MATH 6350 – HOMEWORK #3

Basel Najjar, Brian Le, Soo Jong Cho

31st October 2021

Table of Contents

Scope

This report outlines the implementation of the Random Forest automatic classification algorithm to digitized images of typed fonts. Four fonts were utilized in this analysis:

- Yi Baiti: 𑢠𑢡𑢢𑢣𑢤𑢥𑢦𑢧𑢨𑢩𑢪𑢫𑢬𑢭𑢮𑢯
- SWIS721 Aa Bb Cc 1 2 3
- Tahoma: Aa Bb Cc 1 2 3
- Serif: **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. Each pixel row and column position correspond to a feature value X_1 through X_{400}:

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

An example quantized image is below. The image is not of a font; however, it illustrates the concept of image quantization:

Original image (left) compared to quantized image (right).

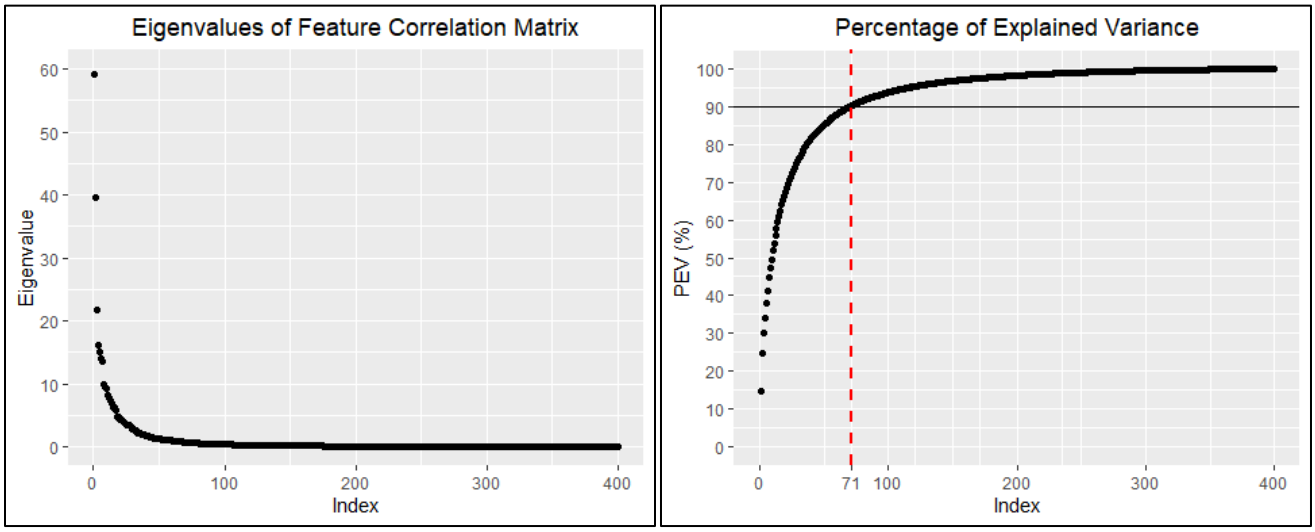
Data Pre-Treatment and Dimensionality Reduction

Data Pre-Treatment

Several pre-treatment steps were performed on the data before 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, bold-face and italicized characters were eliminated. Only normal strength and non-italic characters were considered in this analysis. Finally, feature values were standardized across the global data set so that each feature (X_1, X_2, …, X_{400}) had a mean of 0 and standard deviation of 1.

Dimensionality Reduction

Principal Component Analysis (PCA) was then applied to the cleaned and standardized data set. PCA allows the dimensionality of a data set to be reduced while still capturing much of the variance of the data. Each of the 400 features across all four classes were correlated against one another to generate a 400x400 feature correlation coefficient matrix. The eigenvalues and eigenvectors were then computed for this 400x400 feature correlation matrix. The cumulative sum of eigenvalues divided by 400 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. The first 71 principal components explain 90% of the variance in the data; therefore, the dimensionality of this data set can be reduced from 400 to 71 (r = 71).



The cumulative sum of eigenvalues divided by 400 was used to compute PEV.

Question 1

1.1 – Data Set Summary

The size and composition of the data set with respect to each class is as follows:

Font	Class	Size	Percent of Data
Yi Baiti	CL_1	1,529	13%
SWIS721	CL_2	3,360	29%
Tahoma	CL_3	3,323	29%
Serif	CL_4	3,324	29%
Total	N	11,536	100%

1.2 – Data Set Partitioning

The data was randomly partitioned into training and test sets. First, the global data set was separated into four separate data sets, each containing one class (CL_1 through CL_4). 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. Random partitioning mitigates sample bias, and decomposition into single-class data sets ensures even representation of all four classes in the training and test sets. The size of training and test sets are described below:

Class	Training Set	Percent of Training Data
CL_1	1,223	13%
CL_2	2,688	29%
CL_3	2,658	29%
CL_4	2,659	29%
Total	9,228	100%

Class	Test Set	Percent of Test Data
CL_1	306	13%
CL_2	672	29%
CL_3	665	29%
CL_4	665	29%
Total	2,308	100%

1.3 - SMOTE

Class CL_1 is a minority class in this data set. An unbalanced data set can hinder automatic classification performance. The size of CL_1 was increased using the Synthetic Minority Oversampling Technique (SMOTE). SMOTE generates synthetic data through linear interpolation between data points in $\mathbb{R}^{71}$ feature space. Inputs to the SMOTE function include oversample percentage and *k* nearest neighbors. The size of CL_1 was doubled (100% oversampling) to re-balance the data set and k = 5 nearest neighbors were used for interpolation. It is worth noting that CL_1 data was amplified separately in the training and test sets. In other words, the global data set was randomly partitioned into training and test sets before SMOTE implementation.

1.4 – Re-balanced Data Set Summary

The size of the rebalanced data set is outlined below:

Class	Re-balanced Training Set	Percent of Re-balanced Training Data
CL_1^*	2,446	23%
CL_2	2,688	26%
CL_3	2,658	25%
CL_4	2,659	25%
Total	10,451	100%

Class	Re-balanced Test Set	Percent of Re-balanced Test Data
CL_1^*	612	23%
CL_2	672	26%
CL_3	665	25%
CL_4	665	25%
Total	2,614	100%

Question 2

2.1 – Random Forest Algorithm Overview

Random forest (RF) is a supervised machine learning algorithm that can be used for both classification and regression tasks. This report utilizes the classification approach of this model. The RF algorithm is an extension of two other tree-based methods: Decision trees and Bagging.

Decision trees segment data based on feature values in the data set. This is accomplished by sequential partitioning of all cases in the data set. The model segments data by minimizing the Gini index of resulting sub-groups ("nodes") at each partition. A feature (F) and threshold (th) are automatically determined such that cases with feature value $F \leq th$ are partitioned into node A and cases with feature value $F > th$ are partitioned into node B. Nodes A and B will have minimum possible Gini index (maximum possible purity). Gini index for a given node S can be computed using the following equation:

$$Gini(S) = \sum_{i=1}^{k} f_{CL_i}(1 - f_{CL_i})$$

Where:

- f_{CL_i} represents the fraction of cases belonging to CL_i in node *S.*
- *Sum i = 1 to k* represents the sum over all class CL_1 … CL_k

In working with tree-based methods, the Bias-Variance trade off must be considered. Decision trees are generally considered low bias as no underlying target function is assumed. This typically results in trees with low interpretability. Trees can have high variance when the tree it is too specific to the training data set (overfit). Models with high variance often achieve poor test set accuracy. A single decision tree may be interpretable but will often be overfit to the training data. The RF algorithm mitigates overfit by combining bagging with feature sub-setting.

Bagging is a form of bootstrap for trees that improves performance by using several trees. Many trees are generating using a different random sample of cases (bootstrapping). Model predictions are the "majority vote" of all bagged trees. A single tree in the bag will have low bias and high variance but utilizing average predictions of the entire bag results in a model with low bias and low variance.

One major issue with bags of trees is that individual trees are often highly correlated. There is often a small number of important features that dictate node partitioning early in the tree. This results in many similar trees. The RF model generates de-correlated trees to reduce the variance of a single tree. Unlike bagging, the trees created by the RF algorithm are low in both variance and bias. While a typical bagging model considers all features at each given node or split, RF trees only consider a small random subset of features at each node. The number of features selected (*m)* is fixed. In general, *m* is much smaller than the total number features in the data set (*p)*. In notational terms, $m << p$. A common heuristic for selecting *m* is to use $m = \sqrt{p}$. It is worth noting that if *m = p*, the resulting model is identical to a bag of trees model.

A random forest may contain hundreds or thousands of trees. For explanatory proposes, only a single tree will be considered. A random subset of cases in the data set is used to construct a single tree; this is the previously described bootstrapping method. The algorithm will randomly select *m* features at a given node. The node will be partitioned conditionally by a feature and a threshold that result in minimum Gini index for the resulting new nodes. All node partitions are binary. For any node that is created with Gini of 0 and purity of 1, the model will not create any more splits following this given node. These terminal nodes are known as "leaves" of the tree. This process of randomly selecting *m* predictors, determining a feature and threshold, and creating new nodes will continue until there is purity at every node at the leaves of the tree. In some cases, size or Gini index value limits are set to prevent the generation of leaves with a small number of cases. The process of creating trees continues with new random subsets for each tree until a fixed number of trees have been generated.

Like the bag of trees model, the RF model use a majority vote method. New (test) data points are classified based the predicted class of the majority of trees in the forest. Each tree in the forest will generate one prediction, and the majority vote of all trees in the forest dictates the class of every new point.

2.2 – The *randomForest* Function

The random forest function within *R* has several inputs, outputs, and attributes. The *randomForest* function comes from the *randomForest* library in *R*. Key arguments are described in the table below:

Argument	Description
y	The response variable describes the true class of each case in the training data set.
data	The *data* input contains feature values for each case. The function utilizes a random subset *m* of these features classify each case.
mtry	The number of randomly selected features the model will use at each node. This value is typically fixed at $\sqrt{(p)}$ for a classification model and $\frac{p}{3}$ for a regression model.
ntree	The number of trees that the random forest function will generate. Several values of forest size are typically tested to balance performance with computation time. For robustness, it is ideal to have a value of *ntree* that is not too small so that each feature in the given dataset is used more than once. If the value of *ntree* is too small, all features may not be used and can result in a high number of classification errors.
importance	Boolean input that determines feature importance should be considered in determining the model. This instructs the algorithm to track the mean reduction in Gini index and mean reduction in error for each feature in the given dataset.

The *randomForest* function output is an object that contains a variety of information concerning the model. The following table describes the outputs that can be called after a random forest model is generated. These outputs relate to classification models; additional outputs can be called for regression models. Regression outputs are not discussed in this report.

Output	Description
Type	Returns the type of RF model produced. There are three possible values that can be returned: regression, classification, or unsupervised.
Predicted	Produces a predicted value or classification of each of the data points based on the out of bag sample for each given tree.
Importance	The importance output is a matrix of size p x (k + 2). Each row in the matrix describes a feature in the dataset. The first k columns produced are the mean decrease in accuracy for each class (CL_1, … , CL_k) in the model. This is the decrease in accuracy for an individual class if a given feature was removed. The k^{th}+1 column outlines *Mean Decrease Accuracy.* This is the overall mean of the decrease in accuracy of all k classes. This value can be interpreted as the mean decrease in model accuracy if the feature was not included in the model. The final column is *Mean Decrease Gini.* The interpretation is identical to the *Mean Decrease Accuracy* column; however, the scale of Gini values is different. When a feature is important, it will tend to be used more often in the model. The model selects features that lead to the largest reduction in Gini index value at each node.

