

**MATH 6350 Fall 2020 MSDS**  
**Homework 5: Random Forest**

Sara Nafaryeh

Tony Nguyen

Thomas Su

All authors played an equal part

## Introduction

In this report, we will continue to add to what we have performed in Homework 2,3, and 4 by looking into improving our data through Random Forest. A brief reminder of what was done in Homework 2,3, and 4: we selected three font data sets consisting of digitized images of typed characters and determined using the KNN function in R to predict our best K. Next we used PCA, a dimensionality reduction method that reduces the dimensionality of a large data set, by capturing as much of the data information as possible into a smaller one that contains most of the information of the large set. And lastly, K-means clustering was used to minimize the distance within a cluster all while maximizing the distance between different clusters.

The files were downloaded from a zip file from the following link:  
<https://archive.ics.uci.edu/ml/machine-learning-databases/00417/>

The font types that we chose were: COMIC, BOOKMAN, and MONOTYPE. Each has 412 column features along with total observed cases of 2669, 2388, and 2388 respectively. Each case describes numerically a digitized image of some specific character typed in each of the fonts. The images have  $20 \times 20 = 400$  pixel sizes, each with its own "gray level" indicated by an integer value of 0 to 255. All the fonts have 412 feature columns, where 400 of them describe the 400 pixels named:

**$\{r0c0, r0c1, r0c2, \dots, r19c17, r19c18, r19c19\}$**

**"rLcM"** = gray level image intensity for pixel in position **{Row L, Column M}**.

As stated earlier, in Homework 5, we will be more focused on using the random forest to help improve our prediction outcome.

## DATA Set up

A quick reminder of our data set up from Homework 2, we will briefly go over this section as a refresher. The following steps are taken to get R ready as well as organize our data for the steps that follow. Other than the 400 columns that are associated with the pixels, the data set font files each have the following 12 names:

*{ font, fontVariant, m\_label, strength, italic, orientation, m\_top, m\_left, originalH, originalW, h, w }*

Of these 12 we need to discard the following 9:

*{fontVariant, m\_label, orientation, m\_top, m\_left, originalH, originalW, h, w}*

And keep the following 3: *{font, strength, italic}* as well as the 400 pixel columns named: *{ r0c0, r0c1, r0c2, ... , r19c17, r19c18, r19c19}* therefore we are left with 403 columns. After these steps are completed, we define three CLASSES on images of the “normal” character were we extract all the rows in which our three fonts have both strength of 0.4 and italic of 0:

CL1 = all rows of **comic.csv** file for which {strength = 0.4 and italic=0}

CL2 = all rows of **bookman.csv** file for which {strength = 0.4 and italic=0}

CL3 = all rows of **monotype.csv** file for which {strength = 0.4 and italic=0}

And left with the following row outputs:

```
> n1 = nrow(CL1)
> n2 = nrow(CL2)
> n3 = nrow(CL3)
> N = sum(n1, n2, n3)
> n1;n2;n3;N
[1] 597
[1] 667
[1] 667
[1] 1931
```

The respected row sizes for CLASS1, CLASS2, and CLASS3 are named

**n1, n2, n3** = 597, 667, 667 and there sum → N = 1931 cases

We combine them all together into a data set named DATA using the function **rbind()** and see it has dimensions of 1931 rows and 403 columns: the 403 being font,

strength, italic, +400 pixels we will call features X1, X2, ... X400. Each such feature  $X_j$  is observed N times, and its N observed values are listed in the column "j" of DATA.

Once again, we make sure to standardize the 400 features. Define the **start\_col** and **end\_col** to be the feature X1 starts and X400 ends respectfully. Now that we have created our [1931x403] **DATA** set table, we observe the mean,  $m = \text{mean}(X1) \dots \text{mean}(X400)$ , and standard deviation,  $s = \text{sd}(X1) \dots \text{sd}(400)$  for each of our pixel features 1 to 400. The most important step to do before running our program is to standardize the data features in order to set everything at a mean of 0 and a standard deviation at 1. It is important to standardize because when we compare measurements they can have different units as one can observe in the table above. In the table below observe the variance of the data before and after standardization. Notice the variance of the DATA of X1 and X2 features before, and after when they all equal to 1. Variables that are measured at different scales do not contribute equally to the analysis and might end up creating a bias. We scale the data and name it SDATA. Note that X is a data frame from our DATA where we start at feature X1 and end at feature X400.

```
> var(DATA[,4]); var(DATA[,5])
[1] 7328.596
[1] 10008.27
> X = DATA[,start_col: end_col]
> SDATA = scale(X)
> var(SDATA[,4]); var(SDATA[,5])
[1] 1
[1] 1
```

Finally part of our data set up we explore our data outcome from Homework 3, PCA. Our Percentage of Variance Explained we calculated using the **cumsum(D)/sum(D)** functions in R, where D is defined as our eigenvalues. 95% (PVE(r) = 95%) of the variance, we identified to be  $r = 112$  dimensions (or about 28%). Therefore, we concluded that 95% of our data falls within 1 to 112 dimensions and computed the PCA: new features also known as principal components.  $Y1(n), Y2(n), \dots, Yr(n)$  for each case in n and  $r = 112$ .

```
> #transpose of the matrix W^T, ==> principal components Y1(n)...Yr(n)
> #Y1(n) = Q11 * X1(n) + Q21* X2(n) + ... + Qr1 * Xr(n)
> Qt = t(eigen(R)$vector)[1:6,1:6]; Qt
      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
[1,] 0.037554270 0.042860673 0.04316449 0.04698484 0.04820854 0.052766042
[2,] 0.009644978 0.013215923 0.01723940 0.01635823 0.01393625 0.013510422
[3,] -0.044836953 -0.054798554 -0.04991597 -0.03201898 -0.01691300 -0.009282794
[4,] 0.124525697 0.136898628 0.14209661 0.14214624 0.13047797 0.102531423
[5,] -0.048293288 -0.041227763 -0.04463198 -0.05573747 -0.05368905 -0.054488360
[6,] -0.005032767 -0.003129964 -0.00605311 -0.01219605 -0.01925138 -0.017942660
```

```
> Qt = t(eigen(R)$vector)[112,1:6]; Qt
[1] -0.017344514 0.039593019 -0.002696742 -0.019101426 -0.032370772 0.035545753
```

***That being said:***

$$Y1(n) = 0.0375 \cdot F1(n) + 0.0428 \cdot F2(n) + 0.0431 \cdot F3(n) + 0.0469 \cdot F4(n) + 0.0482 \cdot F5(n) + 0.0527 \cdot F6(n)$$

$$Y2(n) = 0.0096 \cdot F1(n) + 0.0132 \cdot F2(n) + 0.0172 \cdot F3(n) + 0.0163 \cdot F4(n) + 0.0139 \cdot F5(n) + 0.01351 \cdot F6(n)$$

...

$$Y112(n) = -0.0050 \cdot F1(n) + 0.0395 \cdot F2(n) - 0.0026 \cdot F3(n) - 0.0191 \cdot F4(n) - 0.0323 \cdot F5(n) - 0.0355 \cdot F6(n)$$

In conclusion, the goal of PCA was to find a low-dimensional representation of the observation that explains a good fraction of the variance and in Homework 3 we believe we found a reasonable value of new features  $r = 112$  out of our 400. Compression helps eliminate unnecessary, redundant, or noisy information. Using linear compression through PCA is helpful when we have large numerical features.

