CSC529 HW3 Jonggoo Kang

Problem 1: (Handwriting recognition using support vector machines)

```
# library
library(caret)
library(e1071)
library(ggplot2)
library(lattice)
library(rpart)

# load the data
setwd('/Users/jaygkay/Desktop/CSC529/HW3')
usdat=read.table('uspsdata.txt',header=F)
usclass=read.table("uspscl.txt",header=F)
dim(usdat); dim(usclass)

## [1] 200 256

## [1] 200   1

# combining the data
uspsdata <- as.matrix(usdat)
uspscl <- as.matrix(usclass)
colnames(uspscl) <- "class"
data <- cbind(uspsdata, uspscl)
data <- as.data.frame(data)
dim(data)

## [1] 200 257

# split the data with 33% of a Test set
intrain <- createDataPartition(y=data$class, p=0.66, list = FALSE)
train <- data[intrain,]
test <- data[-intrain,]
train[["class"]] = factor(train[["class"]])
test[["class"]] = factor(test[["class"]])
dim(train); dim(test)

## [1] 132 257

## [1]  68 257
```

1-a

```
# Train linear SVM
### perform cross validation using 10 folds and repeat this precess 3 times
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
### train linear SVM with soft margin
```

```r
gridLinear <- expand.grid(C = c(0.1))
linearSVM <- train(class ~., data = train,
                   method = "svmLinear",
                   tuneGrid = gridLinear,
                   trControl = trctrl,
                   tuneLength = 10) #preProcess = c("center","scale")
linearSVM
```

```
## Support Vector Machines with Linear Kernel
##
## 132 samples
## 256 predictors
##   2 classes: '-1', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 119, 120, 118, 120, 119, 119, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9419414  0.8837879
##
## Tuning parameter 'C' was held constant at a value of 0.1
```

It shows that the train model has the accuracy of 94.19%

```r
# Test Prediction
test_pred_linear <- predict(linearSVM, newdata = test)
test_pred_linear
```

```
##  [1] 1  1  1  -1 1  1  1  1  -1 1  -1 1  -1 -1 -1 1  -1 1  -1 -1 1  1  -1
## [24] -1 1  1  -1 1  1  -1 -1 -1 -1 -1 1  -1 -1 1  1  -1 -1 -1 1  -1 1  -1
## [47] -1 -1 -1 1  -1 1  1  -1 1  1  -1 -1 1  -1 -1 1  -1 1  -1 1  1  1
## Levels: -1 1
```

```r
# accuracy for testing
confusionMatrix(test_pred_linear, test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 34  1
##         1   0 33
##
##                Accuracy : 0.9853
##                  95% CI : (0.9208, 0.9996)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
```

```
##
##                   Kappa : 0.9706
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9706
##          Pos Pred Value : 0.9714
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5147
##       Balanced Accuracy : 0.9853
##
##        'Positive' Class : -1
##
```

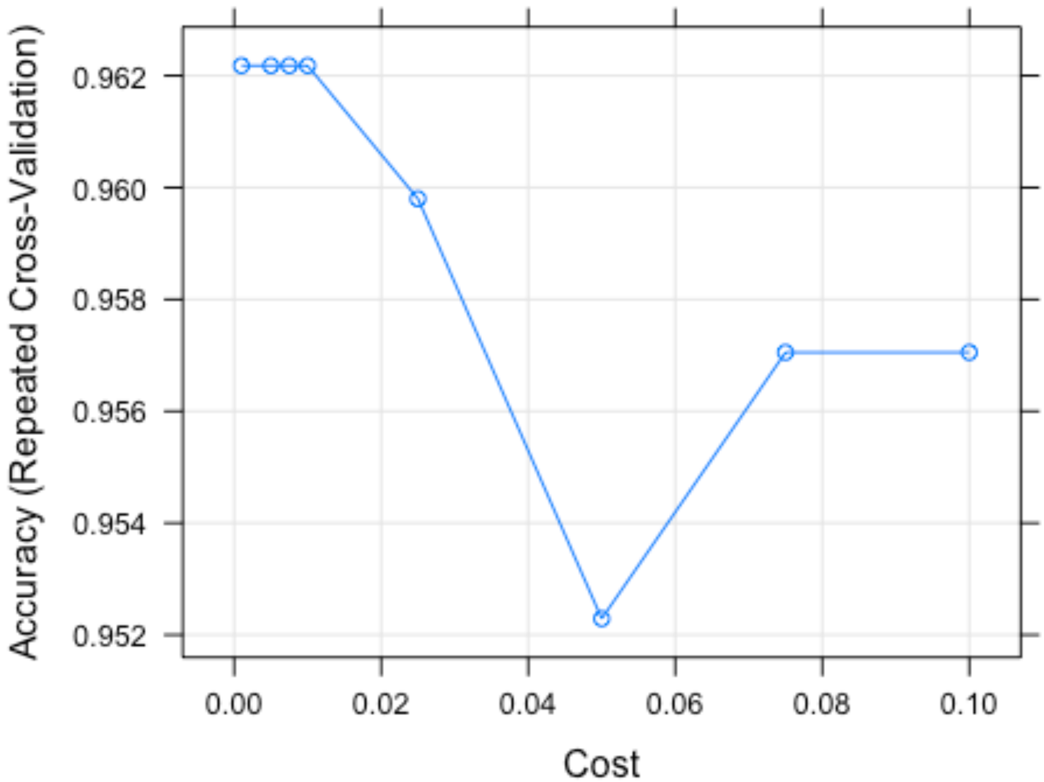
The result shows that the accuracy of test is 98.53%. Thus, this model is well trained.

```
# Vary the soft margin parameters
gridLinear1 <- expand.grid(
  C = c(0, 0.1, 0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.025, 0.001))
linearSVM_grid <- train(class ~., data = train,
                        method = "svmLinear",
                        tuneGrid = gridLinear1,
                        trControl = trctrl,
                        tuneLength = 10)
linearSVM_grid
```

```
## Support Vector Machines with Linear Kernel
##
## 132 samples
## 256 predictors
##   2 classes: '-1', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 118, 120, 119, 118, 119, 119, ...
## Resampling results across tuning parameters:
##
##   C       Accuracy   Kappa
##   0.0000        NaN        NaN
##   0.0010  0.9621795  0.9246157
##   0.0050  0.9621795  0.9246157
##   0.0075  0.9621795  0.9246157
##   0.0100  0.9621795  0.9246157
##   0.0250  0.9597985  0.9198538
##   0.0500  0.9522894  0.9047730
##   0.0750  0.9570513  0.9142968
##   0.1000  0.9570513  0.9142968
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.001.

plot(linearSVM_grid)
```