2.3 – The *predict* Function

The *predict* function in *R* can be used to classify new data using a random forest model. The inputs to this function include the random forest model object generated by the *randomForest* function and a data set to be classified by the model. The output of the predict function is a predicted class for each case in the input data set.

Question 3

3.1 & 3.2 – Random Forest Implementation

In this section, the random forest automatic classification algorithm was applied to the fonts data set. Input parameters were fixed at *ntree* = 100 (number of trees in the RF model) and *mtry* = $\sqrt{71} \approx 8$ (number of features considered at each node). The training set of size 10,451 was used to build the model. Classification accuracy was evaluated using both the training set as well as the smaller test set of size 2,614. To generate a RF model with100 trees, computation time was 9 seconds. Confusion matrices for both the training and test sets are outlined below are outlined below:

Training Set (100 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	99 %	1 %	0 %	0 %
	CL_2	0 %	100 %	0 %	0 %
	CL_3	0 %	2 %	97 %	1 %
	CL_4	0 %	2 %	2 %	96 %

Test Set (100 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	83 %	9 %	4 %	4 %
	CL_2	4 %	82 %	5 %	8 %
	CL_3	1 %	5 %	67 %	27 %
	CL_4	3 %	7 %	34 %	57 %

When applied to the training set, classification by the RF model was nearly perfect. Almost all cases in the training set were correctly classified. Since the model has "seen" the training data before, good classification accuracy is expected. The global accuracy of the training and test datasets are 98.0 ± 0.2% and 72.0 ± 1.4%, respectively. The margin of error surrounding these accuracy figures is the 90% confidence interval. Class CL_1 was correctly classified most frequently in the test set with an accuracy of 83%. The most common CL_1 misclassification was CL_2, with an error rate of 9%. The second-best classification performance is that of CL_2 with 82% accuracy in the test set. Classes CL_3 and CL_4 exhibited worse performance compared to CL_1 and CL_2. It is worth noting that CL_1 was amplified using SMOTE; this could explain part of why CL_1 classification performance is so high. The most significant misclassification errors in the test set occur between CL_3 and CL_4, with 34% of CL_4 points misclassified as CL_3 and 27% of CL_3 points misclassified as CL_4.

Question 4

4.1 – Finding Optimal Forest Size (RF*)

Random forest models with 200 and 300 trees were generated to determine the optimal number of trees. For an RF model with 200 trees, computation time was 15 seconds. Confusion matrices are outlined below

Training Set (200 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	99 %	1 %	0 %	0 %
	CL_2	0 %	100 %	0 %	0 %
	CL_3	0 %	2 %	97 %	1 %
	CL_4	0 %	2 %	2 %	96 %

Test Set (200 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	82 %	9 %	5 %	3 %
	CL_2	4 %	83 %	4 %	8 %
	CL_3	2 %	5 %	68 %	26 %
	CL_4	3 %	7 %	34 %	56 %

Like the previous implementation, classification performance of the training set is nearly perfect with 98.0 ± 0.2% accuracy. Classification performance of the test set was slightly higher at 72.4 ± 1.4%. Class CL_2 was correctly classified most frequently with an accuracy of 83%. The most misclassifications of CL_2 were within CL_1 with an error rate of 9%. There was a slight decrease in CL_1 classification performance, however the difference is not statistically significant. CL_3 and CL_4 again shared the most common misclassification errors.

For an RF model with 300 trees, computation time was 35 seconds. Confusion matrices are outlined below:

Training Set (300 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	99 %	1 %	0 %	0 %
	CL_2	0 %	100 %	0 %	0 %
	CL_3	0 %	2 %	97 %	1 %
	CL_4	0 %	2 %	2 %	96 %

Test Set (300 Trees)		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	83 %	9 %	4 %	4 %
	CL_2	4 %	83 %	5 %	8 %
	CL_3	1 %	5 %	67 %	27 %
	CL_4	3 %	6 %	36 %	55 %

Classification performance of the training set was again 98 ± 0.2%. Classes CL_1 and CL_2 were correctly classified most frequently within the test set, while classes CL_3 and CL_4 were misclassified most frequently. The global training set accuracy was 70.5 ± 1.4%, a slight decrease compared to previous implementations. Based on the consistent poor classification accuracy of classes CL_3 and CL_4, RF models with a larger number of trees are not expected to result in increased performance.

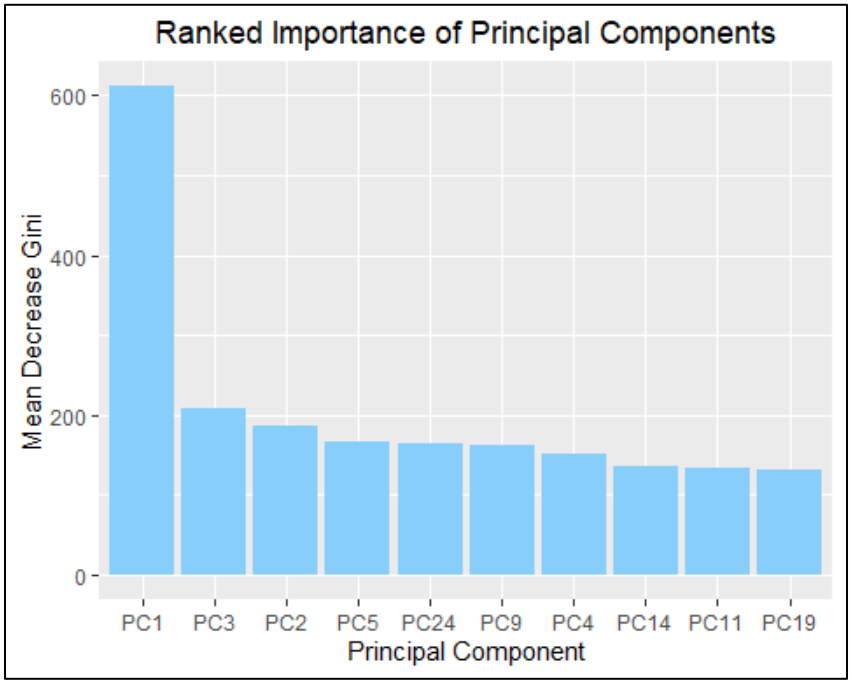
Considering overall accuracy, misclassification error rates, and computation time the best model was selected as *ntree** = 200 trees (*RF**). Results for the three random forest implementations are summarized below:

Number of Trees (*ntree*)	90% Confidence Interval of Test Set Accuracy
100	70.6% – 73.4%
200	**71.0% – 73.8%**
300	69.1% - 71.9%

Question 5

5.1 – Feature Importance

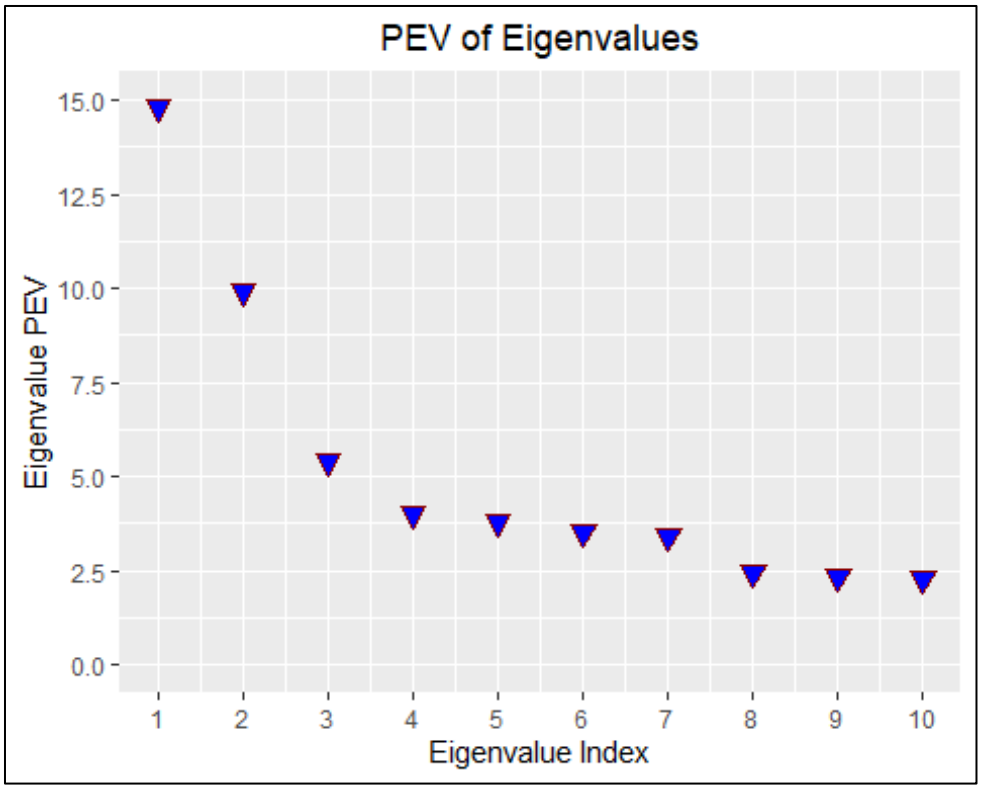
Using the RF model containing 200 trees (RF*), importance values were calculated for each of the 71 features. Importance is an optional Boolean argument of the *randomForest* function. With this option set to *True*, the model computing time is slightly longer. For RF* containing 200 trees, computation time increased from 15 seconds to 23 seconds with the addition of importance calculations. The importance argument returns two values for each feature in the data set: Mean Decrease in Accuracy (MDA) and Mean Decrease in Gini Index (MDG). MDA for a feature variable represents the expected accuracy reduction in predicting out-of-bag samples when that feature is excluded from the model. Similarly, MDG for a feature variable represents the overall decrease in Gini index (or mean increase in Node purity) that results from splits over that variable, averaged over all trees in the model. Interestingly, feature importance ranked among all 71 features does not strictly follow the order of principal components. The bar graph and corresponding table below outline the top ten most important feature variables ranked by MDA and MDG.



Importance Rank	PC_k	CL_1	CL_2	CL_3	CL_4	MDA	MDG	Eigenvalue
1	PC_1	21%	11%	7%	6%	11%	620	59
2	PC_3	5%	6%	2%	3%	4%	206	22
3	PC_{24}	6%	3%	2%	2%	3%	177	4
4	PC_5	6%	5%	2%	3%	3%	173	15
5	PC_2	4%	4%	1%	2%	3%	159	40
6	PC_9	4%	4%	2%	2%	3%	155	9
7	PC_4	5%	4%	2%	2%	3%	153	16
8	PC_{14}	4%	3%	2%	2%	2%	143	7
9	PC_{11}	3%	4%	2%	2%	3%	131	8
10	PC_{19}	4%	3%	1%	2%	3%	131	5

The expected reduction in classification accuracy for each class CL_1 through CL_4 is outlined in the table above. If the most important feature (PC_1) was excluded from the model, CL_1 classification accuracy would reduce by over 20%. Class CL_3 and CL_4, on the other hand, would only exhibit a reduction in accuracy of 7% and 6%, respectively. Subsequent features are far less important than PC_1 for all individual classes as well as the overall data set.

