MSDM5054 Final project, Fall 2022

ZHANG Juntao - 20908272

13th December, 2022

Part I: Classification on 20newsgroup Data

Perform data processing and split the dataset into two parts: train(90%), test(10%).

Problem 1.1

Model with 20newsgroup dataset through random forest algorithm and report the 5-fold cross validation value of the misclassification error.

Firstly, we perform parameter tuning in order to get the parameters with best performance in prediction. Use “randomForest” function in R, there are two parameters: “mtry” and “ntree”, which represent the number of predictors and the number of trees respectively.

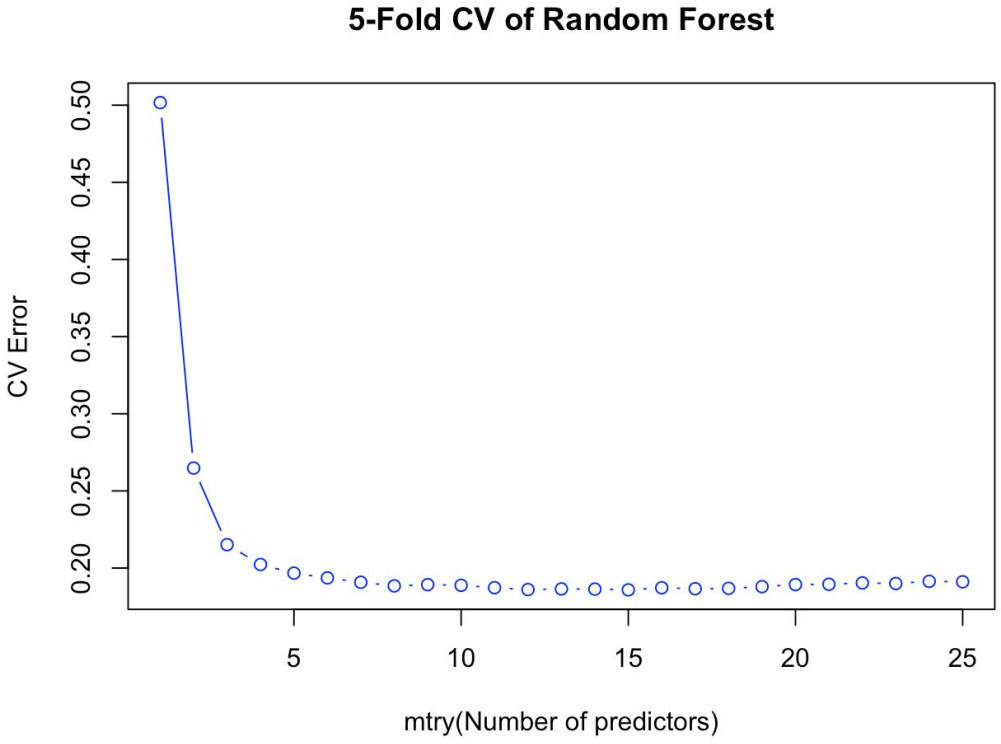
We first select the best “mtry” with “ntree”=100, i.e. find the best number of variables randomly sampled as candidates at each split with 100 trees to grow.

And set the range of “mtry” to be [1, 2, 3,...., 23, 24, 25], then we get the following results:

```
43  # parameter tuning - find the best mtry, with ntree=100
44  set.seed(7)
45  flds <- createFolds(1:14618, k = 5, list = TRUE, returnTrain = FALSE)
46  CVErr<-rep(0,25)
47  for(k in 1:25){
48    err<-0
49    for(fold in flds){
50      set.seed(7+k)
51      train_cv <- data.frame(train[-fold,])
52      test_cv <- data.frame(train[fold,])
53      rf = randomForest(group~.,data=train_cv, mtry=k, ntree=100, na.action=na.omit)
54      pred = predict(rf, newdata=test_cv, type="class")
55      conf <- table(pred, test_cv$group)
56      err <- err + 1 - sum(diag(conf))/sum(conf)
57    }
58    CVErr[k]<-err/5
59  }
60
61  plot(1:25,CVErr,type='b',col='blue', xlab="mtry(Number of predictors)",ylab="CV Error"
62  CVErr
63  min(CVErr)
64  which(CVErr==min(CVErr))
```

```
> CVErr
 [1] 0.5016615 0.2648027 0.2151812 0.2023142 0.1967111 0.1936319 0.1908000 0.1884606 0.1892608
[10] 0.1888296 0.1872909 0.1860593 0.1864904 0.1863671 0.1858748 0.1872291 0.1866133 0.1868598
[19] 0.1879682 0.1893225 0.1895074 0.1903694 0.1900001 0.1914160 0.1911080
> min(CVErr)
[1] 0.1858748
> which(CVErr==min(CVErr))
[1] 15
```

From which we can see that the 5-fold CV error is the smallest when “mtry” is equal to 15, and the smallest CV error is 0.1858748, the figure of 5-fold CV error with “mtry” is as follow figure shows.