The above plot is showing that the classifier is giving best accuracy on c = 0.001 and it recommends the best model has 96.22% of accuracy.

1-b

```
### train Radial SVM with soft margin
gridRadial <- expand.grid(sigma = c(0.0025), C=(0.1))
radialSVM <- train(class ~., data = train,
                   method = "svmRadial",
                   trControl = trctrl,
                   tuneLength = 10,
                   tuneGrid = gridRadial) #preProcess = c("center","scale"),
tuneGrid = grid,

radialSVM
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 132 samples
## 256 predictors
##   2 classes: '-1', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 119, 118, 119, 119, 119, 118, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9394383  0.8787796
##
## Tuning parameter 'sigma' was held constant at a value of 0.0025
##
## Tuning parameter 'C' was held constant at a value of 0.1
```

The accuracy of train is 93.94% in the case with soft margin.

```
# Test Prediction
test_pred_radial <- predict(radialSVM, newdata = test)
test_pred_radial
```

```
##  [1] 1  1  1  -1 1  1  1  1  -1 1  -1 1  -1 -1 -1 1  -1 1  -1 -1 1  1  -1
## [24] -1 1  1  -1 1  1  -1 -1 -1 -1 -1 1  -1 -1 1  1  -1 -1 -1 1  -1 1  -1
## [47] -1 -1 -1 1  -1 1  1  -1 1  1  -1 -1 1  -1 -1 1  -1 1  -1 1  1  1
## Levels: -1 1
```

```
# accuracy for testing
confusionMatrix(test_pred_radial, test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 34  1
##         1   0 33
##
##                Accuracy : 0.9853
##                  95% CI : (0.9208, 0.9996)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9706
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
##             Specificity : 0.9706
```

```
##          Pos Pred Value : 0.9714
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5147
##       Balanced Accuracy : 0.9853
##
##        'Positive' Class : -1
##
```