5.2 – Interpretation of Eigenvalues



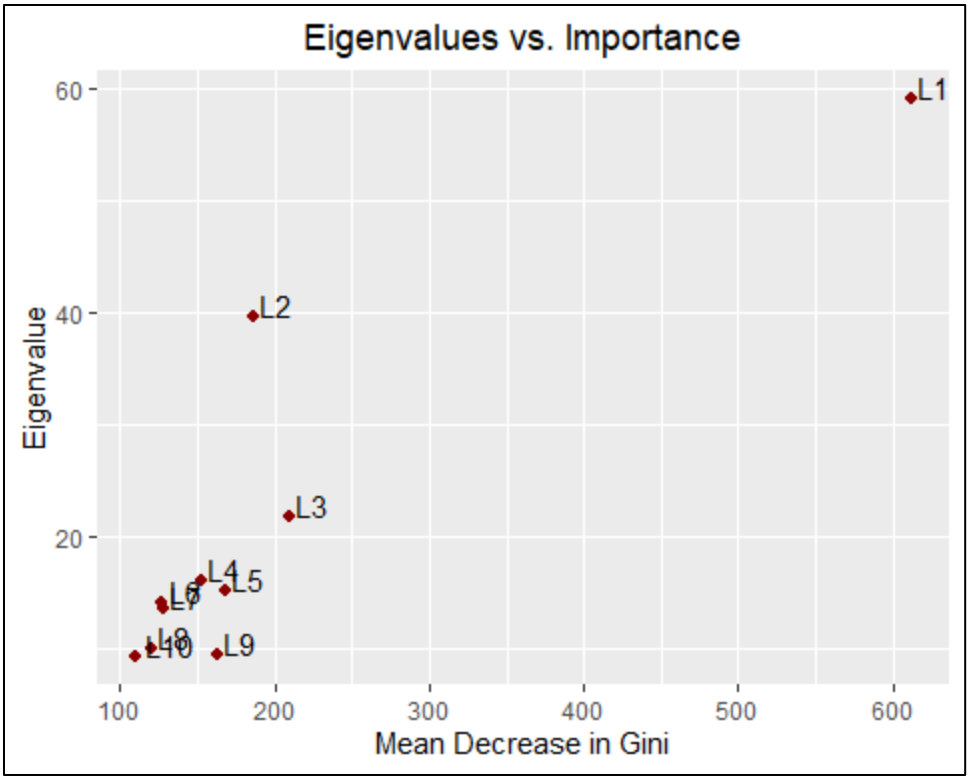
As described in previous sections, each principal component feature (PC_1 through PC_{71}) has a corresponding eigenvalue (L_1 through L_{71}). The cumulative sum of eigenvalues divided by 400 represents percentage of explained variance (PEV). The first principal component explains 15% of the variance in the data, the second explains 10%, and so forth. For this data set, r was computed to be 71 based on the desired PEV of 90%.

$$\sum_{i=1}^{r=71} \frac{L_i}{400} = PEV_r = 90\%$$

In other words, the first 71 principal components explain 90% of the variance in the data. The sum of all 400 eigenvalues divided by 400 would equal 100%.

5.3 – Relationship Between Eigenvalues and Importance

The table and figures below outline MDA and MDG for the first ten principal components in the RF* model. The first principal component (PC_1) is unsurprisingly the most important feature in the data set. If PC_1 was excluded from the model, the expected decrease in classification accuracy would be 11.2%.



Principal Component	Eigenvalue	Mean Decrease in Accuracy (MDA)	Mean Decrease in Gini Index (MDG)
PC_1	59	11.2 %	621
PC_2	40	2.7 %	159
PC_3	22	3.9 %	206
PC_4	16	3.0 %	153
PC_5	15	3.8 %	173
PC_6	14	1.9 %	127
PC_7	14	2.0 %	127
PC_8	10	2.0 %	123
PC_9	9	2.9 %	155
PC_{10}	9	1.2 %	111

In general, larger eigenvalues (first several principal components) are associated with a higher degree of importance within the model. This is particularly true for PC_1 with an eigenvalue of 59, MDA of 11.2%, and MDG of 621. This is not a strict rule, however; it is only a general trend. For example, PC_2 and PC_3 have eigenvalues of 40 and 22, respectively. Despite PC_2 having a larger eigenvalue, PC_3 ranks higher in terms of importance within the model.

Question 6

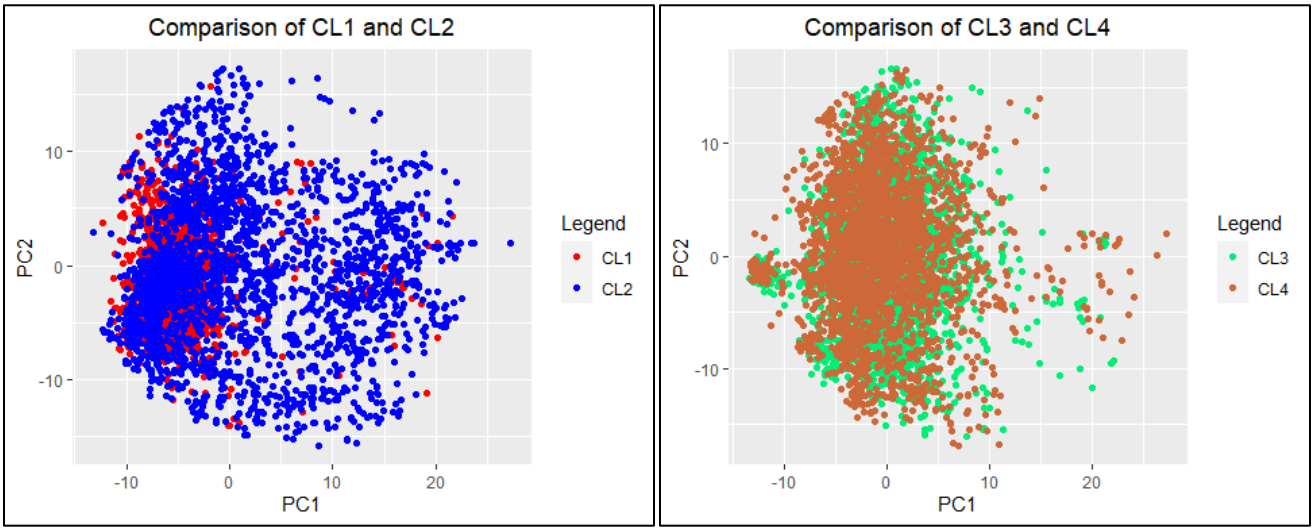
6.1 – Misclassification Errors Using RF*

The confusion matrix for the RF* classifier containing 200 trees along with 90% confidence intervals for classification accuracy of each class (CL_1 through CL_4) is outlined below:

RF* Test Set		Predicted Class			
		CL_1	CL_2	CL_3	CL_4
True Class	CL_1	82 %	9 %	5 %	3 %
	CL_2	4 %	83 %	4 %	8 %
	CL_3	2 %	5 %	68 %	26 %
	CL_4	3 %	7 %	34 %	56 %

Class	90% Confidence Interval for Test Set Accuracy
CL_1	82 ± 0.24 %
CL_2	81 ± 0.21%
CL_3	66 ± 0.20 %
CL_4	56 ± 0.18 %

Classes CL_3 and CL_4 exhibited the worst overall performance in the model. 34% of CL_4 points were misclassified as CL_3 and 26% of CL_3 points were misclassified as CL_4. Plots of CL_1 vs. CL_2 and CL_3 vs. CL_4 with respect to the first two principal components are outlined below. It is worth noting that these plots are a 2-dimensional projection of 71-dimension feature space. The first two principal components account for approximately 25% of the variance in the data.



Though both pairs of classes appear to be poorly separable, the concentration of points in CL_3 and CL_4 appears visually more similar than CL_1 and CL_2. Clustering and separability are discussed in detail in a later section of this report.

6.2 – New RF Classifier for Poor Performing Classes

A new RF classification model containing 200 trees was generated to attempt binary classification between CL_3 and CL_4. The confusion matrix for this new RF classifier compared to CL_3 and CL_4 classification from RF* is outlined below:

Binary RF Test Set		Predicted Class	
		CL_3	CL_4
True Class	CL_3	68 %	32 %
	CL_4	36 %	64 %

RF* CL3/CL4 Accuracy		Predicted Class	
		CL_3	CL_4
True Class	CL_3	68 %	26 %
	CL_4	34 %	56 %

Similar CL_3 classification accuracy was achieved using the new binary RF model. CL_3 classification accuracy was 68% in both models. CL_4 classification accuracy improved considerably, however. The new RF model achieved 64% classification accuracy for CL_4, compared to only 56% for the RF* model. In both models, many misclassifications occur between CL_3 and CL_4. The elimination of CL_1 and CL_2 in the binary model does not substantially improve overall classification accuracy.

6.3 – Methods to Improve Classification

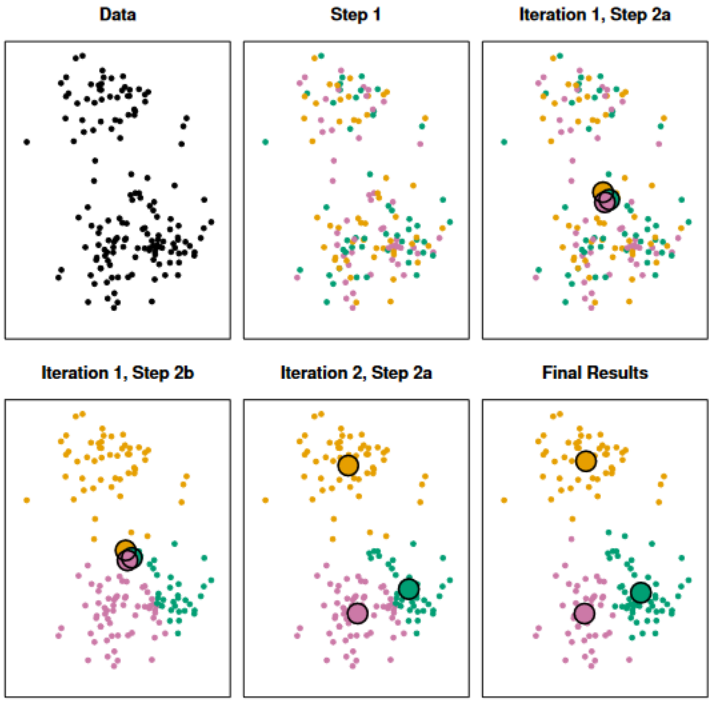
In this section, methods for improved classification accuracy are implemented on the fonts data set. Two methods are tested: unsupervised clustering using the K-Means algorithm, and multiple binary RF classification.