## Question 1

1.1) Selecting random train and test sets for each of the three classes:

```
set.seed(123)
pd1 = sample(x = 2, size = nrow(CL1), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL1
pd2 = sample(x = 2, size = nrow(CL2), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL2
pd3 = sample(x = 2, size = nrow(CL3), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL3
trainCL1 = CL1[pd1==1, ]
testCL1 = CL1[pd1==2, ]
trainCL2 = CL2[pd2==1, ]
testCL2 = CL2[pd2==2, ]
trainCL3 = CL3[pd3==1, ]
testCL3 = CL3[pd3==2, ]
```

```
> table(DATA$font)
```

COMIC	BOOKMAN	MONOTYPE
597	667	667

As you recall from our Data Set Up we see each class has the following cases:

***n1, n2, n3*** = 597, 667, 667

After the train and test sets are separated by 80% and 20% respectfully for each class, we have the following cases for each class:

```

> dim(trainCL1);dim(testCL1) # Comic
[1] 477 113
[1] 120 113
> dim(trainCL2);dim(testCL2) # Bookman
[1] 541 113
[1] 126 113
> dim(trainCL3);dim(testCL3) # Monotype
[1] 527 113
[1] 140 113

```

1.2) As we can see in the dimensions above, we observe trainCL1:trainCL3 with the size 477, 541, and 527 respectively (79.9%, 81.1%, and 79.0% of the train set) and testCL1: trainCL3 with the size 102, 126, and 140 respectively (17.0%, 18.9%, and 21.0% of the test set) therefore the gap between these percentage is too small to justify cloning or undersampling.

1.3) Now we have new pairs of the train and test sets. Re-define our true train/test set: **TRAINSET\_TARGET** / **TESTSET\_TARGET** by regrouping across classes using row combining these new pairs. The same goes for the **TRAINSET** / **TESTSET**:

```

TRAINSET_TARGET = rbind(trainCL1, trainCL2, trainCL3)[,1] # true train set
TESTSET_TARGET = rbind(testCL1, testCL2, testCL3)[,1]      # true test set

TRAINSET = rbind(trainCL1, trainCL2, trainCL3)[,-1]
TESTSET = rbind(testCL1, testCL2, testCL3)[,-1]

> dim(TRAINSET); dim(TESTSET)
[1] 1545 112
[1] 386 112

```

Once again, as we can see in the dimensions above, we observe TRAINSET = 1545 cases at 80% separated and TESTSET = 386 at 20% separated, where our data sets' total number of cases is  $N = 1931$ .



## Question 2

2.1) Random Forest(RF) is the combination of tree predictors where each tree depends on the values of random vectors sampled independently with the same distribution for all trees. It helps avoid overfitting and, such as our data, it can deal with a large number of features to help make predictions.

2.2) To get started in performing random forest, we first make sure to set our *TRAINSET\_TARGET* into factors using `as.factor()`. Next using the function `randomForest()`, from the package: `library(randomForest)` in R. Defined as **rf\_CL** we input the factored *TRAINSET\_TARGET* as our model as a function of all the other remaining variables and set the data = *TRAINSET*.

```
library(randomForest)
TRAINSET_TARGET <- as.factor(TRAINSET_TARGET)
rf_CL <- randomForest(TRAINSET_TARGET~.,data=TRAINSET)

> print(rf_CL)

Call:
randomForest(formula = TRAINSET_TARGET ~ ., data = TRAINSET)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 10

OOB estimate of error rate: 22.46%
Confusion matrix:
      BOOKMAN  COMIC  MONOTYPE  class.error
BOOKMAN      398     49       94    0.2643253
COMIC         41    407       29    0.1467505
MONOTYPE      94     40      393    0.2542694
```

From the results, we can see that we have a classification type of random forest with the default number of trees (ntree) set to 500. The number of variables tried at each split (mtry) is 10. We note that the default is  $\sqrt{p}$ , where  $p$  = number of features. In our case, the number of features we have in both our TRAINSET and TESTSET is based on our results from our PCA in the Data Set-Up,  $r = 112$ . (from here

and moving forward, we will note  $r = p = 112$ ). Going back to our randomForest mtry =  $\sqrt{P} = \sqrt{112} \approx 10$

The Out Of Bag (OOB) estimate of error rate = 22.46%  $\approx (100 * (1 - .7745) = 22.55\%$  accuracy) it is a good accuracy, but could be better/improved. Looking at the confusion matrix, we see that predictions are the best when predicting for class 1, COMIC with the lowest class error of 14.7% on the train set. In the terms of predicting our other two classes, we see a higher class error with 26.4% for Bookman and 25.4% for Monotype on the train set. For the OOB accuracy of 77.45%, we need to remember that it is the data this particular tree has not seen yet, therefore the accuracy is based on the error calculation.

2.3) Using the *predict()* function we input **rf\_CL** associated to the randomForest and train = TRAINSET. We observe the following outputs:

```
p0 <- predict(rf_CL, TRAINSET)

> head(p0)
 668  669  670  673  674  676
COMIC COMIC COMIC COMIC COMIC COMIC
Levels: BOOKMAN COMIC MONOTYPE

> p0[10:15]
 681  682  684  685  686  689
BOOKMAN BOOKMAN COMIC COMIC COMIC COMIC
Levels: BOOKMAN COMIC MONOTYPE
```

Using the *head()* function we see what is contained in the first 6 outputs of our predict function. The first 6 predictions are seen to be the class1: Comic. For curiosity, we see what the predictions are for [10:15], and for 10,11 we see it is class2: Bookman and rest are once again class1:Comic.

2.4) Using the *confusionMatrix()* function, we input our prediction **p0** associated to the randomForest and true train = TRAINSET\_TARGET. We observe the following outputs for the table and overall accuracy:

```

> confusionMatrix(p0,TRAINSET_TARGET)$table
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN    540     8    27
COMIC       1   469     3
MONOTYPE    0     0   497
> confusionMatrix(p0,TRAINSET_TARGET)$overall[1]
Accuracy
0.9747573

```

As we can see in our table, we have the reference values/ actual classes as the column and the prediction values of the three classes. We see the correct classification is observed 540 times as the font: Bookman while the model also predicted it to be Bookman. The same goes for 469 times for Comic and 497 times for Monotype to be classified as the respected fonts while actually being at the font. We can also see that there are a total of  $(8 + 27 + 1 + 3) = 39$  misclassifications.

As for our accuracy we see that total classified correct/ total cases in the TRAINSET =  $[(540 + 469 + 497) / (1545)] = [1506/1545]100\% = 97.4\%$  accuracy. (classification error is less than 1%). Unlike the OOB estimation, we have a much higher accuracy using the prediction function. This is due to the training data provided to the randomForest model based on our TRAINSET data of 1545 cases, resulting in higher accuracy.

### Question 3

3.1) Fixing the number of trees (ntree) set to 100, and the number of variables tried at each split (mtry) to  $\sqrt{p}$  we repeat the steps done in part 2.2) applying random forest to classify our data set into the 3 classes. Note we are using the  $p = 112$  new features we found in our PCA result.

```
> set.seed(123)
> rf = randomForest(TRAINSET_TARGET ~ ., data = TRAINSET, ntree = 100, mtry = sqrt(p))
> print(rf)
```

Call:  
randomForest(formula = TRAINSET\_TARGET ~ ., data = TRAINSET,  
ntree = 100, mtry = sqrt(p))  
Type of random forest: classification  
Number of trees: 100  
No. of variables tried at each split: 11

OOB estimate of error rate: 27.25%

Confusion matrix:

	BOOKMAN	COMIC	MONOTYPE	class.error
BOOKMAN	382	63	96	0.2939002
COMIC	66	365	46	0.2348008
MONOTYPE	109	41	377	0.2846300

We observe the classification random forest, at  $ntree = 100$  and  $mtry = 11$ . Compared to our randomForest in part 2.2) we observe a slight increase the OOB estimate of the error rate to 27.25% resulting in a slight decrease in accuracy at  $(1 - 0.2725) \times 100\% = 72.75\%$  (whereas in part 2.2 it was 77.45%).