The result shows that the accuracy of test is 98.53%

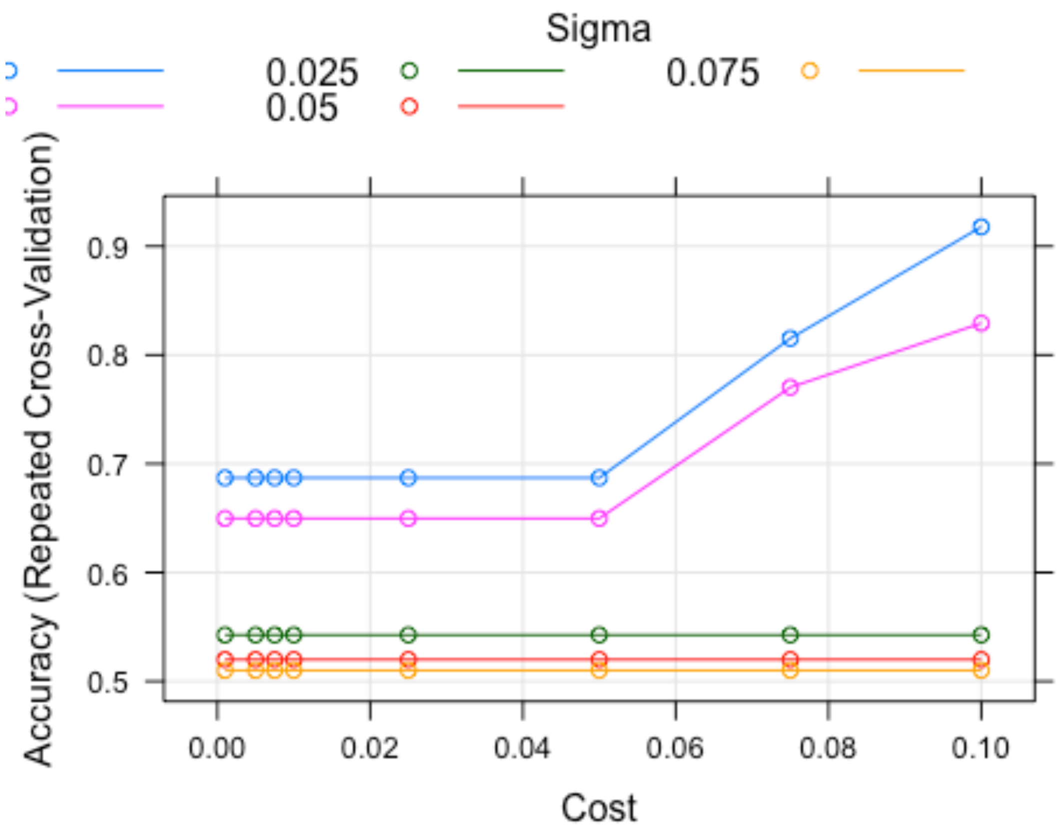
```
# vary the soft margin parameters
gridRadial1<- expand.grid(
  sigma = c(0.01, 0.025, 0.05, 0.075, 0.001),
  C = c(0, 0.1, 0.075, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.025, 0.001))

radialSVM_grid <- train(class ~., data = train,
                        method = "svmRadial",
                        trControl = trctrl,
                        tuneLength = 10,
                        tuneGrid = gridRadial1)
radialSVM_grid
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 132 samples
## 256 predictors
##   2 classes: '-1', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 118, 120, 118, 119, 118, 120, ...
## Resampling results across tuning parameters:
##
##   sigma  C       Accuracy   Kappa
##   0.001  0.0000        NaN         NaN
##   0.001  0.0010  0.6870269  0.41507937
##   0.001  0.0050  0.6870269  0.41507937
##   0.001  0.0075  0.6870269  0.41507937
##   0.001  0.0100  0.6870269  0.41507937
##   0.001  0.0250  0.6870269  0.41507937
##   0.001  0.0500  0.6870269  0.41507937
##   0.001  0.0750  0.8152320  0.64503338
##   0.001  0.1000  0.9177961  0.83728937
##   0.010  0.0000        NaN         NaN
##   0.010  0.0010  0.6497253  0.34047619
##   0.010  0.0050  0.6497253  0.34047619
##   0.010  0.0075  0.6497253  0.34047619
```

```
##   0.010  0.0100  0.6497253  0.34047619
##   0.010  0.0250  0.6497253  0.34047619
##   0.010  0.0500  0.6497253  0.34047619
##   0.010  0.0750  0.7702381  0.55779800
##   0.010  0.1000  0.8292125  0.66388661
##   0.025  0.0000        NaN         NaN
##   0.025  0.0010  0.5425824  0.12619048
##   0.025  0.0050  0.5425824  0.12619048
##   0.025  0.0075  0.5425824  0.12619048
##   0.025  0.0100  0.5425824  0.12619048
##   0.025  0.0250  0.5425824  0.12619048
##   0.025  0.0500  0.5425824  0.12619048
##   0.025  0.0750  0.5425824  0.12619048
##   0.025  0.1000  0.5425824  0.12619048
##   0.050  0.0000        NaN         NaN
##   0.050  0.0010  0.5199634  0.08095238
##   0.050  0.0050  0.5199634  0.08095238
##   0.050  0.0075  0.5199634  0.08095238
##   0.050  0.0100  0.5199634  0.08095238
##   0.050  0.0250  0.5199634  0.08095238
##   0.050  0.0500  0.5199634  0.08095238
##   0.050  0.0750  0.5199634  0.08095238
##   0.050  0.1000  0.5199634  0.08095238
##   0.075  0.0000        NaN         NaN
##   0.075  0.0010  0.5100427  0.06111111
##   0.075  0.0050  0.5100427  0.06111111
##   0.075  0.0075  0.5100427  0.06111111
##   0.075  0.0100  0.5100427  0.06111111
##   0.075  0.0250  0.5100427  0.06111111
##   0.075  0.0500  0.5100427  0.06111111
##   0.075  0.0750  0.5100427  0.06111111
##   0.075  0.1000  0.5100427  0.06111111
## 
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.001 and C = 0.1.
```

```
plot(radialSVM_grid)
```



SVM RBF kernel shows that the accuracy is 91.78 % with sigma = 0.001 and c =0.1

1-c

```
# linearSVM Test Prediction
test_pred_linear_grid <- predict(linearSVM_grid, newdata = test)
test_pred_linear_grid

##  [1] 1  1  1  -1 1  1  1  1  -1 1  -1 1  -1 -1 -1 1  -1 1  -1 -1 1  1  -1
## [24] -1 1  1  -1 1  1  -1 -1 -1 -1 -1 1  -1 -1 1  1  1  -1 -1 1  -1 1  -1
## [47] -1 -1 -1 1  -1 1  1  -1 1  1  -1 -1 1  -1 -1 1  -1 1  -1 1  1  1
## Levels: -1 1

# accuracy for testing
confusionMatrix(test_pred_linear_grid, test$class)

## Confusion Matrix and Statistics
##
##             Reference
## Prediction -1  1
##         -1 33  1
##         1   1 33
```

```
##
##                Accuracy : 0.9706
##                  95% CI : (0.8978, 0.9964)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9412
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9706
##             Specificity : 0.9706
##          Pos Pred Value : 0.9706
##          Neg Pred Value : 0.9706
##              Prevalence : 0.5000
##          Detection Rate : 0.4853
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9706
##
##        'Positive' Class : -1
##
```