Unsupervised Clustering Using K-Means

The K-Means algorithm is an unsupervised learning technique that attempts to cluster points based relative distance to computed cluster centers. The K-Means function accepts several inputs:

- An "unlabeled" feature data set containing N cases and p features (with no response variable).
- Number of clusters to be found (G_1 through G_k).
- Number of sequential implementations.
- Maximum number of iterations per implementation (optional).

The algorithm works by first assigning all points in the data set to a random cluster (G_1 through G_k). Next, the center of each cluster ($cent_1$ through $cent_k$) is computed by taking the average of all feature values for points in that cluster. Points are then re-assigned to the nearest cluster center. The nearest cluster center is that which has the smallest Euclidian distance to any given point. Centers are re-computed, and points are re-assigned iteratively until point assignment becomes static or a fixed maximum number of iterations has been reached.



Iterations of the K-means algorithm

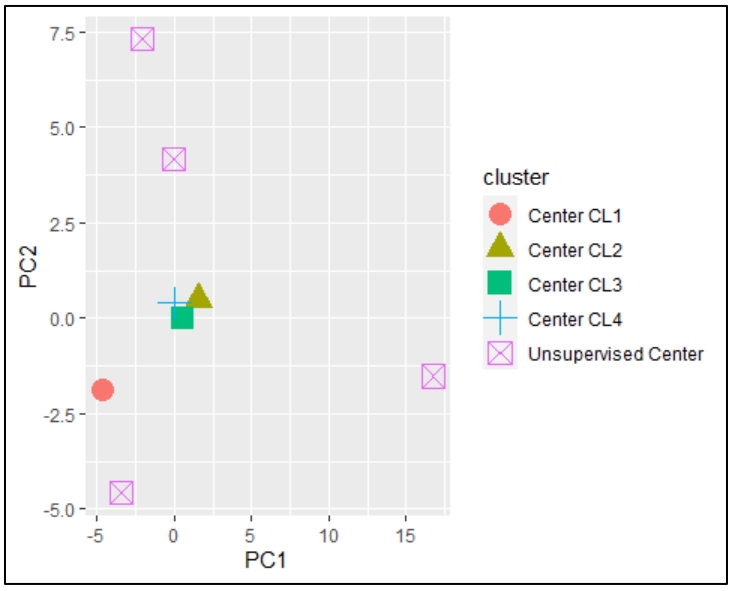
Image source: James, Gareth, et al. "Chapter 12." *An Introduction to Statistical Learning: With Applications in R*, 2nd ed., Springer, New York, NY, 2021, pp. 520–520.

The resulting clusters will have minimum intra-cluster dispersion; this is only a local minimum, however, and is based wholly on the random initial point assignments. Therefore, it is common practice to implement multiple K-means sequences – on the order of 20 – 100 times – and keep the best solution of all implementations. Here, 20 implementations were utilized, and the computation time was 10 seconds. The best clustering result (minimum intra-cluster dispersion) of these 20 implementations is outlined below. It should be noted that only original CL_1 data was used; no synthetic points are included in the clusters.

Clustering Results		True Class			
		CL_1	CL_2	CL_3	CL_4
Cluster	G_1	4 %	61 %	17 %	18 %
	G_2	6 %	20 %	37 %	37 %
	G_3	21 %	25 %	27 %	26 %
	G_4	9 %	32 %	28 %	30 %

Cluster Purity		Gini Index	Purity
Cluster	G_1	0.56	44 %
	G_2	0.68	32 %
	G_3	0.75	25 %
	G_4	0.71	29 %

The cluster numbers (G_1 through G_4) are arbitrary and do not in any way correspond to class numbers (CL_1 through CL_4). The clustering itself is not arbitrary, however. If the classes were well-separable the four clusters should be relatively homogenous with respect to one single class. The left-hand table shows the composition of each cluster. Each cluster is heterogenous as there is significant representation of every class in every cluster. The only exception to this is cluster G_1 with 6% of cases belonging to CL_2. The purity of each cluster can be computed by taking 1 – Gini Index. In general, clusters are not pure. The cluster with the highest purity is again cluster G_1 with 44%. Interestingly, cluster G_3 consists of approximately equal proportions of all four classes. This is supported by the purity value of 25% for G_3. The cluster centers found by the K-means algorithm are displayed in the following figure, along with true centers of each of the four classes. The cluster centers are unlabeled as the numbering itself is arbitrary. The plot is a 2-d projection of 71-dimensional feature space; the first two principal components explain approximately 25% of the variance in the data.



The consistently high classification accuracy of class CL_1 in all RF implementations is unsurprising considering the distance of CL_1 center compared to the other class centers. The true centers of CL_2, CL_3, and CL_4 appear to be close in this 2-d projection. Visualization of higher-dimensional space is not possible, but the true Euclidian distance between centers can be computed. Using the K-means algorithm with one class and one center at a time, the true center of each class was found. The following is an outline of pairwise distances between each true class center:

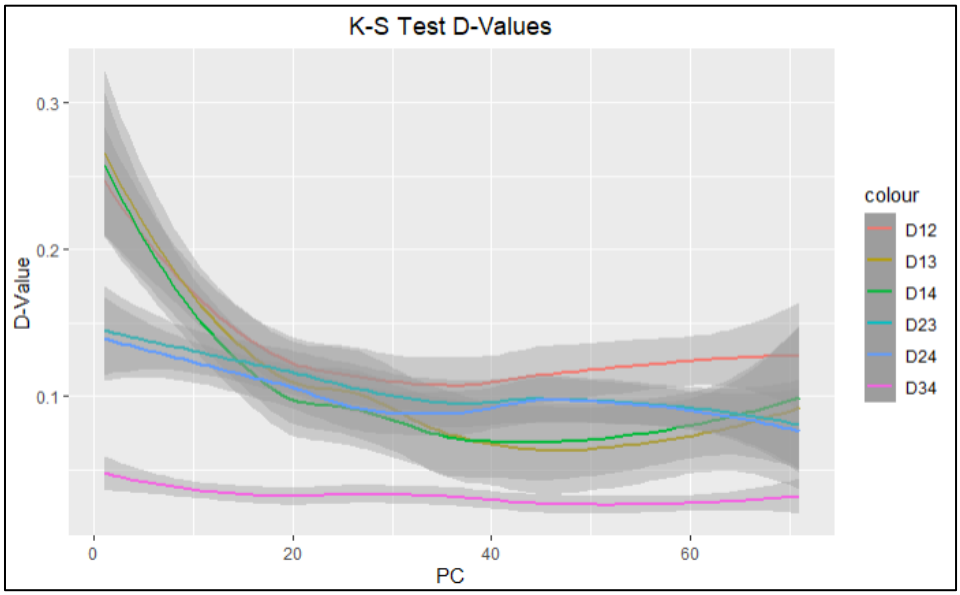
Class Pairs	CL1 vs. CL2	CL1 vs. CL3	CL1 vs. CL4	CL2 vs. CL3	CL2 vs. CL4	CL3 vs. CL4
Distance Between True Centers	6.18	2.41	0.87	0.88	0.37	**0.07**

Based on these distances, CL_1 appears to be the most unique, with distances of 6.18 and 2.14 from CL_2 and CL_3, respectively. As shown in the 2-d projection above, there is minimal separation between the centers of CL_2, CL_3, and CL_4. Interestingly, the distance between CL_3 and CL_4 is at least one order of magnitude smaller than any other distance, at .07 units between these two centers. This can potentially explain the overall similarity of the classes and the consistently poor accuracy between CL_3 and CL_4.

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.

The plot below shows the pairwise D-value of all principal components between classes:



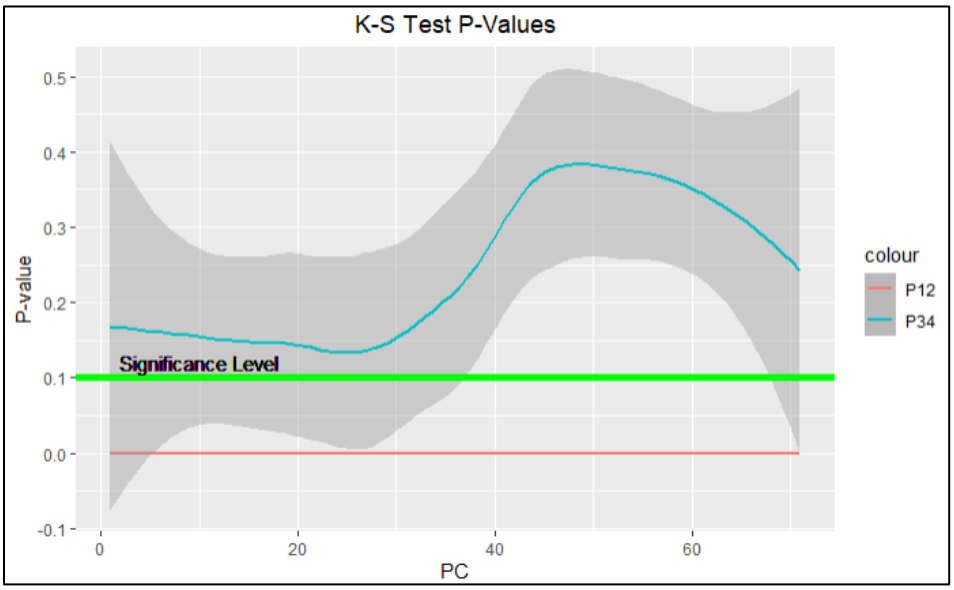
In general, the distances between CL_1 and all other classes is relatively large. For the first few principal components, the vertical distance of CL_1 to any other class is up to .25; this indicates that there is a distinct difference between CL_1 and the other classes. D-value also tends to decrease with increasing principal component index. The percent explained variance of each principal component also decreases with increasing index; the distance values reflect this property. The D-value between CL_3 and CL_4 is relatively small. This indicates the distribution of feature variables is similar in both classes and is another explanation for the poor classification performance between CL_3 and CL_4 within random forest models.

The similarity between CL_3 and CL_4 can be further analyzed using the KS-test P-values. P-value represents the threshold at which the null hypothesis can be rejected or fail to be rejected. Within the context of the KS-test, 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 < α
- Where $F_i(X)$ and $F_j(X)$ represent the cumulative distribution function (cdf) for two distributions.

KS-test P-values for each principal component of CL_1 vs. CL_2 and CL_3 vs. CL_4 is plotted below. The solid green line indicates the significance level of $\alpha = 0.1$.



Based on these results, there is sufficient evidence at the $\alpha = 0.1$ significance level (90% confidence level) to conclude that the distributions of CL_1 and CL_2 are different. The P-value of CL_1 vs. CL_2 is essentially zero, so $P_{12} < \alpha$. Conversely, there is insufficient evidence at the $\alpha = 0.1$ significance level (90% confidence level) to conclude that the distributions of CL_3 and CL_4 are different. The P-value of CL_3 vs. CL_4 is larger than α for all principal components, so $P_{34} > \alpha$.

Multiple Binary Random Forest Classifiers

In this section, six binary RF classification models were generated for pairwise classification between each class. Each of the binary random forest models predict results between two given classes. It is unable to predict any cases outside of the binary classes present. Each class has three RF models that can generate predictions. RF models can be combined to predict based on majority vote. The results of this majority vote classifier will not directly correspond to a particular class but rather will predict whether a case is correctly classified in CL_1 through CL_4. Each class will have three votes that determine the predicted class.