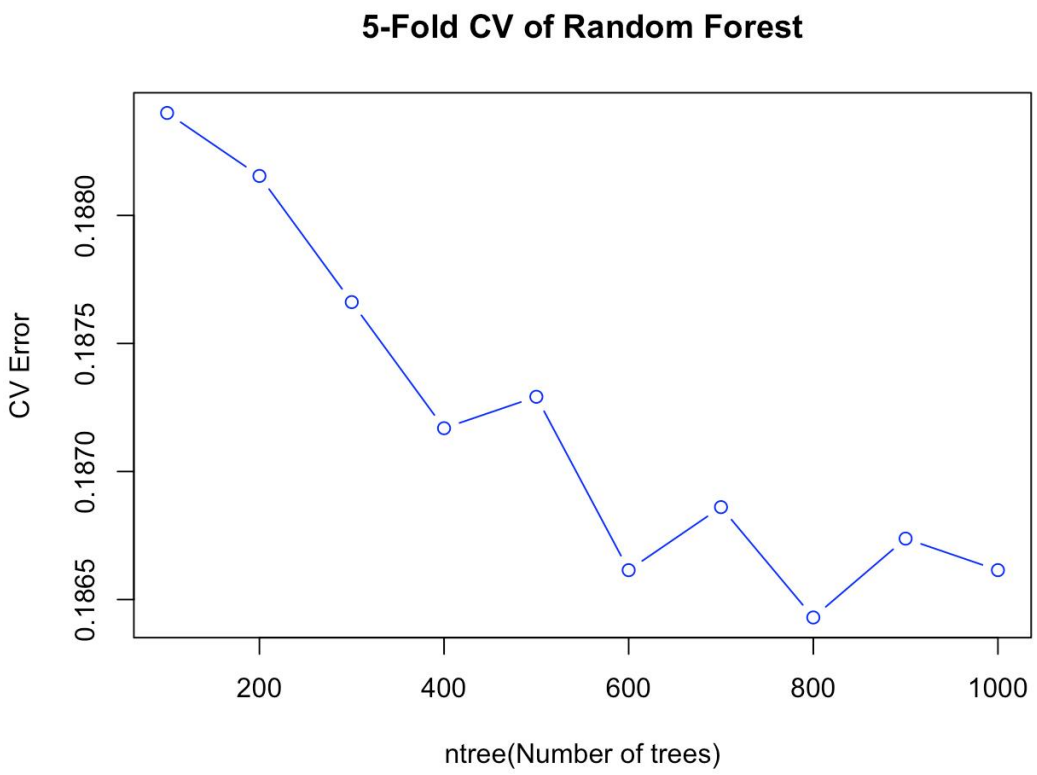
So we select “mtry”=15, then find the best value of “ntree” with “mtry”=15, i.e. find the best number of trees to grow, with 15 variables randomly sampled as candidates at each split. Set the range of “ntree” to be [100, 200, 300,....,800, 900, 1000], then get the following results:

```
# parameter tuning - find the best ntree, with mtry=15
ntree_list <- seq(100, 1000, 100)
set.seed(17)
flds <- createFolds(1:14618, k = 5, list = TRUE, returnTrain = FALSE)
CVErr<-rep(0,10)
for(k in 1:10){
  err<-0
  for(fold in flds){
    set.seed(7+k)
    train_cv <- data.frame(train[-fold,])
    test_cv <- data.frame(train[fold,])
    rf = randomForest(group~.,data=train_cv, mtry=15, ntree=100*k, na.action=na.omit)
    pred = predict(rf, newdata=test_cv, type="class")
    conf <- table(pred, test_cv$group)
    err <- err + 1 - sum(diag(conf))/sum(conf)
  }
  CVErr[k]<-err/5
}

plot(ntree_list,CVErr,type='b',col='blue', xlab="ntree(Number of trees)",ylab="CV Error"
CVErr
min(CVErr)
which(CVErr==min(CVErr))
```

```
> CVErr
 [1] 0.1884001 0.1881541 0.1876615 0.1871691 0.1872919 0.1866147 0.1868609 0.1864301 0.1867380
[10] 0.1866148
> min(CVErr)
[1] 0.1864301
> which(CVErr==min(CVErr))
[1] 8
```

From the results we can see the CV error is smallest when “ntree”=800, and the smallest CV error is 0.1864301, so we get the best CV error: 0.1864301
The figure of 5-fold CV error with “ntree” is as follow figure shows.



Therefore, we finished parameter tuning and get the best parameters: “mtry”=15, “ntree”=800. Use these two best parameters and train dataset to build a random forest, and get the prediction on test dataset, the result is shown in the following figure:

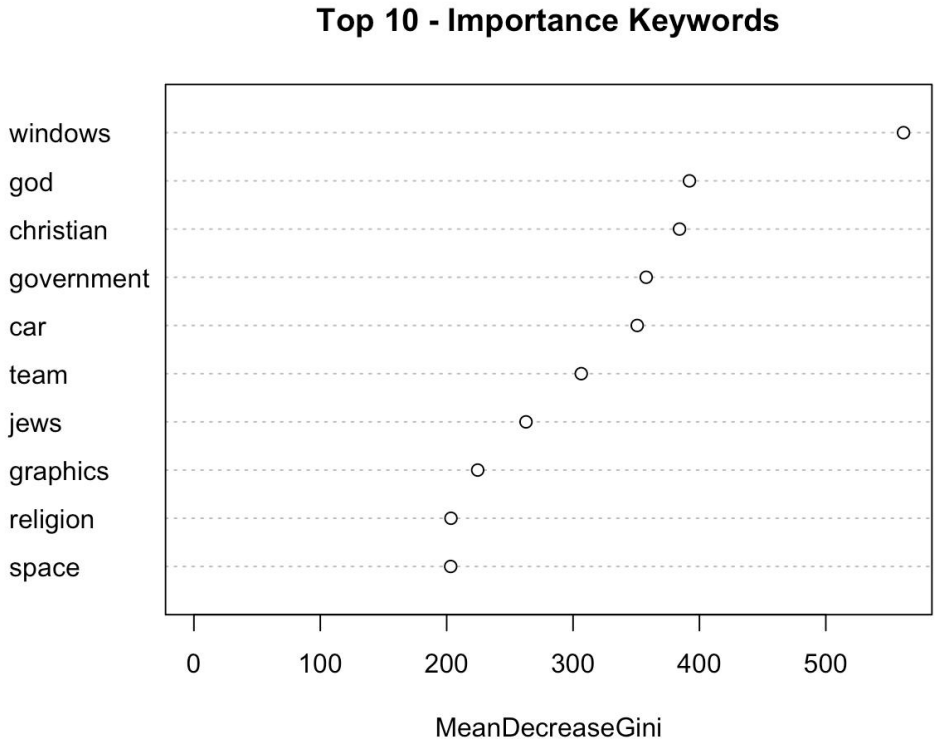
```
> rf = randomForest(group~.,data=train, mtry=15, ntree=800, na.action=na.omit)
> pred=predict(rf,newdata=test, mtry=15, ntree=800, na.action=na.omit, type="class")
> conf <- table(pred,test$group)
> conf

pred   1   2   3   4
   1 425  24  51  29
   2  10 275  11  11
   3  22  16 150  22
   4  23  34  42 479
> 1 - sum(diag(conf))/1624
[1] 0.1816502
```