The results of confusion matrix show that this time the accuracy on the test set is 97%

```
# Test Prediction
test_pred_radial_grid <- predict(radialSVM, newdata = test)
test_pred_radial_grid
```

```
##  [1] 1  1  1  -1 1  1  1  1  -1 1  -1 1  -1 -1 -1 1  -1 1  -1 -1 1  1  -1
## [24] -1 1  1  -1 1  1  -1 -1 -1 -1 -1 1  -1 -1 1  1  -1 -1 -1 1  -1 1  -1
## [47] -1 -1 -1 1  -1 1  1  -1 1  1  -1 -1 1  -1 -1 1  -1 1  -1 1  1  1
## Levels: -1 1
```

```
# accuracy for testing
confusionMatrix(test_pred_radial_grid, test$class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1 34  1
##         1   0 33
##
##                Accuracy : 0.9853
##                  95% CI : (0.9208, 0.9996)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9706
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 1.0000
```

```
##              Specificity : 0.9706
##           Pos Pred Value : 0.9714
##           Neg Pred Value : 1.0000
##               Prevalence : 0.5000
##           Detection Rate : 0.5000
##     Detection Prevalence : 0.5147
##        Balanced Accuracy : 0.9853
##
##         'Positive' Class : -1
##
```

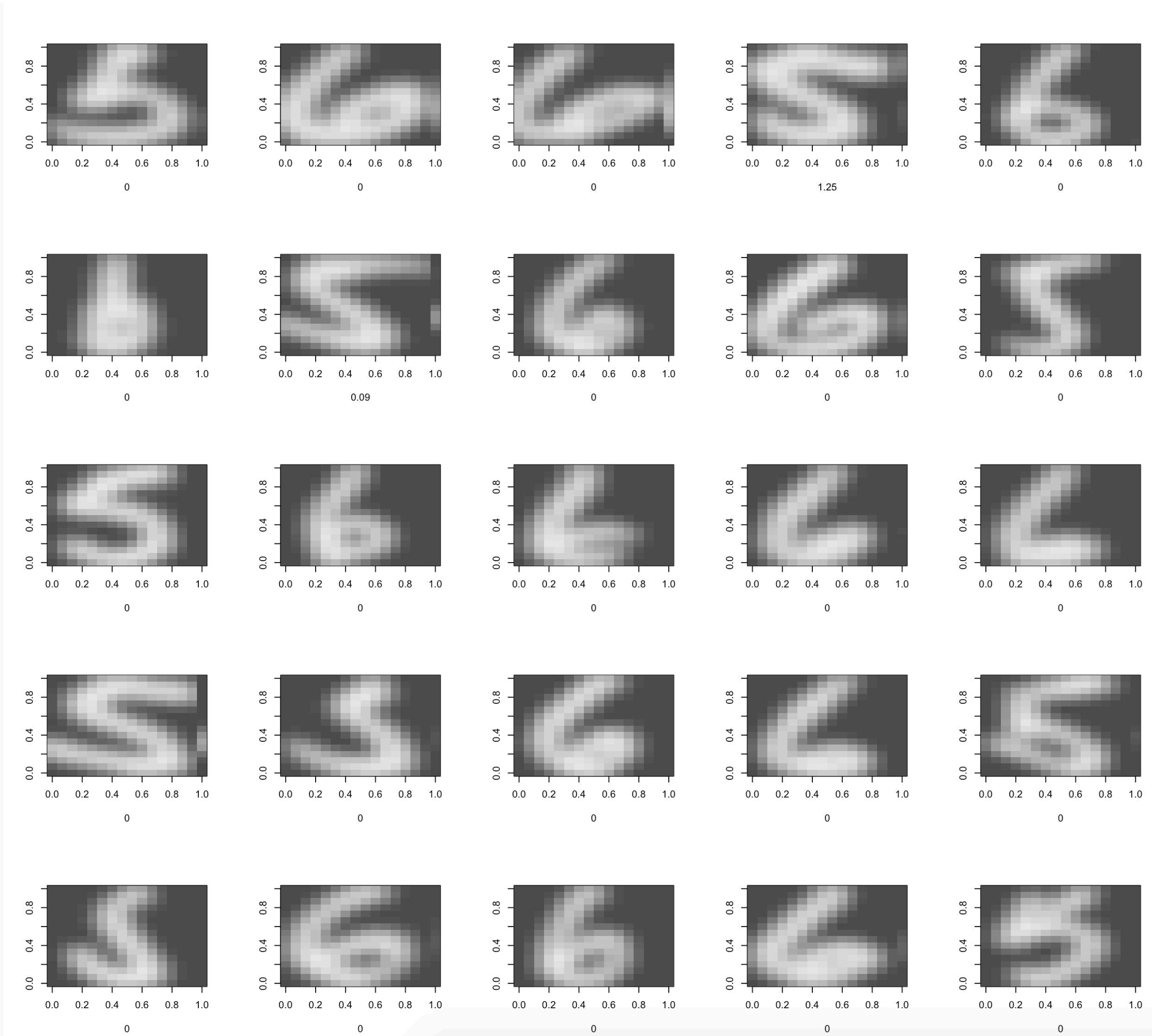
The results of confusion matrix show that this time the accuracy on the test set is 98.53% Linear SVM has better accuracy

