10/31/2021

MATH 6350

Font Classifications with Random Forest

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 an 20×20 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. 4 files are selected out of 153 files.

The size of each class will be compared and if they are imbalanced, cloning, perturbation, or SMOTE will be applied to oversample the smallest class to get a more balanced data set. Otherwise, the classifier will be biased.

After the data is cleaned, treated, and applied principal component analysis (PCA) to reduce the number of features, the data will be split into 20% of test set and 80% of training set randomly.

This report is done in R programming language. The algorithmic principles for basic random forest will be interpreted. Several important or most common used inputs, parameters, options will be explained. The outputs of the random forest will be discussed as well. Also, the predict function in R will be listed and explained as well.

When everything is ready, apply the Random Forest Automatic Classifier to build the model. 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.

Then, use these two sets to find out the best number of trees in the random forest model. Also, the importance of first 10 principal components will be discussed and compared to the percentage of explained variance. Final section will discuss several potential methods to improve the classification.

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 style 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 $20 \times 20 = 400$ 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

Class	File	Size	Percentage
CL1	GILL.csv	1459	27%
CL2	LEELAWADEE.csv	1378	25%
CL3	ROMAN.csv	1194	22%
CL4	TECHNIC.csv	1380	26%
Total		5411	100%

Oversampling the data set

Since each class has about $22 \sim 27\%$ 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 5411×400 data frame.

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. Fortunately, R provides a built-in function, prcomp(), for PCA and the eigenvalue, eigenvector, and the principal components can be easily obtained.

Percentage of Explained Variance (PEV)

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 400×400 eigenvector will be computed together with 400 eigenvalue.

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 1 shows the PEV versus m.

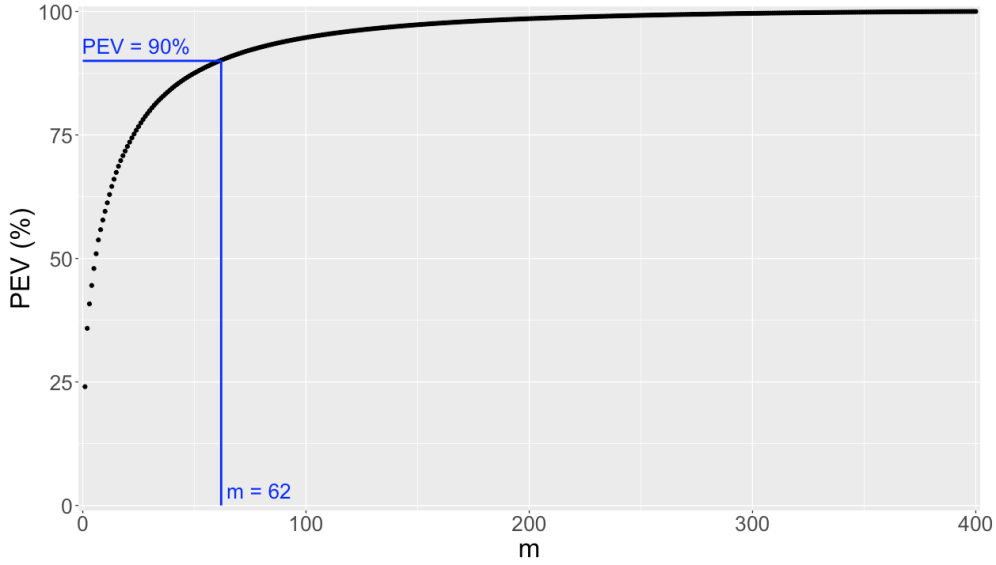

Figure 1 Percentage of Explained Variance (PEV) vs m

Data after applying Principal Component Analysis

By selecting the first 62 principal component, the final data set will have dimension of 5411×62. Combine with the true classification of each case the final data set has dimension of 5411×63.

Table 2 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

1 Data Splitting

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 3 shows that there are 1080 cases of test set and 4331 cases of training set.

Table 3 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 Random Forest Algorithm Description

2.1 Outline

Random Forest is a supervised learning method (it also can be used in unsupervised mode for proximities among data points) that comprises a large group of specialized decision trees. The response variable can be either qualitative or quantitative. Random forests add an additional degree of randomness that is not available to other similar methods, such as bagging. The additional randomness is due to when the algorithm searches for the best feature variable for splitting a node, random forests will select a feature from a random subset of all the features. This creates many diverse trees that can be analyzed for their predictive capabilities. The subset is typically equal to the square root of the total number of features.

Another impressive feature of the random forest algorithm is that it can compute the importance of each feature. It computes this by analyzing the accuracy of trees that do not contain this particular feature. If the accuracy decreases substantially, then the feature is important to our model and should be included to increase the predictive power. The predictive power of the model can be improved in several ways. Firstly, you can increase the number of total trees that the algorithm will create. This will stabilize the accuracies as well as increasing the predictive capability of the algorithm. Secondly, you can change the number of features that the algorithm will consider while constructing each node of the tree. This will change the degree of randomness leading to a large contrast between the types of trees created.

2.2 Random Forest

2.2.1 Inputs, Parameters, Options

The input for the random forest classifier is the features' values of training set and the true classification of the training set. The most important parameters:

- ntree: Specifies the number of trees you would like the algorithm to create. The default value is 500.
- mtry: The number of variables randomly selected as candidates for the features used in the splitting a specific node. The default values are the square root of the total number of features for classification and the total number of features divided by three for regression.

- importance: This parameter is used if you would like the algorithm to consider the importance of each feature before it samples the number of features to be used in node creation. The default value is FALSE, indication of a perfectly random selection of features.

Optional extra parameters:

- nodesize: Indicates the minimum size of terminal nodes. Setting this number larger will cause smaller trees to be grown and decrease computational time.
- maxnodes: Indicates the maximum number of terminal nodes that trees in the random forest can have. If left unspecified, the trees will have the maximum number of nodes possible.

2.2.2 Outputs

Below are most common examined outputs for a random forest model

- predicted: Returns the predicted values of the input data based on the out-of-bag samples.
- type: Which type of method is being implemented (regression, classification, or unsupervised).
- importance: Calculates the mean decrease in accuracy when a specific feature is removed from the algorithm. It computes the importance of all the features.
- err.rate: Computes the error rates of the predictions for each class as well as the out of bag error rate for the classification.
- confusion: Creates the confusion matrix of the result.

2.3 Prediction

For most of the built models in R will be used to predict the result for the test set. Thus, the predict() function can predict the result with a defined model.

2.3.1 Inputs

The most common used inputs for predict() function in R are as below:

- object: a model project for which predictions are desired. The built model such as linear regression model and random forest model and the test set to be predicted.
- newdata: the data which is desired to be predicted.

2.3.2 Outputs

The output for the predict() function will be the predicted responses for the selected model object.

3 Random Forest Application

To compare with the performance of kNN automatic classifier, the training set and test set has been intentionally selected as same as they are in Homework 2. Table 4 shows that confusion matrix by using kNN automatic classifier with $k = 3$.

