further checked by using Student's t-test.

*Table 2 Mean of Feature X210*

| | Mean |
|---|---|
| CL1 | 146.7 |
| CL2 | 68.3 |
| CL3 | 59.1 |
| CL4 | 82.7 |

### *1.1.2 Student's t-test*

All the two-samples Student's t-test has Null Hypothesis as below:

$$H_0: \mu_{class\ A} = \mu_{class\ B}$$

$H_0$ can be rejected if the p-value is smaller than the significance level $\alpha$, which is set as 0.1 for the Student's t-tests. Table 3 shows the results from the two-samples Student's t-tests for each pair of four classes.

*Table 3 Student's t-test between two classes and compare P-value with $\alpha = 0.1$*

| 1st Class | 2nd Class | p-value | p-value < 0.1 |
|---|---|---|---|
| CL1 | CL2 | 8.07E-74 | TRUE |
| CL1 | CL3 | 3.51E-92 | TRUE |
| CL1 | CL4 | 2.92E-48 | TRUE |
| CL2 | CL3 | 1.88E-02 | TRUE |
| CL2 | CL4 | 4.79E-04 | TRUE |
| CL3 | CL4 | 6.60E-09 | TRUE |

Since all the p-values are essentially close to 0 and significantly smaller than $\alpha$, all the Null Hypothesis, $H_0$, can be rejected. The t-test results support the observation in Section 1.1.1 that the class means are significantly different.

### *1.1.3 Histogram of X210 Feature for each class*

The histograms are plotted with density instead of frequency since the size are different for each class. From Figure 1, it can be observed that Class 1 (Upper Left) has a higher density of pixels with a grey level of 255, and a lower grey value of 0. The other three classes have a high density of grey levels at the 0 value and a lower density near the 255 values.

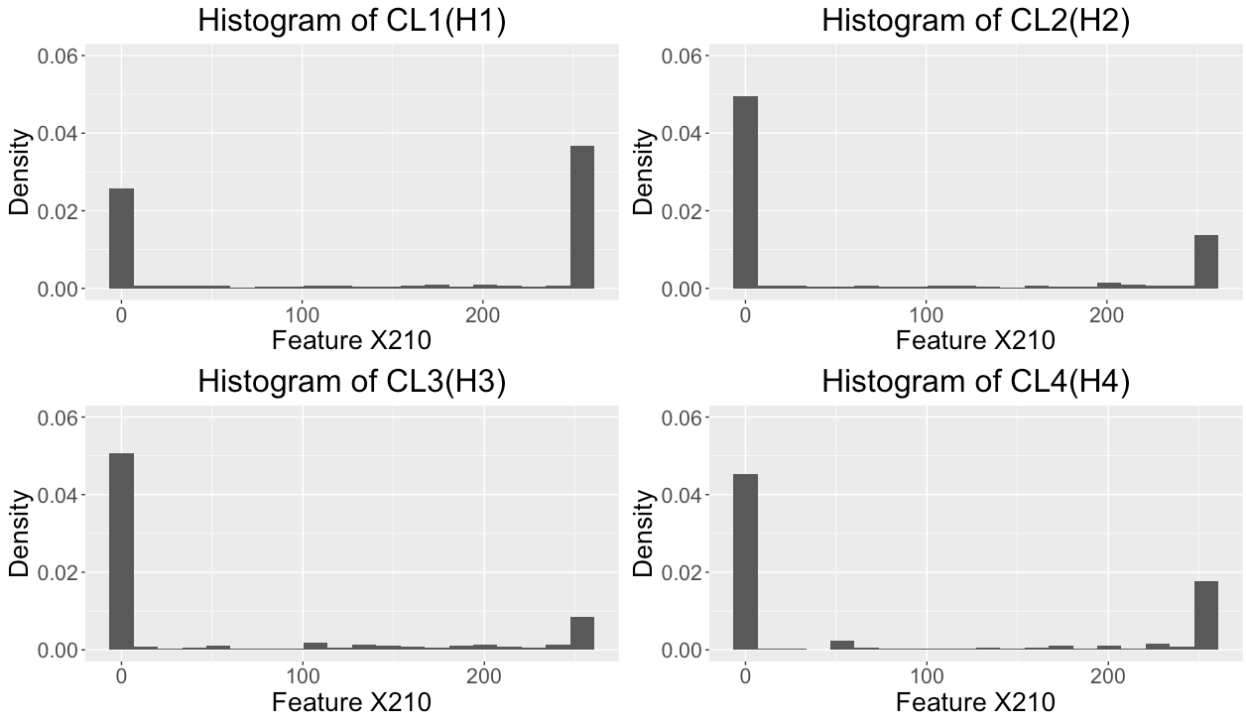


*Figure 1 Histogram of X210 feature of each class. Up Left: GILL (Class 1), Up Right: LEELAWADEE (Class 2), Down Left: ROMAN (Class 3), Down Right: TECHNIC (Class 4)*

However, visual comparison is not enough. Thus, the two-samples Kolmogorov-Smirnov tests which are used to compare the cumulative distribution between two classes need to be conducted to support the interpretation from histograms.

### 1.1.4 *Kolmogorov-Smirnov test*

All the two-samples ks-test has Null Hypothesis as below:

$$H_0: Class\ A \text{ and } Class\ B \text{ have same distributions}$$

$H_0$ can be rejected if the p-value is smaller than $\alpha$. Table 4 shows the results from the two-samples ks-tests and the p-value for each pair of four classes.

*Table 4 4 Kolmogorov-Smirnov Tests between two classes*

| 1st Class | 2nd Class | p-value | p-value < 0.1 |
|---|---|---|---|
| CL1 | CL2 | 0.00E+00 | TRUE |
| CL1 | CL3 | 0.00E+00 | TRUE |
| CL1 | CL4 | 0.00E+00 | TRUE |
| CL2 | CL3 | 7.42E-04 | TRUE |
| CL2 | CL4 | 8.71E-04 | TRUE |
| CL3 | CL4 | 2.85E-09 | TRUE |

Since all p-values are essentially close to 0 and significantly smaller than $\alpha$, all the Null Hypothesis, $H_0$ , can be rejected which means that all the classes have different distribution at Feature X210.

Overall, we can conclude that Feature X210 has a good discriminate power between the four classes.

## 1.2 Correlation between each feature

### 1.2.1 *Correlation Value for 400 features and top 10 pairs of correlated pixels*

Table 5 shows the top 10 highest correlation values among 400 features. It can be obviously observed that 10 out of 10 of these paired pixels are located adjacent to each other.

It is reasonable because any kind of character in any font will have thickness and to represent the thickness of a font, the adjacent pixels will have to be both in black (value = 255). Meanwhile, any character will have large white space. To represent this big white space, several adjacent pixels will have to be all in white (value = 0).

*Table 5 Top 10 highest correlated pixels*

| | Pixel 1 | Pixel 2 | Correlation |
|---|---|---|---|
| 1 | (7,19) | (8,19) | 0.93 |
| 2 | (9,19) | (10,19) | 0.93 |
| 3 | (9,18) | (10,18) | 0.93 |
| 4 | (19,9) | (19,10) | 0.93 |
| 5 | (11,0) | (12,0) | 0.93 |
| 6 | (9,0) | (10,0) | 0.93 |
| 7 | (5,9) | (5,10) | 0.93 |
| 8 | (18,9) | (18,10) | 0.93 |
| 9 | (12,1) | (13,1) | 0.93 |
| 10 | (10,0) | (11,0) | 0.93 |

### 1.2.2 *Correlation Value* Comparison

In contrast, two pixels which are far away from each other will have less chance to have same color since different character will have different pattern. Because the data set is about the "font", it is reasonable to take letter "L" and "H" as examples.

If the letters are cut into 20 x 20 pixels as shown in Figure 2, the left middle pixel which is highlighted by yellow box is black but the right middle pixel which is highlighted in red is white for "L" but for "H", both are black. There are some letters fall between either the first category as letter "L" or second category as letter "H". Thus, it is difficult to predict the color of a pixel by using a pixel far away from it.

Refer to Section 1.2.1, it can be observed that if the two pixels are adjacent, it is highly possible that they will be both black (highlighted in green in Figure 2) or both white (highlighted in blue in Figure 2) or one white and one black if they are at the transition zone of the letter and the white space.

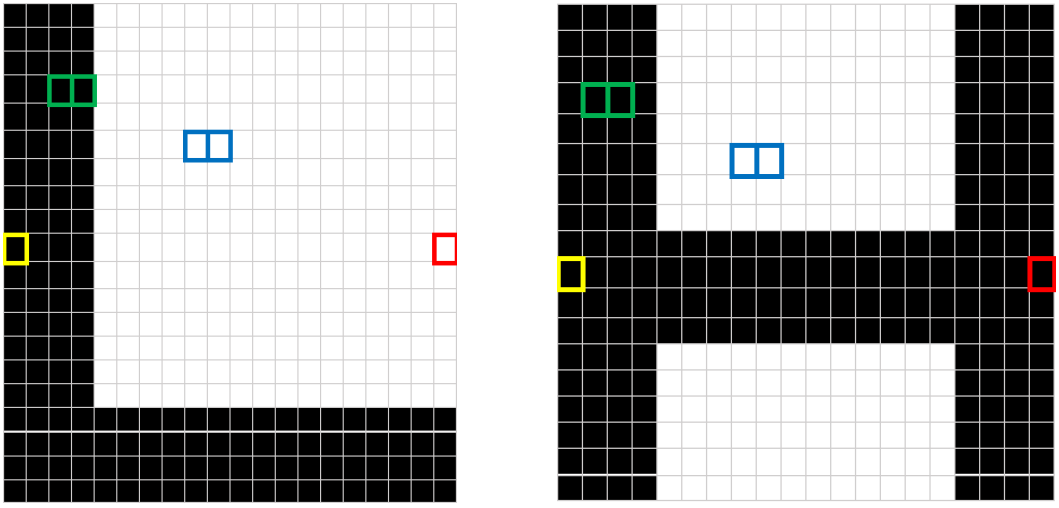

*Figure 2 Letter "L" and "H" as examples to explain the correlation value*

Table 6 shows a good example on this comparison. While comparing the correlation value of Pixel(6, 13) and Pixel(7, 13) which are next to each other and correlation value of Pixel(6, 1) and Pixel(6, 18) which are apart, higher correlation appears to the former one and lower correlation happens to the latter one.

*Table 6 Correlation Value between Pixel(6, 13) and Pixel(7, 13) and between Pixel(6, 1) and Pixel(6, 18)*

| Pixel 1 | Pixel 2 | Correlation |
|---|---|---|
| (6,13) | (7,13) | 0.82 |
| (6,1) | (6,18) | 0.58 |

## 1.3 Standardize the Data Set

To have more meaningful distance (eliminate the unit) between two cases while apply kNN algorithm in next section, all the features will be standardized as below.

$$SX(n) = \frac{[X(n) - Mean(X)]}{Standard\ Deivation(X)}$$

The data after standardizing will have size of 5411 x 400. Table 7 only shows the first 6 cases with first 7 features.