The corresponding confusion matrix in test dataset is as above figure shown, and the misclassification error is 0.1816502

```
> important <- importance(rf)
> important[order(-important),]
   windows        god  christian government        car       team       jews   graphics   religion
561.498335 392.091389 384.227603 357.890286 350.757786 306.487996 262.766937 224.533985 203.382378
     space        gun   baseball        mac      games     hockey   software       card        bmw
203.156651 187.269821 179.502952 171.561066 163.765560 159.860134 132.505499 132.053070 131.196004
```

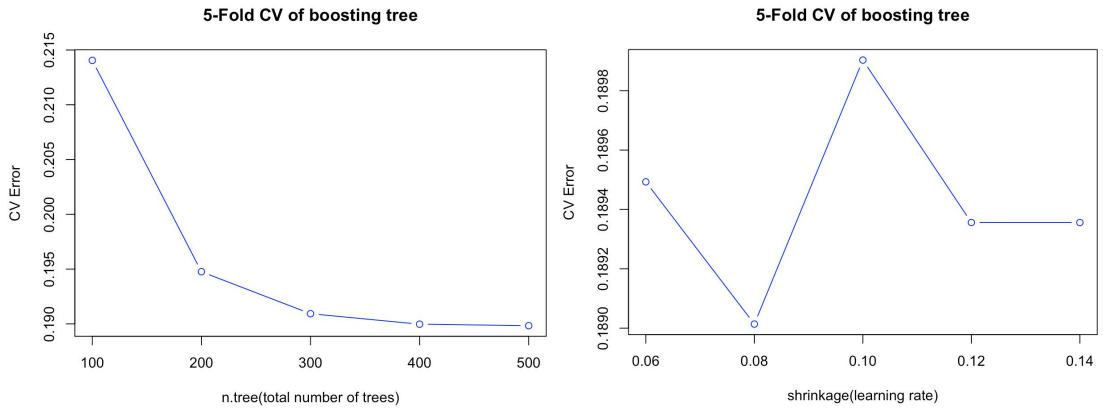
We can also see the ten most important keywords based on variable importance are: windows, god, christian, government, car, team, jews, graphics, religion, space. The Top 10 - Importance Keywords is as following figure shows.



Problem 1.2

Build a boosting tree for this dataset, use “gbm” function in R, there are three main parameters: “n.trees”, “shrinkage” and “interaction.depth”. Here, we consider tuning two parameters “n.trees” and “shrinkage”, because“interaction.depth”= 1 often works well, in which case each tree is a stump, consisting of a single split.

Firstly, set the range of “n.trees” to be [100, 200, 300, 400, 500] and “shrinkage”=0.1, find the best value of “n.trees” through 5-fold CV. The 5-fold CV error is smallest when “n.trees”=500, as the following figure shown, and the CV error is 0.1898349



Then find the best “shrinkage” through 5-fold CV, (through historical experience) set the range of “shrinkage” to be [0.06, 0.08, 0.1, 0.12, 0.14] and “n.trees”=500. And find that “shrinkage”=0.08 has the best performance, as the above figure shown, and

the smallest CV error is 0.1890137. So we get the best CV error is 0.1890137. And the tuning parameters are: “n.trees”=500, “shrinkage”=0.08, “interaction.depth”= 1
Finally, use the above selected parameters and train dataset to build a boosting tree, and get the prediction on test dataset, get the result as the following figure shown:

```
> conf

yhat   1   2   3   4
   1 396  30  63  28
   2  11 250  17  10
   3  30  13 138  36
   4  15  33  39 515
> 1 - sum(diag(conf))/sum(conf)
[1] 0.2001232
```

The confusion matrix is as above figure shown, the misclassification error is 0.2001232

Problem 1.3

Compare the results from random forest and boosting trees:
We can see that, after parameter tuning, the misclassification error of random forest in test dataset is 0.1816502 and the misclassification error of boosting tree in the test dataset is 0.2001232. So we can conclude that random forest has a better predicting performance than boosting tree in this problem. Besides, on train dataset, the best CV errors of random forest and boosting tree are 0.1864301 and 0.1890137 respectively, which are very close.
These results show that random forest perform better in test dataset, and boosting tree perform better in train dataset, which agrees with the properties of these two model: random Forest improves performance by reducing model’s variance; boosting tree improves performance by reducing model’s bias.

Problem 1.4

Use all predictors to build a multi-class LDA classifier and get the 5-fold CV error is 0.2025579, as the following figure(left) shows.

```
> set.seed(7)
> flds <- createFolds(1:14618, k = 5, list = TRUE, returnTrain = FALSE)
> err<-0
> for(fold in flds){
+   train_cv <- data.frame(train[-fold,])
+   test_cv <- data.frame(train[fold,])
+   lda.fit<-lda(group~.,data=train_cv)
+   pred<-predict(lda.fit,test_cv)
+   conf <- table(pred$class,test_cv$group)
+   err <- err + 1 - sum(diag(conf))/sum(conf)
+ }
> err/5
[1] 0.2025579
```

```
> lda.fit<-lda(group~.,data=train)
> pred<-predict(lda.fit,test)
> conf <- table(pred$class,test$group)
> conf

     1   2   3   4
  1 408  48  62  17
  2   5 255   8  14
  3  19  15 140  22
  4  26  32  51 502
> 1 - sum(diag(conf))/sum(conf)
[1] 0.1964286
```

Then use the train dataset use build a multi-class LDA classifier and get prediction on test dataset, the confusion matrix is as above figure(right) shows. And the misclassification error in test dataset is 0.1964286

Problem 1.5

Use several important predictors got from Problem 1.1 to build a multi-class QDA classifier in order to avoid “rank deficiency” probelm. The selected six predictors are: windows, god, christian, car, government, team. Get the following results:

```
> set.seed(17)
> flds <- createFolds(1:14618, k = 5, list = TRUE, returnTrain = FALSE)
> err<-0
> for(fold in flds){
+   train_cv <- data.frame(train[-fold,])
+   test_cv <- data.frame(train[fold,])
+   qda.fit<-qda(group~windows+god+christian+car+government+team,data=train_cv)
+   pred<-predict(qda.fit,test_cv)
+   conf <- table(pred$class,test_cv$group)
+   err <- err + 1 - sum(diag(conf))/sum(conf)
+ }
> err/5
[1] 0.4906967
```

```
> conf

     1   2   3   4
1 439 210 234 309
2   1 106  11   9
3   0   1   0   0
4  12   9  12 271
> 1 - sum(diag(conf))/sum(conf)
[1] 0.4975369
```

The confusion matrix is as the above figure(right) shows, and the 5-fold CV error is 0.4906967, as above figure(left) shows. And the misclassification error in test dataset is 0.4975369.