Table 4 Confusion Matrix by using kNN classifier with k = 3

		True			
		1	2	3	4
Predicted	1	82.8	10.5	4.2	6.2
	2	3.4	65.5	3.8	1.8
	3	2.4	8.4	83.6	0.4
	4	11.3	15.6	8.4	91.7

Now, the training set will be used to construct the random forest model with below hyperparameters as initial value:

1. number of trees = 100
2. number of features = $\sqrt{r}$, where r is the smallest number to have $PEV(r) \geq 90\%$

After the random forest classifier is built, using the training set and the test set to compute global accuracy and two confusion matrices respectively.

Since the random forest is used as a supervised classifier in this report, the training set prediction accuracy and test set prediction accuracy can be computed as below.

Table 5 Global Accuracy of Training Set and Test Set

Set	Accuracy
Training	98.3
Test	86.9

It is not surprising that the global training set prediction accuracy is very high since that the training set has been used to construct the model. Due to the algorithm of bagging, each training case has very high chance ($\approx 63.2\%$) to be used in building some of the decision trees in a random forest. If a training case has been used to build any decision tree in a random forest, the prediction of that training case for that specific decision tree will always be correct. Thus, the overall prediction accuracy of training set will be very high. On the other hand, since the test set is not used to build the model, it will not have the situation mentioned above.

To observe the prediction accuracy of each class, the confusion matrices for both training set and test set are computed as below. In Table 6, the all the misclassified cases in Class 2, Class 3, and Class 4 are classified as Class 1. It can be guessed that Class 1 has some "special features" that the algorithm will identify them as very important key to classify as Class 1. However, all these misclassified cases may have some features very similar to those "special features" so the classifier will misclassify them.

From Table 7, it can be observed that same as confusion matrix of training set, there a are a few misclassified cases in Class 2, Class 3, and Class 4 are classified as Class 1 in the test set. Also, Class 2 has the lowest prediction accuracy among the 4 classes which is consistent with the result of kNN automatic classifier as shown in Table 4. However, the accuracy is improved from 65.5% to 78.5%.

Table 6 Confusion Matrix of Training Set

		True			
		1	2	3	4
Predicted	1	99.9	3.0	2.3	1.7
	2	0.1	97.0	0	0
	3	0	0	97.7	0
	4	0	0	0	98.3

Table 7 Confusion Matrix of Test Set

		True			
		1	2	3	4
Predicted	1	90.3	13.1	4.6	6.2
	2	7.6	78.5	8.4	0.7
	3	0.0	3.3	85.3	0.0
	4	2.1	5.1	1.7	93.1

4 Random Forest Application with different number of trees

To find out the best number of trees for random forest classifier, two different number of trees i.e., 200 and 300 are used to construct and compare the accuracies on the test set.

From Figure 2, it can be observed that the accuracy reaches the highest value for both point estimation and 90% confidence interval when number of trees equals to 100 and 300. Thus, 100 trees classifier is selected as best RF classifier for the rest of sections for this report to reduce the computation time.

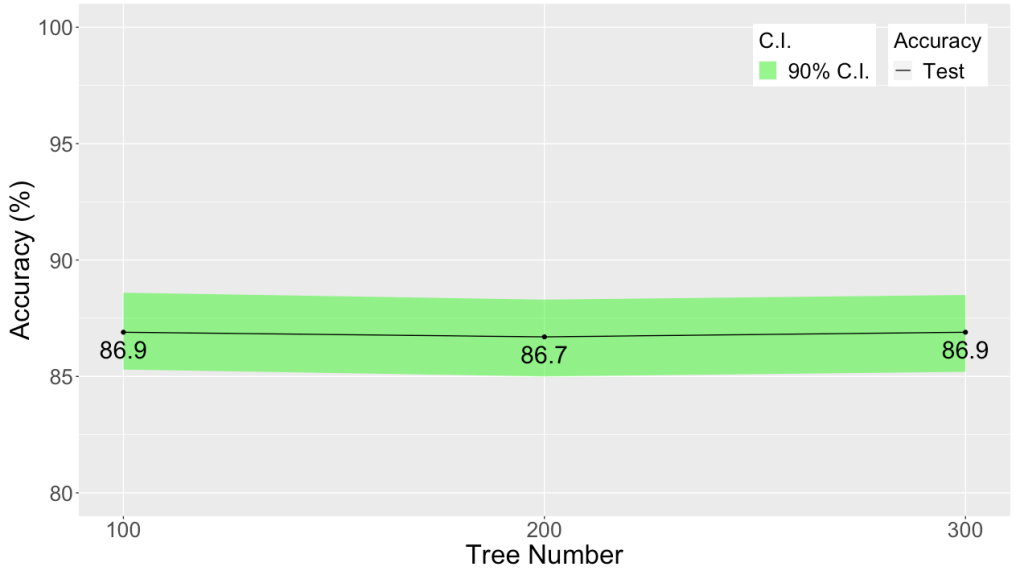

Figure 2 Prediction Accuracy for number of trees = 100, 200, 300 in random forest model

5 Feature Importance

5.1 Accuracy Loss

To calculate the importance of each feature (principal component for our data set), the feature importance shall be calculated. We would like to know how important is each feature that will impact the classification result. Since the data set has been applied PCA, it can be expected that the first principal component will be the most important feature and second principal component will follow.

Fortunately, randomForest() function in R has already calculated this for us. "MeanDecreaseAccuracy" in the importance output from the random forest model gives

a rough estimate of the loss in prediction performance, accuracy loss, when a particular feature is omitted from the training set i.e., the importance for that particular feature.

$$Accuracy\ Loss = Accuracy\ (Full\ Model) - Accuracy\ (Reduced\ Model)$$

It is noted that if two variables are somewhat redundant, then omitting one of them may not lead to massive gains in prediction performance but would make the second variable more important.

From Figure 3, top 10 important features are shown. It can be observed that the first principal component is the most important feature which contributes around 19%.