*Table 7 First 6 cases with First 7 features of Standardized Data*

| | r0c0 | r0c1 | r0c2 | r0c3 | r0c4 | r0c5 | r0c6 |
|---|---|---|---|---|---|---|---|
| 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 2 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 3 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 4 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 5 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.9485551 | -1.0578108 |
| 6 | 1.62125672 | 0.36714278 | -0.1556462 | -0.742095 | -0.8469891 | -0.9485551 | -1.0578108 |

## 1.4 Generate the complete Data Set with Response Variable and Features

Since kNN algorithm is a supervised classifier, it is important to have the original response variable for the data set to calculate the accuracy of prediction. Thus, each case will be assigned a number to represent its class number as the response variable. This column will be named as "TRUC" and combined with the standardized data set to get the complete data set to be ready for applying kNN in next step.

The standardized data together with true class column i.e., TRUC will have dimension of 5411 x 401. Table 8 only shows the first 6 cases with True Class and first 7 features.

*Table 8 First 6 cases with True Class and First 7 features of Standardized Data*

| | TRUC | r0c0 | r0c1 | r0c2 | r0c3 | r0c4 | r0c5 | r0c6 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 2 | 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 3 | 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 4 | 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.7320565 | -0.2619737 |
| 5 | 1 | -0.6364723 | -0.691387 | -0.7248442 | -0.778314 | -0.8469891 | -0.9485551 | -1.0578108 |
| 6 | 1 | 1.62125672 | 0.36714278 | -0.1556462 | -0.742095 | -0.8469891 | -0.9485551 | -1.0578108 |

# 2 Data Automatic Classification

## 2.1 Split the Standardize Data Set into Training Set and Test Set

To reduce the bias toward any class for either training set or test set, evenly and randomly sampling the data shall be performed. The complete test set will be randomly selected from 20% of all the cases in each class and then combine as a complete test set. The remaining 80% cases in each class are combined as complete training set. Table 9 shows that there are 1080 cases of test set and 4331 cases of training set.

*Table 9 Size of Test Set and Training Set*

| | Test | Train |
|---|---|---|
| CL1 | 291 | 1168 |
| CL2 | 275 | 1103 |
| CL3 | 238 | 956 |
| CL4 | 276 | 1104 |
| Total | 1080 | 4331 |

## 2.2 Preliminary k-Nearest Neighbor (kNN) Automatic Classifier

Both training and test sets are used to determine the best value of $k$. Since there are no pre-defined statistical methods to find the most favorable value of $k$, this section and next section will repeat the kNN Automatic Classifier to find the best $k$ which will end up with higher prediction accuracy i.e., performance in both training and test sets for the model without overfitting and underfitting. However, there are some empirical choices for the k as below:

1. $k_{Max} \leq 50$
2. $k_{Max} \leq \sqrt{size\ of\ training\ set}$

Since the complete training data set has 4331 cases, the first empirical choice is going to be applied. Thus, the preliminary kNN Automatic Classifier will be conducted for $k = 5, 10, 15, 20, 30, 40, 50$.

From Figure 3 below, both training and test performance curves decrease as $k$ goes up with peak performance at $k = 5$. There is no overfitting or underfitting for any $k$. This is a good sign. Thus, it is wise to conduct a second kNN Automatic Classifier with $k$ between 3 to 10 ($k = 1$ or 2 are not good choices for any situation) to check if there is any value of $k$ that can get a higher performance than $k = 5$.

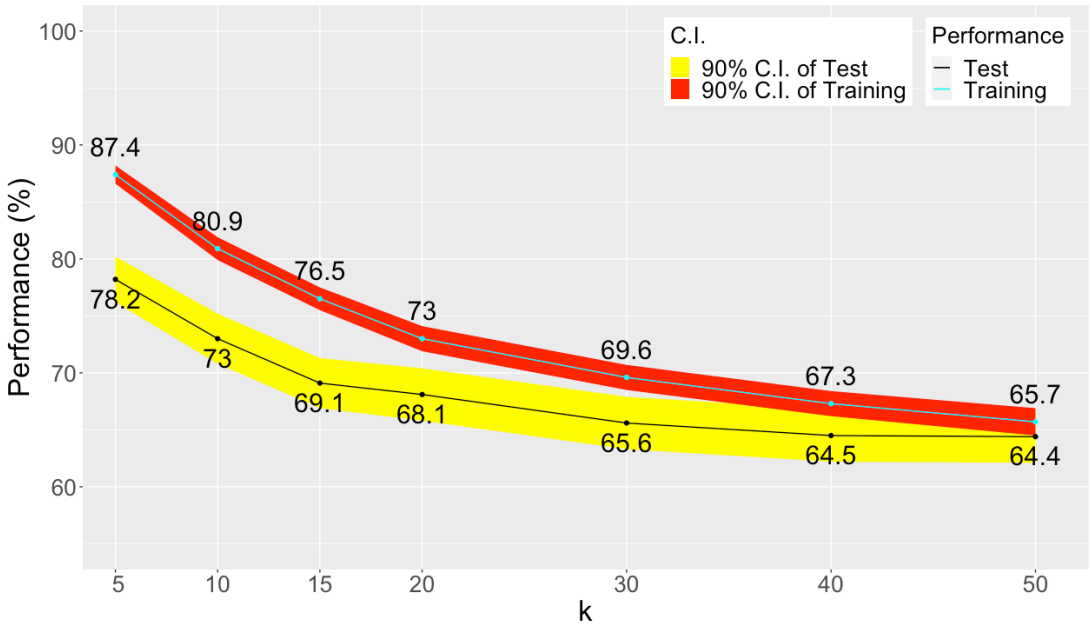


*Figure 3 Training and Test Performance with k = 5, 10, 15, 20, 30, 40, 50*

## 2.3 Secondary k-Nearest Neighbor (kNN) Automatic Classifier

### *2.3.1 Repeat the kNN process*

From Figure 4 below, it can be observed that test performance reaches around $80\%$ while $k = 3, 4, 5$. There is no overfitting and underfitting for any $k$. Besides, 90% confidence interval of the test performance reach the highest point when $k = 3$. Thus, we choose $k = 3$ as the best $k$ for these data set.

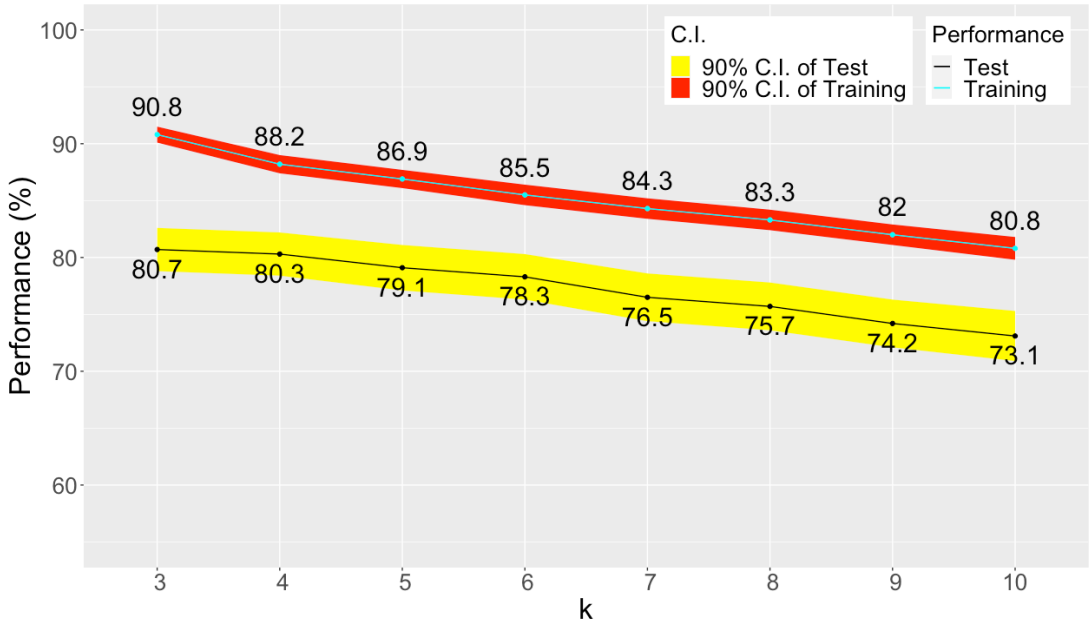


*Figure 4 Training and Test Performance with k = 3, 4, 5, 6, 7, 8, 9, 10*

### 2.3.2 *Determine the best k*

Redo the kNN Automatic Classifier again with $k = 3$ and compute the test performance, 90% confidence interval of the test performance, and the computing time for future comparison.

$$90\%\ confidence\ interval = Test\ Performance \pm 1.6 \times \text{Margin on Performance}$$

where

- $\text{Margin on Performance} = \sqrt{\frac{Test\ Performance \times (1 - Test\ Performance)}{data\ size}}$

Margin on Performance will decrease as the data set size increase. Table 10 is the Performance Table for $k = 3$. The performing computer is Apple M1 chip (3.2 GHz) with 16 GB of RAM.

*Table 10 Performance Table for Test Set when k = 3*

| | Computing Time (s) | Test Performance (%) | 90% C.I. (%) |
|---|---|---|---|
| Best k | 2.8 | 80.8 | [ 78.9 , 82.7 ] |

## 2.4 Confusion Matrix

The confusion Matrix are built to have a better understanding about the correct classification and misclassification rate of each class. Table 11 shows that Class 4 has the best performance among all the classes. Meanwhile, Class 2 has the lowest performance.

*Table 11 Confusion Table in Percentage when k = 3.*

| | True CL1 | True CL2 | True CL3 | True CL4 |
|---|---|---|---|---|
| Pred CL1 | 82.8 | 10.5 | 4.2 | 6.2 |
| Pred CL2 | 3.4 | 65.5 | 3.8 | 1.8 |
| Pred CL3 | 2.4 | 8.4 | 83.6 | 0.4 |
| Pred CL4 | 11.3 | 15.6 | 8.4 | 91.7 |

Figure 5 below shows the 90% confidence interval for all the 4 classes. In conclusion, we have 90% confidence that the performance will be in below ranges for each class:

1. Class 1: Between 79.3% to 86.3%
2. Class 2: Between 60.9% to 70.1%
3. Class 3: Between 79.8% to 87.4%
4. Class 4: Between 89.0% to 94.4%

It is obvious that Class 1 and Class 3 have big portion of overlap for the confidence interval which means that even though the test performance of Class 3 is higher than Class 2, there will be some other data set cause the test performance of Class 2 higher than Class 3.