The prediction and confusion matrix of the train and test set are as follow at  $ntree = 100$  and  $mtry = \sqrt{p}$ :

```

> # prediction & confusion matrix - train data
> library(caret)
> pred1 = predict(rf, TRAINSET)
> trainconf = confusionMatrix(pred1, TRAINSET_TARGET)$table # or table(pred1,
  TRAINSET_TARGET)
> print(trainconf)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      538      9      25
COMIC         0    467      1
MONOTYPE       3      1    501
> trainperf = confusionMatrix(pred1, TRAINSET_TARGET)$overall[1] # or sum(diag
  (trainconf))/sum(trainconf)
> print(trainperf)
Accuracy
0.9747573

```

For the TRAINSET prediction, we observe the same percentage of accuracy as in our TRAINSET in part 2.3 and 2.4) at 97.4%. However that being said, we do notice a difference in the correct classification. Though slightly different, we observed 538 (*compared to 540*) times as the font: Bookman while the model also predicted it to be Bookman. The same goes for 467 (*compared to 469*) times for Comic and 501 (*compared to 497*) times for Monotype to be classified as the respected fonts while actually being at font. We can also see that there is still a total of  $(3+1+9+25+1) = 39$  misclassifications. (*compared to  $(8 + 27+1 +3) = 39$  in part 2.4*) therefore, we can suggest the changing from default ntree = 500, to ntree = 100 did not make much of a change.

```

> # prediction & confusion matrix - test data
> pred2 = predict(rf, TESTSET)
> testconf = confusionMatrix(pred2, TESTSET_TARGET)$table
> print(testconf)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      86     12     25
COMIC        17    102      5
MONOTYPE     23      6    110
> testperf = confusionMatrix(pred2, TESTSET_TARGET)$overall[1]
> print(testperf)
Accuracy
0.7720207

```

For the TESTSET prediction of the true test: TESTSET\_TARGET, we observe the percent accuracy has decreased down to 77.2%. this testset data has not been seen by the random forest model, we can conclude this is a more accurate assessment accuracy of the model.

3.2) We will not explore the random forest at a fix  $n_{\text{try}} = \sqrt{p}$  and explore the the testset confusion matrices along with their performance accuracy at  $n_{\text{tree}} = \{10, 50, 100, 200, 300, 400\}$

```
> conf(1)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      80     20      30
COMIC         20     82      12
MONOTYPE      26     18     98

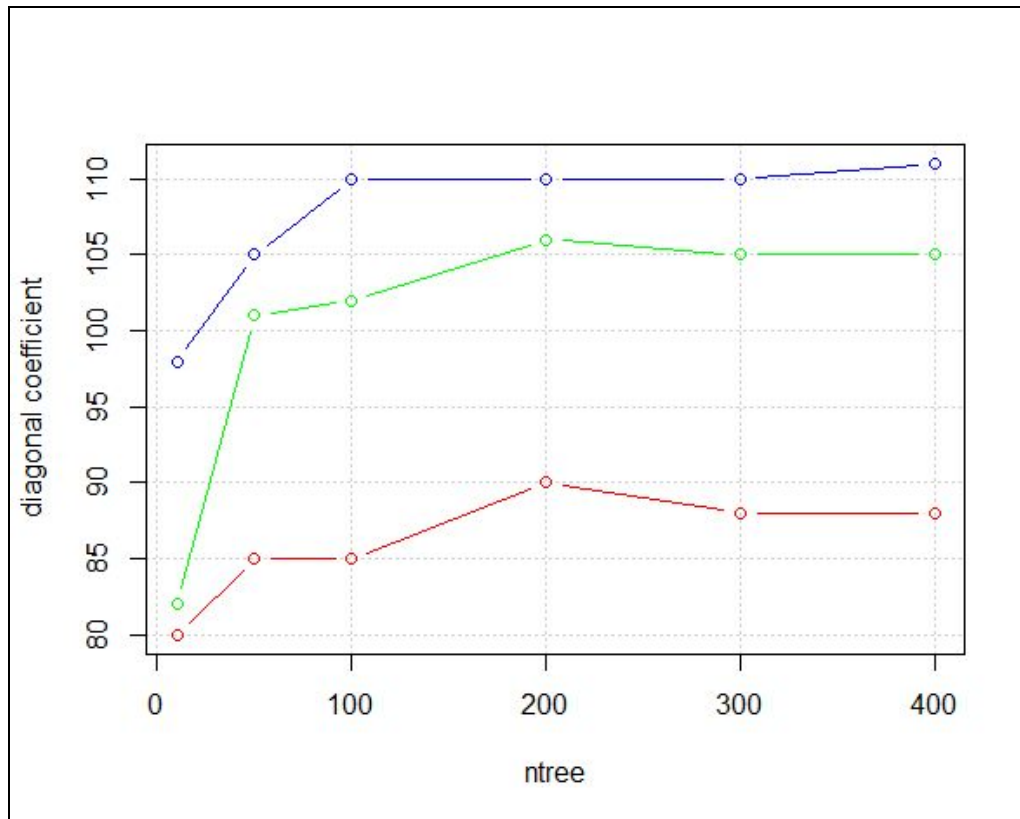
> conf(2)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      85     14      28
COMIC         18    101       7
MONOTYPE      23      5    105

> conf(3)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      85     12      25
COMIC         17    102       5
MONOTYPE      24      6    110

> conf(4)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      90     10      25
COMIC         14    106       5
MONOTYPE      22      4    110

> conf(5)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      88     10      27
COMIC         15    105       3
MONOTYPE      23      5    110

> conf(6)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      88     10      26
COMIC         15    105       3
MONOTYPE      23      5    111
```



The following curve above represents the diagonal coefficients vs, the ntree of each class. Compare it to 6 confusion matrices above, we observe

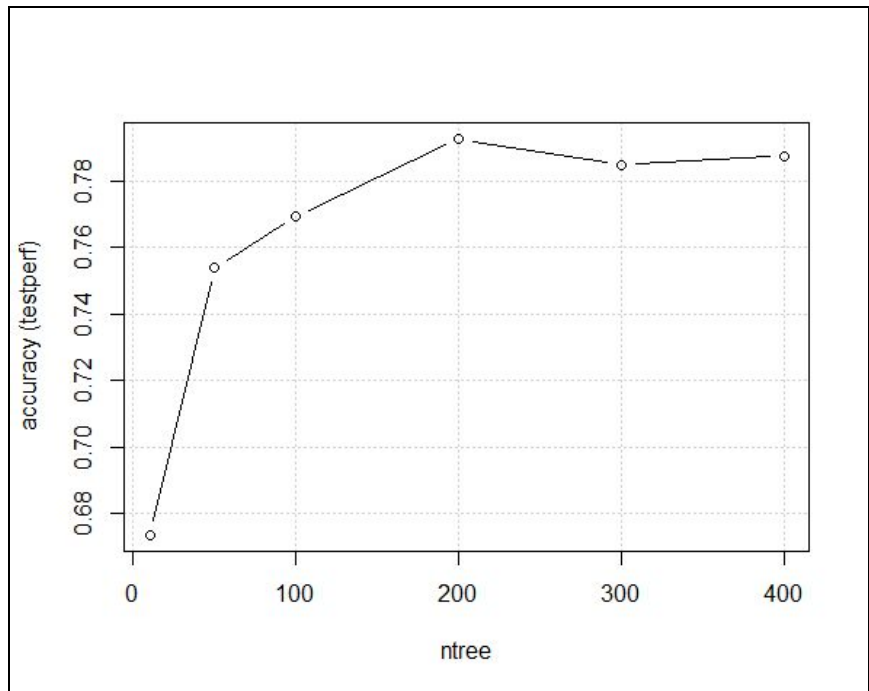
**Red:** The first diagonal coefficients of the 6 matrices (1,1), {80, 85, 85, 90, 88, 88}