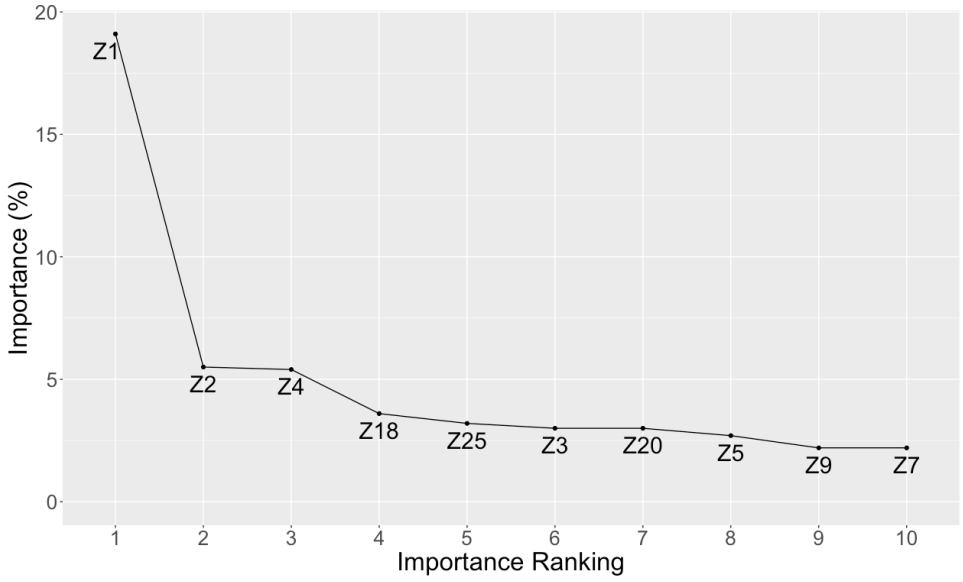

Figure 3 Feature Importance of First 10 Principal Components

5.2 Meaning of Eigenvalue of PCA

The eigenvalue, L, of a correlation matrix is a useful measure of explained variance. The explained variance tells us how much information (variance) can be attributed to each of the principal components. In another word, it can be imaged that $L_k/400$ means how much information can be provided by the principal component (Z_k) related to the particular eigenvalue (L_k) with respect to the original 400 features. The sum of the eigenvalues will equal to the number of features of the original data set i.e., data before PCA. For our cases,

$$\sum_{k=1}^{400} \frac{L_k}{400} = 1$$

Thus, the higher the eigenvalue, the more information that particular principal component will provide.

5.3 Importance vs Eigenvalue

Comparing the importance of a principal component and its respective eigenvalue, it can be observed that there is positive linear association between accuracy loss and eigenvalue.

The scatterplot between Eigenvalue and Accuracy Loss (Feature Importance) is shown in Figure 4.

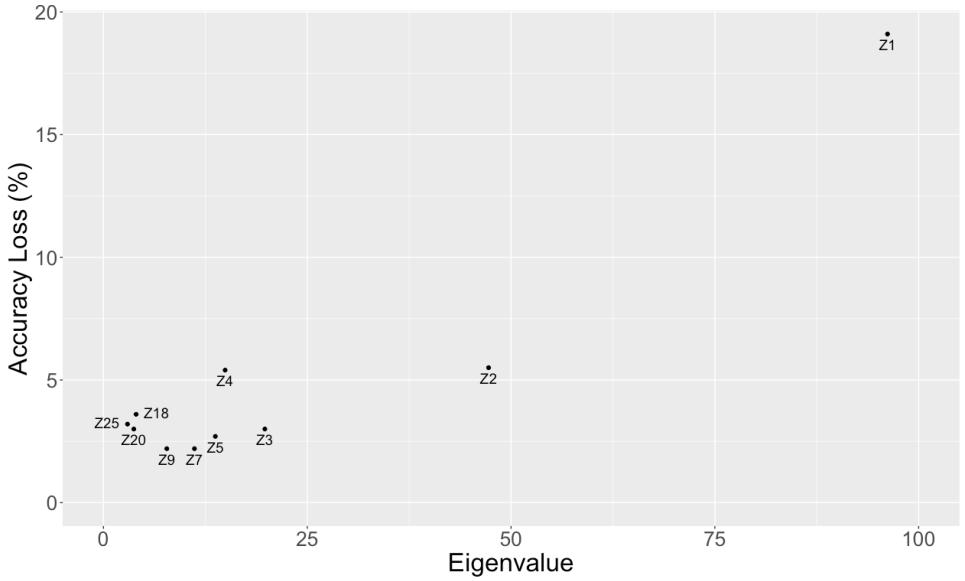

Figure 4 Accuracy Loss versus Eigenvalue

To verify the estimation, the correlation between Accuracy Loss (Feature Importance) and Eigenvalue of top 10 important feature is $r^2 = 0.935$ which implies that there is strong linear association between the Accuracy Loss and Eigenvalue. In other word, the higher the Accuracy Loss (Feature Importance), the higher the Eigenvalue it will be.

6 Performance Comparison and Automatic Classifier Improvement

6.1 Worst Classification among two classes

Refer to Section 4, it can be determined that the best number of trees among 100, 200, and 300 for the RF classifier is 100. The confusion matrix of best RF i.e., number of trees = 100 is shown in Table 8. It can be observed that the classification between Class 1 and Class 2 is the worst. Table 9 shows that misclassified rate between 2 Classes while the RF is built based on all 4 Classes included. Thus, this misclassified rate between two classes will be impact by all 4 Classes. So, it is reasonable to build the RF model only based on two classes and see the accuracy rate in Section 6.2.

Table 8 Confusion Matrix of Test Set

		True			
		1	2	3	4
Predicted	1	90.3	13.1	4.6	6.2
	2	7.6	78.5	8.4	0.7
	3	0.0	3.3	85.3	0.0
	4	2.1	5.1	1.7	93.1

Table 9 Misclassified Rate between 2 Classes

CL_i	CL_j	Misclassified Rate
1	2	20.7
1	3	4.6
1	4	8.3
2	3	11.7
2	4	5.8
3	4	1.7

6.2 Test Accuracy between two classes

In this section, 6 RF models are built. Each of them is constructed by only using 2 classes. Thus, we can easily compare which two classes are most difficult to be distinguished without the impact by other 2 unused classes. Table 10 to Table 15 show the confusion matrices. It is obvious that Class 1 and Class 2 are more difficult to classify than other pairs. However, the accuracy is still high. Also, most of the RF classifiers provide an accuracy rate larger than 90%.

Compared to the result in Section 6.1, it can be observed that without Class 3 and Class 4, there is no improvement of classification between Class 1 and Class 2.

Table 10 Confusion Matrix between Class 1 and Class2

		True	
		1	2
Predicted	1	90.4	13.1
	2	9.6	86.9

Table 11 Confusion Matrix between Class 1 and Class 3

		True	
		1	3
Predicted	1	99.3	8.0
	3	0.7	92.0

Table 12 Confusion Matrix between Class 1 and Class 4

		True	
		1	4
Predicted	1	96.2	6.9
	4	3.8	93.1

Table 13 Confusion Matrix between Class 2 and Class 3

		True	
		2	3
Predicted	2	96.0	13.9
	3	4.0	86.1

Table 14 Confusion Matrix between Class 2 and Class 4

		True	
		2	4
Predicted	2	94.2	6.2
	4	5.8	93.8

Table 15 Confusion Matrix between Class 3 and Class 4

		True	
		3	4
Predicted	3	92.9	3.3
	4	7.1	96.7

6.3 Improvement

Although the prediction accuracy is already high, it is reasonable to find some methods to improve the random forest models. Below subsections provide 5 methods and we would like to see if by any chance they can improve the accuracy. To have a meaningful comparison on the result, the test set will be fixed (20% of the entire data set created in Section 1).

6.3.1 Increase the number of trees

Since we only built RF model with tree number = 100, 200, and 300, we are wondering if increasing the number of trees to a larger number will the accuracy be improved or not. Thus, we fix the number of features = $\sqrt{r}$ and change the number of trees to get the accuracy. Figure 5 shows the accuracy for different tree numbers increased by 10 from 10 to 500. It can be observed that the accuracy falls around 85% and 87.5% after tree numbers > 50. Since it will require more computation time for larger number of trees, it is wise to stick with tree numbers = 100.