1-d (extra credit)

```
# create a 16x16 matrix with pixel color values
m = matrix(unlist(data[1,-1]), nrow = 16, byrow = TRUE)

# plot the matrix

image(m, col = grey.colors(255))
# reverse (rotate the matrix)
rotate <- function(x) t(apply(x,2,rev))
# plot some of images
par(mfrow=c(4,4))
lapply(1:16,
       function(x) image(
         rotate(matrix(unlist(train[x,-1]), nrow = 16, byrow = TRUE)),
         col = grey.colors(255),
         xlab = train[x,1]))
```




```
test_pred_linear_grid
```

-1 indicates 5 and 1 indicates 6

```
[1] 1  1  1  -1 1
[6] 1  1  1  -1 1
[11]-1 1  -1 -1 -1
[16] 1  -1 1  -1 -1
[21] 1  1  -1 -1 1
```

Misclassification colored blue

Correctly classification colored black

Problem 2: (E-commerce Customer Identification using ensemble of classifiers)

More details about analysis attached behind.

2-1

DT Training Results	Sensitivity/precision	Specificity/Recall
Raw data	1	0
Raw data with balanced data	0.7657	0.4191
Normalize attributes	0.7657	0.4191
Feature selection	0.7657	0.4191

2-2

RF Training Results	Sensitivity/precision	Specificity/Recall
Raw data	0.9996	0.9197
Raw data with balanced data	0.9813	0.9505
Normalize attributes	0.9912	0.9813
Feature selection	0.9912	0.9714

2-3

DT Testing Results	Sensitivity/precision	Specificity/Recall
Raw data	1	0
Raw data with balanced data	0.7614	0.4292
Normalize attributes	0.7614	0.4292
Feature selection	0.7614	0.4292

RF Testing Results	Sensitivity/precision	Specificity/Recall
Raw data	0.9781	0.0394
Raw data with balanced data	0.5847	0.5719
Normalize attributes	0.5389	0.6060
Feature selection	0.5644	0.5772

Decision Tree is the algorithm which calculates using information. This fact shows there is no changes on both sensitivity and specificity as the dataset transforms. In this reason, trainsforming the dataset during Decision Tree is not ciritically necessary. However, Randome Foreset shows different results. RF is affected by transfored dataset. After applying balanced, normalized, and feature-selected dataset, the sensitity and specificity have changed. It means that exploratory analysis on dataset will derive ciritical changes on classification.

2-4

Checking missing values

By checking with command sum(is.na()), I noticed that here is no missing values in the dataset.

```
train <- cbind(x_train, y_train)
train$Class <- as.factor(train$Class)

test <- cbind(x_test, y_test)
test$Class <- as.factor(test$Class)
dim(train); dim(test)

## [1] 10000   335

## [1] 10000   335

###### Checking missing values
sum(is.na(train))

## [1] 0

sum(is.na(test))

## [1] 0
```

No missing values are detected

Balancing the dataset

```
table(train$Class)

## 
##    0    1 
## 9091  909
```

In the train set, there are 9091 0s and 909 1s. It is absolutely an imbalced class. To balance the data, I will do down-sampling.

```
# balance_train
bal_train <- downSample(x = train[, -ncol(train)], y = train$Class)
table(bal_train$Class)

## 
##   0   1 
## 909 909

dim(bal_train)

## [1] 1818  335
```

Now the classes are well balanced.

```r
# balnace_test
table(test$Class)

##
##    0    1
## 9061  939

bal_test <- downSample(x = test[, -ncol(test)], y = test$Class)
table(bal_test$Class)

##
##   0   1
## 939 939

dim(bal_test)

## [1] 1878  335
```