**Green:** The second diagonal coefficients of the 6 matrices (2,2),{82,101,102,106,105,105}

**Blue:** The third diagonal coefficient of the 6 matrices (3,3), {98,105,110,110,110,111}.

Once again, these coefficients represent the times a font is correct while the model also predicted it to be that specific font.

```
> testperf(ntrees = 10)
Accuracy
0.6735751
> testperf(50)
Accuracy
0.753886
> testperf(100)
Accuracy
0.7694301
> testperf(200)
Accuracy
0.7927461
> testperf(300)
Accuracy
0.7849741
> testperf(400)
Accuracy
0.7875648
```



The following curve above is the test set accuracy vs ntrees. This time we are looking at our accuracy of (total classified correct) / ( total cases in the TESTSET) , where the TESTSET= 386 cases. As we can see, it appears the highest testset accuracy is at when ntrees= 200 with an accuracy of 79.27%

Therefore, we select the best ntree value, bntr = 200



## Question 4

4.1) Setting our best ntree value, bntr = 200 and number of variables to split the tree, mtry = sqrt(p) where  $p = 112$ , we define **bestRF** as our randomForest importance IM1...IMp of our 112 features as seen below. We observe the OOB estimate of error rate to be 24.85% (with an accuracy of  $(1 - 0.2485) \times 100\% = 75.15\%$ ). We also observe that once again class font Comic displays the lowest class error at 19.7% while Bookman and Monotype are followed behind with 28.1% and 26.2%. Compared to our results from 2.2) our data has slightly decreased in accuracy.

```
> set.seed(123)
> bestRF = randomForest(TRAINSET_TARGET ~ ., data = TRAINSET, ntree = bntr, mtry = sqrt(p))
> print(bestRF)
```

Call:  
randomForest(formula = TRAINSET\_TARGET ~ ., data = TRAINSET,  
ntree = bntr, mtry = sqrt(p))  
Type of random forest: classification  
Number of trees: 200  
No. of variables tried at each split: 11

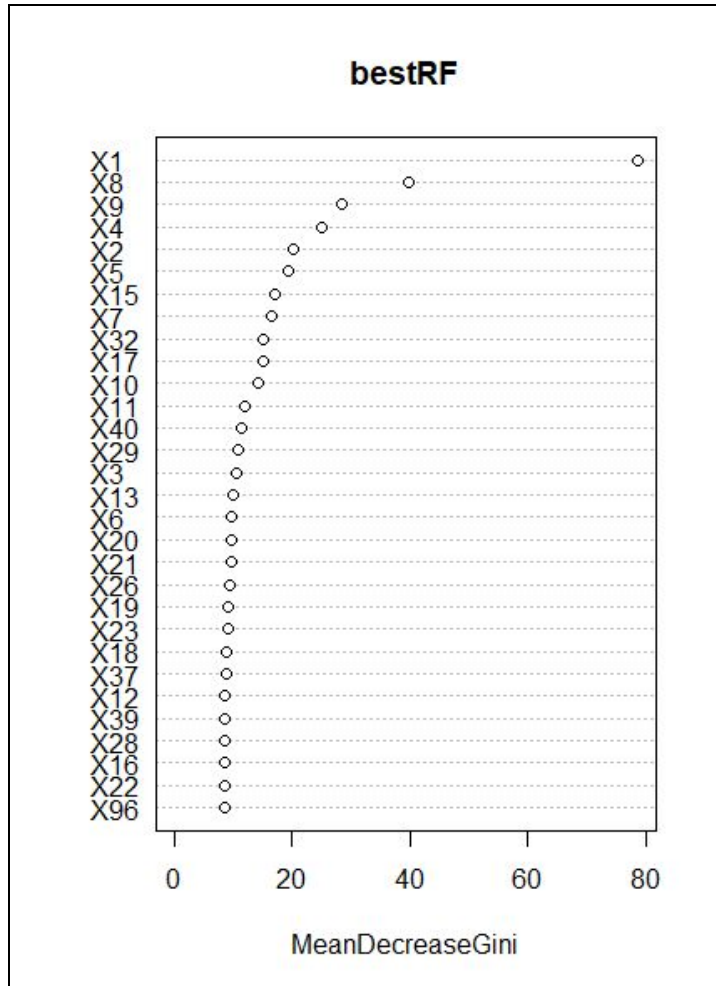
OOB estimate of error rate: 24.85%

Confusion matrix:

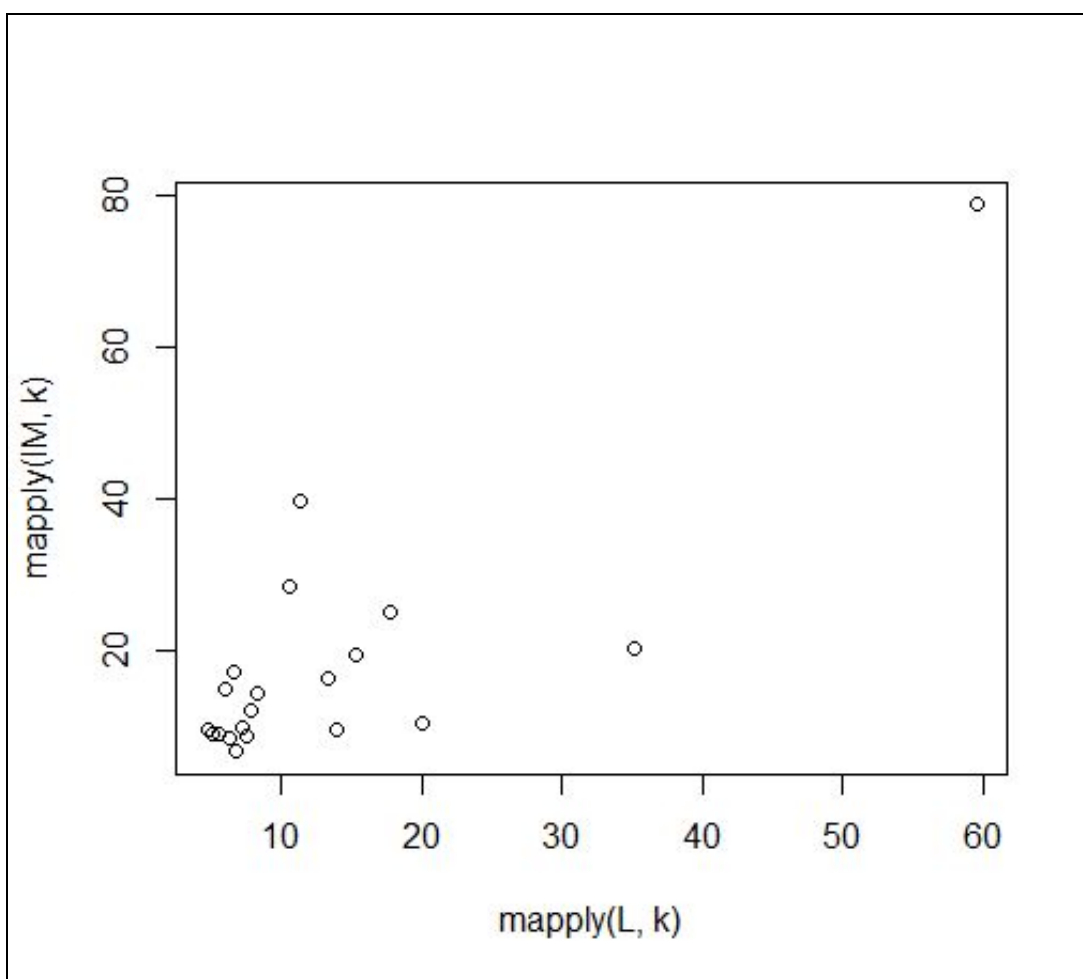
	BOOKMAN	COMIC	MONOTYPE	class.error
BOOKMAN	389	60	92	0.2809612
COMIC	63	383	31	0.1970650
MONOTYPE	103	35	389	0.2618596