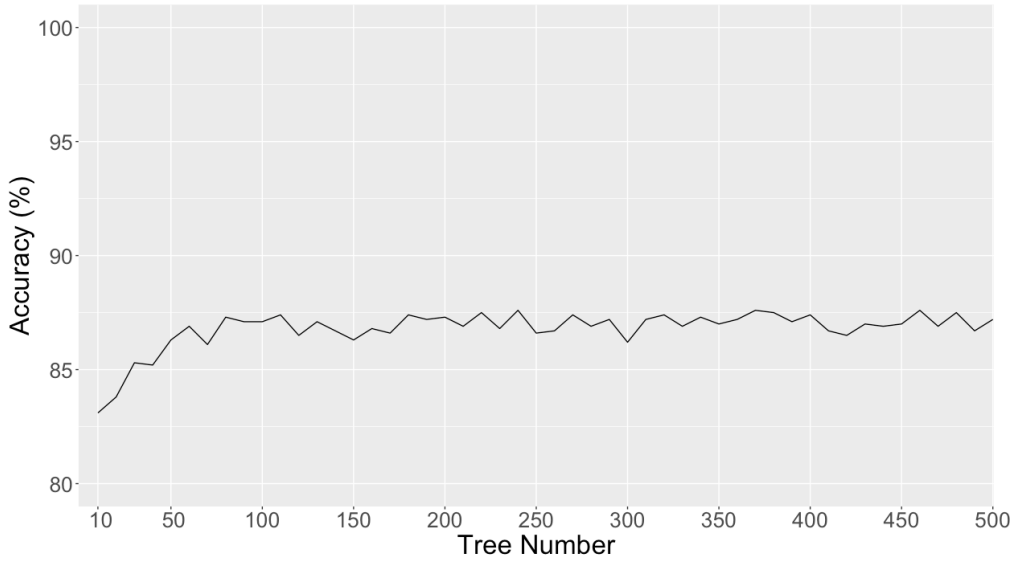

Figure 5 Accuracy vs Tree Number

6.3.2 Change the number of features to be used in each tree in random forest

Since the default number of features for RF model is the squared root of total number of features, it is interesting to see if the increase or decrease the number of features will have impact on the result. Thus, we fix the tree number = 100 and change the number of features to get the accuracy. Figure 6 shows the accuracy for different number of features (from 1 to 61) being used for RF model. Noted that if all 62 features are used, it becomes a bagging model. It can be observed that the accuracy is decreasing as we include more features while building the RF model.

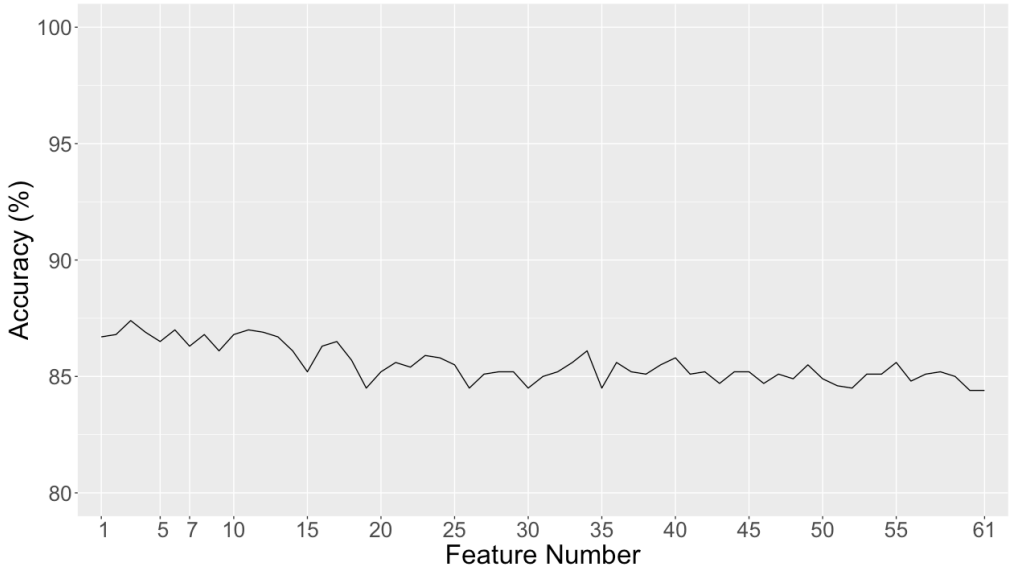

Figure 6 Accuracy vs Feature Number

6.3.3 Preliminary clustering by k-means before RF Classification

Firstly, repeat k-means 20 times to find out a good dispersion of the clustering i.e., minimize the total dispersion. The training data set is split into 4 clusters G_1, G_2, G_3, G_4 with centers c_1, c_2, c_3, c_4 and sizes of N_1, N_2, N_3, N_4. Second, construct RF classifiers separately for each cluster which will provide 4 classifiers RF_1, RF_2, RF_3, RF_4.

Each test case, X_n, will be clustered to its closest center c_j with respect to cluster G_j. Then, classify X_n into class predicted by RF_{Gj}.

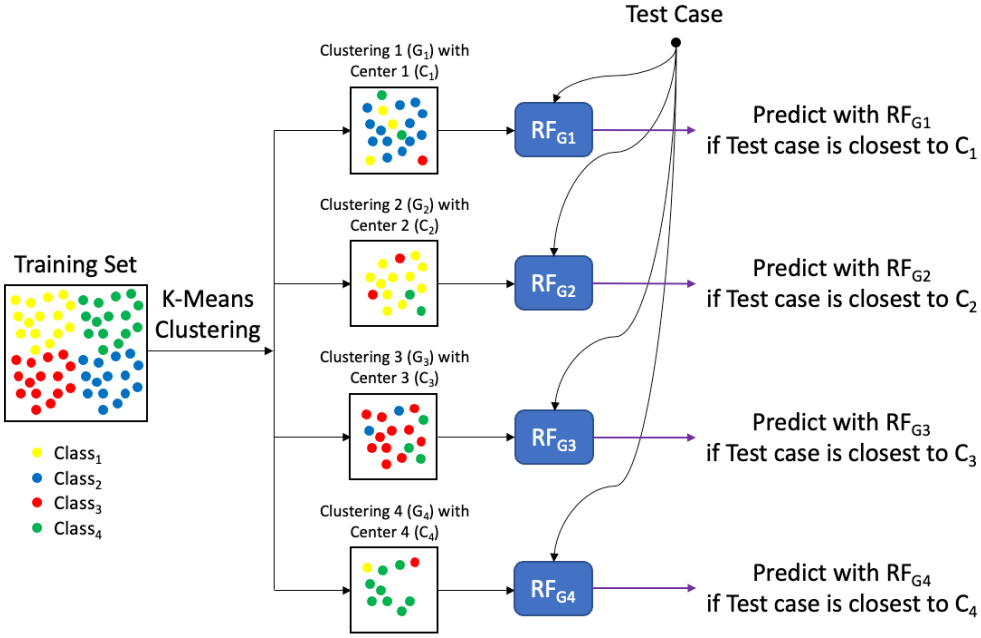

Figure 7 Flow Chart for Preliminary clustering by k-means before RF Classification

Table 16 shows that there is no improvement on the classification.