Problem 1.6

Compare the performances of all above methods.

List the misclassification errors in test dataset of the above four methods(from best to worest):

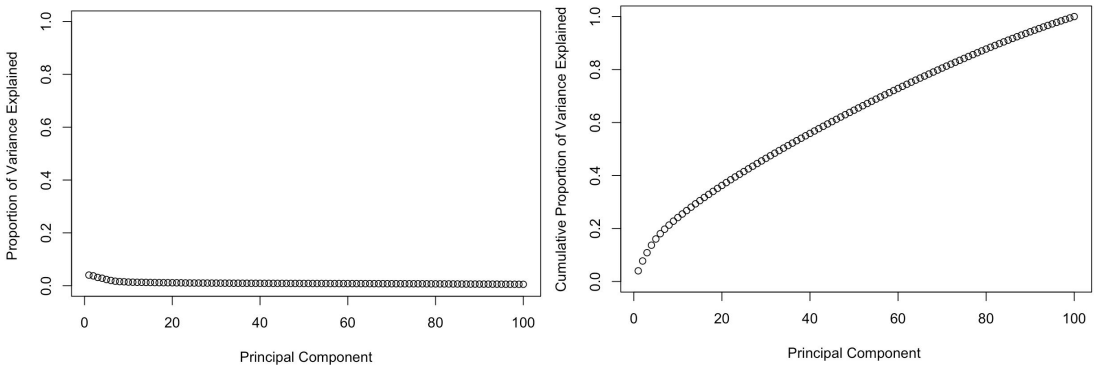
1. random forest: 0.1816502
2. multi-class LDA classifier: 0.1964286
3. boosting tree: 0.2001232
4. multi-class QDA classifier: 0.4975369

From the above results, we can see that in test dataset, random forest has best performance, multi-class LDA classifier has the second-greatest performance, boosting tree has the third-greatest performance, and multi-class QDA classifier has the worest performance. Since we only use six predictors to build multi-class QDA classifier in order to avoid “rank deficiency” probelm, the cost is losing a lot of information, resulting in a large misclassification error.

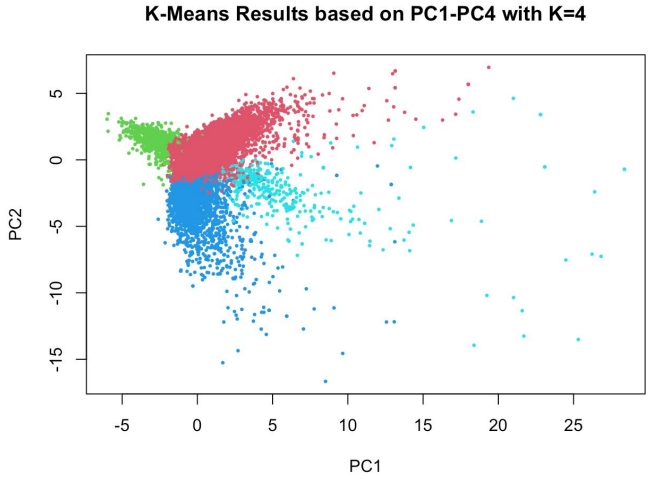
Part II. Spectral Clustering (PCA + K-means)

Problem 2.1

Firstly, apply PCA on the binary occurrence matrix, get the principal components' proportion of variance explained and cumulative proportion of Variance explained, as the following figures shown.



Then take the top 4 left singular vectors of the occurrence matrix and apply K-means on the rows of these singular vectors with K=4, display the clustering result in the two-dimensional plane, as the following figure shows.



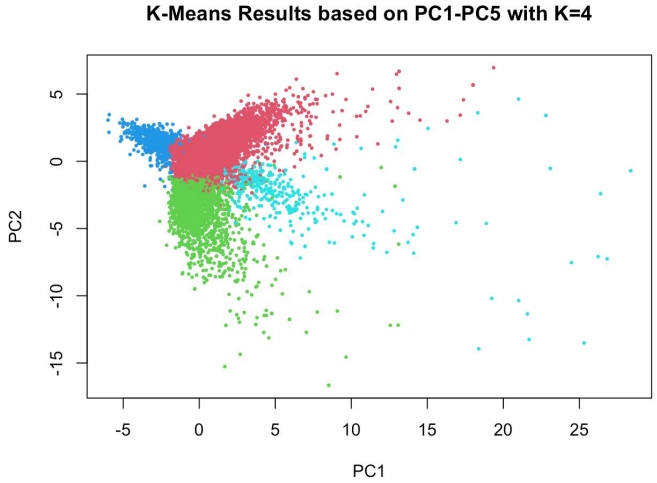

```
> # mis-clustering error rate
> num1<-1
> err1<-rep(0,24)
> for(i in 1:4){
+   for(j in 1:4){
+     for(h in 1:4){
+       for(k in 1:4){
+         if((i!=j)&(i!=h)&(i!=k)&(j!=h)&(j!=k)&(h!=k)){
+           dt<-as.factor(groups$V1)
+           dt[which(dt=='1')]<-i
+           dt[which(dt=='2')] <-j
+           dt[which(dt=='3')] <-h
+           dt[which(dt=='4')] <-k
+           err1[num1]<-0
+           for(row in 1:16242){
+             if(dt[row]!=km.out$cluster[row]){
+               err1[num1]<-err1[num1]+1
+             }
+           }
+           num1<-num1+1
+         }
+       }
+     }
+   }
+ }
```

Use the method as above figure shows to get the mis-clustering error and the mis-clustering error rate, as the following figure shows. The mis-clustering error is 3919 and mis-clustering error rate is 0.241288

```
> min(err1)
[1] 3919
> min(err1)/16242
[1] 0.241288
```

Problem 2.2

Take the top 5 left singular vectors of the occurrence matrix and apply K-means on the rows of these singular vectors with K=4. Display the clustering result in the two-dimensional plane, as the following figure shows.