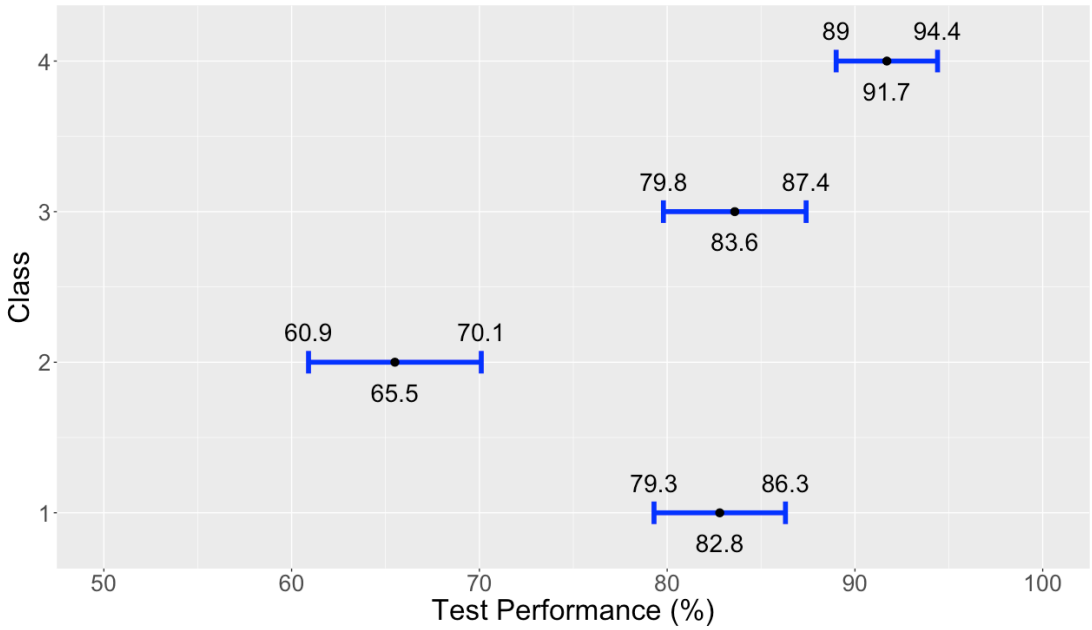


*Figure 5 90% Confidence Interval of Each Class Test Performance*

## 2.5 Misclassified Cases

Since the best value of $k$ is 3, the kNN algorithm is to find out the 3 nearest neighbor of each case and figure out which classes are these neighbors in. Then, the case will then be classified as the class which the majority of its 3 nearest neighbors from. Thus, to identify why these misclassified cases are misclassified, it is wise to list all the $k$ nearest neighbors for these cases. Table 12 shows all the cases which are under Class 2 but misclassified as Class 1, 3 or 4. The first case of each ERR list are picked and computed the distance between it and all cases.

*Table 12 Cases which have True Class = 2 but misclassified as other classes. Left: Misclassified as Class 1. Middle: Misclassified as Class 3. Right: Misclassified as Class 4. The highlighted cases are selected to interpret the misclassification situation.*

| Test Result | Case Number | TRUC |
|---|---|---|
| 1 | 2161 | 2 |
| 1 | 1468 | 2 |
| 1 | 2182 | 2 |
| 1 | 2737 | 2 |
| 1 | 2717 | 2 |
| 1 | 1928 | 2 |
| 1 | 2645 | 2 |
| 1 | 2355 | 2 |
| 1 | 2298 | 2 |
| 1 | 2531 | 2 |
| 1 | 2771 | 2 |
| 1 | 2171 | 2 |
| 1 | 2173 | 2 |
| 1 | 1666 | 2 |
| 1 | 2790 | 2 |
| 1 | 2175 | 2 |
| 1 | 1606 | 2 |
| 1 | 2048 | 2 |
| 1 | 2158 | 2 |
| 1 | 1494 | 2 |
| 1 | 2134 | 2 |
| 1 | 2763 | 2 |
| 1 | 2628 | 2 |
| 1 | 2824 | 2 |
| 1 | 2142 | 2 |
| 1 | 2743 | 2 |
| 1 | 2295 | 2 |
| 1 | 2775 | 2 |
| 1 | 1486 | 2 |

| Test Result | Case Number | TRUC |
|---|---|---|
| 3 | 1886 | 2 |
| 3 | 2033 | 2 |
| 3 | 1951 | 2 |
| 3 | 1540 | 2 |
| 3 | 1466 | 2 |
| 3 | 2484 | 2 |
| 3 | 1591 | 2 |
| 3 | 1599 | 2 |
| 3 | 1777 | 2 |
| 3 | 2185 | 2 |
| 3 | 1552 | 2 |
| 3 | 1955 | 2 |
| 3 | 2793 | 2 |
| 3 | 2289 | 2 |
| 3 | 1845 | 2 |
| 3 | 1784 | 2 |
| 3 | 2091 | 2 |
| 3 | 2253 | 2 |
| 3 | 2117 | 2 |
| 3 | 2575 | 2 |
| 3 | 1908 | 2 |
| 3 | 2288 | 2 |
| 3 | 2591 | 2 |

| Test Result | Case Number | TRUC |
|---|---|---|
| 4 | 1566 | 2 |
| 4 | 1992 | 2 |
| 4 | 2582 | 2 |
| 4 | 2815 | 2 |
| 4 | 2681 | 2 |
| 4 | 2059 | 2 |
| 4 | 2829 | 2 |
| 4 | 2836 | 2 |
| 4 | 2054 | 2 |
| 4 | 2411 | 2 |
| 4 | 2557 | 2 |
| 4 | 2685 | 2 |
| 4 | 2255 | 2 |
| 4 | 2833 | 2 |
| 4 | 1970 | 2 |
| 4 | 2786 | 2 |
| 4 | 2101 | 2 |
| 4 | 2488 | 2 |
| 4 | 1683 | 2 |
| 4 | 2713 | 2 |
| 4 | 2102 | 2 |
| 4 | 1976 | 2 |
| 4 | 1512 | 2 |
| 4 | 2741 | 2 |
| 4 | 2016 | 2 |
| 4 | 1799 | 2 |
| 4 | 2581 | 2 |
| 4 | 2804 | 2 |
| 4 | 2118 | 2 |
| 4 | 2113 | 2 |
| 4 | 2143 | 2 |
| 4 | 1562 | 2 |
| 4 | 2801 | 2 |
| 4 | 2658 | 2 |
| 4 | 2624 | 2 |
| 4 | 1746 | 2 |
| 4 | 2688 | 2 |
| 4 | 2807 | 2 |
| 4 | 2286 | 2 |
| 4 | 1624 | 2 |
| 4 | 2802 | 2 |
| 4 | 2041 | 2 |
| 4 | 1556 | 2 |

As per the description of knn function in R, *If there are ties for the kth nearest vector, all candidates are included in the vote.* Thus, the $k$ nearest neighbor may include more than number of $k$.

Table 13 to Table 15 shows that $k$ nearest neighbor for each misclassified case in ERR21, ERR23, and ERR24.

*Table 13 3 nearest neighbors to the case#2161 whose true class is 2 but predicted as class 1*

| | Case # | Distance | TRUC |
|---|---|---|---|
| Neighbor 1 | 1472 | 6.60166414 | 2 |
| Neighbor 2 | 884 | 10.0779043 | 1 |
| Neighbor 3 | 1176 | 15.7395261 | 1 |

*Table 14 3 nearest neighbors to the case#1886 whose true class is 2 but predicted as class 3*

| | Case # | Distance | TRUC |
|---|---|---|---|
| Neighbor 1 | 2575 | 7.92002605 | 2 |
| Neighbor 2 | 3782 | 19.0384011 | 3 |
| Neighbor 3 | 3057 | 19.4566534 | 3 |

*Table 15 3 nearest neighbors to the case#1566 whose true class is 2 but predicted as class 4*

| | Case # | Distance | TRUC |
|---|---|---|---|
| Neighbor 1 | 2255 | 8.15466677 | 2 |
| Neighbor 2 | 1562 | 11.577114 | 2 |
| Neighbor 3 | 4054 | 12.976193 | 4 |
| Neighbor 4 | 4298 | 12.976193 | 4 |

In Table 15, noted that Case# 1566 has 4 nearest neighbors even though the chosen $k$ value is 3 due to the same distances for 3$^{rd}$ and 4$^{th}$ neighbor. Since there is a tie for the vote, kNN model will randomly pick from Class 2 or Class 4 which means that if we redo the kNN again, this case may not be misclassified.

In conclusion, most of the nearest neighbors for all the 3 misclassified cases don't belong to Class 2. That's the reason of them being misclassified.

# 3 Principal Component Analysis (PCA)

Dimension reduction is often a key step in automatic classification. One would like to have a new number of features much less than the number of features in the original data, but the new features should still provide as much information as the original ones. Principal Component Analysis constructs new features as linear combination of the original features. Also, the new features are standardized and decorrelated to avoid redundancy.

## 3.1 Eigenvalue and Eigenvector

To perform the PCA, eigenvalue and eigenvector are calculated. Since there will be information loss between the data compression and decompression, it is important to keep as many information as possible. Thus, the Mean Squared Error of Decompression (MSE) i.e., information loss shall be as small as possible. Percentage of Explain Variance (PEV) which is a function of MSE is set to be 90% in this report.

$$PEV(r) = 1 - \frac{MSE(r)}{m} = \frac{\sum_{i=1}^{r} L_i}{m}$$

where

- $m$ is the number of original features
- $L_i$ is the sorted eigenvalue

In our case, there are $400$ features in the original data set. A $400 \times 400$ eigenvector will be computed together with $400$ eigenvalue. Figure 6 shows the eigenvalue (L) versus m.

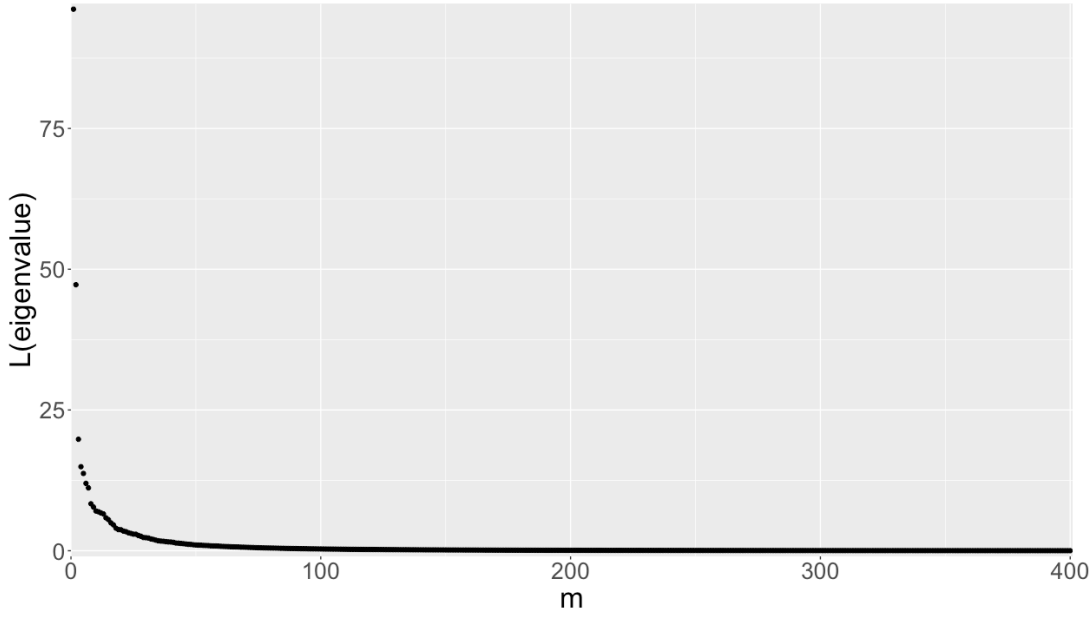


*Figure 6 Eigenvalues vs m*

## 3.2 Percentage of Explained Variance (PEV)

In this report, PEV is set to be equal to or larger than 90%. Thus, the smallest integer "$r$" such that $PEV(r) \geq 90\%$ is 62. Figure 7 shows the PEV versus m.