Table 16 Confusion Matrix of Preliminary k-means Clustering before RF Classification

		True			
		1	2	3	4
Predicted	1	86.9	10.9	3.4	5.4
	2	9.6	78.9	9.2	1.4
	3	0.3	4.0	85.7	0.0
	4	3.1	6.2	1.7	93.1

6.3.4 RF Classifiers for pairs of classes

Same as Section 6.2, we would like to construct separately RF_{ij} which is classifier of CL_i versus CL_j. Since we have 4 classes, we will get 6 classifiers as RF_{12}, RF_{13}, RF_{14}, RF_{23}, RF_{24}, RF_{34}. Then, each test case X_n will get 6 "possible" classes prediction $RF_{12}(X_n)$, $RF_{13}(X_n)$, $RF_{14}(X_n)$, $RF_{23}(X_n)$, $RF_{24}(X_n)$, $RF_{34}(X_n)$. Compute the class which has majority as follows

- Number of votes for CL_1 = v1 which is computed based on $RF_{12} + RF_{13} + RF_{14}$
- Number of votes for CL_2 = v2 which is computed based on $RF_{12} + RF_{23} + RF_{24}$
- Number of votes for CL_3 = v3 which is computed based on $RF_{13} + RF_{23} + RF_{34}$

- Number of votes for CL_4 = v4 which is computed based on $RF_{14} + RF_{24} + RF_{34}$

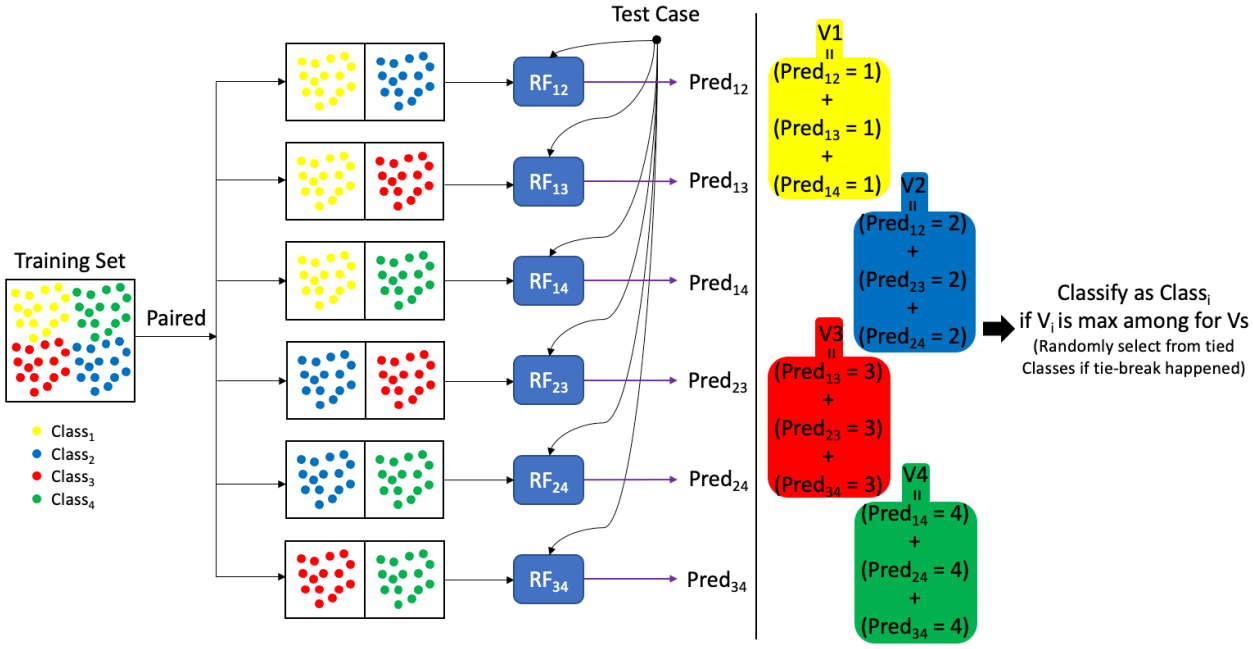

Figure 8 Flow Chart for RF Classifiers for pairs of classes

Table 17 also shows that there is no improvement on the classification.

Table 17 Confusion Matrix of RF Classifiers for pairs of classes

		True			
		1	2	3	4
Predicted	1	89.0	11.6	3.8	7.6
	2	7.2	79.3	12.6	0.0
	3	0.3	3.3	81.9	0.4
	4	3.4	5.8	1.7	92.0

6.3.5 Bag of Random Forest Classifiers

Even though RF model is already a bagging classifier, we would like to apply bagging on several RF classifiers. First, randomly choose 5 set of sub training set from the original training set. Each sub training set will have size of 80% of original training set. Then, use each sub training set to separately train a RF classifier to get 5 RF classifiers as RF_{tr1}, RF_{tr2}, ..., RF_{tr5}. For each test case $X_{n,}$ we get 5 "possible" classes $RF_{tr1}(X_n)$, ..., $RF_{tr5}(X_n)$. The predicted class of X_n will be the class which has majority of occurrences among those 5 predictions from the bag of RF classifiers.