Use the same method in **Problem 2.1** to get the mis-clustering error and the mis-clustering error rate, as the following figure shows. The mis-clustering error is 3907 and mis-clustering error rate is 0.2405492

```
> min(err1)
[1] 3907
> min(err1)/16242
[1] 0.2405492
```

Problem 2.3

Compare with the performances from part I:

Firstly, use top 5 left singular vectors to apply K-means with K=4 get a smaller mis-clustering error rate than use top4 left singular vectors since more information was used.

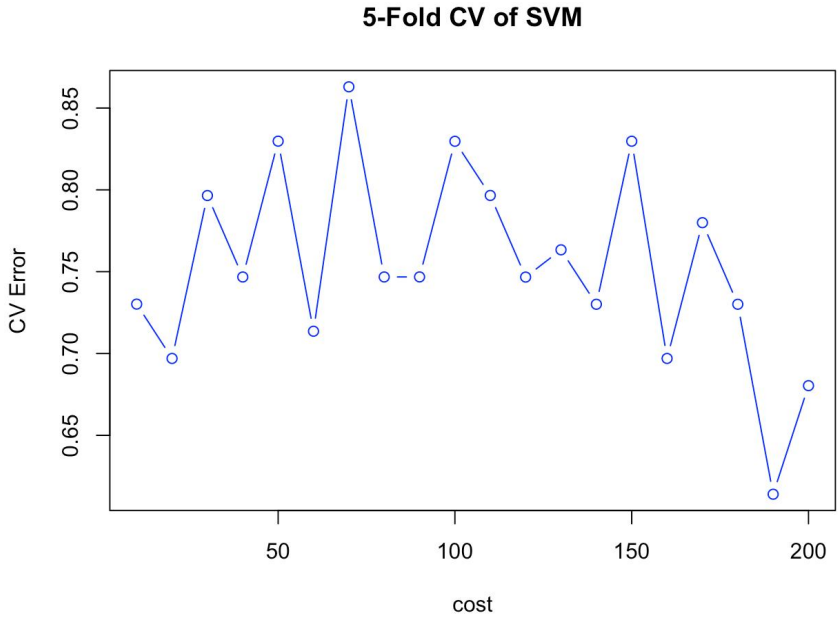
And compare with the performances from supervised methods of part I, we can see that supervised methods have better performance except QDA method. This is consistent with our expection, since more information was used in methods of part I, not only the information of label, but also the information of predictiors.

Part III. Classification on MNIST Data

Problem 3.1

Use only the digit images of 3 and 6 to build an SVM with linear kernel classifier for binary classification. And choose the best cost parameter by 5-fold cross validation, set the searching range of cost parameter to be [10, 20, 30, ... , 180, 190, 200]. Get the 5-fold CV errors as the following figure shows, as we can see, the 5-fold CV error is the smallest when cost=190 and the smallest CV error is 0.6139978

```
cost_list <- seq(10, 200, 10)
CVerr <- rep(0, length(cost_list))
for(i in 1:length(cost_list)){
  svm1=svm(label~., data=train1, kernel="linear", cost=cost_list[i], scale=TRUE, cross=5)
  CVerr[i] <- mean(100-svm1$accuracies)
}
```



So we select cost=190 and linear kernel, use train1 dataset to build a SVM binary classification, the time cost of training this model is about 3 senconds.
Then apply it on the test1 dataset, the confusion matrix is as following figure shown, and the misclassification error 0.009341998=0.9341998%

```
> mean(pred!=test1$label)
[1] 0.009341998
> confusionMatrix(pred, test1$label)
Confusion Matrix and Statistics

          Reference
Prediction    3    6
         3 1251   12
         6   11 1188
```

Problem 3.2

Use only the digit images of 3 and 6 to build an SVM with radial kernel classifier for binary classification. And choose the best cost parameter, gamma parameter by 5-fold cross validation, set the seraching range, cost: [10, 50, 100, 150, 200], gamma: [0.001, 0.003, 0.005, 0.007, 0.009, 0.01, 0.015, 0.02, 0.025]. Then perform grid search, get the results of CV errors, as the following figure shows.

```
cost_list <- c(10, 50, 100, 150, 200)
gamma_list <- c(0.001, 0.003, 0.005, 0.007, 0.009, 0.01, 0.015, 0.02, 0.025)
CVerr <- matrix(0, nrow = length(gamma_list), ncol = length(cost_list))
for(i in 1:length(gamma_list)){
  for(j in 1:length(cost_list)){
    svm2=svm(label~., data=train1, kernel="radial", gamma=gamma_list[i],
             cost=cost_list[j], scale=TRUE, cross=5)
    CVerr[i,j] <- mean(100-svm2$accuracies)
  }
}
CVerr
```

```
> CVerr
          [,1]      [,2]      [,3]      [,4]      [,5]
 [1,] 0.4148689 0.4480640 0.4646615 0.3982990 0.5310240
 [2,] 0.4315077 0.3485064 0.3650764 0.4480777 0.3484927
 [3,] 0.5310378 0.3817015 0.4314527 0.3982714 0.4148827
 [4,] 0.6471790 0.6970266 0.6140391 0.6139703 0.6471790
 [5,] 0.7800004 0.7633341 0.7633616 0.7633203 0.7965291
 [6,] 0.8463079 0.8463216 0.8629329 0.9127805 0.8629329
 [7,] 1.2446344 1.1782030 1.1781618 1.1782443 1.1947593
 [8,] 1.8586322 1.7424909 1.7091169 1.8586459 1.6761696
 [9,] 2.2568623 2.3565299 2.3564061 2.3398911 2.3232799
```

About the above figure of CV error, row represent different gamma and column represent different cost. As we can see, the CV error is the smallest at position [2,5], i.e. when gamma=0.003, cost=200, and the smallest CV error is 0.3484927