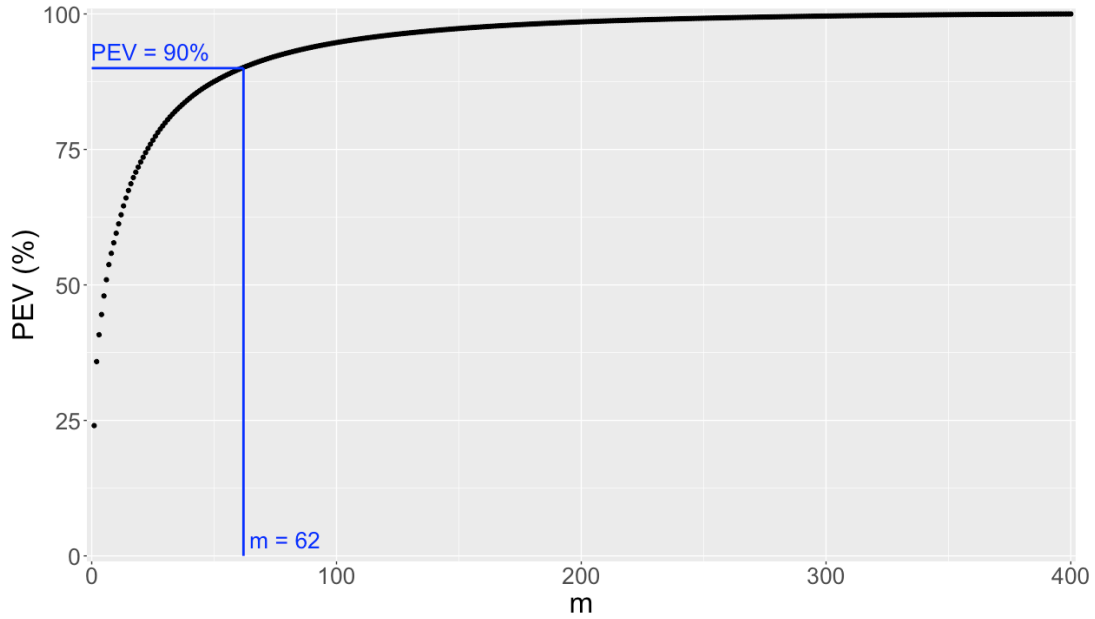


*Figure 7 Percentage of Explained Variance (PEV) vs m*

## 3.3 Apply Principal Component Analysis

The new features after Principal Component Analysis (PCA) are produced by the multiplication of the Standardize Data (SDATA) and the Eigenvector (W).

$$ZDATA_{N\times 400} = SDATA_{N\times 400} \times W_{400\times 400}$$

The final dimension of the data after PCA, ZDATA, is 5411 x 400.

However, since we have $r = 62$ to get $PEV \geq 90\%$, only the first 62 principal components will be taken from ZDATA to have a data set with size of 5411 x 62. Together with the TRUC column, the final data set with size of 5411 x 63 is generated.

Since the data after PCA has dimension of 5411 x 63, Table 16 only shows the first 6 cases with TRUC and first 7 features in below.

*Table 16 First 6 cases with First 7 features of Data after PCA*

| | TRUC | V1 | V2 | V3 | V4 | V5 | V6 | V7 |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 20.3309051 | -1.2448531 | -2.770295 | 2.10850216 | 2.09203383 | -1.4044534 | 2.92570844 |
| 2 | 1 | 17.8302992 | -0.657016 | -4.1377791 | 3.397947 | -0.1339552 | 0.72800757 | 2.77131667 |
| 3 | 1 | 20.3309051 | -1.2448531 | -2.770295 | 2.10850216 | 2.09203383 | -1.4044534 | 2.92570844 |
| 4 | 1 | 17.8302992 | -0.657016 | -4.1377791 | 3.397947 | -0.1339552 | 0.72800757 | 2.77131667 |
| 5 | 1 | -5.7027004 | 3.04603417 | -5.9035089 | -1.2941225 | 4.68990459 | 0.80037051 | -2.4853168 |
| 6 | 1 | 7.2542036 | 1.27501693 | 1.64668112 | 1.47610997 | -5.8213176 | -1.2358181 | 1.10017072 |

## 3.4 Apply kNN Automatic Classifier to Data Set after Principal Component Analysis

To know the power of PCA and the performance against the original data, the newly obtained PCA data set will be applied to kNN automatic classifier at best $k$ as concluded in Section 2.3. Figure 8 shows the comparison of test performance between original data set and data set after PCA. It is observed that two kinds of data sets have a big portion of

overlapping in their confidence intervals which means doing PCA will not compromise too much on the performance for kNN.

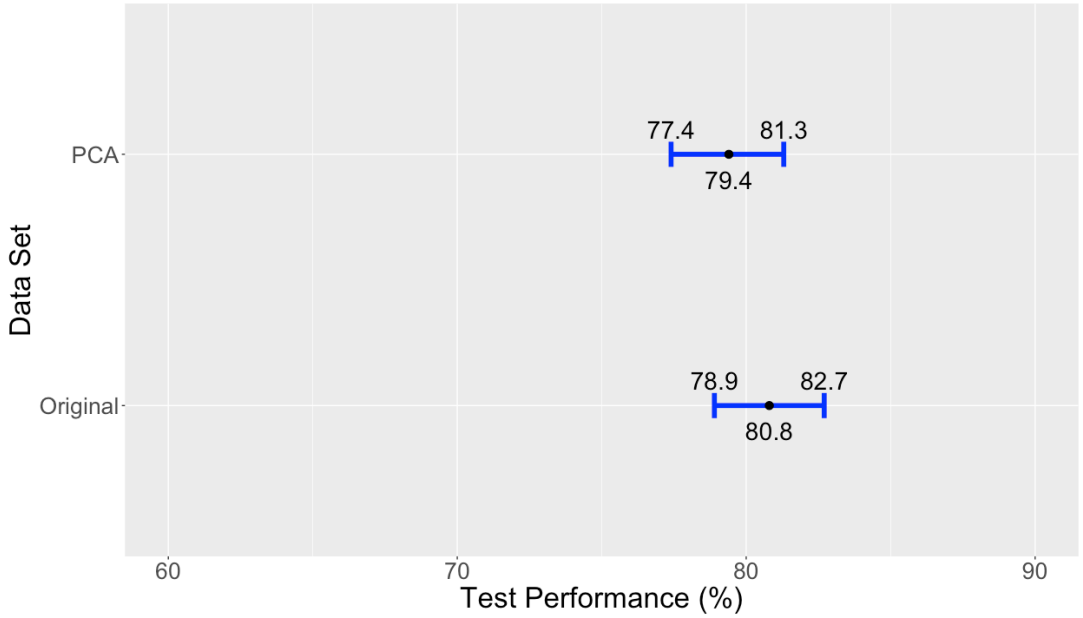


*Figure 8 90% Confidence Interval of Test Performance of Original Data and PCA Data*

Table 17 shows that the Test Performance decreased by 1.4% while it can save up to 72.3% of the computing time which is a good trade off.

*Table 17 Comparison between Standardize Data Set and Data after Principal Component Analysis*

| | Computing Time (s) | Test Performance (%) |
|---|---|---|
| Original | 2.8 | 80.8 |
| PCA | 0.9 | 79.4 |

## 3.5 Visualize the First Two Component

From PEV versus m in Figure 7, we can know that $PEV(2) \cong 36\%$. Therefore, first 2 principal components can only explain about 36% of information provided by the features, which is not powerful enough to accurately separate the classes on a 2D diagram. However, we may still get some information or surprise from these plots.

In Figure 9, we can observe a good separation between Class 1 and Class 2. Most of the Class 1 spread at right side and Class 2 spread at left side of the figure. However, there are still some overlapping around V1 = 0. This may cause the difficulty of classification.

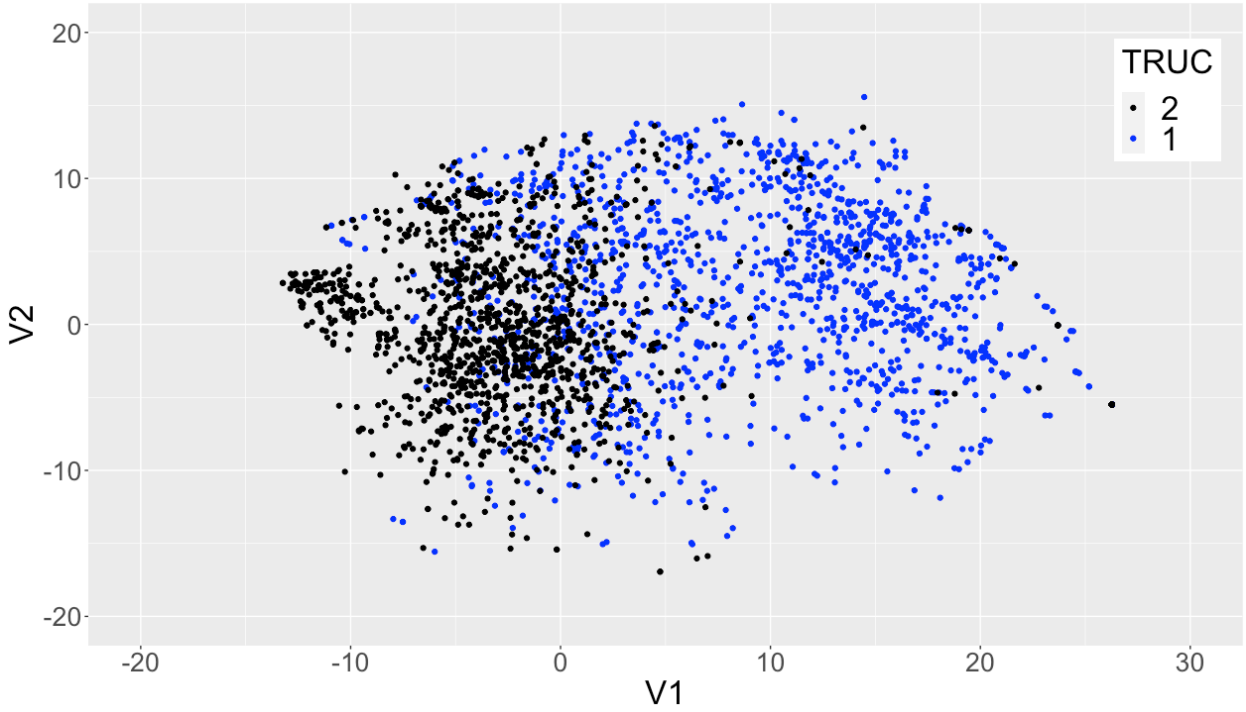


*Figure 9 Class 2 and Class 1 Distribution on a planar graph based on first 2 principal components, V1 and V2.*

In Figure 10, we can observe Class 3 and Class 2 have a slightly worse separation compared to Class 1 and Class 2, since there are some points of Class 2 which are covered by the points for Class 3 around V1 from -15 to 0 and V2 from -5 to 5.