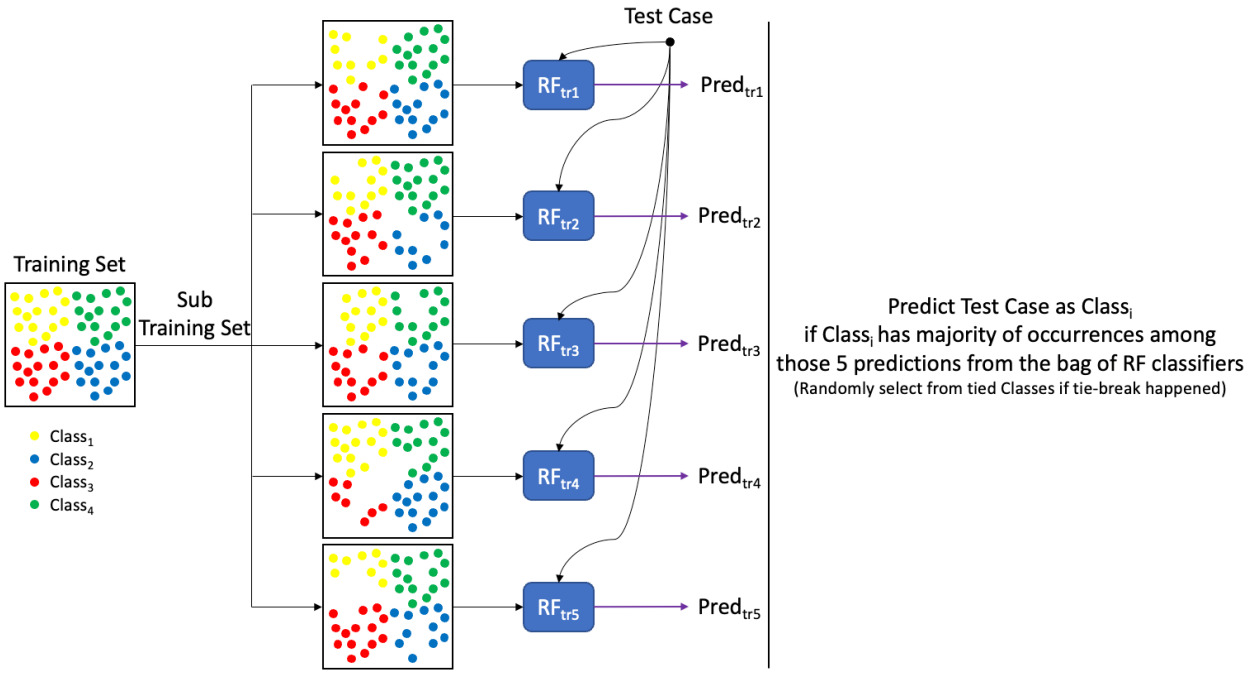

Figure 9 Flow Chart for Bag of RF Classifiers

Table 18 shows that there is no improvement on the classification.

Table 18 Confusion Matrix of Bag of RF Classifiers

		True			
		1	2	3	4
Predicted	1	89.3	13.1	4.2	7.2
	2	7.2	79.3	10.1	1.1
	3	0.7	2.5	84.9	0.0
	4	2.7	5.1	0.8	91.7

6.3.6 Conclusion

Even though we have tried several methods to improve the accuracy of the prediction, it seems that none of them provided a better classification result. Table 19 shows the global accuracy of the RF model with number of trees = 100 and global accuracy of those improvement methods. This could be the case since the original prediction accuracy is already high ($\approx 86.9\%$). Hence, it is hard to find a better way to improve it. However, comparing Random Forest classifier to kNN (≈79.4% per previous report), we can see improvement. It can be studied further with other classifiers in the future to see if there exists a better model to classify different fonts.

Table 19 Global Accuracy of best RF and RF with Improvement

Case Study	Accuracy
Best Tree Number = 100	86.9
K-Means Clustering	86.2
Paired Classes	85.7
Bag of RF	86.4