So we select cost=200, gamma=0.003 and radial kernel, use train1 dataset to build a SVM binary classifier,the time cost of training this model is about 4 seconds.
Then apply it on the test dataset, the confusion matrix is as following figure shown, and the misclassification error 0.00528026=0.528026%

```
> svm2=svm(label~., data=train1, kernel="radial",gamma=0.003,cost=200,scale=TRUE)
> pred <- predict(svm2, newdata = test1)
> mean(pred!=test1$label)
[1] 0.00528026
> confusionMatrix(pred, test1$label)
Confusion Matrix and Statistics

          Reference
Prediction    3    6
         3 1258    9
         6    4 1191
```

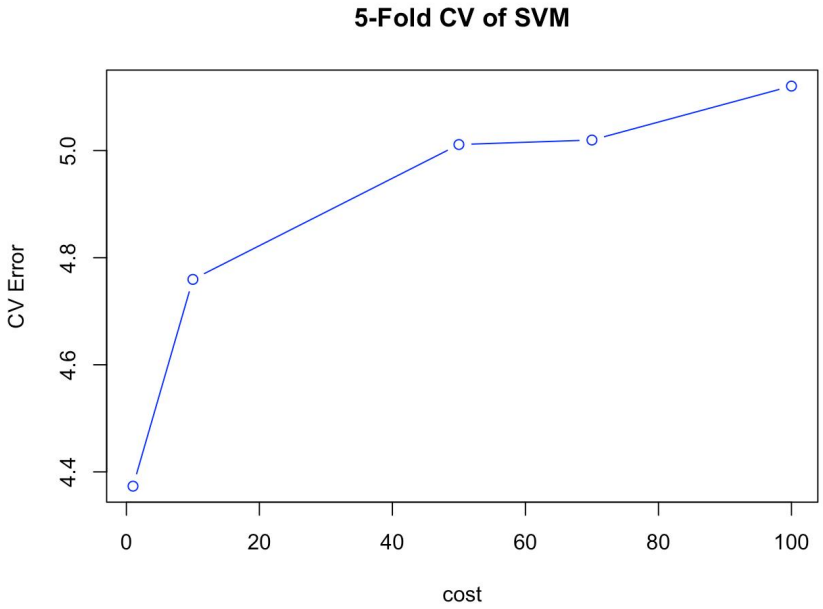
Problem 3.3

Compare the results of the above two models:

As we can see from the above results, in test dataset, the misclassification error of model-1(linear kernel) is about 0.9% and the misclassification error of model-2(radial kernel) is about 0.5%. So after parameter tuning, SVM with radial kernel has a better performance then SVM with linear kernel. I think one of the reasons is that SVM with radial kernel has more flexibility since it has two parameters could be tune to fit different situations.

Problem 3.4

Use only the digit images of1,2,5,8 to build an SVM with linear kernel classifier for multi-class classification. And choose the best cost parameter by 5-fold cross validation, set the searching range of cost parameter to be [1, 10, 50, 70, 100]. Get the 5-fold CV errors as the following figure shows, as we can see, the 5-fold CV error is the smallest when cost=1and the smallest CV error is 4.373341.



So we select cost=1 and linear kernel, use train2 dataset to build a SVM mult-class classifier, the time cost of training this model is about 17 seconds.

Then apply it on the test2 dataset, the confusion matrix is as the following figure shown, and the misclassification error 0.04598419=4.598419%

```
> svm3=svm(label~., data=train2, kernel="linear", cost=1, scale=TRUE)
> pred <- predict(svm3, newdata = test2)
> mean(pred!=test2$label)
[1] 0.04598419
> confusionMatrix(pred, test2$label)
Confusion Matrix and Statistics

          Reference
Prediction    1    2    5    8
         1 1344    2   14   21
         2   11 1142   18   29
         5    0   17 1057   40
         8    8   24   37 1042
```

Problem 3.5

Use the complete dataset of train and test to build an SVM classifier for classifying all 10 classes.

Historical experience: when the number of features is very large, which is similar to the number of samples, select the SVM with linear kernel; and when the number of features is much smaller than the sample size, and the sample size is normal(not very large or very small), select SVM with radial kernel.

So based on historical experience and our experiments on the complete dataset, we choose SVM with radial kernel. Then perform parameter tuning to choose the best cost parameter and gamma parameter. Since the heavy calculation, we firstly determine a rough range of these two parameters through experiments, and then set the searching range, cost: [10, 20, 30, 40], gamma: [0.001, 0.003, 0.005, 0.007, 0.009, 0.01]. Perfrom grid search based on above ranges and select best parameters by 5-fold CV. Get the results as the following figure shown.

```
> CVerr
          [,1]     [,2]     [,3]     [,4]
[1,] 4.736667 4.350000 4.186667 4.093333
[2,] 3.546667 3.473333 3.420000 3.530000
[3,] 3.406667 3.370000 3.466667 3.420000
[4,] 3.573333 3.430000 3.526667 3.553333
[5,] 3.686667 3.556667 3.656667 3.730000
[6,] 3.793333 3.876667 3.776667 3.873333
```

Aboout the above figure, row represent different gamma and column represent different cost. As we can see, the CV error is the smallest at position [3,2], i.e. when gamma=0.005, cost=20, and the smallest CV error is 3.37

So we select cost=20, gamma=0.005 and radial kernel, use train dataset to build a SVM mult-class classifier, the time cost to train this mdoel is about 120 seconds. Then apply it on the test dataset, the confusion matrix is as following figure shown, and the misclassification error 0.03325=3.325%

```
> svm4=svm(label~., data=train, kernel="radial", cost=20, gamma=0.005, scale=TRUE)
> pred <- predict(svm4, newdata = test)
> mean(pred!=test$label)
[1] 0.03325
> confusionMatrix(pred, test$label)
Confusion Matrix and Statistics

          Reference
Prediction    0    1    2    3    4    5    6    7    8    9
         0 1121    0    2    0    2    4    4    1    2    2
         1    0 1341    0    4    2    2    0    1    3    2
         2    3   14 1154   29   11    4    9    5   10    8
         3    2    1    5 1197    0   18    0    2    9    3
         4    0    1    7    0 1141    0    5    7    4   16
         5    4    0    3   12    3 1071    8    2   12    3
         6    7    1    1    1    3   13 1171    0    1    0
         7    0    1    6    6    2    2    0 1227    3   16
         8    2    3    4    5    1    9    3    0 1079    4
         9    1    1    3    8   10    3    0   19    9 1099
```