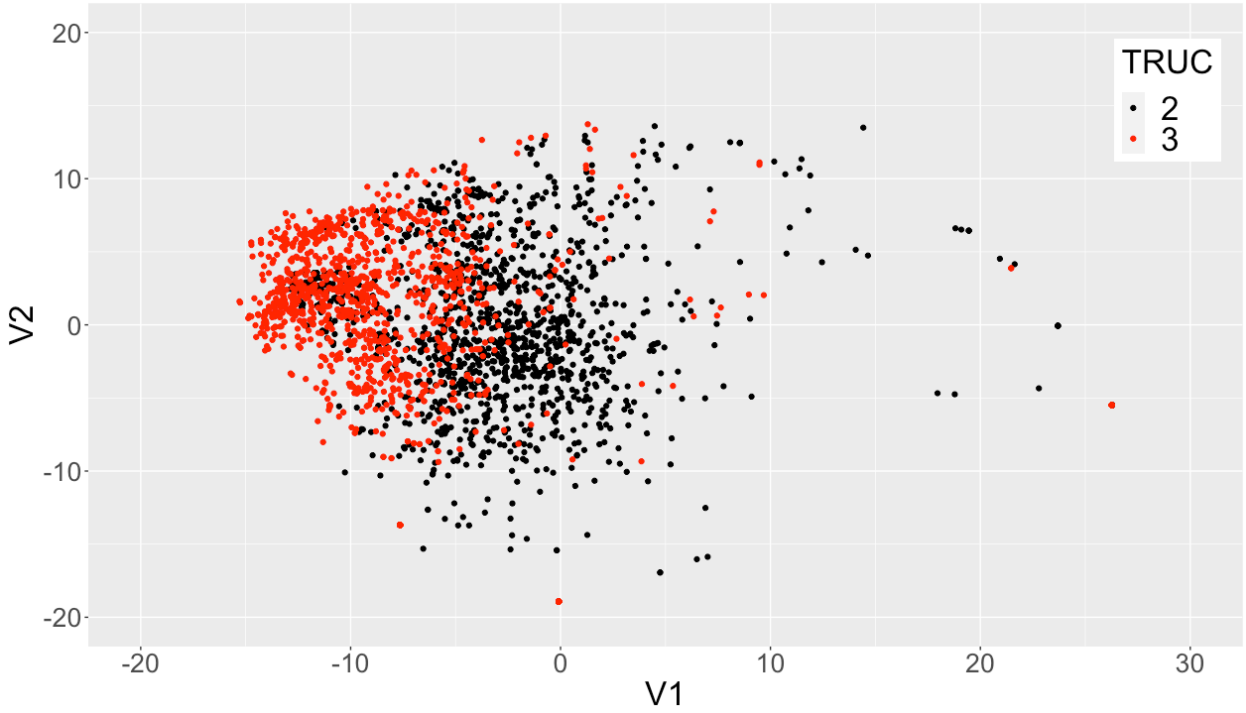


*Figure 10 Class 2 and Class 3 Distribution on a planar graph based on first 2 principal components, V1 and V2.*

In Figure 11, we can observe a poor separation between Class 2 and Class 4. It is difficult to distinguish the two classes from the plot. Most of the Class 2 and Class 4 are spread between V1 from -10 to 5 and V2 from -10 to 10. This is not a surprise that among all the misclassified situations, there are 15.6% of the Class 2 cases being classified as Class 4 which is the highest error rate in the entire confusion matrix in Section 2.4.

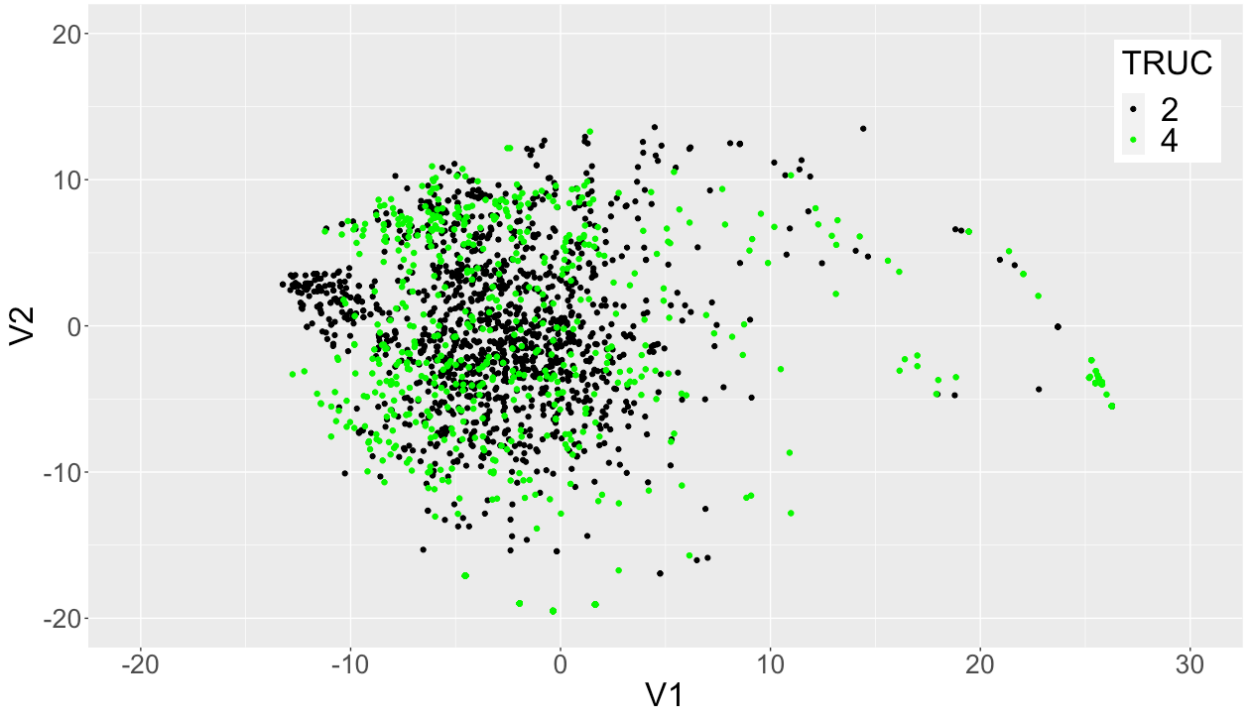


*Figure 11 Class 2 and Class 4 Distribution on a planar graph based on first 2 principal components, V1 and V2.*

## 3.6 Visualize misclassified cases

Figure 12 demonstrates that the misclassified Class 2 points seem to correspond with the range and density of the points in the other classes. For example, the Class 2 points that were misclassified as Class 3 are in the net area of V1 from -15 to 0 and V2 from -10 to 10, which is where the Class 3 points are mostly concentrated. Points misclassified as Class 1 and Class 4 also are in areas with high concentrations of their incorrect classes.

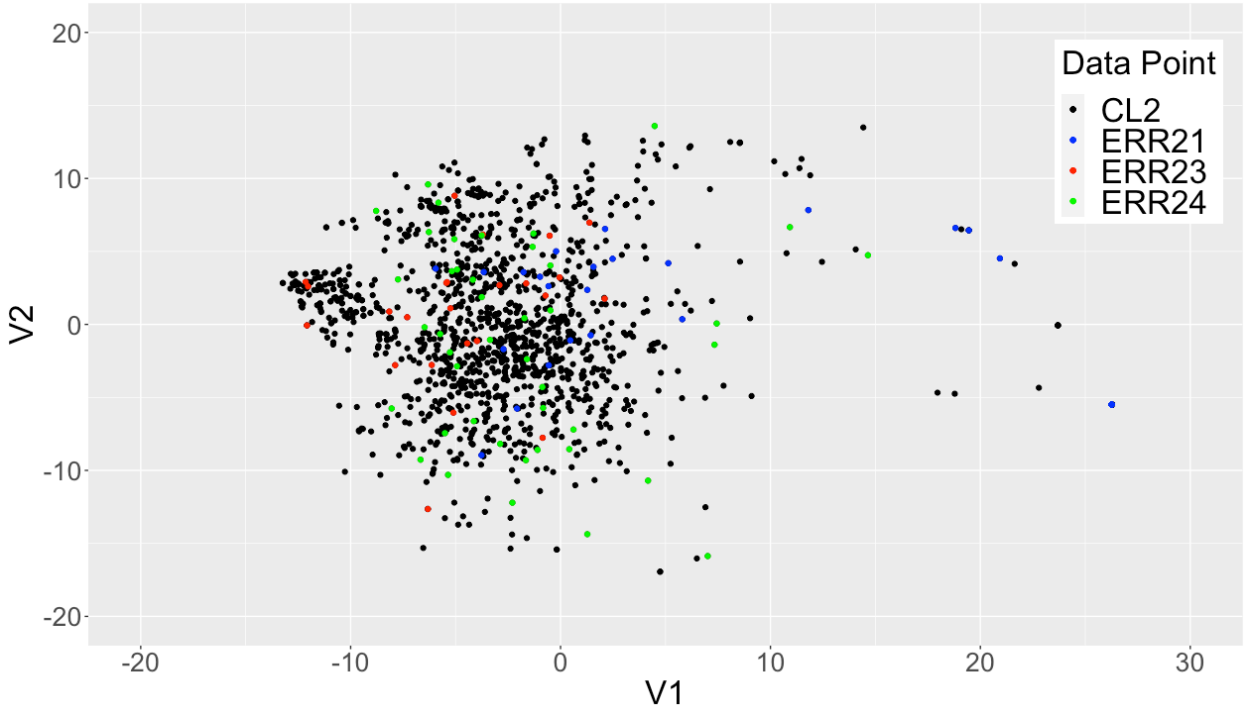


*Figure 12 All the Class 2 cases on the planar projection and highlighted cases which shall be belongs to Class 2 but misclassified as Class 1, 3, or 4*

As mentioned before, since $PEV(2) \cong 36\%$, there are a lot of information is missing while we try to classify the point visually on a 2-dimensional plot. Even if we plot them into 3-dimensional, i.e., $PEV(3)$ we still have only around 40% of the information. It is hard to visually identify the which class shall each case belongs to. So, the best way to classify these data is using the kNN algorithm which means to calculate the distance between cases and figure out which are the k-nearest neighbor so that it can vote for the classification and interpret mathematically and easily.