Using the *importance()* function, we can view the importance of each variable. We get the measure of the total decrease in node impurity that results for splits over that feature, averaged over all trees. In our plot **varImpPlot(bestRF)** we can clearly see how pure the nodes are at the end of the tree without each variable and feature X1 stands out the most out of all the features with the highest mean gini decrease. In the output above the bestFR plot, we displayed the values of the top 10 highest Mean Gini Decreases. We notice that  $X1 = 78.70$ , while the next highest is  $X8 = 39.77$ .

```
> bestRF$importance[c(1,8,9,4,2,5,15,7,32,17),]
      x1      x8      x9      x4      x2
78.70201 39.77565 28.55033 25.15356 20.20513
      x5     x15      x7     x32     x17
19.35398 17.06720 16.45548 15.14708 14.93685
```



The scatter plot (eigenvalues vs the RF variable importance) shows that our first eigenvalue in our data is the most important than the previous eigenvalues and that the rest of the eigenvalues are not as important. We can clearly see that the one point in the top right corner is our first/highest eigenvalue  $D[1] = 59.507$ , and `bestRF$importance[1] = 78.702`



4.2) In this section we will select a new random train and test set by `setseed()` different from part 1.1). We will also set the `mtry = sqrt(p)`, `ntree=bntr`. So we do not mix up our randomly selected data from part 1.1) we apply the `randomForest()` on the new train set defined as `TRAINSET_new`, and new test set `TESTSET_new`, along with their new train and test sets: `TRAINSET_TARGET_new`, `TESTSET_TARGET_new`.

```
> set.seed(111)
> bestRF_new = randomForest(TRAINSET_TARGET_new ~ ., data = TRAINSET_new,
  ntree = bntr, mtry = sqrt(p))
> print(bestRF_new)
```

Call:

```
randomForest(formula = TRAINSET_TARGET_new ~ ., data = TRAINSET_new,
  ntree = bntr, mtry = sqrt(p))
  Type of random forest: classification
    Number of trees: 200
No. of variables tried at each split: 11
```

OOB estimate of error rate: 23.81%

Confusion matrix:

	BOOKMAN	COMIC	MONOTYPE	class.error
BOOKMAN	409	65	82	0.2643885
COMIC	58	398	24	0.1708333
MONOTYPE	98	44	380	0.2720307

```
> set.seed(111)
> pred2_new = predict(bestRF_new, TESTSET_new)
> testconf_new = confusionMatrix(pred2_new, TESTSET_TARGET_new)$table
> testperf_new = confusionMatrix(pred2_new, TESTSET_TARGET_new)$overall
[1]
> print(testconf_new)
```

	Reference		
Prediction	BOOKMAN	COMIC	MONOTYPE
BOOKMAN	82	7	31
COMIC	14	104	11
MONOTYPE	15	6	103

```
> print(testperf_new)
Accuracy
0.7747989
```

At `ntree = bntr = 200`, and `mtry = sqrt(112) = 11`, we get the test performance accuracy of our new randomly selected data to be `testperf_new = 77.47%`.

Comparing the test set accuracy and confusion matrix of our past random sample (bestRF) to our new random sample (newRF) we see that our accuracy has decreased only slightly. Based on a 95% confidence interval, we compare the percentages we observe that the *testpref\_b* of the **bestRF** confidence interval is narrower compared to the *testpref\_new* **newRF** by only  $[(0.8171899 - 0.7324080) - (0.8331825 - 0.7523097)] = 0.0039091 \Rightarrow \mathbf{0.39\%}$ . It is a small change and therefore not much of a difference between the two randomly selected samples.

```
> CI(p = testperf_b, n = nrow(TESTSET))
[1] 0.7523097 0.8331825
> CI(p = testperf_new, n = nrow(TESTSET_new))
[1] 0.7324080 0.8171899
>
> print(testconf_b)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      90     10      25
COMIC        14    106       5
MONOTYPE     22      4     110
> print(testconf_new)
      Reference
Prediction BOOKMAN COMIC MONOTYPE
BOOKMAN      82      7      31
COMIC        14    104      11
MONOTYPE     15      6     103
```

## Question 5

5.1) Once again, we set the `ntree = bntr = 200`, and `mtry = sqrt(112) = 11`; however, this time, we will apply the RF function to 3 problems:

**D1** = CL1 vs **D0** = {CL2 and CL3}

**E1** = CL2 vs **E0** = {CL1 and CL3}

**F1** = CL3 vs **F0** = {CL1 and CL2}

According to the case numbers of the new two classes (**D1** vs **D0**, or **E1** vs **E0**, or **F1** vs **F0**) of all three problems, there is a serious issue regarding strong imbalances that will lead to the skewed or biased prediction. As demonstrated in the table below, **D0**, **E0**, and **F0** are twice the case size of **D1**, **E1**, and **F1**.

```
> nrow(D1);nrow(D0)
[1] 597
[1] 1334
> nrow(E1);nrow(E0)
[1] 667
[1] 1264
> nrow(F1);nrow(F0)
[1] 667
[1] 1264
```

To combat the imbalanced classes, we clone to increase the number of cases in the smaller class until the case numbers of the training set and the test set between both classes respectively match.

So, here is how it works. For the training set of the smaller class (**D1** or **E1** or **F1**), we randomly select 80% of the cases as its training set at first. Then, we clone each case of its training set *n* times, where “*n*” is the ceiling value of the training set’s size ratio of bigger class to smaller class. Basically, the number of “*n*” is either 2 or 3 in all three questions here. Now, the modified training set of the smaller class definitely has a larger size than the intact training set of the bigger class (**D0** or **E0** or **F0**) does. To reach a balanced state between both classes’ training sets, we randomly select “*m*” quantities of only cloned cases in the modified training set of the smaller class to

remove out, where “m” is the size difference between the modified training set of smaller class and the intact training set of the bigger class.

For the test set of the smaller class, because we selected the training set in the original data set just now, the rest of 20% of the cases are automatically classified as its test set. Then the next procedures are all the same as how we did in the training set just now.

```
> print(RF1)
Call:
randomForest(formula = as.factor(trainset_target_D) ~ ., data = trainset_D,
  ntree = bntr, mtry = sqrt(p))
  Type of random forest: classification
    Number of trees: 200
No. of variables tried at each split: 11

  OOB estimate of  error rate: 15.32%
Confusion matrix:
      D1  D0 class.error
D1 279 258 0.48044693
D0 69 1528 0.04320601
> print(RF2)
Call:
randomForest(formula = as.factor(trainset_target_E) ~ ., data = trainset_E,
  ntree = bntr, mtry = sqrt(p))
  Type of random forest: classification
    Number of trees: 200
No. of variables tried at each split: 11

  OOB estimate of  error rate: 10.88%
Confusion matrix:
      E0  E1 class.error
E0 1533  16 0.01032924
E1  204 269 0.43128964
> print(RF3)
Call:
randomForest(formula = as.factor(trainset_target_F) ~ ., data = trainset_F,
  ntree = bntr, mtry = sqrt(p))
  Type of random forest: classification
    Number of trees: 200
No. of variables tried at each split: 11

  OOB estimate of  error rate: 6.43%
Confusion matrix:
      F0  F1 class.error
F0 918  93 0.09198813
F1  37 974 0.03659743
>
```