- RF_{12} Model for CL_1 vs. CL_2
- RF_{13} Model for CL_1 vs. CL_3
- RF_{14} Model for CL_1 vs. CL_4
- RF_{23} Model for CL_2 vs. CL_3
- RF_{24} Model for CL_2 vs. CL_4
- RF_{34} Model for CL_3 vs. CL_4

For example, CL_1 cases can be classified the 2-out-of-3 voting between RF_{12}, RF_{13}, and RF_{14}.

Results for the CL_1 vs. CL_2, CL_3 and CL_4 binary classification models are summarized below. Accuracy for CL_1 is similar to the RF* model, but accuracy for other classes is significantly higher at 95% for CL_2, 97% for CL_3, and 97% for CL_4.

CL_1 vs. CL_2		Predicted Class	
		CL_1	CL_2
True Class	CL_1	86 %	14 %
	CL_2	5 %	95 %

CL_1 vs. CL_3		Predicted Class	
		CL_1	CL_3
True Class	CL_1	88 %	12 %
	CL_3	3 %	97 %

CL_1 vs. CL_4		Predicted Class	
		CL_1	CL_4
True Class	CL_1	87 %	13 %
	CL_4	3 %	97 %

For each of the binary random forest models, CL_1 displayed consistently high classification accuracy compared to all classes with a range of accuracy results between 86% and 88%. Misclassifications ranged from 12% to 14%. Classification accuracy for CL_2, CL_3, and CL_4, on the other hand, ranged from 95% to 97%. This is a significant improvement over the original RF* model. In fact, class CL_1 misclassifications were higher than any other class when using the binary RF classifier.

Results for the CL_2 vs. CL_3 and CL_4 binary classification models are summarized below. Accuracy for CL_2 is similar to the RF* model in the CL_2 vs. CL_3 binary model, but CL_2 accuracy for the CL_2 vs. CL_4 binary model is lower at 76%. Classification accuracy for CL_3 and CL_4 significantly higher at 91% for CL_3, 97% for CL_3, and 84% for CL_4.

CL_2 vs. CL_3		Predicted Class	
		CL_2	CL_3
True Class	CL_2	89 %	11 %
	CL_3	9 %	91 %

CL_2 vs. CL_4		Predicted Class	
		CL_2	CL_4
True Class	CL_2	76 %	24 %
	CL_4	16%	84 %

When using separate RF classifiers, CL_2 was more commonly misclassified than CL_3 but the difference is small. Only 11% of CL_2 cases were correctly classified, compared to 9% of CL_3. For the CL_2 vs. CL_4 binary model, classification accuracy was significantly better for CL_4 but worse for CL_2.

Results for the CL_3 vs. CL_4 and CL_4 binary classification models are summarized below. This is identical to the binary classification algorithm generated in section 6.2.

CL_3 vs. CL_4		Predicted Class	
		CL_3	CL_4
True Class	CL_3	64 %	36 %
	CL_4	37 %	63 %

Both CL_3 and CL_4 display consistently poor classification accuracy with frequent misclassifications. The feature distributions for these two classes share much in common. The hypothesis testing outlined previously discusses the similarity between the two distributions in detail. The test set for the RF voting classifier is the same test set used for previous random forest models in all prior sections. The following are the output results of the RF voting classifier applied to a global test set:

Class	Test Classification Accuracy
CL1	85.6 ± 0.20 %
CL2	99.1 ± 0.21 %
CL3	94.3 ± 0.20 %
CL4	96.2 ± 0.18 %

Direct improvement in classification accuracy can be observed across all classes, particularly in CL_3 and CL_4. CL_1 did not show significant improvement in correct classifications. Interestingly, the worst-performing classes in the original RF* model yielded better results in the RF voting classifier. Based on the previous binary classification results, CL_3 and CL_4 were less frequently misclassified as CL_1 compared to CL_1 misclassifications as CL_3 and CL_4. It is therefore likely that the RF voting classifier more accurately differentiated between CL_3 and CL_4 based on the "free vote" of non-CL_1. CL_2 has the highest overall accuracy of all classes with 99% accuracy. Again, based on the previous binary classification results, CL_2 was less frequently misclassified as CL_1 compared to CL_1 misclassifications as CL_2. The same "free vote" phenomena may be occurring within the CL_2 RF voting classifier. Classes CL_3 and CL_4 consistently showed poor classification accuracy in all previous models. In the RF voting classifier, classification accuracy of these two classes improves greatly. If the voting criteria is raised to 3-out-of-3 voting for all RF classifiers, the accuracy of CL_3 and CL_4 are reduced to 60% and 55%, respectively. This indicates that a single random forest model can have major influence over how the overall voting model will perform. On the other hand, a 2-out-of-3 voting system reduces the overall variance of the model and leads to a more robust classification algorithm overall.

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

R Code

Scope

```
CL1_raw = read.csv("YI BAITI.csv", header = T)      #import datasets of fonts
CL2_raw = read.csv("SWIS721.csv", header = T)
CL3_raw = read.csv("TAHOMA.csv", header = T)
CL4_raw = read.csv("SERIF.csv", header = T)

#Removing columns containing unneeded information
CL1 = CL1_raw[c(-2:-3,-6:-12)]
CL2 = CL2_raw[c(-2:-3,-6:-12)]
CL3 = CL3_raw[c(-2:-3,-6:-12)]
CL4 = CL4_raw[c(-2:-3,-6:-12)]

#Removing rows that contain missing data
CL1 = CL1[complete.cases(CL1),]
CL2 = CL2[complete.cases(CL2),]
CL3 = CL3[complete.cases(CL3),]
CL4 = CL4[complete.cases(CL4),]
#Defining "normal" font type as classes CL1,...,CL4
#"normal" means not bold nor italicized
CL1 = subset(CL1, strength == 0.4 & italic == 0)
CL2 = subset(CL2, strength == 0.4 & italic == 0)
CL3 = subset(CL3, strength == 0.4 & italic == 0)
CL4 = subset(CL4, 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]

#Changing name in 'font' to CL1,...,CL4
CL1$font = "CL1"
CL2$font = "CL2"
CL3$font = "CL3"
CL4$font = "CL4"

#Renaming the feature columns of CL1,...,CL4 to X1,...,X400
names_list = NULL #initialize an empty list
names_list[[1]] = "TRUC" #renaming font to "TRUC"

for (i in 1:400){ #naming 400 columns X1 to X400
 names_list[[i+1]] = paste("X",i,sep = "")

 }
colnames(CL1)[1:401] = c(names_list)
colnames(CL2)[1:401] = c(names_list)
colnames(CL3)[1:401] = c(names_list)
colnames(CL4)[1:401] = c(names_list)

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

colnames(DATA)[1] = c("TRUC") #renaming font to "TRUC"
DATA$TRUC = as.factor(DATA$TRUC) #converting TRUC to factor

#Defining number of rows
nDATA = nrow(DATA) #calculating rows in DATA
nCL1 = nrow(CL1)
```

```
nCL2 = nrow(CL2)
nCL3 = nrow(CL3)
nCL4 = nrow(CL4)

perc1 = nCL1/nDATA
perc2 = nCL2/nDATA
perc3 = nCL3/nDATA
perc4 = nCL4/nDATA

fDATA<- as.data.frame(scale(DATA[2:401])) #Standardized data function
SDATA <- cbind(DATA[1], fDATA) #Standardized data frame SDATA

corr = cor(fDATA)      #create correlation matrix of standardized data
corr_ev = eigen(corr)      #Get the eigen values and vectors of the correlation matrix

L_values = corr_ev$values
W_vectors = corr_ev$vectors

#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))

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)

W_vectors_opt = W_vectors[,1:PEV_opt]  #gather the eigenvectors of eigenvalues that total 90%

#Creating 71 feature length ZDATA matrix from PCA
ZDATA = c()

for (case in 1:nrow(fDATA)){ #1 to N rows
  case_pc = c()
  case_feature = unlist(fDATA[case,])
  for (pc in 1:dim(W_vectors_opt)[2]){ #1 to r columns
    case_pc[pc] = W_vectors_opt[,pc] %*% case_feature                    #matrix multiply of the original cases and eigenva
lues to create new PC values
}
  ZDATA = rbind(ZDATA,case_pc)
}

ZDATA = as.data.frame(ZDATA)                    #create a dataframe from the list of PC
ZDATA = cbind(SDATA$TRUC,ZDATA,stringsAsFactors = TRUE)

colnames(ZDATA) = c("TRUC",paste0("PC",1:PEV_opt))                        #relabel the column names of the ZDATA dataframe
rownames(ZDATA) = c(1:nrow(ZDATA))                    #relabel row names for the ZDATA dataframe
dim(ZDATA)
```

1.2

```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))
}

Zsplit = random_subset(ZDATA) #randomly partitioning data into train/test set
Ztrain = Zsplit[[1]] #extracting training set
Ztest = Zsplit[[2]]#extracting test set

table(Ztrain$TRUC)   #return the counts for each given class

##
##  CL1  CL2  CL3  CL4
## 1223 2688 2658 2659

table(Ztest$TRUC)   #return the counts for each given class

##
## CL1 CL2 CL3 CL4
## 306 672 665 665

#below is the use of the SMOTE function in R. We will SMOTE on the Train and test individually. We will only SMOTE on clas
s CL1


# 1.3
CL1_train = subset(Ztrain,Ztrain$TRUC == "CL1")     #subset data from Ztrain that of CL1
CL1_train$TRUC = 1
CL1_train_SMOTE = SMOTE(CL1_train[,-1],CL1_train[,1],dup_size = 1)       #SMOTE process with inputs of the labels data, an
d size of smote.
CL1_train = cbind(TRUC = "CL1", CL1_train_SMOTE$data)
CL1_train = within(CL1_train, rm("class"))

CL1_test = subset(Ztest,Ztest$TRUC == "CL1")     #subset data from Ztest that of CL1
CL1_test$TRUC = 1
CL1_test_SMOTE = SMOTE(CL1_test[,-1],CL1_test[,1],dup_size = 1)       #SMOTE process with inputs of the labels data, and s
ize of smote.
CL1_test = cbind(TRUC = "CL1", CL1_test_SMOTE$data)
CL1_test = within(CL1_test, rm("class"))

CL1_train_synthetic = cbind(TRUC = "Synthetic", CL1_train_SMOTE$syn_data[1:2])      #create plots
CL1_train_original = cbind(TRUC = "Original", CL1_train_SMOTE$data[1:2])
```

```
CL1_train_plot = rbind(CL1_train_synthetic,CL1_train_original)

CL1_test_synthetic = cbind(TRUC = "Synthetic", CL1_test_SMOTE$syn_data[1:2])
CL1_test_original = cbind(TRUC = "Original", CL1_test_SMOTE$data[1:2])
CL1_test_plot = rbind(CL1_test_synthetic,CL1_test_original)
```

1.4

```
Ztrain = rbind(CL1_train,subset(Ztrain, Ztrain$TRUC != "CL1"))   #recombine data with the original train data
Ztest = rbind(CL1_test,subset(Ztest, Ztest$TRUC != "CL1"))       #recombine data with the original test data

Ztrain$TRUC = as.factor(Ztrain$TRUC)                            #turn the true class to factors
Ztest$TRUC = as.factor(Ztest$TRUC)
table(Ztrain$TRUC)

table(Ztest$TRUC)

round(table(Ztrain$TRUC)/nrow(Ztrain)*100)              #percent of data in each class after SMOTE

round(table(Ztest$TRUC)/nrow(Ztest)*100)

nrow(Ztrain)

nrow(Ztest)
```