R Script

```
rm(list = ls())
# ============================================================================
library(dplyr)
library(caret)
library(ggplot2)
library(tree)
library(randomForest)
# ============================================================================
# Problem 0
file_wd = "/Users/jerrychien/Desktop/OneDrive - University Of Houston/6350 - Statistical Learning and Data Mining/HW/HW 3/fonts"
selected_file = c("GILL.csv", "LEELAWADEE.csv", "ROMAN.csv", "TECHNIC.csv") # Each class 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 un necessary 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", "originalH", "originalW", "h", "w")
for (i in 1:length(selected_file)){
  assign(paste("CL", i, sep = ""),
         filter(select(read.csv(paste(file_wd, "/", selected_file[i], 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
class_size[, "Percentage"] = round((class_size[, "Size"] / N) * 100)

# Combine all the 4 classes and assign to a data frame.
DATA = rbind(CL1, CL2, CL3, CL4)

# True Class for each case
TRUC = data.frame()
```

```r
for (i in CL_list){
  temp = as.data.frame(rep(substring(i, 3), dim(get(i))[1]))
  colnames(temp) = "TRUC"
  TRUC = rbind(TRUC, temp)
}

# PCA
PCA = prcomp(DATA, center = TRUE, scale = TRUE) # Use built-in PCA
L = PCA$sdev^2 # Eigenvalue for finding the PEV >= 90%

PEV = NULL
for (m in 1:400){
  PEV[m] = sum(L[c(1:m)]) / 400 * 100
}
smallest_r = sum(PEV < 90) + 1 # Smallest r to have PEV >= 90%

ZDATA = PCA$x[ ,c(1:smallest_r)] # Truncate the data to first "r" principal c
omponents

final_ZDATA = cbind(TRUC, ZDATA) # Final data after PCA with true class.

# ============================================================================
# Problem 1
## 1.2
data_split = function(portion = 0.2, nclass = 4, df){
  test_set = data.frame()
  training_set = data.frame()
  size_test_training = data.frame()
  for (i in 1:nclass){
    temp = filter(df, TRUC == i)
    sampled_number = sample(dim(temp)[1], dim(temp)[1] * portion)

    temp_training = temp[-sampled_number, ]
    temp_test = temp[sampled_number, ]

    test_set = rbind(test_set, temp_test)
    training_set = rbind(training_set, temp_training)

    size_test_training[paste("CL", i, sep = ""), "Test"] = dim(temp_test)[1]
    size_test_training[paste("CL", i, sep = ""), "Training"] = dim(temp_train
ing)[1]
  }
  size_test_training["Total", "Test"] = dim(test_set)[1]
  size_test_training["Total", "Training"] = dim(training_set)[1]

  return(list(training_set, test_set, size_test_training))
}
set.seed(20211004) # Use seed = 20211004 to get same data set as HW2
split_data = data_split(portion = 0.2, nclass = 4, df = final_ZDATA)
training_set = split_data[[1]]
```

```r
test_set = split_data[[2]]
size_test_training = split_data[[3]]

# ============================================================================
# Problem 3
## 3.1 Built the random forest with 100 trees.
rf_100 = randomForest(x = training_set[, -1], y = as.factor(training_set[,
1]), ntree = 100, importance = TRUE)
pred_training_100 = predict(rf_100, training_set[, -1])
pred_test_100 = predict(rf_100, test_set[, -1])

## 3.2 Calculate the global performance and confusion matrix
global_acc = data.frame("Set" = c("Training", "Test"),
                        "Accuracy" = c(round(mean(pred_training_100 == traini
ng_set[, 1]) * 100, 1), round(mean(pred_test_100 == test_set[, 1]) * 100,
1)))

conf_training_100 = round(prop.table(confusionMatrix(data = pred_training_10
0, reference = as.factor(training_set[, 1]))$table, margin = 2) * 100, 1)
conf_test_100 = round(prop.table(confusionMatrix(data = pred_test_100, refere
nce = as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)

# ============================================================================
# Problem 4
## 4.1 Built the random forest with 200 and 300 trees.
rf_200 = randomForest(x = training_set[, -1], y = as.factor(training_set[,
1]), ntree = 200, importance = TRUE)
pred_training_200 = predict(rf_200, training_set[, -1])
pred_test_200 = predict(rf_200, test_set[, -1])
conf_test_200 = round(prop.table(confusionMatrix(data = pred_test_200, refere
nce = as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)

rf_300 = randomForest(x = training_set[, -1], y = as.factor(training_set[,
1]), ntree = 300, importance = TRUE)
pred_training_300 = predict(rf_300, training_set[, -1])
pred_test_300 = predict(rf_300, test_set[, -1])
conf_test_300 = round(prop.table(confusionMatrix(data = pred_test_300, refere
nce = as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)

## 4.2 Compare the performance and select the best tree number.
diff_number_tree_comp = data.frame()
for (i in 1:3){
  diff_number_tree_comp[i, "Number_of_Trees"] = i * 100
  temp_test_acc = mean(get(paste("pred_test_", i * 100, sep = "")) == test_se
t[, 1])
  diff_number_tree_comp[i, "Test_Accuracy"] = round(temp_test_acc * 100, 1)
  SE = sqrt(temp_test_acc * (1 - temp_test_acc) / dim(test_set)[1])
  diff_number_tree_comp[i, "LL_90_Test"] = round((temp_test_acc - 1.6 * SE) *
 100, 1)
  diff_number_tree_comp[i, "UL_90_Test"] = round((temp_test_acc + 1.6 * SE) *
```

```
 100, 1)
}

ggplot(data = diff_number_tree_comp) +
  geom_ribbon(aes(x = Number_of_Trees, ymax = UL_90_Test, ymin = LL_90_Test,
fill = "90% C.I."), alpha = 0.5) +
  geom_point(aes(x = Number_of_Trees, y = Test_Accuracy)) +
  geom_line(aes(x = Number_of_Trees, y = Test_Accuracy, color = "Test")) +
  geom_text(aes(Number_of_Trees, Test_Accuracy, label = Test_Accuracy), hjust
 = 0.5, vjust = 1.5, size = 8) +
  theme(text = element_text(size = 25)) +
  ylab("Accuracy (%)") + xlab("Tree Number") + expand_limits(x = c(90, 310),
y = c(80, 100)) +
  scale_x_discrete(limits = c(100, 200, 300)) +
  scale_color_manual(name = "Accuracy", values = ("Test" = "black")) +
  scale_fill_manual(name = "C.I.", values = c("90% C.I." = "green")) +
  theme(legend.position = c(0.85, 0.90), legend.text = element_text(size = 2
0), legend.title = element_text(size = 20), legend.box = "horizontal")

best_tree_number = as.integer(diff_number_tree_comp[diff_number_tree_comp["Te
st_Accuracy"] == max(diff_number_tree_comp["Test_Accuracy"])][1])

# ============================================================================
## 5.1 Feature Importance
sorted_IM = sort(get(paste("rf_", best_tree_number, sep = ""))$importance[, "
MeanDecreaseAccuracy"], decreasing = T)[1:10]
feature_IM = data.frame()
for (i in 1:10) {
  feature_number = as.integer(substring(names(sorted_IM)[i], 3))
  feature_IM[i, "Feature"] = feature_number
  feature_IM[i, "Importance"] = round(sorted_IM[i] * 100, 1)
  feature_IM[i, "Eigenvalue"] = L[feature_number]
}

ggplot() +
  geom_point(aes(x = c(1:10), y = feature_IM[, "Importance"])) +
  geom_line(aes(x = c(1:10), y = feature_IM[, "Importance"])) +
  geom_text(aes(c(1:10) + c(-0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0), feature_IM[,"Im
portance"], label = paste("Z", feature_IM[, "Feature"], sep = "")), hjust =
0.5, vjust = 1.5, size = 8) +
  theme(text = element_text(size = 25)) +
  ylab("Importance (%)") + xlab("Importance Ranking") + expand_limits(x = c
(1,10), y = c(0, 2.5)) +
  scale_x_discrete(limits = factor(c(1:10)))

## 5.3
ggplot() +
  geom_point(aes(x = feature_IM[, "Eigenvalue"], y = feature_IM[, "Importance
"])) +
  geom_text(aes(feature_IM[, "Eigenvalue"] + c(0, 0, 0, 2.5, -2.5, 0, 0, 0,
```

```
0, 0), feature_IM[, "Importance"] + c(0, 0, 0, 0.5, 0.5, 0, 0, 0, 0, 0), labe
l = paste("Z", feature_IM[, "Feature"], sep = "")), hjust = 0.5, vjust = 1.5,
 size = 5) +
  theme(text = element_text(size = 25)) +