After applying our random forest to the three classifiers: RF1, RF2, and RF3, we observe that RF3 generated the lowest OOB error rate at 6.43 while RF1 had the highest at 15.32%.

```
> print(A1)
Accuracy
0.8483146
> print(A2)
Accuracy
0.8616601
> print(A3)
Accuracy
0.8003953
>
```

```
> print(M1)
      Reference
Prediction D1 D0
D1      69  20
D0      61 384
> print(M2)
      Reference
Prediction E0 E1
E0     365  53
E1      17  71
> print(M3)
      Reference
Prediction F0 F1
F0     223  71
F1       30 182
```

For our testset confusion matrix M1 of **D0** vs **D1**, we observe it to have an accuracy of 84.8%. For the other two confusion matrix of **E1** vs **E0**, and **F1** vs **F0**, we observe prediction accuracies of 86.1% and 80.0%

5.2) Now, applying separately the predictor bestRF already computed to the classify, CL1 vs {CL2 and CL3}, CL2 vs {CL1 and CL3}, CL3 vs {CL1 and CL2}, we yield accuracies B1,B2,and B3 along with their testset confusion matrices BM1, BM2, and BM3.

```
>
>
> print(B1)
Accuracy
0.9400749
> print(B2)
Accuracy
0.9743083
> print(B3)
Accuracy
0.9367589
>
```

```
> print(BM1)
      Reference
Prediction D1 D0
D1     122  24
D0       8 380
> print(BM2)
      Reference
Prediction E0 E1
E0     372   3
E1      10 121
> print(BM3)
      Reference
Prediction F0 F1
F0     249  28
F1        4 225
```



Compared to our results of question 5.1), our accuracy all increased for each classifier, specifically as below:

- The accuracy B1 increases 9.2% if compared to the accuracy A1 (B1 is 94% and A1 is 84.8%)
- The accuracy B2 increases 11.2% if compared to the accuracy A2 (B2 is 97.4% and A2 is 86.2%)
- The accuracy B3 increases 13.7% if compared to the accuracy A3 (B3 is 93.7% and A3 is 80.0%)

All in all, we get fairly better results for all 3 accuracies B1, B2, B3 in question 5.2) if compared to the accuracies A1, A2, A3 in question 5.1). Again, we believe our results can still be improved further in the future given that we explore and apply different approaches.

## R Codes

```
rm(list=ls()) # to remove all objects from a specified environment
cat("\f") # to clean console
```

### # Data Setup

```
library(readr)
setwd("~/Library/Mobile Documents/com~apple~CloudDocs/Study Files (Graduate)/MATH
6350/Homework/HW03/fonts/")
```

```
# We selected COMIC.csv, BOOKMAN.csv, MONOTYPE.csv.
COMIC <- data.frame(read_csv("COMIC.csv"))
BOOKMAN <- data.frame(read_csv("BOOKMAN.csv"))
MONOTYPE <- data.frame(read_csv("MONOTYPE.csv"))
```

```
c.names = names(COMIC) # All three font files shares same categories of columns
```

```
c.discard = match(c("fontVariant", "m_label", "orientation", "m_top", "m_left", "originalH",
                    "originalW", "h", "w"), c.names) # discard 9 columns listed
# na.omit() function is to omit all rows that contain NA values
comic = na.omit(COMIC[, -c.discard])
bookman = na.omit(BOOKMAN[, -c.discard])
monotype = na.omit(MONOTYPE[, -c.discard])
```

```
CL1 = bookman[bookman[, match("strength", names(bookman))] == 0.4 &
              bookman[, match("italic", names(bookman))] == 0,]
CL2 = comic[comic[, match("strength", names(comic))] == 0.4 &
            comic[, match("italic", names(comic))] == 0,]
CL3 = monotype[monotype[, match("strength", names(monotype))] == 0.4 &
               monotype[, match("italic", names(monotype))] == 0,]
DATA = rbind(CL1, CL2, CL3)
str(DATA)
```

### # Standardization

```
true_set = as.factor(DATA[,1])
start_col = match("r0c0", names(DATA))
end_col = ncol(DATA)
X = DATA[, start_col: end_col]
m = sapply(X, mean) # same as apply(X, MARGIN = 2, mean)
s = sapply(X, sd)
SDATA = scale(X)
```

### # Correlation Matrix Computation

```
R = cor(X)
# sigma = cov(X) # variance-covariance matrix "sigma"
# R=cov2cor(sigma) # either way to find correlation matrix
# all.equal(cor(X), cov2cor(sigma))
```

### # Eigenvalues and Eigenvectors

```
D = eigen(R)$values
# or D = (summary(pca)$sdev)^2
Q = eigen(R)$vectors
```

```
p = order(abs(cumsum(D)/sum(D) - 0.95), decreasing = FALSE)[1]
V = Q[, 1:p]
```

```

X = as.matrix(scale(X))
Y = data.frame(true_set, X%%*%V)
CL2 = Y[Y[,match("true_set", names(Y))] == "COMIC",]
CL1 = Y[Y[,match("true_set", names(Y))] == "BOOKMAN",]
CL3 = Y[Y[,match("true_set", names(Y))] == "MONOTYPE",]

```

```

n1 = nrow(CL1)
n2 = nrow(CL2)
n3 = nrow(CL3)
N = sum(n1, n2, n3)

```

```

N == nrow(DATA) # verification

```

```

SX = Y[,-1]

```

### # Question 1

```

set.seed(123)
pd1 = sample(x = 2, size = nrow(CL1), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL1
pd2 = sample(x = 2, size = nrow(CL2), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL2
pd3 = sample(x = 2, size = nrow(CL3), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL3

```

```

trainCL1 = CL1[pd1==1, ]
testCL1 = CL1[pd1==2, ]
trainCL2 = CL2[pd2==1, ]
testCL2 = CL2[pd2==2, ]
trainCL3 = CL3[pd3==1, ]
testCL3 = CL3[pd3==2, ]

```

```

TRAINSET_TARGET = rbind(trainCL1, trainCL2, trainCL3)[,1] # true train set
TESTSET_TARGET = rbind(testCL1, testCL2, testCL3)[,1] # true test set

```

```

TRAINSET = rbind(trainCL1, trainCL2, trainCL3)[,-1]
TESTSET = rbind(testCL1, testCL2, testCL3)[,-1]

```

### # Question 2: Random Forest(RF)

```

# reference: https://www.youtube.com/watch?v=dJclNIN-TPo
library(randomForest)
?randomForest
# convert to factors
# TRAINSET_TARGET <- as.factor(TRAINSET_TARGET)
# TESTSET_TARGET <- as.factor(TESTSET_TARGET)
rf_CL <- randomForest(TRAINSET_TARGET~.,data=TRAINSET)
print(rf_CL)
attributes(rf_CL)
rf_CL$confusion
# predication
confusionMatrix(p0,TRAINSET_TARGET)$table
confusionMatrix(p0,TRAINSET_TARGET)$overall[1]

p00 <- predict(rf_CL,TESTSET)
head(p00)
confusionMatrix(p00,TESTSET_TARGET)$table
confusionMatrix(p00,TESTSET_TARGET)$overall[1]