3.1 and 3.2

```
ntree = 100    #set number of trees to build in the random forest
ntry = round(sqrt(PEV_opt))     #set the number of variables to randomly select at each node in the random forest

system.time(RF_model <- randomForest(TRUC~. , data = Ztrain, mtry = ntry , ntree = ntree))        #run random forest model
 with system time function

RF_predict_train <- predict(RF_model , Ztrain)                  #predict the result with the train dataset
predict_train_table <- table(Ztrain$TRUC , RF_predict_train)       #obtain confusion matrix of the results of the train d
ataset.

round(prop.table(predict_train_table, 1),2)*100       #round the confusion matrix and turn values to percents

RF_predict_test <- predict(RF_model , Ztest)                    #predict the result with the test dataset
predict_test_table <- table(Ztest$TRUC , RF_predict_test)        #obtain confusion matrix of the results of the test datas
et.

round(prop.table(predict_test_table, 1),2)*100      #round the confusion matrix and turn values to percents
```

4.1

```
ntree = 200    #set number of trees to build in the random forest
ntry = round(sqrt(PEV_opt))         #set the number of variables to randomly select at each node in the random forest

system.time(RF_model <- randomForest(TRUC~. , data = Ztrain, mtry = ntry , ntree = ntree))     #run random forest model wi
th system time function

RF_predict_train <- predict(RF_model , Ztrain)                  #predict the result with the train dataset
predict_train_table <- table(Ztrain$TRUC , RF_predict_train)    #obtain confusion matrix of the results of the train datas
et.

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model , Ztest)                     #predict the result with the test dataset
predict_test_table <- table(Ztest$TRUC , RF_predict_test)        #obtain confusion matrix of the results of the test data
set.
```

```
round(prop.table(predict_test_table, 1),2)*100

ntree = 300               #set number of trees to build in the random forest
ntry = round(sqrt(PEV_opt))         #set the number of variables to randomly select at each node in the random forest

system.time(RF_model <- randomForest(TRUC~. , data = Ztrain, mtry = ntry , ntree = ntree))

RF_predict_train <- predict(RF_model , Ztrain)                #predict the result with the train dataset
predict_train_table <- table(Ztrain$TRUC , RF_predict_train)          #obtain confusion matrix of the results of the trai
n dataset.

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model , Ztest)                 #predict the result with the test dataset
predict_test_table <- table(Ztest$TRUC , RF_predict_test)          #obtain confusion matrix of the results of the test da
taset.

round(prop.table(predict_test_table, 1),2)*100

5.1

ntree = 200    #set number of trees to build in the random forest
ntry = round(sqrt(PEV_opt))         #set the number of variables to randomly select at each node in the random forest

system.time(RF_model <- randomForest(TRUC~. , data = Ztrain, mtry = ntry , ntree = ntree, importance = TRUE))     #run ran
dom forest model with system time function

RF_predict_train <- predict(RF_model , Ztrain)                  #predict the result with the train dataset
predict_train_table <- table(Ztrain$TRUC , RF_predict_train)    #obtain confusion matrix of the results of the train datas
et.

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model , Ztest)                      #predict the result with the test dataset
predict_test_table <- table(Ztest$TRUC , RF_predict_test)         #obtain confusion matrix of the results of the test data
set.

round(prop.table(predict_test_table, 1),2)*100

kable(round(RF_model$importance[1:10,5:6],3))         #obtain top ten imporatance of the random forest model

IMk = RF_model$importance[1:10,5]
qplot(seq_along(IMk),IMk*100) +                           #plot the importance factors of the model with Mean reduce in
accuracy
  geom_point(shape=23, fill="blue", color="darkred", size=3) +
  labs(title = "Importance of Principal Components",
       x="PC",y="Mean Reduction in Accuracy (%)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,12,1),limits = c(0,12)) +
  scale_x_continuous(breaks = seq(1,10,1), limits = c(1,10))

ginik = RF_model$importance[1:10,6]
qplot(seq_along(ginik),ginik) +                           #plot the importance factors of the model with Mean decrease i
n gini
  geom_point(shape=21, fill="blue", color="darkred", size=3) +
  labs(title = "Mean Decrease in Gini Index",x="PC",y="Decrease in Gini Index") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,650,50),limits = c(0,650)) +
  scale_x_continuous(breaks = seq(1,10,1), limits = c(1,10))
```

```
important_df <- as.data.frame(RF_model$importance)                    #sort the top ten important PC in the model
important_df$PC <- rownames(important_df)
important_df <- important_df[order(-important_df$MeanDecreaseGini),][1:10,]

ggplot(important_df[10:1,], aes(x=reorder(PC, -MeanDecreaseGini), weight=MeanDecreaseGini)) +     #create a bar graph of t
he top 10 PC in the model by importance
  geom_bar(fill = "lightskyblue") +
  scale_fill_discrete(name="Variable Group") +
  ylab("Mean Decrease Gini") +
  xlab("Principal Component") +
  ggtitle("Ranked Importance of Principal Components") +
  theme(plot.title = element_text(hjust = 0.5))

PC_df <- data.frame(c(paste0("PC",1:PEV_opt)))        #return dataframe of importance factors with eigen values sorted by
 importance in the model.
PC_df <- cbind(PC_df, L_values[1:PEV_opt])
colnames(PC_df) = c("PC", "Eigenvalues")
important_df <- as.data.frame(RF_model$importance)
important_df$PC <- rownames(important_df)
important_PC <- merge(x = important_df, y = PC_df, by = "PC", all = TRUE)
important_PC <- important_PC[order(-important_PC$MeanDecreaseGini),][1:10,]
rownames(important_PC) <- important_PC$PC
important_PC <- important_PC[,-1]
important_PC
```

5.2

```
L_10 = L_values[1:10]
qplot(seq_along(L_10),L_10/400*100) +
  geom_point(shape=25, fill="blue", color="darkred", size=3) +
  labs(title = "PEV of Eigenvalues",x="Eigenvalue Index",y="Eigenvalue PEV") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,15,2.5), limits = c(0,15)) +
  scale_x_continuous(breaks = seq(1,10,1), limits = c(1,10))
```

5.3

```
scatterplot_5.3 = as.data.frame(cbind(IMk,L_10))         #create a dataframe that contains important PC in the model with
 the respective eigenvalue
names(scatterplot_5.3) = c("IMk","L_values")

PC = rownames(scatterplot_5.3)

ggplot(scatterplot_5.3, aes(x= IMk, y= L_values, label=PC))+       #create a scatter plot of the top ten important PC in t
he model with the respective eigenvalue
  geom_point(shape=20, fill="blue", color="darkred", size=3) +
  geom_text(aes(label=PC),hjust=-.2, vjust=0) +
  labs(title = "Eigenvalues vs. Importance",x="Importance",y="Eigenvalue") +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_y_continuous(breaks = seq(0,60,5), limits = c(0,60)) +
  scale_x_continuous(breaks = seq(0,.15,.025), limits = c(0,.15))

kable(scatterplot_5.3)
```

6.1

```r
round(prop.table(predict_test_table, 1),2)*100                #return the confusion matrix of the RF* model. the selected model.

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

CL1_Zsub = subset(ZDATA, TRUC == "CL1")[1:3]           #obtain the first two PC of each class
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"           #rename each of the first columns in the dataframes
names(CL2_Zsub)[1] = "CLASS"
names(CL3_Zsub)[1] = "CLASS"
names(CL4_Zsub)[1] = "CLASS"

CL12_Zsub = rbind(CL1_Zsub,CL2_Zsub)           #subset data and create dataframes of each combination of classes. Total combinations total 6. (4 * 3 ) / 2
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() +      #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("red", "blue"),name = "Legend") +                    #CL1 and CL2
  labs(title = "Comparison of CL1 and CL2") +
  theme(plot.title = element_text(hjust = 0.5))

ggplot(CL13_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +       #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("red", "springgreen2"),name = "Legend") +                    #CL1 and CL3
  labs(title = "Comparison of CL1 and CL2") +
  theme(plot.title = element_text(hjust = 0.5))

ggplot(CL14_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +       #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("red", "sienna3"),name = "Legend") +                    #CL1 and CL4
  labs(title = "Comparison of CL1 and CL4") +
  theme(plot.title = element_text(hjust = 0.5))

ggplot(CL23_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +       #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("blue", "springgreen2"),name = "Legend") +                    #Cl2 and CL3
  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() +       #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("blue", "sienna3"),name = "Legend") +                    #Cl2 and CL4
  labs(title = "Comparison of CL2 and CL4") +
  theme(plot.title = element_text(hjust = 0.5))

ggplot(CL34_Zsub,aes(x = PC1, y =PC2, color = factor(CLASS))) + geom_point() +       #create plots of the first two PC of each datapoint and plot them by class.
  scale_color_manual(values=c("springgreen2", "sienna3"),name = "Legend") +                    #Cl3 and Cl4
```

```
  labs(title = "Comparison of CL3 and CL4") +
  theme(plot.title = element_text(hjust = 0.5))
```

6.2

```
ntree = 200                          #create a random forest model only with the two worse preforming models in Cl3 and CL4
ntry = round(sqrt(PEV_opt))

Z_star = Ztrain[which(Ztrain$TRUC == "CL3" | Ztrain$TRUC == "CL4"),]                  #obtain data that is Cl3 and CL4 fo
r our train
Z_star$TRUC = factor(Z_star$TRUC)

Z_star_test = Ztest[which(Ztest$TRUC == "CL3" | Ztest$TRUC == "CL4"),]                  #obtain data that is Cl3 and CL4
for our test
Z_star_test$TRUC = factor(Z_star_test$TRUC)

system.time(RF_model <- randomForest(TRUC~. , data = Z_star, mtry = ntry , ntree = ntree))       #run random forest models
 on these two Classes

RF_predict_test <- predict(RF_model , Z_star_test)
predict_test_table <- table(Z_star_test$TRUC , RF_predict_test)         #return predicted class models on the test set

round(prop.table(predict_test_table, 1),2)*100
```