## R Script

```
rm(list = ls())
# ================================================================================
library(dplyr)
library(class)
library(reshape2)
library(ggplot2)
library(amap)
library(caret)
library(gridExtra)
# ================================================================================
file_wd = "/Users/jerrychien/Desktop/OneDrive - University Of Houston/6350 - Statisti
cal Learning and Data Mining/HW/HW 2/fonts"
selected_file = c("GILL.csv", "LEELAWADEE.csv", "ROMAN.csv", "TECHNIC.csv") # Each cl
ass around 1300. Report now base on this case.
file_class = data.frame()
for (i in c(1:4)){
  file_class[paste("Class", i), "File"] = selected_file[i]
}

# Read the 4 csv files and assign to 4 different data frame and delete the unnecessar
y columns.
# Filter all the 4 data frames with strength = 0.4 and italic = 0 and assign them to
4 different class.
col_to_be_skipped = c("fontVariant", "m_label", "orientation", "m_top", "m_left", "or
iginalH", "originalW", "h", "w")
CL1 = filter(select(read.csv(paste(file_wd, "/", selected_file[1], sep = ""), skipNul
 = T), -col_to_be_skipped), strength == 0.4 & italic == 0)[, -c(1, 2, 3)]
CL2 = filter(select(read.csv(paste(file_wd, "/", selected_file[2], sep = ""), skipNul
 = T), -col_to_be_skipped), strength == 0.4 & italic == 0)[, -c(1, 2, 3)]
CL3 = filter(select(read.csv(paste(file_wd, "/", selected_file[3], sep = ""), skipNul
 = T), -col_to_be_skipped), strength == 0.4 & italic == 0)[, -c(1, 2, 3)]
CL4 = filter(select(read.csv(paste(file_wd, "/", selected_file[4], sep = ""), skipNul
 = T), -col_to_be_skipped), strength == 0.4 & italic == 0)[, -c(1, 2, 3)]
CL_list = c("CL1", "CL2", "CL3", "CL4")
# Print out the size of each class and the total size of 4 classes.
class_size = data.frame()
N = 0
for (i in CL_list){
  class_size[i, "Size"] = dim(get(i))[1]
  N = N + dim(get(i))[1]
}
class_size["Total", "Size"] = N
DATA = rbind(CL1, CL2, CL3, CL4) # Combine all the 4 classes and assign to a data fra
me.

# ================================================================================
# Problem 1
# 1.1
X210_mean = data.frame() # Calculate the mean value of feature X210 for each class.
for (i in CL_list){
  X210_mean[i, "Mean"] = round(mean(get(i)[, 210]), 1)
}
```

```
t_test = data.frame() # Conduct t-test for 4 classes.
paired_list = combn(CL_list, 2)
for (i in 1:ncol(paired_list)){
  t_test[i, "1st Class"] = paired_list[, i][1]
  t_test[i, "2nd Class"] = paired_list[, i][2]
  t_test[i, "p-value"] = formatC(t.test(get(paired_list[, i][1])[, 210], get(paired_l
ist[, i][2])[, 210])[["p.value"]], format = "e", digits = 2)
  t_test[i, "p-value < 0.1"] = t.test(get(paired_list[, i][1])[, 210], get(paired_lis
t[, i][2])[, 210])[["p.value"]] < 0.1
}

for (i in c(1:4)){ # Plot the histogram of Feature X210 of each class.
  assign(paste("H", i, sep = ""),
         ggplot(data = get(paste("CL", i, sep = ""))) +
           geom_histogram(aes(x = r10c9, y = ..density..), bins = 20) +
           ggtitle(paste("Histogram of CL", i, "(H", i, ")", sep = "")) +
           theme(plot.title = element_text(hjust = 0.5), text = element_text(size = 2
0)) +
           ylab("Density") + xlab("Feature X210") + ylim(0, 0.06)
  )
}
grid.arrange(H1, H2, H3, H4, ncol = 2)

ks_test = data.frame() # Conduct KS-test for 4 classes.
for (i in 1:ncol(paired_list)){
  ks_test[i, "1st Class"] = paired_list[, i][1]
  ks_test[i, "2nd Class"] = paired_list[, i][2]
  ks_test[i, "p-value"] = formatC(ks.test(get(paired_list[, i][1])[, 210], get(paired
_list[, i][2])[, 210])[["p.value"]], format = "e", digits = 2)
  ks_test[i, "p-value < 0.1"] = ks.test(get(paired_list[, i][1])[, 210], get(paired_l
ist[, i][2])[, 210])[["p.value"]] < 0.1
}
# 1.2
CORR = cor(DATA[, 1:400]) # Correlation Values
upper_CORR = CORR * upper.tri(CORR)
melted_uCORR = melt(abs(upper_CORR))
melted_ordered_uCORR = melted_uCORR[order(melted_uCORR$value, decreasing = T), ]
top_10_corr_value = round(melted_ordered_uCORR[c(1:10), "value"], 2)
top_10_corr_index = melted_ordered_uCORR[c(1:10), c("Var1", "Var2")]
for (i in 1:10){
  top_10_corr_index[i, "Pixel1"] = gsub("r", "(", top_10_corr_index[i, "Var1"])
  top_10_corr_index[i, "Pixel2"] = gsub("r", "(", top_10_corr_index[i, "Var2"])
  top_10_corr_index[i, "Pixel1"] = gsub("c", ",", top_10_corr_index[i, "Pixel1"])
  top_10_corr_index[i, "Pixel2"] = gsub("c", ",", top_10_corr_index[i, "Pixel2"])
  top_10_corr_index[i, "Pixel1"] = paste(top_10_corr_index[i, "Pixel1"], ")", sep = "
")
  top_10_corr_index[i, "Pixel2"] = paste(top_10_corr_index[i, "Pixel2"], ")", sep = "
")
}

top_10_corr_index[, c("Var1", "Var2")] = NULL
top_10_corr = cbind(top_10_corr_index, "Correlation" = top_10_corr_value)
rownames(top_10_corr) = c(1:10)

corr_compare = data.frame("Pixel1" = c("(6,13)", "(6,1)"), "Pixel2" = c("(7,13)", "
```

```
(6,18)"), "Correlation" = c(round(CORR["r6c13", "r7c13"], 2), round(CORR["r6c1", "r6c
18"], 2)))
rm(CORR)

# 1.3
# Calculate the mean and standard deviation of each feature. Then, standardize each f
eature.
mean_table = data.frame()
sd_table = data.frame()
SDATA = DATA
for (j in colnames(DATA)[1:400]){
  mean_table["Mean", j] = mean(DATA[, j])
  sd_table["Standard Deviation of", j] = sd(DATA[, j])
  SDATA[, j] = (DATA[, j] - mean(DATA[, j])) / sd(DATA[, j])
}

# 1.4
# Assign class number to each case respectively.
TRUC = data.frame()
for (i in CL_list){
  temp = as.data.frame(rep(substring(i, 3), dim(get(i))[1]))
  colnames(temp) = "TRUC"
  TRUC = rbind(TRUC, temp)
}

SDATA_with_TRUC = cbind(TRUC, SDATA)

case = data.frame("Case_Number" = c(1: dim(SDATA)[1]))
final_SDATA = cbind(case, TRUC, SDATA)

# ============================================================================
# Problem 2
# 2.1
# Split the data set into training set and test set
test_set = data.frame()
training_set = data.frame()
size_test_train = data.frame()
set.seed(20211004)
for (i in 1:4){
  temp = filter(final_SDATA, TRUC == i)
  sampled_number = sample(dim(temp)[1], dim(temp)[1] * 0.2)
  assign(paste("sampled_number_for_CL", i, sep = ""), sampled_number)

  temp_test = temp[sampled_number, ]
  temp_train = temp[-sampled_number, ]
  assign(paste("testCL", i, sep = ""), temp_test)
  test_set = rbind(test_set, temp_test)
  assign(paste("trainCL", i, sep = ""), temp_train)
  training_set = rbind(training_set, temp_train)

  size_test_train[paste("CL", i, sep = ""), "Test"] = dim(temp_test)[1]
  size_test_train[paste("CL", i, sep = ""), "Train"] = dim(temp_train)[1]
}
```

```
size_test_train["Total", "Test"] = dim(test_set)[1]
size_test_train["Total", "Train"] = dim(training_set)[1]

# 2.2
# Preliminary kNN
k = c(5, 10, 15, 20, 30, 40, 50)
preliminary_perf_table = data.frame(k , "Training_Perf" = 0, "Test_Perf" = 0)
for (i in 1:length(k)){
  temp_starting_time_training = Sys.time()
  preliminary_training_result = knn(train = training_set[, -c(1,2)], test = training_
set[, -c(1,2)], cl = training_set[, 2], k = k[i])
  temp_end_time_training = Sys.time()
  temp_starting_time_test = Sys.time()
  preliminary_test_result = knn(train = training_set[, -c(1,2)], test = test_set[, -c
(1,2)], cl = training_set[, 2], k = k[i])
  temp_end_time_test = Sys.time()
  preliminary_perf_table[i, "Training_Perf"] = round(sum(preliminary_training_result
== training_set[, 2]) / length(preliminary_training_result), 3) * 100
  preliminary_perf_table[i, "Test_Perf"] = round(sum(preliminary_test_result == test_
set[, 2]) / length(preliminary_test_result), 3) * 100
  preliminary_perf_table[i, "Training_Set_Computing_Time"] = round(temp_end_time_trai
ning - temp_starting_time_training, 1)
  preliminary_perf_table[i, "Test_Set_Computing_Time"] = round(temp_end_time_test - t
emp_starting_time_test, 1)

  temp_standard_error = sqrt((preliminary_perf_table[i, "Test_Perf"] / 100) * (1 - pr
eliminary_perf_table[i, "Test_Perf"] / 100) / dim(test_set)[1])
  preliminary_perf_table[i, "UL_90_Test"] = round((preliminary_perf_table[i, "Test_Pe
rf"] / 100 + 1.6 * temp_standard_error) * 100, 1)
  preliminary_perf_table[i, "LL_90_Test"] = round((preliminary_perf_table[i, "Test_Pe
rf"] / 100 - 1.6 * temp_standard_error) * 100, 1)
  temp_standard_error = sqrt((preliminary_perf_table[i, "Training_Perf"] / 100) * (1
- preliminary_perf_table[i, "Training_Perf"] / 100) / dim(training_set)[1])
  preliminary_perf_table[i, "UL_90_Training"] = round((preliminary_perf_table[i, "Tra