Normalization

I normalized the data with library(caret) by centring and scaling the dataset, and it gives me well normalized and scaled datasets.

```r
# nomal_train
process_train <- preProcess(bal_train[1:334], method = c("center", "scale"))
norm_train <- predict(process_train, bal_train[1:335])
str(norm_train)

## 'data.frame':    1818 obs. of  335 variables:
##  $ V1   : num  0.993 -1.006 0.993 -1.006 0.993 ...
##  $ V2   : num  -1.889 -0.193 -1.7 -0.193 -0.758 ...
##  $ V3   : num  -0.55 -0.55 -0.55 -0.55 -0.55 ...
##  $ V4   : num  -1.751 -1.578 -1.578 -0.195 -0.714 ...
##  $ V5   : num  1.9532 1.9532 1.9532 0.0774 0.0774 ...
##  $ V6   : num  -0.523 1.91 -0.523 -0.523 -0.523 ...
##  $ V7   : num  -0.82 -0.82 1.22 -0.82 1.22 ...
##  $ V8   : num  0.696 -1.435 -1.435 -1.435 0.696 ...
##  $ V9   : num  -0.638 -0.638 -0.638 -0.638 -0.638 ...
##  $ V10  : num  0.991 -1.009 0.991 -1.009 0.991 ...
##  $ V11  : num  -0.0203 -0.7784 -0.4941 -1.489 0.5008 ...
##  $ V12  : num  0.752 -0.989 -0.119 -0.119 -0.989 ...
##  $ V13  : num  0.59 0.59 0.59 0.59 0.59 ...
##  $ V14  : num  -0.512 0.332 0.332 -0.192 -0.978 ...
##  $ V15  : num  0.993 -1.006 0.993 -1.006 0.993 ...
##  $ V16  : num  -0.113 -1.505 -0.113 -0.61 -0.113 ...
##  $ V17  : num  -2.372 -0.332 -0.151 -0.564 -0.435 ...
##  $ V18  : num  -0.147 -0.438 -0.264 -0.205 -0.641 ...
##  $ V19  : num  -0.453 -0.04 -0.162 -0.964 -0.502 ...
##  $ V20  : num  2.646 -0.236 0.587 -1.06 0.587 ...
##  $ V21  : num  2.5742 -0.6819 0.0435 -1.3744 0.4309 ...
##  $ V22  : num  -1.785 -0.122 -0.122 2.538 0.21 ...
```

```
##  $ V23  : num  1.026 -0.691 0.106 -1.287 0.597 ...
##  $ V24  : num  -1.6548 -0.6091 -0.0863 2.528 0.4366 ...
##  $ V25  : num  4.258 -0.378 -0.29 -1.015 0.289 ...
##  $ V26  : num  0.417 -0.426 -0.426 2.104 -0.426 ...
##  $ V27  : num  -1.896 -0.327 0.277 -0.689 0.881 ...
##  $ V28  : num  -0.201 -0.197 -0.234 -0.206 -0.197 ...
##  $ V29  : num  -0.22 -0.22 -0.22 -0.22 -0.22 ...
##  $ V30  : num  -0.654 -0.654 1.878 -0.654 0.686 ...
##  $ V31  : num  -0.0475 1.9858 0.3156 -0.338 -0.5558 ...
##  $ V32  : num  5.954 -0.266 -0.159 -0.855 0.318 ...
##  $ V33  : num  3.942 -0.357 0.288 -1.217 0.503 ...
##  $ V34  : num  2.087 -0.043 -0.043 -0.753 0.667 ...
##  $ V35  : num  -1.304 -0.537 0.23 1.764 -0.154 ...
##  $ V36  : num  -0.858 -0.858 -0.167 0.524 0.87 ...
##  $ V37  : num  -0.082 -0.44 -0.798 -0.44 0.635 ...
##  $ V38  : num  -0.283 -1.988 -0.351 0.331 0.4 ...
##  $ V39  : num  0.00404 -1.9999 -0.29284 0.37514 0.52358 ...
##  $ V40  : num  -2.0771 -0.1674 0.0688 -1.1321 -0.1674 ...
##  $ V41  : num  -1.6897 -0.0533 -0.3226 -1.6069 0.1331 ...
##  $ V42  : num  -3.3424 -0.2948 0.056 -0.3167 -0.0756 ...
##  $ V43  : num  -0.7679 -0.4071 0.0311 -0.0463 -0.3298 ...
##  $ V44  : num  -1.9129 -0.1384 -0.0772 -1.3826 -0.016 ...
##  $ V45  : num  2.496 -0.167 0.277 -1.055 0.721 ...
##  $ V46  : num  1.702 -0.588 0.156 -1.386 0.661 ...
##  $ V47  : num  0.396 -0.302 -0.264 -1.477 0.44 ...
##  $ V48  : num  1.278 -0.752 -0.141 -1.472 0.692 ...
##  $ V49  : num  0.4957 -0.6402 0.0143 -1.2478 0.6967 ...
##  $ V50  : num  2.774 -0.917 0.468 -1.094 0.774 ...
##  $ V51  : num  2.624 -0.446 1.215 -1.113 0.388 ...
##  $ V52  : num  3.467 -0.389 1.981 -0.888 0.744 ...
##  $ V53  : num  -0.701 -0.467 0.562 -0.818 0.131 ...
##  $ V54  : num  -1.47905 0.00321 0.24129 -1.17184 0.29505 ...
##  $ V55  : num  -1.578 -0.821 -0.317 2.711 0.188 ...
##  $ V56  : num  -2.051 -0.262 0.159 -0.472 1.001 ...
##  $ V57  : num  -1.584 -0.823 -0.316 2.726 0.191 ...