6.3

```
set.seed(1)                                          #preform K means on the entire dataset to find clusters of dat
a.
KDATA = ZDATA
KDATA$TRUC=as.factor(KDATA$TRUC)
levels(KDATA) = c(1:4)
system.time(KM_model <- kmeans(KDATA[-1], centers = 4, nstart = 100))              #run k means with 4 clusters and 100 r
andom starting points to class by

class_matrix <-table(KM_model$cluster,KDATA$TRUC)            #return the results of each cluster and the amount of each cl
ass that each class contains

CL11 = class_matrix[1,1]/sum(class_matrix[1,])#obtain gini for Cluster 1 class 1
CL12 = class_matrix[1,2]/sum(class_matrix[1,])#obtain gini for Cluster 1 class 2
CL13 = class_matrix[1,3]/sum(class_matrix[1,])#obtain gini for Cluster 1 class 3
CL14 = class_matrix[1,4]/sum(class_matrix[1,]) #obtain gini for Cluster 1 class 4

CL21 = class_matrix[2,1]/sum(class_matrix[2,]) #obtain gini for Cluster 2 class 1
CL22 = class_matrix[2,2]/sum(class_matrix[2,])#obtain gini for Cluster 2 class 2
CL23 = class_matrix[2,3]/sum(class_matrix[2,])#obtain gini for Cluster 2 class 3
CL24 = class_matrix[2,4]/sum(class_matrix[2,]) #obtain gini for Cluster 2 class 4

CL31 = class_matrix[3,1]/sum(class_matrix[3,])#obtain gini for Cluster 3 class 1
CL32 = class_matrix[3,2]/sum(class_matrix[3,])#obtain gini for Cluster 3 class 2
CL33 = class_matrix[3,3]/sum(class_matrix[3,])#obtain gini for Cluster 3 class 3
CL34 = class_matrix[3,4]/sum(class_matrix[3,])#obtain gini for Cluster 3 class 4

CL41 = class_matrix[4,1]/sum(class_matrix[4,])#obtain gini for Cluster 1 class 1
CL42 = class_matrix[4,2]/sum(class_matrix[4,])#obtain gini for Cluster 2 class 2
CL43 = class_matrix[4,3]/sum(class_matrix[4,])#obtain gini for Cluster 3 class 3
CL44 = class_matrix[4,4]/sum(class_matrix[4,])#obtain gini for Cluster 4 class 4

Gini_G1 <- CL11*(1-CL11)+CL12*(1-CL12)+CL13*(1-CL13)+CL14*(1-CL14#obtain total gini for cluster 1
Gini_G2 <- CL21*(1-CL21)+CL22*(1-CL22)+CL23*(1-CL23)+CL24*(1-CL24)       #obtain total gini for cluster 1
Gini_G3 <- CL31*(1-CL31)+CL32*(1-CL32)+CL33*(1-CL33)+CL34*(1-CL34)       #obtain total gini for cluster 1
```

```
Gini_G4 <- CL41*(1-CL41)+CL42*(1-CL42)+CL43*(1-CL43)+CL44*(1-CL44)       #obtain total gini for cluster 1

Gini_total <- Gini_G1 + Gini_G2 + Gini_G3+ Gini_G4                      #total gini for the model
gini_matrix = matrix(c(Gini_G1,Gini_G2,Gini_G3,Gini_G4), nrow = 4, ncol = 1)
purity_matrix = matrix(c(1,1,1,1), nrow = 4, ncol = 1) - gini_matrix                      #total purity for the model

round(gini_matrix,2)

round(purity_matrix,2)*100

#preform k means to find the centers of the data of each class
set.seed(1)
KDATA_1 = subset(KDATA, TRUC == "CL1")   #subset data by class
KDATA_2 = subset(KDATA, TRUC == "CL2")
KDATA_3 = subset(KDATA, TRUC == "CL3")
KDATA_4 = subset(KDATA, TRUC == "CL4")

KM_model_1 <- kmeans(KDATA_1[-1], centers = 1, nstart = 20)      #run k means for each class with only one center to find
the true or local centers
KM_model_2 <- kmeans(KDATA_2[-1], centers = 1, nstart = 20)
KM_model_3 <- kmeans(KDATA_3[-1], centers = 1, nstart = 20)
KM_model_4 <- kmeans(KDATA_4[-1], centers = 1, nstart = 20)

c1_pred = KM_model$centers[1,1:2]                        #obtain predicted centers of previous models
c2_pred = KM_model$centers[2,1:2]
c3_pred = KM_model$centers[3,1:2]
c4_pred = KM_model$centers[4,1:2]

c1_act = KM_model_1$centers[1,1:2]                       #obtain true centers for each class in the model
c2_act = KM_model_2$centers[1,1:2]
c3_act = KM_model_3$centers[1,1:2]
c4_act = KM_model_4$centers[1,1:2]

centers = as.data.frame(rbind(c1_pred, c2_pred, c3_pred, c4_pred,               #combine all the predicted centers and tru
e centers in one dataframe
                              c1_act, c2_act, c3_act, c4_act))

cluster = c("Unsupervised Center","Unsupervised Center",              #create labels for each row in the dataframe
            "Unsupervised Center","Unsupervised Center",
            "Center CL1","Center CL2",
            "Center CL3","Center CL4")

label_names = c("Center CL1","Center CL2", "Center CL3","Center CL4")


centers = as.data.frame(cbind(cluster, centers))
names(centers) = c("cluster","PC1","PC2")
centers$cluster = as.factor(centers$cluster)

ggplot(centers, aes(x = PC1, y = PC2, color = cluster)) +      #create plot of the centers produced by kmeans and true cen
ters for the first 2 Principal components
  geom_point(size = 5, aes(shape = cluster, color = cluster))

dist12 = dist(rbind(KM_model_1$centers[1],KM_model_2$centers[1]))         #obtain distances between each class to another
respective class from the k means.
dist13 = dist(rbind(KM_model_1$centers[2],KM_model_3$centers[1]))
dist14 = dist(rbind(KM_model_1$centers[3],KM_model_4$centers[1]))
dist23 = dist(rbind(KM_model_2$centers[4],KM_model_3$centers[1]))
dist24 = dist(rbind(KM_model_2$centers[4],KM_model_4$centers[1]))
```

```
dist34 = dist(rbind(KM_model_3$centers[4],KM_model_4$centers[1]))

kable(matrix(c("CL1 vs. CL2", "CL1 vs. CL3",
                       "CL1 vs. CL4", "CL2 vs. CL3",
                       "CL2 vs. CL4","CL3 vs. CL4",
                       round(dist12,2),round(dist13,2),
                       round(dist14,2),round(dist23,2),
                       round(dist24,2),round(dist34,2)),
                     ncol = 2))

D_values12 = NULL
D_values13 = NULL
D_values14 = NULL
D_values23 = NULL
D_values24 = NULL
D_values34 = NULL

for (i in 1:PEV_opt){     #preform k-s test for each column in each class to another respective class. total 6 combination
s
  colname = paste("PC",i,sep = "")
  D_values12[i] = as.numeric(ks.test(KDATA_1[[colname]], KDATA_2[[colname]])[1])
  D_values13[i] = as.numeric(ks.test(KDATA_1[[colname]], KDATA_3[[colname]])[1])
  D_values14[i] = as.numeric(ks.test(KDATA_1[[colname]], KDATA_4[[colname]])[1])
  D_values23[i] = as.numeric(ks.test(KDATA_2[[colname]], KDATA_3[[colname]])[1])
  D_values24[i] = as.numeric(ks.test(KDATA_2[[colname]], KDATA_4[[colname]])[1])
  D_values34[i] = as.numeric(ks.test(KDATA_3[[colname]], KDATA_4[[colname]])[1])
}

D_values = as.data.frame(cbind(c(1:PEV_opt),D_values12,D_values13,              #create a dataframe of the resulting D val
ues
                 D_values14,D_values23,D_values24,D_values34))

names(D_values) = c("PCk","D12","D13","D14","D23","D24","D34")

ggplot(D_values, aes(x = PCk, group = 1)) +              #create a smoothing plot of the resulting D value
  geom_smooth(aes(y=D12,color = "D12")) +
  geom_smooth(aes(y=D13,color = "D13")) +
  geom_smooth(aes(y=D14,color = "D14")) +
  geom_smooth(aes(y=D23,color = "D23")) +
  geom_smooth(aes(y=D24,color = "D24")) +
  geom_smooth(aes(y=D34,color = "D34")) +
  ylab("Vertical Distance between CDFs") +
  ggtitle("K-S Test D-Values") +
  theme(plot.title = element_text(hjust = 0.5))

KM_clusters = cbind(KM_model$cluster,KDATA)
names(KM_clusters) = c("Gk","TRUC",paste0("PC",1:PEV_opt, sep = ""))
G1 = subset(KM_clusters,Gk == 1)
G2 = subset(KM_clusters,Gk == 2)
G3 = subset(KM_clusters,Gk == 3)
G4 = subset(KM_clusters,Gk == 4)

#randomly partitioning data into train/test set
G1_split = random_subset(G1)
G1_train = G1_split[[1]] #extracting training set
G1_test = G1_split[[2]]#extracting test set

G2_split = random_subset(G2)
G2_train = G2_split[[1]] #extracting training set
```

```
G2_test = G2_split[[2]]#extracting test set

G3_split = random_subset(G3)
G3_train = G3_split[[1]] #extracting training set
G3_test = G3_split[[2]]#extracting test set

G4_split = random_subset(G4)
G4_train = G4_split[[1]] #extracting training set
G4_test = G4_split[[2]]#extracting test set

ntree = 200
ntry = round(sqrt(PEV_opt))

#G1
system.time(RF_1 <- randomForest(TRUC~. , data = G1_train[-1],                    #preform random forest for cluster 1
                                 mtry = ntry , ntree = ntree))

##    user  system elapsed
##   0.612   0.006   0.619

RF_predict_train_1 <- predict(RF_1 , G1_train)                      #obtain the training results of the random forest
predict_train_table_1 <- table(G1_train$TRUC , RF_predict_train_1)

round(prop.table(predict_train_table_1, 1),2)*100

RF_predict_test_1 <- predict(RF_1 , G1_test)                     #obtain the test results on the test data for the random
forest
predict_test_table_1 <- table(G1_test$TRUC , RF_predict_test_1)
round(prop.table(predict_test_table_1, 1),2)*100

ntree = 200
ntry = round(sqrt(PEV_opt))

#G2
system.time(RF_2 <- randomForest(TRUC~. , data = G2_train[-1],                    #preform random forest for cluster 2
                                 mtry = ntry , ntree = ntree))

RF_predict_train_2 <- predict(RF_2 , G2_train)
predict_train_table_2 <- table(G2_train$TRUC , RF_predict_train_2)

round(prop.table(predict_train_table_2, 1),2)*100

RF_predict_test_2 <- predict(RF_2 , G2_test)
predict_test_table_2 <- table(G2_test$TRUC , RF_predict_test_2)
round(prop.table(predict_test_table_2, 1),2)*100

ntree = 200
ntry = round(sqrt(PEV_opt))

#G3
system.time(RF_3 <- randomForest(TRUC~. , data = G3_train[-1],                    #preform random forest for cluster 3
                                 mtry = ntry , ntree = ntree))

RF_predict_train_3 <- predict(RF_3 , G3_train)
predict_train_table_3 <- table(G3_train$TRUC , RF_predict_train_3)

round(prop.table(predict_train_table_3, 1),2)*100

RF_predict_test_3 <- predict(RF_3 , G3_test)
predict_test_table_3 <- table(G3_test$TRUC , RF_predict_test_3)
round(prop.table(predict_test_table_3, 1),2)*100
```

```
ntree = 200
ntry = round(sqrt(PEV_opt))

#G4
system.time(RF_4 <- randomForest(TRUC~. , data = G4_train[-1],                    #preform random forest for cluster 4
                                 mtry = ntry , ntree = ntree))

RF_predict_train_4 <- predict(RF_4 , G4_train)
predict_train_table_4 <- table(G4_train$TRUC , RF_predict_train_4)

round(prop.table(predict_train_table_4, 1),2)*100

RF_predict_test_4 <- predict(RF_4 , G4_test)
predict_test_table_4 <- table(G4_test$TRUC , RF_predict_test_4)
round(prop.table(predict_test_table_4, 1),2)*100