```

### # Question 3.1

```
set.seed(123)
rf = randomForest(TRAINSET_TARGET~., data = TRAINSET, ntree = 100, mtry = sqrt(p))
print(rf)
attributes(rf)
rf$confusion # the confusion matrix of the prediction (based on Out-of-Bag data)

# prediction & confusion matrix - train data
library(caret)
pred1 = predict(rf, TRAINSET)
trainconf = confusionMatrix(pred1, TRAINSET_TARGET)$table # or table(pred1, TRAINSET_TARGET)
print(trainconf)
trainperf = confusionMatrix(pred1, TRAINSET_TARGET)$overall[1] # or sum(diag(trainconf))/sum(trainconf)
print(trainperf)

# prediction & confusion matrix - test data
pred2 = predict(rf, TESTSET)
testconf = confusionMatrix(pred2, TESTSET_TARGET)$table
print(testconf)
testperf = confusionMatrix(pred2, TESTSET_TARGET)$overall[1]
print(testperf)
```

### # Question 3.2

```
rf = function(ntrees) {
  set.seed(123)
  rf = randomForest(TRAINSET_TARGET~., data = TRAINSET, ntree = ntrees, mtry = sqrt(p))
  return(rf)
}
testconf = function(ntrees) {
  pred2 = predict(rf(ntrees), TESTSET)
  testconf = confusionMatrix(pred2, TESTSET_TARGET)$table
  return(testconf)
}
n = c(10, 50, 100, 200, 300, 400)
conf = function(x) {
  n = n[x]
  conf = testconf(n)
  return(conf)
}
testperf = function(ntrees) { # accuracy
  pred2 = predict(rf(ntrees), TESTSET)
  testperf = confusionMatrix(pred2, TESTSET_TARGET)$overall[1]
  return(testperf)
}
conf(1)
conf(2)
conf(3)
conf(4)
conf(5)
conf(6)

testperf(ntrees = 10)
testperf(50)
testperf(100)
testperf(200)
```

```

testperf(300)
testperf(400)

plot(n, mapply(testperf, n), type = "b",
      panel.first=grid(),
      xlab = "ntree", ylab = "accuracy (testperf)")

coeff = function(k) {
  coeff = NULL
  for (i in c(1:6)) {
    coeff[i] = diag(matrix(mapply(conf, i), 3))[k]
  }
  return(coeff)
}
plot(n, coeff(1), type = "b", col = "red", ylim = c(80, 111),
      panel.first=grid(), xlab = "ntree", ylab = "diagonal coefficient")
lines(n, coeff(2), type = "b", col = "green")
lines(n, coeff(3), type = "b", col = "blue")

```

```
bntr = 200
```

#### # Question 4.1

```

set.seed(111)
bestRF = randomForest(TRAINSET_TARGET~, data = TRAINSET, ntree = bntr, mtry = sqrt(p))
print(bestRF)

# prediction & confusion matrix - test data (for bestRF)
pred2_b = predict(bestRF, TESTSET)
testconf_b = confusionMatrix(pred2_b, TESTSET_TARGET)$table
testperf_b = confusionMatrix(pred2_b, TESTSET_TARGET)$overall[1]

```

```

IM = function(k) {
  IM = bestRF$importance[k]
  return(IM)
}
L = function(k) {
  L = D[k]
  return(L)
}
k = 1:p
mapply(IM, k) # IM_1, IM_2, ..., IM_p
plot(mapply(L, k), mapply(IM, k), xlab = "eigenvalue: L_k", ylab = "bestRF's Importance: IM_k")

```

#### # Question 4.2

```

set.seed(111) # not set.seed(123), bc we need a different starting number used to generate a sequence of random numbers
pd1_new = sample(x = 2, size = nrow(CL1), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL1
pd2_new = sample(x = 2, size = nrow(CL2), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL2
pd3_new = sample(x = 2, size = nrow(CL3), replace = TRUE, prob = c(0.8, 0.2)) # partition data of CL3

trainCL1_new = CL1[pd1_new==1, ]
testCL1_new = CL1[pd1_new==2, ]
trainCL2_new = CL2[pd2_new==1, ]
testCL2_new = CL2[pd2_new==2, ]
trainCL3_new = CL3[pd3_new==1, ]

```

```

testCL3_new = CL3[pd3_new==2, ]

TRAINSET_TARGET_new = rbind(trainCL1_new, trainCL2_new, trainCL3_new)[,1] # true train set
TESTSET_TARGET_new = rbind(testCL1_new, testCL2_new, testCL3_new)[,1] # true test set

TRAINSET_new = rbind(trainCL1_new, trainCL2_new, trainCL3_new)[,-1]
TESTSET_new = rbind(testCL1_new, testCL2_new, testCL3_new)[,-1]

bestRF_new = randomForest(TRAINSET_TARGET_new~., data = TRAINSET_new, ntree = bntr, mtry = sqrt(p))
print(bestRF_new)

## prediction & confusion matrix - train data (new)
#library(caret)
#pred1_new = predict(bestRF_new, TRAINSET_new)
#trainconf_new = confusionMatrix(pred1_new, TRAINSET_TARGET_new)$table # or table(pred1,
#TRAINSET_TARGET)
#print(trainconf_new)
#trainperf_new = confusionMatrix(pred1_new, TRAINSET_TARGET_new)$overall[1] # or
#sum(diag(trainconf))/sum(trainconf)
#print(trainperf_new)

# prediction & confusion matrix - test data (new)
pred2_new = predict(bestRF_new, TESTSET_new)
testconf_new = confusionMatrix(pred2_new, TESTSET_TARGET_new)$table
testperf_new = confusionMatrix(pred2_new, TESTSET_TARGET_new)$overall[1]

CI = function(p, CL = 0.95, n) {
  ci = p + c(-1, 1)*qnorm(1-(1-CL)/2)*sqrt(p*(1-p)/n)
  return(ci)
}

print(testperf_b)
print(testperf_new)

CI(p = testperf_b, n = nrow(TESTSET))
CI(p = testperf_new, n = nrow(TESTSET_new))

print(testconf_b)
print(testconf_new)

# Question 5.1
# CL1 versus {CL2+CL3}
D1 = CL1
D0 = rbind(CL2, CL3)

set.seed(123)
x = sample(x = nrow(D1), size = as.integer(0.8*nrow(D1)), replace = FALSE)
time1 = ceiling(as.integer(0.8*nrow(D0))/length(x))
i = sample(x = setdiff(c(1:(length(x)*time1)), seq(1, length(x)*time1, time1)),
  size = length(x)*time1 - as.integer(0.8*nrow(D0)),
  replace = FALSE)
pd_D1train = rep(x, each = time1)[-i]
setdiff(x, pd_D1train) # verification for all the original 80% of cases are not missed, so we expect the output be
integer(0)