ining_Perf"] / 100 + 1.6 * temp_standard_error) * 100, 1)
  preliminary_perf_table[i, "LL_90_Training"] = round((preliminary_perf_table[i, "Tra
ining_Perf"] / 100 - 1.6 * temp_standard_error) * 100, 1)
}

# Plot the k value against prediction performance.
ggplot(data = preliminary_perf_table) +
  geom_ribbon(aes(x = k, ymax = UL_90_Test, ymin = LL_90_Test, fill = "90% C.I. of Te
st"), alpha = 1) +
  geom_ribbon(aes(x = k, ymax = UL_90_Training, ymin = LL_90_Training, fill = "90% C.
I. of Training"), alpha = 1) +
  geom_line(aes(k, Test_Perf, color = "Test")) +
  geom_point(aes(k, Test_Perf)) +
  geom_line(aes(k, Training_Perf, color = "Training")) +
  geom_point(aes(k, Training_Perf), color = "cyan") +
  theme(text = element_text(size = 25)) +
  geom_text(aes(k, Test_Perf, label = round(Test_Perf, 2)), hjust = 0.5, vjust = 1.5,
 size = 8) +
  geom_text(aes(k, Training_Perf, label = round(Training_Perf, 2)), hjust = 0.5, vjus
t = -1, size = 8) +
  scale_color_manual(name = "Performance", breaks = c("Test", "Training"), values = c
```

```
("black", "cyan")) +
  scale_fill_manual(name = "C.I.", values = c("90% C.I. of Test" = "yellow", "90% C.
I. of Training" = "red")) +
  scale_x_discrete(limits = c(5, 10, 15, 20, 30, 40, 50)) +
  ylab("Performance (%)") + expand_limits(x = c(4,52), y = c(55, 100)) +
  theme(legend.position = c(0.78, 0.90), legend.text = element_text(size = 20), legen
d.title = element_text(size = 20), legend.box = "horizontal") +
  guides(fill = guide_legend(order = 1), colour = guide_legend(order = 2))

# 2.3
# Test k between 3 and 10
k = c(3, 4, 5, 6, 7, 8, 9, 10)
secondary_perf_table = data.frame(k , "Training_Perf" = 0, "Test_Perf" = 0)
for (i in 1:length(k)){
  temp_starting_time_training = Sys.time()
  secondary_training_result = knn(train = training_set[, -c(1,2)], test = training_se
t[, -c(1,2)], cl = training_set[, 2], k = k[i])
  temp_end_time_training = Sys.time()
  temp_starting_time_test = Sys.time()
  secondary_test_result = knn(train = training_set[, -c(1,2)], test = test_set[, -c
(1,2)], cl = training_set[, 2], k = k[i])
  temp_end_time_test = Sys.time()
  secondary_perf_table[i, "Training_Perf"] = round(sum(secondary_training_result == t
raining_set[, 2]) / length(secondary_training_result), 3) * 100
  secondary_perf_table[i, "Test_Perf"] = round(sum(secondary_test_result == test_set
[, 2]) / length(secondary_test_result), 3) * 100
  secondary_perf_table[i, "Training_Set_Computing_Time"] = round(temp_end_time_traini
ng - temp_starting_time_training, 1)
  secondary_perf_table[i, "Test_Set_Computing_Time"] = round(temp_end_time_test - tem
p_starting_time_test, 1)

  temp_standard_error = sqrt((secondary_perf_table[i, "Test_Perf"] / 100) * (1 - seco
ndary_perf_table[i, "Test_Perf"] / 100) / dim(test_set)[1])
  secondary_perf_table[i, "UL_90_Test"] = round((secondary_perf_table[i, "Test_Perf"]
 / 100 + 1.6 * temp_standard_error) * 100, 1)
  secondary_perf_table[i, "LL_90_Test"] = round((secondary_perf_table[i, "Test_Perf"]
 / 100 - 1.6 * temp_standard_error) * 100, 1)

  temp_standard_error = sqrt((secondary_perf_table[i, "Training_Perf"] / 100) * (1 -
secondary_perf_table[i, "Training_Perf"] / 100) / dim(training_set)[1])
  secondary_perf_table[i, "UL_90_Training"] = round((secondary_perf_table[i, "Trainin
g_Perf"] / 100 + 1.6 * temp_standard_error) * 100, 1)
  secondary_perf_table[i, "LL_90_Training"] = round((secondary_perf_table[i, "Trainin
g_Perf"] / 100 - 1.6 * temp_standard_error) * 100, 1)
}

# Plot the k value against prediction performance.
ggplot(data = secondary_perf_table) +
  geom_ribbon(aes(x = k, ymax = UL_90_Test, ymin = LL_90_Test, fill = "90% C.I. of Te
st"), alpha = 1) +
  geom_ribbon(aes(x = k, ymax = UL_90_Training, ymin = LL_90_Training, fill = "90% C.
I. of Training"), alpha = 1) +
  geom_line(aes(k, Test_Perf, color = "Test")) +
  geom_point(aes(k, Test_Perf)) +
  geom_line(aes(k, Training_Perf, color = "Training")) +
```

```
  geom_point(aes(k, Training_Perf), color = "cyan") +
  theme(text = element_text(size = 25)) +
  geom_text(aes(k, Test_Perf, label = round(Test_Perf, 2)), hjust = 0.5, vjust = 1.5,
 size = 8) +
  geom_text(aes(k, Training_Perf, label = round(Training_Perf, 2)), hjust = 0.5, vjus
t = -1, size = 8) +
  scale_color_manual(name = "Performance", breaks = c("Test", "Training"), values = c
("black", "cyan")) +
  scale_fill_manual(name = "C.I.", values = c("90% C.I. of Test" = "yellow", "90% C.
I. of Training" = "red")) +
  scale_x_discrete(limits = c(3, 4, 5, 6, 7, 8, 9, 10)) +
  ylab("Performance (%)") + expand_limits(x = c(3,10), y = c(55, 100)) +
  theme(legend.position = c(0.78, 0.90), legend.text = element_text(size = 20), legen
d.title = element_text(size = 20), legend.box = "horizontal") +
  guides(fill = guide_legend(order = 1), colour = guide_legend(order = 2))
best_k = filter(secondary_perf_table, Test_Perf == max(secondary_perf_table$Test_Per
f))[, "k"]

# Test on best k.
training_result = knn(train = training_set[, -c(1,2)], test = training_set[, -c(1,
2)], cl = training_set[, 2], k = best_k)
start_time = Sys.time()
test_result = knn(train = training_set[, -c(1,2)], test = test_set[, -c(1,2)], cl = t
raining_set[, 2], k = best_k)
end_time = Sys.time()
computing_time = end_time - start_time

# Compute the 90% Confidence Interval of Test Performance
test_perf = sum(test_result == test_set[, 2]) / length(test_result)
best_k_Standard_Error = sqrt(test_perf * (1 - test_perf) / length(test_result))
best_k_CI_UL = test_perf + 1.6 * best_k_Standard_Error
best_k_CI_LL = test_perf - 1.6 * best_k_Standard_Error

best_k_test_perf = data.frame()
best_k_test_perf["Best k", "Computing_Time"] = round(computing_time, 1)
best_k_test_perf["Best k", "Test_Perform (%)"] = round(test_perf, 3) * 100
best_k_test_perf["Best k", "90% C.I."] = paste("[", round(best_k_CI_LL, 3) * 100, ",
", round(best_k_CI_UL, 3) * 100, "]")

# 2.4
# Compute the confusion matrix
confusion_matrix_test = round(prop.table(confusionMatrix(test_result, as.factor(test_
set[, 2]))$table, margin = 2)* 100, 1)
colnames(confusion_matrix_test) = c("True CL1", "True CL2", "True CL3", "True CL4")
rownames(confusion_matrix_test) = c("Pred CL1", "Pred CL2", "Pred CL3", "Pred CL4")
CI_for_each_case = data.frame()
for (i in 1:4){
  Standard_Error = sqrt(confusion_matrix_test[i, i] / 100 * (1 - confusion_matrix_tes
t[i, i] / 100) / dim(filter(test_set, TRUC == i))[1])
  CI_for_each_case[i, "CL"] = i
  CI_for_each_case[i, "CI_LL"] = round(confusion_matrix_test[i, i] / 100 - 1.6 * Stan
dard_Error, 3) * 100
  CI_for_each_case[i, "Test_Performance"] = round(confusion_matrix_test[i, i] / 100,
3) * 100
  CI_for_each_case[i, "CI_UL"] = round(confusion_matrix_test[i, i] / 100 + 1.6 * Stan
```

```
dard_Error, 3) * 100
}

ggplot(data = CI_for_each_case, aes(x = Test_Performance, y = CL)) +
  geom_errorbar(aes(xmin = CI_LL, xmax = CI_UL), width = 0.15, color = "blue", size =
 2) +
  geom_point(size = 3) +
  geom_text(aes(x = Test_Performance, y = CL - 0.2, label = Test_Performance), size =
 7.5) +
  geom_text(aes(x = CI_LL, y = CL + 0.2, label = CI_LL), size = 7.5) +
  geom_text(aes(x = CI_UL, y = CL + 0.2, label = CI_UL), size = 7.5) +
  xlab("Test Performance (%)") + ylab("Class") +
  theme(plot.title = element_text(hjust = 0.5), text = element_text(size = 25)) +
  xlim(50, 100)

# 2.5
# Error List
ERR21 = filter(cbind(test_result, test_set[, c(1, 2)]), TRUC == 2 & test_result == 1)
ERR23 = filter(cbind(test_result, test_set[, c(1, 2)]), TRUC == 2 & test_result == 3)
ERR24 = filter(cbind(test_result, test_set[, c(1, 2)]), TRUC == 2 & test_result == 4)

dist_matrix = as.matrix(Dist(final_SDATA[, -c(1,2)]))
melted_dist_matrix = reshape2::melt(dist_matrix)

if (dim(ERR21)[1] == 0){
  ERR21_case = NULL
} else {
  ERR21_case = ERR21[1, "Case_Number"]
  melted_dist_matrix_ERR21_case = filter(melted_dist_matrix, Var1 == ERR21_case & Var
2 != ERR21_case)
  ordered_melted_dist_matrix_ERR21_case = melted_dist_matrix_ERR21_case[order(melted_
dist_matrix_ERR21_case$value, decreasing = F), ]
  ERR21_case_neighbor = data.frame()
  for (i in c(1:best_k)){
    ERR21_case_neighbor[paste("Neighbor", i), "Case #"] = ordered_melted_dist_matrix_
ERR21_case[i, "Var2"]
    ERR21_case_neighbor[paste("Neighbor", i), "Distance"] = filter(ordered_melted_dis