CL1_ZsubTrain = subset(Ztrain, TRUC == "CL1")                     #subset data base on the class of the data.
CL2_ZsubTrain = subset(Ztrain, TRUC == "CL2")                     #CL2 subset for train
CL3_ZsubTrain = subset(Ztrain, TRUC == "CL3")                     #CL3 subset for train
CL4_ZsubTrain = subset(Ztrain, TRUC == "CL4")                     #CL4 subset for train


CL1_ZsubTest = subset(Ztest, TRUC == "CL1")                      #CL1 subset for test
CL2_ZsubTest = subset(Ztest, TRUC == "CL2")                      #CL2 subset for test
CL3_ZsubTest = subset(Ztest, TRUC == "CL3")                      #CL3 subset for test
CL4_ZsubTest = subset(Ztest, TRUC == "CL4")                      #CL4 subset for test


CL12_ZsubTrain = rbind(CL1_ZsubTrain,CL2_ZsubTrain)                    #partition subset combinations of each class. to
tal of 6 combinations
CL13_ZsubTrain = rbind(CL1_ZsubTrain,CL3_ZsubTrain)                    #CL1 and CL3 train set
CL14_ZsubTrain = rbind(CL1_ZsubTrain,CL4_ZsubTrain)                    #CL1 and CL4 train set
CL23_ZsubTrain = rbind(CL2_ZsubTrain,CL3_ZsubTrain)                    #CL2 and CL3 train set
CL24_ZsubTrain = rbind(CL2_ZsubTrain,CL4_ZsubTrain)                    #CL2 and CL4 train set
CL34_ZsubTrain = rbind(CL3_ZsubTrain,CL4_ZsubTrain)                    #CL4 and CL3 train set


CL12_ZsubTest = rbind(CL1_ZsubTest,CL2_ZsubTest)                    #CL1 and CL2 test set
CL13_ZsubTest = rbind(CL1_ZsubTest,CL3_ZsubTest)                    #CL1 and CL3 test set
CL14_ZsubTest = rbind(CL1_ZsubTest,CL4_ZsubTest)                    #CL1 and CL4 test set
CL23_ZsubTest = rbind(CL2_ZsubTest,CL3_ZsubTest)                    #CL2 and CL3 test set
CL24_ZsubTest = rbind(CL2_ZsubTest,CL4_ZsubTest)                    #CL2 and CL4 test set
CL34_ZsubTest = rbind(CL3_ZsubTest,CL4_ZsubTest)                    #CL3 and CL4 test set

ntree = 200
ntry = round(sqrt(PEV_opt))

CL12_ZsubTrain[] <- lapply(CL12_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)   #reset factors on each dataset
 since we have less factors now
CL12_ZsubTest[] <- lapply(CL12_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_12 <- randomForest(TRUC~. , data = CL12_ZsubTrain, mtry = ntry , ntree = ntree))   #create random for
est model on CL1 and CL2

#obtain results to the random forest model
RF_predict_train <- predict(RF_model_12 , CL12_ZsubTrain)
predict_train_table <- table(CL12_ZsubTrain$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100
```

```
RF_predict_test <- predict(RF_model_12 , CL12_ZsubTest)
predict_test_table_12 <- table(CL12_ZsubTest$TRUC , RF_predict_test)
round(prop.table(predict_test_table_12, 1),2)*100

ntree = 200
ntry = round(sqrt(PEV_opt))

CL13_ZsubTrain[] <- lapply(CL13_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)
CL13_ZsubTest[] <- lapply(CL13_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_13 <- randomForest(TRUC~. , data = CL13_ZsubTrain, mtry = ntry , ntree = ntree))  #create random fore
st model on CL1 and CL3

#obtain results to random forest model
RF_predict_train <- predict(RF_model_13 , CL13_ZsubTrain)
predict_train_table <- table(CL13_ZsubTrain$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model_13 , CL13_ZsubTest)
predict_test_table_13 <- table(CL13_ZsubTest$TRUC , RF_predict_test)

round(prop.table(predict_test_table_13, 1),2)*100

ntree = 200
ntry = round(sqrt(PEV_opt))

CL14_ZsubTrain[] <- lapply(CL14_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)
CL14_ZsubTest[] <- lapply(CL14_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_14 <- randomForest(TRUC~. , data = CL14_ZsubTrain, mtry = ntry , ntree = ntree))  #create random fore
st model on CL1 and CL4

#obtain results to random forest model
RF_predict_train <- predict(RF_model_14 , CL14_ZsubTrain)
predict_train_table <- table(CL14_ZsubTrain$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model_14 , CL14_ZsubTest)
predict_test_table_14 <- table(CL14_ZsubTest$TRUC , RF_predict_test)

round(prop.table(predict_test_table_14, 1),2)*100

ntree = 200
ntry = round(sqrt(PEV_opt))

CL23_ZsubTrain[] <- lapply(CL23_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)
CL23_ZsubTest[] <- lapply(CL23_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_23 <- randomForest(TRUC~. , data = CL23_ZsubTrain, mtry = ntry , ntree = ntree))  #create random fore
st model on CL2 and CL3

#obtain results to random forest model
RF_predict_train <- predict(RF_model_23 , CL23_ZsubTrain)
predict_train_table <- table(CL23_ZsubTrain$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model_23 , CL23_ZsubTest)
predict_test_table_23 <- table(CL23_ZsubTest$TRUC , RF_predict_test)

round(prop.table(predict_test_table_23, 1),2)*100
```

```
CL24_ZsubTest[] <- lapply(CL24_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
CL24_ZsubTrain[] <- lapply(CL24_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_24 <- randomForest(TRUC~. , data = CL24_ZsubTest, mtry = ntry , ntree = ntree))  #create random fores
t model on CL2 and CL4

#obtain results to random forest model
RF_predict_train <- predict(RF_model_24 , CL24_ZsubTest)
predict_train_table <- table(CL24_ZsubTest$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model_24 , CL24_ZsubTrain)
predict_test_table_24 <- table(CL24_ZsubTrain$TRUC , RF_predict_test)

round(prop.table(predict_test_table_24, 1),2)*100

CL34_ZsubTrain[] <- lapply(CL34_ZsubTrain, function(x) if(is.factor(x)) factor(x) else x)
CL34_ZsubTest[] <- lapply(CL34_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
system.time(RF_model_34 <- randomForest(TRUC~. , data = CL34_ZsubTrain, mtry = ntry , ntree = ntree))  #create random fore
st model on CL3 and CL4

#obtain results to random forest model
RF_predict_train <- predict(RF_model_34 , CL34_ZsubTrain)
predict_train_table <- table(CL34_ZsubTrain$TRUC , RF_predict_train)

round(prop.table(predict_train_table, 1),2)*100

RF_predict_test <- predict(RF_model_34 , CL34_ZsubTest)
predict_test_table_34 <- table(CL34_ZsubTest$TRUC , RF_predict_test)

round(prop.table(predict_test_table_34, 1),2)*100

#obtain global accuracy of each random forest model
Accuracy12 <- round(sum(diag(predict_test_table_12)) / sum(predict_test_table_12)*100,2)
Accuracy13 <- round(sum(diag(predict_test_table_13)) / sum(predict_test_table_13)*100,2)
Accuracy14 <- round(sum(diag(predict_test_table_14)) / sum(predict_test_table_14)*100,2)
Accuracy23 <- round(sum(diag(predict_test_table_23)) / sum(predict_test_table_23)*100,2)
Accuracy24 <- round(sum(diag(predict_test_table_24)) / sum(predict_test_table_24)*100,2)
Accuracy34 <- round(sum(diag(predict_test_table_34)) / sum(predict_test_table_34)*100,2)


#obtain average performance of each model by class
CL1_Accuracy <- (Accuracy12 +Accuracy13 + Accuracy14) / 3
CL1_Accuracy

CL2_Accuracy <- (Accuracy12 +Accuracy23 + Accuracy24) / 3
CL2_Accuracy

CL3_Accuracy <- (Accuracy34 +Accuracy23 + Accuracy13) / 3
CL3_Accuracy

CL4_Accuracy <- (Accuracy34 +Accuracy24 + Accuracy14) / 3
CL4_Accuracy

#reset factors in each subset dataframe
CL1_ZsubTest[] <- lapply(CL1_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
CL2_ZsubTest[] <- lapply(CL2_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
CL3_ZsubTest[] <- lapply(CL3_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)
CL4_ZsubTest[] <- lapply(CL4_ZsubTest, function(x) if(is.factor(x)) factor(x) else x)

#function to obtain Random forest voting for each random forest subjected to the given class
```

```r
Accuracy_aggregator <- function(Test_set, RF1, RF2,RF3){   #inputs of of a test set case and 3 random forest models that h
as been produced.
  df <- c()
  for (i in 1:nrow(Test_set)){                              #for each case in test set
    RF_pred1 <- predict(RF1, Test_set[i,])[1]                              #predict the following class on the a random fo
rest model
    RF_pred2 <- predict(RF2, Test_set[i,])[1]                              #predict the following class on the a random fo
rest model
    RF_pred3 <- predict(RF3, Test_set[i,])[1]                              #predict the following class on the a random fo
rest model
    if (RF_pred1 == Test_set[i,1]) {                                  #if the predicted class from the random forest is equa
l to the true class of the case
    RF_pred1 = 1                                                      #set the value of the predicted to be 1 or true
      } else {                                                        #otherwise if it does not equal to the true class
    RF_pred1 = 0                                                      #set the value of the predicted to be 0 or false
      }

    if (RF_pred2 == Test_set[i,1]) {                                 #repeat the following for each random forest predictor
    RF_pred2 = 1
      } else {
    RF_pred2 = 0
      }

    if (RF_pred3 == Test_set[i,1]) {
    RF_pred3 = 1
      } else {
    RF_pred3 = 0
      }
    row_df <- cbind(RF_pred1,RF_pred2,RF_pred3)                              #create a row of the results for each of the
random forest models
    df <- rbind(df, row_df)                                                  #combine the rows to a dataframe as one

  }

  #
  list <- rowSums(df[ , c("RF_pred1" ,"RF_pred2", "RF_pred3")])              #create a new column that finds the sum of ea
ch row
  df <- cbind(df, list)
  return(sum(df[,4] >=2 )/ nrow(df))                                         #return the percent of rows that are more tha
n 2 in the sum column. Since this is majority voting, if there is 2/3 of the row, it is defined the class to be correct. i
f not, the given class vote is wrong.
}

CL1_aggregator <- Accuracy_aggregator(CL1_ZsubTest, RF_model_12, RF_model_13, RF_model_14)
CL1_aggregator

CL2_aggregator <- Accuracy_aggregator(CL2_ZsubTest, RF_model_12, RF_model_23, RF_model_24)
CL2_aggregator

CL3_aggregator <- Accuracy_aggregator(CL3_ZsubTest, RF_model_13, RF_model_23, RF_model_34)
CL3_aggregator

CL4_aggregator <- Accuracy_aggregator(CL4_ZsubTest, RF_model_14, RF_model_24, RF_model_34)
CL4_aggregator
```