```

```

# setdiff(x, y) is the material that is in x, that is not in y

y = setdiff(c(1:nrow(D1)), x)
TIME1 = ceiling((nrow(D0) - as.integer(0.8*nrow(D0)))/length(y))
j = sample(x = setdiff(c(1:(length(y)*TIME1)), seq(1, (length(y)*TIME1), TIME1)),
          size = length(y)*TIME1 - (nrow(D0) - as.integer(0.8*nrow(D0))),
          replace = FALSE)
pd_D1test = rep(y, each = TIME1)[-j]
setdiff(y, pd_D1test) # verification for all the original 20% of cases are not missed, so we expect the output be
integer(0)

intersect(pd_D1train, pd_D1test) # verification for test set and training set do not intersect

trainD1 = D1[pd_D1train,]
testD1 = D1[pd_D1test,]

pd_D0train = sample(x = nrow(D0), size = round(0.8*nrow(D0)), replace = FALSE)
trainD0 = D0[pd_D0train,]
testD0 = D0[-pd_D0train,]

trainset_D = rbind(trainD1, trainD0)[-1]
testset_D = rbind(testD1, testD0)[-1]

trainset_target_D = as.factor(rbind(trainD1, trainD0)[,1])
testset_target_D = as.factor(rbind(testD1, testD0)[,1])

library(plyr) # for mapvalues() function
trainset_target_D = mapvalues(trainset_target_D, c("BOOKMAN", "COMIC", "MONOTYPE"), c("D1", "D0",
"D0"))
testset_target_D = mapvalues(testset_target_D, c("BOOKMAN", "COMIC", "MONOTYPE"), c("D1", "D0", "D0"))

RF1 = randomForest(as.factor(trainset_target_D)~., data = trainset_D, ntree = bntr, mtry = sqrt(p))

pred_D = predict(RF1, testset_D)
testconf_D = confusionMatrix(pred_D, as.factor(testset_target_D))$table
M1 = testconf_D
testperf_D = confusionMatrix(pred_D, as.factor(testset_target_D))$overall[1]
A1 = testperf_D

# CL2 versus {CL1+CL3}
E1 = CL2
E0 = rbind(CL1, CL3)

set.seed(123)
x2 = sample(x = nrow(E1), size = as.integer(0.8*nrow(E1)), replace = FALSE)
time2 = ceiling(as.integer(0.8*nrow(E0))/length(x2))
i2 = sample(x = setdiff(c(1:(length(x2)*time2)), seq(1, length(x2)*time2, time2)),
          size = length(x2)*time2 - as.integer(0.8*nrow(E0)),
          replace = FALSE)
pd_E1train = rep(x2, each = time2)[-i2]
setdiff(x2, pd_E1train) # verification for all the original 80% of cases are not missed, so we expect the output be
integer(0)

y2 = setdiff(c(1:nrow(E1)), x2)

```

```

TIME2 = ceiling((nrow(E0) - as.integer(0.8*nrow(E0)))/length(y2))
j2 = sample(x = setdiff(c(1:(length(y2)*TIME2)), seq(1, (length(y2)*TIME2), TIME2)),
            size = length(y2)*TIME2 - (nrow(E0) - as.integer(0.8*nrow(E0))),
            replace = FALSE)
pd_E1test = rep(y2, each = TIME2)[-j2]
setdiff(y2, pd_E1test) # verification for all the original 20% of cases are not missed, so we expect the output be
integer(0)

intersect(pd_E1train, pd_E1test) # verification for test set and training set do not intersect

trainE1 = E1[pd_E1train,]
testE1 = E1[pd_E1test,]

pd_E0train = sample(x = nrow(E0), size = round(0.8*nrow(E0)), replace = FALSE)
trainE0 = E0[pd_E0train,]
testE0 = E0[-pd_E0train,]

trainset_E = rbind(trainE1, trainE0)[-1]
testset_E = rbind(testE1, testE0)[-1]

trainset_target_E = as.factor(rbind(trainE1, trainE0)[,1])
testset_target_E = as.factor(rbind(testE1, testE0)[,1])

trainset_target_E = mapvalues(trainset_target_E, c("BOOKMAN", "COMIC", "MONOTYPE"), c("E0", "E1",
"E0"))
testset_target_E = mapvalues(testset_target_E, c("BOOKMAN", "COMIC", "MONOTYPE"), c("E0", "E1", "E0"))

RF2 = randomForest(as.factor(trainset_target_E)~, data = trainset_E, ntree = bntr, mtry = sqrt(p))

pred_E = predict(RF2, testset_E)
testconf_E = confusionMatrix(pred_E, as.factor(testset_target_E))$table
M2 = testconf_E
testperf_E = confusionMatrix(pred_E, as.factor(testset_target_E))$overall[1]
A2 = testperf_E

# CL3 versus {CL1+CL2}
F1 = CL3
F0 = rbind(CL1, CL2)

set.seed(123)
x3 = sample(x = nrow(F1), size = as.integer(0.8*nrow(F1)), replace = FALSE)
time3 = ceiling(as.integer(0.8*nrow(F0))/length(x3))
i3 = sample(x = setdiff(c(1:(length(x3)*time3)), seq(1, length(x3)*time3, time3)),
            size = length(x3)*time3 - as.integer(0.8*nrow(F0)),
            replace = FALSE)
pd_F1train = rep(x3, each = time3)[-i3]
setdiff(x3, pd_F1train) # verification for all the original 80% of cases are not missed, so we expect the output be
integer(0)

y3 = setdiff(c(1:nrow(F1)), x3)
TIME3 = ceiling((nrow(F0) - as.integer(0.8*nrow(F0)))/length(y3))
j3 = sample(x = setdiff(c(1:(length(y3)*TIME3)), seq(1, (length(y3)*TIME3), TIME3)),
            size = length(y3)*TIME3 - (nrow(F0) - as.integer(0.8*nrow(F0))),

```



```

        replace = FALSE)
pd_F1test = rep(y3, each = TIME3)[-j3]
setdiff(y3, pd_F1test) # verification for all the original 20% of cases are not missed, so we expect the output be
integer(0)

intersect(pd_F1train, pd_F1test) # verification for test set and training set do not intersect

trainF1 = F1[pd_F1train,]
testF1 = F1[pd_F1test,]

pd_F0train = sample(x = nrow(F0), size = round(0.8*nrow(F0)), replace = FALSE)
trainF0 = F0[pd_F0train,]
testF0 = F0[-pd_F0train,]

trainset_F = rbind(trainF1, trainF0)[-1]
testset_F = rbind(testF1, testF0)[-1]

trainset_target_F = as.factor(rbind(trainF1, trainF0)[,1])
testset_target_F = as.factor(rbind(testF1, testF0)[,1])

trainset_target_F = mapvalues(trainset_target_F, c("BOOKMAN", "COMIC", "MONOTYPE"), c("F0", "F0",
"F1"))
testset_target_F = mapvalues(testset_target_F, c("BOOKMAN", "COMIC", "MONOTYPE"), c("F0", "F0", "F1"))

RF3 = randomForest(as.factor(trainset_target_F)~, data = trainset_F, ntree = bntr, mtry = sqrt(p))

pred_F = predict(RF3, testset_F)
testconf_F = confusionMatrix(pred_F, as.factor(testset_target_F))$table
M3 = testconf_F
testperf_F = confusionMatrix(pred_F, as.factor(testset_target_F))$overall[1]
A3 = testperf_F

print(RF1)
print(RF2)
print(RF3)

print(A1)
print(A2)
print(A3)

print(M1)
print(M2)
print(M3)

```

### # Question 5.2

```

B1 = confusionMatrix(mapvalues(predict(bestRF, testset_D), c("BOOKMAN", "COMIC", "MONOTYPE"),
c("D1", "D0", "D0")),
        as.factor(testset_target_D))$overall[1]
B2 = confusionMatrix(mapvalues(predict(bestRF, testset_E), c("BOOKMAN", "COMIC", "MONOTYPE"), c("E0",
"E1", "E0")),
        as.factor(testset_target_E))$overall[1]
B3 = confusionMatrix(mapvalues(predict(bestRF, testset_F), c("BOOKMAN", "COMIC", "MONOTYPE"), c("F0",
"F0", "F1")),

```

```
as.factor(testset_target_F))$overall[1]

BM1 = confusionMatrix(mapvalues(predict(bestRF, testset_D), c("BOOKMAN", "COMIC", "MONOTYPE"),
c("D1", "D0", "D0"))),
as.factor(testset_target_D))$table
BM2 = confusionMatrix(mapvalues(predict(bestRF, testset_E), c("BOOKMAN", "COMIC", "MONOTYPE"),
c("E0", "E1", "E0"))),
as.factor(testset_target_E))$table
BM3 = confusionMatrix(mapvalues(predict(bestRF, testset_F), c("BOOKMAN", "COMIC", "MONOTYPE"),
c("F0", "F0", "F1"))),
as.factor(testset_target_F))$table

print(B1)
print(B2)
print(B3)

print(BM1)
print(BM2)
print(BM3)
```