Part IV. Additional Bonus(CNN through python)

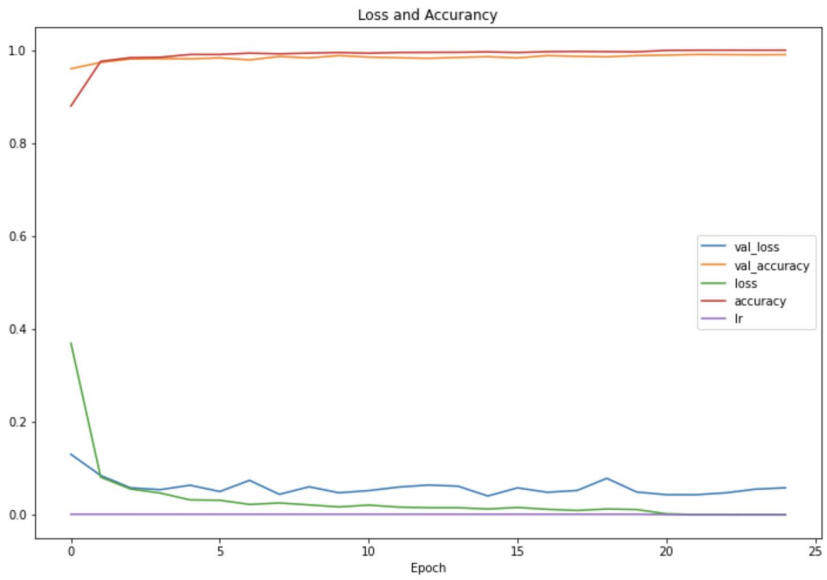
Use Convolutional Neural Network(CNN) on the MNIST dataset. Firstly, perform min-max scaling on the independent variables. Then design a network structure, the self-designed network structure is as the following figure shown.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 12, 12, 64)        640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 12, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 128)         147584
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 6, 6, 192)         221376
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 192)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 192)         921792
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 192)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 768)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               196864
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,601,610
Trainable params: 1,601,610
Non-trainable params: 0
_________________________________________________________________
```

Set model's parameters as the following figure shown, use the model to train and get the figure of training process, the time cost to train this model is about 1.5 hours.

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.3, verbose=1,
                              patience=2, min_lr=0.00000001)
```

```
start = time.time()
history = model.fit(
    X_train, Y_train,
    batch_size=100,
    epochs=25,
    validation_split=0.1,
    callbacks=[reduce_lr],
    verbose=1,
    shuffle=True)
```

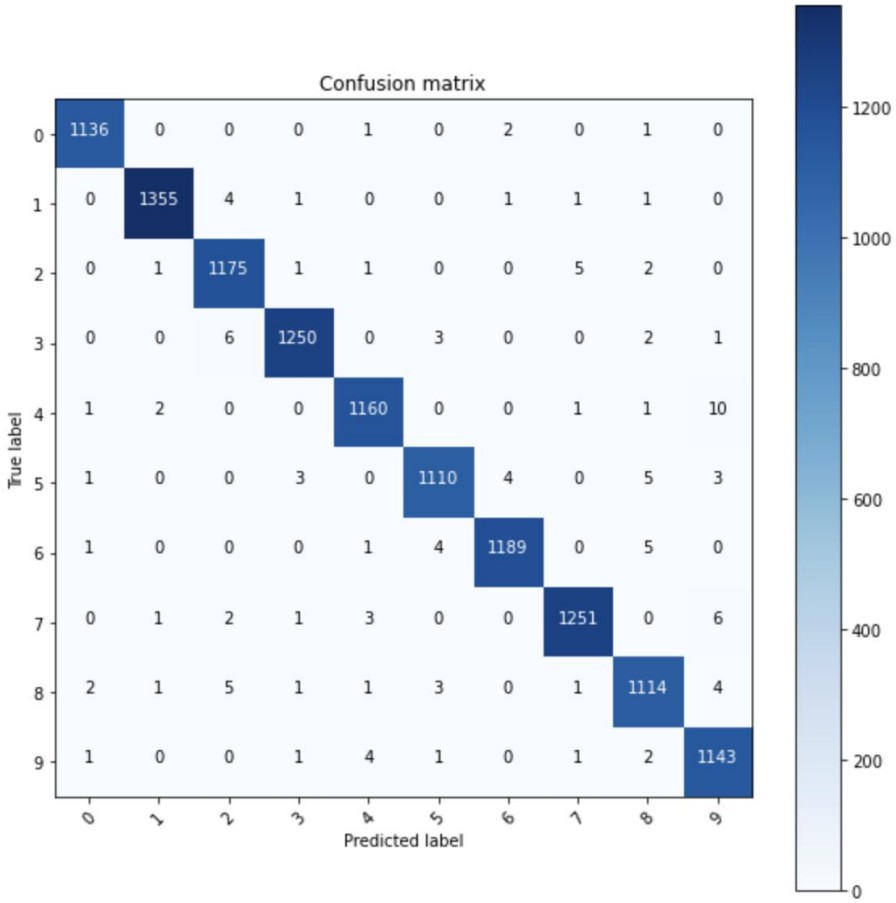


Finally, use the trained model to predict on the test dataset, and get the results as the following figure shown. The accuracy in test dataset is 99.025%, and the misclassification error in test dataset is 0.975%.

```
from sklearn.metrics import confusion_matrix
pred = model.predict_classes(X_test, verbose=2)
conf = confusion_matrix(Y_test, pred)
```

```
accuracy = np.sum(conf.diagonal())/np.sum(conf)
error = 1-accuracy
print('The accuracy in test dataset: ', np.round(accuracy*100,3), '%', sep='')
print('The misclassification error in test dataset:', np.round(error*100, 3), '%', sep='')
```

```
The accuracy in test dataset: 99.025%
The misclassification error in test dataset:0.975%
```



Conclusion:

As we can see from the above results, the test error of CNN is 0.975%, which is much smaller than the test error of SVM(3.325%). This shows the advantage of NN method in prediction. But in time cost aspect, CNN use about 1.5 hours and SVM only use 2 minutes, this shows that SVM is easy to train and NN method often need much more time to train.