##  $ V58  : num  -0.496 -0.496 1.541 -0.34 0.757 ...
##  $ V59  : num  0.177 -1.714 -0.217 0.492 0.965 ...
##  $ V60  : num  -0.2865 -0.0658 0.7677 1.4498 0.7267 ...
##  $ V61  : num  0.975 0.271 0.792 0.731 0.471 ...
##  $ V62  : num  0.8778 0.0789 0.6436 1.4732 0.5205 ...
##  $ V63  : num  2.185 0.863 0.382 -1.661 -0.82 ...
##  $ V64  : num  2.62 -0.373 0.548 -0.373 -0.281 ...
##  $ V65  : num  0.112 0.112 -0.451 -2.331 0.3 ...
##  $ V66  : num  4.7508 1.6224 -0.0982 -1.0367 -0.0982 ...
##  $ V67  : num  2.007 1.375 -0.522 1.375 0.111 ...
##  $ V68  : num  3.005 -0.352 0.801 -0.352 -0.101 ...
##  $ V69  : num  -1.8916 -0.7451 -0.0763 2.5033 0.4969 ...
##  $ V70  : num  -1.502 -0.69 -0.342 2.559 0.471 ...
##  $ V71  : num  3.2814 -0.3528 1.0238 -0.3528 -0.0224 ...
##  $ V72  : num  -1.4084 -0.3755 -0.0312 2.8956 0.141 ...
```

```
##  $ V73  : num  -0.9047 -0.0935 -0.0935 2.3398 -0.0935 ...
##  $ V74  : num  -0.184 -0.184 0.238 -0.184 -0.184 ...
##  $ V75  : num  -0.788 0.353 0.607 -1.549 1.241 ...
##  $ V76  : num  -1.8449 0.0744 0.5543 -1.1252 0.3143 ...
##  $ V77  : num  0.155 0.467 3.276 0.779 0.467 ...
##  $ V78  : num  -0.524 0.315 3.829 -0.577 -0.367 ...
##  $ V79  : num  -1.2 -0.239 -0.719 -1.2 0.722 ...
##  $ V80  : num  1.564 -0.24 -0.466 -1.481 0.775 ...
##  $ V81  : num  -1.9174 -0.0101 0.5508 -0.2345 -0.4589 ...
##  $ V82  : num  3.514 -0.525 -0.525 -1.082 -0.246 ...
##  $ V83  : num  -1.469 1.289 1.683 -0.484 1.092 ...
##  $ V84  : num  -1.3869 0.1569 -0.0146 0.843 -0.8723 ...
##  $ V85  : num  -0.201 0.246 -0.201 0.692 0.246 ...
##  $ V86  : num  -0.215 0.201 -0.215 0.617 0.201 ...
##  $ V87  : num  -0.855 0.19 -0.855 -0.855 0.19 ...
##  $ V88  : num  -0.5636 -0.0375 -1.3529 -1.616 0.2256 ...
##  $ V89  : num  -0.325 0.135 -0.786 -0.786 0.135 ...
##  $ V90  : num  -0.6743 0.0792 -1.1767 -1.3022 0.2048 ...
##  $ V91  : num  -0.423 -0.423 -0.423 -0.423 0.998 ...
##  $ V92  : num  1.585 -0.419 -0.419 -0.419 -0.419 ...
##  $ V93  : num  -0.751 -0.284 -1.451 -1.685 -0.05 ...
##  $ V94  : num  -0.178 -0.512 -0.678 -0.845 -0.512 ...
##  $ V95  : num  -0.0431 0.7139 -0.4216 -0.4216 1.8495 ...
##  $ V96  : num  0.25 -0.397 -0.72 -0.72 -0.397 ...
##  $ V97  : num  -0.147 -0.147 -0.147 -0.147 -0.147 ...
##  $ V98  : num  3.451 0.174 -0.482 -0.482 -0.482 ...
##  $ V99  : num  -0.188 -0.188 -1.45 -1.45 1.075 ...
##   [list output truncated]
```

```r
dim(norm_train)
```

```
## [1] 1818  335
```

```r
# normal_test
process_test <- preProcess(bal_test[1:334], method = c("center", "scale"))
norm_test <- predict(process_test, bal_test[1:335])
str(norm_test)
```

```
## 'data.frame':    1878 obs. of  335 variables:
##  $ V1   : num  -1.04 -1.04 -1.04 -1.04 -1.04 ...
##  $ V2   : num  -0.773 1.335 -2.306 -0.198 -1.923 ...
##  $ V3   : num  -0.543 -0.543 -0.543 -0.543 -0.543 ...
##  $ V4   : num  -0.717 1.174 -2.093 -0.374 1.174 ...
##  $ V5   : num  0.994 -1.125 2.446 0.51 -1.125 ...
##  $ V6   : num  -0.505 -0.505 -0.505 -0.505 -0.505 ...
##  $ V7   : num  -0.796 -0.796 1.255 -0.796 -0.796 ...
##  $ V8   : num  0.689 0.689 -1.45 0.689 -1.45 ...
##  $ V9   : num  -0.61 -0.61 -0.61 -0.61 -0.61 ...
##  $ V10  : num  -0.993 -0.993 -0.993 1.006 -0.993 ...
##  $ V11  : num  0.542 -0.4098 -0.4551 0.0435 0.2248 ...
##  $ V12  : num  -1.009 1.612 -1.009 -0.135 -0.135 ...
```

```