  ylab("Accuracy Loss (%)") + xlab("Eigenvalue") + expand_limits(x = c(0,10
0), y = c(0, 2.5))

cor_L_IM = cor(feature_IM[, "Eigenvalue"], feature_IM[, "Importance"])

# ============================================================================
# Problem 6
## 6.1
conf_test_best_tree = round(prop.table(get(paste("conf_test_", best_tree_numb
er, sep = "")), margin = 2) * 100, 1)
best_tree_acc = round(mean(get(paste("pred_test_", best_tree_number, sep = "
")) == test_set[, 1]) * 100, 1)

paired_CL = combn(c(1:length(selected_file)), 2)
misclassifed = data.frame()
for (i in c(1:dim(paired_CL)[2])){
  misclassifed[i, "CLi"] = paired_CL[1, i]
  misclassifed[i, "CLj"] = paired_CL[2, i]
  misclassifed[i, "Rate"] = conf_test_best_tree[paired_CL[1, i], paired_CL[2,
 i]] + conf_test_best_tree[paired_CL[2, i], paired_CL[1, i]]
}

## 6.2
for (i in c(1:dim(paired_CL)[2])){
  training_set_pair = filter(training_set, TRUC == paired_CL[1, i] | TRUC ==
paired_CL[2, i])
  test_set_pair = filter(test_set, TRUC == paired_CL[1, i] | TRUC == paired_C
L[2, i])
  pred_test_pair = predict(randomForest(x = training_set_pair[, -1], y = as.f
actor(training_set_pair[, 1]), ntree = best_tree_number, importance = TRUE),
test_set_pair[, -1])
  assign(paste("conf_test_pair_", paired_CL[1, i], "_", paired_CL[2, i], sep
= ""), round(prop.table(confusionMatrix(as.factor(pred_test_pair), as.factor
(test_set_pair[, 1]))$table, margin = 2) * 100, 1))
}

## 6.3.1 Enlarge the tree number
different_tree = data.frame()
for (i in seq(0, 500, 10)[-1]){
  temp_rf = randomForest(x = training_set[, -1], y = as.factor(training_set[,
 1]), ntree = i, importance = TRUE)
  temp_predict = predict(temp_rf, test_set[, -1])
  different_tree[i/10, "Tree_Number"] = i
  different_tree[i/10, "Accuracy"] = round(mean(temp_predict == test_set[,
1]) * 100, 1)
}
```

```r
ggplot(data = different_tree) +
  geom_line(aes(x = Tree_Number, y = Accuracy)) +
  theme(text = element_text(size = 25)) +
  ylab("Accuracy (%)") + xlab("Tree Number") + expand_limits(x = c(0, 500), y
 = c(80, 100)) +
  scale_x_discrete(limits = c(10, 50, 100, 150, 200, 250, 300, 350, 400, 450,
 500)) +
  theme(plot.margin = unit(c(10, 15, 10, 10), "point"))

## 6.3.2 Change the number of features of be used in each tree in RF model
different_feature = data.frame()
for (i in 1:(smallest_r - 1)){
  temp_rf = randomForest(x = training_set[, -1], y = as.factor(training_set[,
 1]), ntree = best_tree_number, mtry = i, importance = TRUE)
  temp_predict = predict(temp_rf, test_set[, -1])
  different_feature[i, "Feature_Number"] = i
  different_feature[i, "Accuracy"] = round(mean(temp_predict == test_set[,
1]) * 100, 1)
}

ggplot(data = different_feature) +
  geom_line(aes(x = Feature_Number, y = Accuracy)) +
  theme(text = element_text(size = 25)) +
  ylab("Accuracy (%)") + xlab("Feature Number") + expand_limits(x = c(0, 62),
 y = c(80, 100)) +
  scale_x_discrete(limits = c(1, 5, 7, 10, 15, 20, 25, 30, 35, 40, 45, 50, 5
5, 61))

## 6.3.3 k-means clustering
training_set_k_means = kmeans(training_set[, -1], centers = 4, nstart = 20)
cent = training_set_k_means$centers
cluster_training_set = cbind(training_set_k_means$cluster, training_set)
colnames(cluster_training_set)[1] = "cluster"

purity = data.frame()
for (i in 1:length(selected_file)){
  temp = filter(as.data.frame(cluster_training_set), cluster == i)
  gini = sum(prop.table(table(temp$TRUC)) * (1 - prop.table(table(temp$TRU
C))))
  purity[i, "Cluster"] = i
  purity[i, "Gini"] = round(gini, 2)
  assign(paste("rf_G", i, sep = ""), randomForest(x = temp[, -c(1,2)], y = a
s.factor(temp[, 2]), ntree = best_tree_number, importance = TRUE))
}

pred_test_k_means_rf = NULL
for (i in c(1:dim(test_set)[1])){
  distance = data.frame()
  for (j in 1:length(selected_file)){
```

```
    distance[j, "Cluster"] = j
    distance[j, "Distance"] = dist(rbind(test_set[i, -1], cent[j, ]))[1]
  }
  closest_G = distance[min(distance[, "Distance"]) == distance[, "Distance"],
 "Cluster"]
  pred_test_k_means_rf[i] = predict(get(paste("rf_G", closest_G, sep = "")),
test_set[i, -1])
}

conf_test_k_means_rf = round(prop.table(confusionMatrix(as.factor(pred_test_k
_means_rf), as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)
k_means_rf_acc = round(mean(pred_test_k_means_rf == test_set[, 1]) * 100, 1)

## 6.3.4 6 RF classifiers for 6 pairs of classes
for (i in c(1:dim(paired_CL)[2])){
  training_set_CLi_CLj = filter(training_set, TRUC == paired_CL[1, i] | TRUC
== paired_CL[2, i])
  assign(paste("rf_", paired_CL[1, i], paired_CL[2, i], sep = ""), randomFore
st(x = training_set_CLi_CLj[, -1], y = as.factor(training_set_CLi_CLj[, 1]),
ntree = best_tree_number, importance = TRUE))
}

pred_test_paired_rf = NULL
for (i in c(1:dim(test_set)[1])){
  for (j in c(1:dim(paired_CL)[2])){
    assign(paste("pred_rf_", paired_CL[1, j], paired_CL[2, j], sep = ""), pre
dict(get(paste("rf_", paired_CL[1, j], paired_CL[2, j], sep = "")), test_set
[i, ]))
  }
  vote = data.frame("Class" = c(1:length(selected_file)))
  vote[1, "Vote"] = (pred_rf_12 == 1) + (pred_rf_13 == 1) + (pred_rf_14 == 1)
  vote[2, "Vote"] = (pred_rf_12 == 2) + (pred_rf_23 == 2) + (pred_rf_24 == 2)
  vote[3, "Vote"] = (pred_rf_13 == 3) + (pred_rf_23 == 3) + (pred_rf_34 == 3)
  vote[4, "Vote"] = (pred_rf_14 == 4) + (pred_rf_24 == 4) + (pred_rf_34 == 4)
  max_freq = vote[vote["Vote"] == max(vote["Vote"]), "Class"]
  if (length(max_freq) != 1){
    pred_test_paired_rf[i] = sample(max_freq, 1)
  } else{
    pred_test_paired_rf[i] = max_freq
  }
}

conf_test_paired_rf = round(prop.table(confusionMatrix(as.factor(pred_test_pa
ired_rf), as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)
paired_rf_acc = round(mean(pred_test_paired_rf == test_set[, 1]) * 100, 1)

## 6.3.5 Bag of RF classifiers
bag_rf = 5
for (i in (1:bag_rf)){
  set.seed(i)
```

```
  assign(paste("sub_training_set_", i, sep = ""), data_split(portion = 0.2, n
class = 4, df = training_set)[[1]])
  assign(paste("rf_tr", i, sep = ""), randomForest(x = get(paste("sub_trainin
g_set_", i, sep = ""))[, -1], y = as.factor(get(paste("sub_training_set_", i,
 sep = ""))[, 1]), ntree = best_tree_number, importance = TRUE))
}

pred_test_bag_rf = NULL
for (i in c(1:dim(test_set)[1])){
  for (j in c(1:bag_rf)){
    assign(paste("pred_rf_tr", j, sep = ""), predict(get(paste("rf_tr", j, se
p = "")), test_set[i, ]))
  }
  vote2 = data.frame("Class" = c(1:length(selected_file)))
  vote2[1, "Vote"] = (pred_rf_tr1 == 1) + (pred_rf_tr2 == 1) + (pred_rf_tr3 =
= 1) + (pred_rf_tr4 == 1) + (pred_rf_tr5 == 1)
  vote2[2, "Vote"] = (pred_rf_tr1 == 2) + (pred_rf_tr2 == 2) + (pred_rf_tr3 =
= 2) + (pred_rf_tr4 == 2) + (pred_rf_tr5 == 2)
  vote2[3, "Vote"] = (pred_rf_tr1 == 3) + (pred_rf_tr2 == 3) + (pred_rf_tr3 =
= 3) + (pred_rf_tr4 == 3) + (pred_rf_tr5 == 3)
  vote2[4, "Vote"] = (pred_rf_tr1 == 4) + (pred_rf_tr2 == 4) + (pred_rf_tr3 =
= 4) + (pred_rf_tr4 == 4) + (pred_rf_tr5 == 4)
  max_freq2 = vote2[vote2["Vote"] == max(vote2["Vote"]), "Class"]
  if (length(max_freq2) != 1){
    pred_test_bag_rf[i] = sample(max_freq2, 1)
  } else{
    pred_test_bag_rf[i] = max_freq2
  }
}

conf_test_bag_rf = round(prop.table(confusionMatrix(as.factor(pred_test_bag_r
f), as.factor(test_set[, 1]))$table, margin = 2) * 100, 1)
bag_rf_acc = round(mean(pred_test_bag_rf == test_set[, 1]) * 100, 1)

acc_comp = data.frame("Case Study" = c(paste("Best Tree Number =", best_tree_
number), "K-Means Clustering", "Paired Classes", "Bag of RF"),
                      "Accuracy" = c(best_tree_acc, k_means_rf_acc, paired_rf
_acc, bag_rf_acc))
```