Part V. Semi-supervised Classification

Problem 5.1

Try three classification methods using only labelled training data.

5.1.1 Random forest

Build a random forest classifier, choose the ntree parameter and mtry parameter by 5-fold CV, set ntree_list: [10, 30, 50, 70, 90], mtry_list: [1, 2, 3, 4, 5]
and then perform grid search, get the following results.

```
> CVErr
 [1] 0.2820 0.2632 0.2632 0.2632 0.2632 0.3064 0.2876 0.2876 0.2632 0.2876 0.2484 0.2876 0.2876
[14] 0.2876 0.2632 0.2820 0.2876 0.3064 0.2632 0.2876 0.2632 0.2876 0.2876 0.2876 0.2876
> min(CVErr)
[1] 0.2484
> which(CVErr==min(CVErr))
[1] 11
```

From the CV results, we can determine the best pararmeters: ntree=10, mtry=3. Use them to get prediction on test dataset, get the following result. The test error is 0.094

```
> rf <- randomForest(NSP~.,data=train_label,mtry=3, ntree=10)
> pred=predict(rf,newdata=test, type="class")
> conf <- table(pred,test$NSP)
> conf

pred   1   2   3
   1 376  27   4
   2   2  41   5
   3   2   7  35
> 1 - sum(diag(conf))/sum(conf)
[1] 0.09418838
```

5.1.2 SVM with linear kernel

Build a SVM classifier with linear kernel, choose cost parameter by 5-fold CV, set cost_list: [1, 2, 3, ... , 8, 9, 10], get the following results.

```
> CVerr
 [1] 11.8 11.0 10.4 11.2 10.8 11.4 13.8 12.0 11.4 12.2
> min(CVerr)
[1] 10.4
> which(CVerr==min(CVerr))
[1] 3
```

Then, we can get the best pararmeter: cost=3, use it to get prediction on test dataset, get the following result. The test error is 0.118

```
> svm1=svm(NSP~., data=train_label, kernel="linear", cost=3, scale=TRUE)
> pred <- predict(svm1, newdata = test)
> mean(pred!=test$NSP)
[1] 0.1182365
> confusionMatrix(pred, test$NSP)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 361  24   4
         2  18  45   6
         3   1   6  34
```

5.1.3 KNN

Build a KNN classifier, choose the k parameter by 5-fold CV, set k_list:[1, 2, ..., 9, 10], get the following results.

```
> CVErr
 [1] 0.3200000 0.2800000 0.2133333 0.2133333 0.2533333 0.2533333 0.2533333 0.2533333 0.2533333
[10] 0.2533333
> min(CVErr)
[1] 0.2133333
> which(CVErr==min(CVErr))
[1] 3 4
```

Then, we can get the best pararmeter: k=3 or k=4, choose k=3 and use it to get prediction on test dataset, get the following result. The test error is 0.146

```
> model<-knn3(NSP~.,train_label,k=3)
> pre <-  predict(model, test, type="class")
> t<-table(test$NSP, pre)
> 1-sum(diag(t))/sum(t)
[1] 0.1462926
> t
   pre
      1   2   3
  1 358  18   4
  2  29  40   6
  3  11   5  28
```

5.1.4 Conclusions

From the results of the above three methods, we can get their misclassification errors on test dataset.

Random forest: 0.094

SVM with linear kernel: 0.118

KNN: 0.146

So random forest has the best performance, SVM with linear kernel has the sencond best performance and KNN has the worest performance among these three methods. I think one of the possible reasons is the unbalanced distribution of samples, as the following figure shown. Most of the samples with label-1 in both train and test datasets.

```
> summary(y1)
  1   2   3
388  62  50
> summary(y2)
  1   2   3
380  75  44
```

Since KNN predict smaple's label to be the label with the most occurrences among the k-nearest neighbors, so KNN will be affected greatly when the sample distribution is unbalanced, but random forest and SVM will be affected less in this problem.

Problem 5.2

Try two semi-supervised classification methods using both labelled training data and unlabelled training data.

5.2.1 Semi-KNN

Build a Semi-KNN classifier, and use it to get prediction on test dataset, get the following result. The test error is 0.174

```
> semi1 <- selfTraining(x = x_train, y = y_train, learner = knn3,
+                       learner.pars = list(k = 1), pred = "predict")
> pred<-predict(semi1,x_test)
> table(pred,y_test)
    y_test
pred   1   2   3
   1 354  41  13
   2  18  29   2
   3   8   5  29
> mean(pred!=y_test)
[1] 0.1743487
```

5.2.2 Semi-SVM

Build a Semi-SVM classifier, and use it to get prediction on test dataset, get the following result. The test error is 0.136

```
> semi2 <- selfTraining(x = x_train, y = y_train,
+                       learner = learner,
+                       learner.pars = learner.pars,
+                       pred = pred)
> pred <- predict(semi2, x_test)
> table(pred,y_test)
    y_test
pred   1   2   3
   1 357  26   6
   2  20  43   7
   3   3   6  31
> mean(pred!=y_test)
[1] 0.1362725
```

5.3.3 Comparison with Problem 5.1

List the test error of all these methods firstly.

Random forest: 0.094 SVM with linear kernel: 0.118

KNN: 0.146 Semi-KNN: 0.174 Semi-SVM: 0.136

From the above results, we can see that KNN has better performance than Semi-KNN, SVM has better performance than Semi-SVM, and Semi-SVM only has a better performance than KNN.

So gererally, in this case, supervised learning methods perform better than semi-supervised learning methods. I think there are two possible reasons: 1. small size of the dataset and unbalanced samples; 2. data quality, maybe the data quality is not very good, i.e. exists many noise samples. Since semi-supervised learning assum that unlabeled data must be meaningful, potentially valuable samples, not useless noise samples.