t_matrix_ERR21_case, Var2 == ordered_melted_dist_matrix_ERR21_case[i, "Var2"])["value
"]
    ERR21_case_neighbor[paste("Neighbor", i), "TRUC"] = filter(final_SDATA, Case_Numb
er == ordered_melted_dist_matrix_ERR21_case[i, "Var2"])["TRUC"]
  }
}

if (dim(ERR23)[1] == 0){
  ERR23_case = NULL
} else {
  ERR23_case = ERR23[1, "Case_Number"]
  melted_dist_matrix_ERR23_case = filter(melted_dist_matrix, Var1 == ERR23_case & Var
2 != ERR23_case)
  ordered_melted_dist_matrix_ERR23_case = melted_dist_matrix_ERR23_case[order(melted_
dist_matrix_ERR23_case$value, decreasing = F), ]
  ERR23_case_neighbor = data.frame()
  for (i in c(1:best_k)){
    ERR23_case_neighbor[paste("Neighbor", i), "Case #"] = ordered_melted_dist_matrix_
```

```
ERR23_case[i, "Var2"]
    ERR23_case_neighbor[paste("Neighbor", i), "Distance"] = filter(ordered_melted_dis
t_matrix_ERR23_case, Var2 == ordered_melted_dist_matrix_ERR23_case[i, "Var2"])["value
"]
    ERR23_case_neighbor[paste("Neighbor", i), "TRUC"] = filter(final_SDATA, Case_Numb
er == ordered_melted_dist_matrix_ERR23_case[i, "Var2"])["TRUC"]
  }
}

if (dim(ERR24)[1] == 0){
  ERR23_case = NULL
} else {
  ERR24_case = ERR24[1, "Case_Number"]
  melted_dist_matrix_ERR24_case = filter(melted_dist_matrix, Var1 == ERR24_case & Var
2 != ERR24_case)
  ordered_melted_dist_matrix_ERR24_case = melted_dist_matrix_ERR24_case[order(melted_
dist_matrix_ERR24_case$value, decreasing = F), ]
  ERR24_case_neighbor = data.frame()
  for (i in c(1:4)){
    ERR24_case_neighbor[paste("Neighbor", i), "Case #"] = ordered_melted_dist_matrix_
ERR24_case[i, "Var2"]
    ERR24_case_neighbor[paste("Neighbor", i), "Distance"] = filter(ordered_melted_dis
t_matrix_ERR24_case, Var2 == ordered_melted_dist_matrix_ERR24_case[i, "Var2"])["value
"]
    ERR24_case_neighbor[paste("Neighbor", i), "TRUC"] = filter(final_SDATA, Case_Numb
er == ordered_melted_dist_matrix_ERR24_case[i, "Var2"])["TRUC"]
  }
}

# ============================================================================
# Problem 3
# 3.1
# Eigenvalue and Eigenvector
SCORR = cor(SDATA)
W = eigen(SCORR)$vector
L = eigen(SCORR)$value
write.csv(W, file = "W.csv")
ggplot() +
  geom_point(aes(x = c(1:400), y = L)) +
  ylab("L(eigenvalue)") + xlab("m") +
  scale_y_continuous(expand = c(0, 1)) + scale_x_continuous(expand = c(0, 1)) +
  theme(text = element_text(size = 25), plot.margin = margin(10, 15, 10, 10, "point
"))
rm(SCORR)

# 3.2
# PEV
PEV = NULL
for (m in 1:400){
  PEV[m] = sum(L[c(1:m)]) / 400 * 100
}
smallest_r = sum(PEV < 90) + 1
ggplot() +
  geom_point(aes(x = c(1:400), y = PEV)) +
  geom_segment(aes(x = smallest_r, xend = smallest_r, y = 0, yend = 90), size = 1, co
```

```
lour="blue") +
  geom_segment(aes(x = 0, xend = smallest_r, y = 90, yend = 90), size = 1, colour="bl
ue") +
  geom_text(aes(x = smallest_r + 17, y = 3, label = paste("m =", smallest_r)), size =
 7, color = "blue") +
  geom_text(aes(x = 24, y = 93, label = "PEV = 90%"), size = 7, color = "blue") +
  ylab("PEV (%)") + xlab("m") +
  scale_y_continuous(expand = c(0, 1)) + scale_x_continuous(expand = c(0, 2)) +
  theme(text = element_text(size = 25), plot.margin = margin(10, 15, 10, 10, "point
"))

# 3.3
# PCA Data Set
ZDATA = as.data.frame(as.matrix(SDATA) %*% as.matrix(W))
ZDATA_with_TRUC = cbind(TRUC, ZDATA[, c(1:smallest_r)])
final_ZDATA = cbind(case, TRUC, ZDATA[, c(1:smallest_r)])

# 3.4
# kNN on PCA data set
PCA_test_set = final_ZDATA[test_set[, "Case_Number"], ]
PCA_training_set = final_ZDATA[training_set[, "Case_Number"], ]

start_time <- Sys.time()
PCA_training_result = knn(train = PCA_training_set[, -c(1,2)], test = PCA_training_se
t[, -c(1,2)], cl = PCA_training_set[, 2], k = best_k)
PCA_test_result = knn(train = PCA_training_set[, -c(1,2)], test = PCA_test_set[, -c
(1,2)], cl = PCA_training_set[, 2], k = best_k)
end_time <- Sys.time()
new_computing_time = end_time - start_time

new_test_perf = (sum(PCA_test_result == PCA_test_set[, 2]) / length(PCA_test_result))

PCA_confusion_matrix_test = round(prop.table(confusionMatrix(PCA_test_result, as.fact
or(PCA_test_set[, 2]))$table, margin = 2)* 100, 1)
rownames(PCA_confusion_matrix_test) = c("True CL1", "True CL2", "True CL3", "True CL4
")
colnames(PCA_confusion_matrix_test) = c("Pred CL1", "Pred CL2", "Pred CL3", "Pred CL4
")
for (i in 1:4){
  PCA_Standard_Error = sqrt(PCA_confusion_matrix_test[i, i] / 100 * (1 - PCA_confusio
n_matrix_test[i, i] / 100) / dim(filter(PCA_test_set, TRUC == i))[1])
  PCA_CI_UL = PCA_confusion_matrix_test[i, i] / 100 + 1.6 * PCA_Standard_Error
  PCA_CI_LL = PCA_confusion_matrix_test[i, i] / 100 - 1.6 * PCA_Standard_Error
}

PCA_Standard_Error = sqrt(new_test_perf * (1 - new_test_perf) / length(PCA_test_resul
t))

compare_before_and_after_PCA = data.frame("Data_Set" = c("Original", "PCA"))
compare_before_and_after_PCA[1, "Computing_Time"] = round(computing_time, 1)
compare_before_and_after_PCA[2, "Computing_Time"] = round(new_computing_time, 1)
compare_before_and_after_PCA[1, "Test_Performance"] = round(test_perf * 100, 1)
compare_before_and_after_PCA[2, "Test_Performance"] = round(new_test_perf * 100, 1)
compare_before_and_after_PCA[1, "UL"] = round(best_k_CI_UL, 3) * 100
compare_before_and_after_PCA[1, "LL"] = round(best_k_CI_LL, 3) * 100
```

```
compare_before_and_after_PCA[2, "UL"] = round(new_test_perf + 1.6 * PCA_Standard_Erro
r, 3) * 100
compare_before_and_after_PCA[2, "LL"] = round(new_test_perf - 1.6 * PCA_Standard_Erro
r, 3) * 100

ggplot(data = compare_before_and_after_PCA, aes(x = Test_Performance, y = Data_Set))
+
  geom_errorbar(aes(xmin = LL, xmax = UL), width = 0.1, color = "blue", size = 2) +
  geom_point(size = 3) +
  geom_text(aes(x = Test_Performance, y = c(0.9, 1.9), label = Test_Performance), siz
e = 7.5) +
  geom_text(aes(x = LL, y = c(1.1, 2.1), label = LL), size = 7.5) +
  geom_text(aes(x = UL, y = c(1.1, 2.1), label = UL), size = 7.5) +
  xlab("Test Performance (%)") + ylab("Data Set") +
  theme(plot.title = element_text(hjust = 0.5), text = element_text(size = 25), plot.
margin = margin(10, 10, 10, 10, "point")) +
  xlim(60, 90)

# 3.5
# Plot each class on 2D.
CL24 = filter(final_ZDATA, TRUC == 2 | TRUC == 4)
CL23 = filter(final_ZDATA, TRUC == 2 | TRUC == 3)
CL21 = filter(final_ZDATA, TRUC == 2 | TRUC == 1)
ggplot() +
  geom_point(data = CL24, aes(x = V1, y = V2, color = TRUC)) +
  scale_color_manual(values = c("2" = "black", "4" = "green")) +
  xlim(-20, 30) + ylim(-20, 20) +
  theme(text = element_text(size = 25), legend.position = c(0.935, 0.85), legend.text
 = element_text(size = 25), legend.title = element_text(size = 25))
ggplot() +
  geom_point(data = CL23, aes(x = V1, y = V2, color = TRUC)) +
  scale_color_manual(values = c("2" = "black", "3" = "red")) +
  xlim(-20, 30) + ylim(-20, 20) +
  theme(text = element_text(size = 25), legend.position = c(0.935, 0.85), legend.text
 = element_text(size = 25), legend.title = element_text(size = 25))
ggplot() +
  geom_point(data = CL21, aes(x = V1, y = V2, color = TRUC)) +
  scale_color_manual(values = c("2" = "black", "1" = "blue")) +
  xlim(-20, 30) + ylim(-20, 20) +
  theme(text = element_text(size = 25), legend.position = c(0.935, 0.85), legend.text
 = element_text(size = 25), legend.title = element_text(size = 25))

# 3.6
# Plot class 2 and its misclassified cases on 2D.
PCA_CL2 = filter(final_ZDATA, TRUC == 2)

ZERR21_case = filter(cbind(PCA_test_result, PCA_test_set[, c(1, 2)]), TRUC == 2 & PCA
_test_result == 1)
ZERR23_case = filter(cbind(PCA_test_result, PCA_test_set[, c(1, 2)]), TRUC == 2 & PCA
_test_result == 3)
ZERR24_case = filter(cbind(PCA_test_result, PCA_test_set[, c(1, 2)]), TRUC == 2 & PCA
_test_result == 4)
ZERR21 = final_ZDATA[ZERR21_case$Case_Number, ]
ZERR23 = final_ZDATA[ZERR23_case$Case_Number, ]
ZERR24 = final_ZDATA[ZERR24_case$Case_Number, ]
```

```
ggplot() +
  geom_point(data = PCA_CL2, aes(x = V1, y = V2, color = "CL2")) +
  geom_point(data = ZERR21, aes(x = V1, y = V2, color = "ERR21")) +
  geom_point(data = ZERR23, aes(x = V1, y = V2, color = "ERR23")) +
  geom_point(data = ZERR24, aes(x = V1, y = V2, color = "ERR24")) +
  scale_color_manual(name = "Data Point", values = c("CL2" = "black", "ERR21" = "blue
", "ERR23" = "red", "ERR24" = "green")) +
  xlim(-20, 30) + ylim(-20, 20) +
  theme(text = element_text(size = 25), legend.position = c(0.91, 0.8), legend.text =
 element_text(size = 25), legend.title = element_text(size = 25))
```