##  $ V13  : num  0.601 -1.662 0.601 0.601 -1.662 ...
##  $ V14  : num  -0.788 -1.028 0.384 -0.217 2.127 ...
##  $ V15  : num  -1.04 -1.04 -1.04 -1.04 -1.04 ...
##  $ V16  : num  0.777 -1.035 -0.129 -0.129 0.958 ...
##  $ V17  : num  1.871 -0.885 -0.362 -0.932 0.113 ...
##  $ V18  : num  1.667 -0.749 -0.393 -1.054 -0.113 ...
##  $ V19  : num  2.344 -1.195 -0.815 -1.338 -0.197 ...
##  $ V20  : num  0.182 -1.054 -0.23 -0.23 0.595 ...
##  $ V21  : num  -0.169 -1.08 -0.731 -0.602 0.447 ...
##  $ V22  : num  -1.359 0.584 0.584 1.232 0.26 ...
##  $ V23  : num  -0.241 -1.045 -0.699 -0.441 0.646 ...
##  $ V24  : num  -1.364 0.754 0.225 1.284 0.754 ...
##  $ V25  : num  -0.114 -0.958 -0.985 -0.532 -0.308 ...
##  $ V26  : num  -1.279 0.499 -0.39 1.388 -0.39 ...
##  $ V27  : num  -0.0894 -0.327 0.6235 -0.4459 1.0988 ...
##  $ V28  : num  -0.234 -0.217 -0.243 -0.206 -0.234 ...
##  $ V29  : num  -0.226 -0.226 -0.226 -0.226 -0.226 ...
##  $ V30  : num  -1.21 -0.445 -1.057 -0.445 0.626 ...
##  $ V31  : num  0.596 -0.615 3.019 -1.257 -1.257 ...
##  $ V32  : num  -0.324 -0.728 -0.847 -0.439 -0.214 ...
##  $ V33  : num  -0.327 -1.242 -0.556 -0.327 0.13 ...
##  $ V34  : num  -0.7336 -0.7336 0.0143 -0.7336 0.0143 ...
##  $ V35  : num  -0.0844 0.3198 -0.0844 -0.4885 -0.4885 ...
##  $ V36  : num  -1.303 -0.215 -0.578 0.147 0.147 ...
##  $ V37  : num  0.239 -0.134 -1.251 -0.134 0.612 ...
##  $ V38  : num  -0.063 0.162 -0.35 0.287 0.274 ...
##  $ V39  : num  -0.0722 0.153 -0.3601 0.2656 0.2656 ...
##  $ V40  : num  2.199 -1.244 -0.571 -1.186 0.122 ...
##  $ V41  : num  1.874 -1.112 -0.991 -1.031 0.401 ...
##  $ V42  : num  1.661 -0.73 -0.26 -0.833 0.23 ...
##  $ V43  : num  1.5728 -0.6103 -0.3316 -1.0283 -0.0297 ...
##  $ V44  : num  2.079 -1.192 -0.717 -1.113 0.255 ...
##  $ V45  : num  0.298 -1.038 -0.148 -0.148 0.298 ...
##  $ V46  : num  -0.322 -0.944 -0.67 -0.469 0.485 ...
##  $ V47  : num  0.1925 -0.9618 -0.8432 -0.0633 0.7479 ...
##  $ V48  : num  -0.495 -0.862 -0.776 -0.362 0.74 ...
##  $ V49  : num  -0.843 -0.656 -0.559 -0.381 0.847 ...
##  $ V50  : num  -0.626 -0.881 -0.636 -0.365 0.331 ...
##  $ V51  : num  -0.1897 -1.4472 -0.0768 -0.3479 0.134 ...
##  $ V52  : num  0.786 -1.638 -0.567 -0.326 -0.452 ...
##  $ V53  : num  -0.389 -1.523 -0.19 -0.103 0.425 ...
##  $ V54  : num  -0.101 -0.772 -0.86 -0.581 0.945 ...
##  $ V55  : num  -1.008 1.217 0.228 1.217 0.723 ...
##  $ V56  : num  -0.051 -0.367 0.581 -0.262 1.213 ...
##  $ V57  : num  -1.015 0.985 0.235 1.235 0.735 ...
##  $ V58  : num  -1.115 -0.306 -0.953 -0.306 1.15 ...
##  $ V59  : num  -0.0178 0.1239 -0.3011 0.2002 0.2002 ...
##  $ V60  : num  -0.684 -1.249 -0.806 1.658 1.508 ...
##  $ V61  : num  -0.478 -1.292 -0.792 1.201 0.894 ...
##  $ V62  : num  -0.7 -1.216 -0.745 1.582 1.054 ...
```

```
##  $ V63  : num  1.188 -0.79 0.141 -0.907 -1.256 ...
##  $ V64  : num  -0.571 -0.954 0.099 -0.379 -1.097 ...
##  $ V65  : num  1.354 -1.029 -1.029 -1.029 0.254 ...
##  $ V66  : num  0.725 -0.544 0.161 -0.403 -0.685 ...
##  $ V67  : num  -0.429 -0.429 0.102 1.164 -0.429 ...
##  $ V68  : num  -0.522 -0.883 -0.574 -0.265 -0.985 ...
##  $ V69  : num  -1.653 1.184 0.427 1.279 0.9 ...
##  $ V70  : num  -1.615 0.872 0.279 1.227 0.753 ...
##  $ V71  : num  -0.379 -0.776 -0.606 -0.323 -0.833 ...
##  $ V72  : num  -1.276 0.946 0.39 1.131 0.02 ...
##  $ V73  : num  -0.969 0.255 0.56 0.866 -0.357 ...
##  $ V74  : num  -0.192 -0.192 -0.192 -0.192 0.871 ...
##  $ V75  : num  -0.441 -0.929 0.413 0.291 0.413 ...
##  $ V76  : num  0.887 -0.687 -0.95 -0.687 1.412 ...
##  $ V77  : num  -0.142 -0.952 -0.142 0.667 0.128 ...
##  $ V78  : num  -0.575 0.923 3.198 -0.187 -0.298 ...
##  $ V79  : num  0.709 -2.187 2.157 0.709 1.674 ...
##  $ V80  : num  0.181 -1.37 -0.595 -0.373 0.402 ...
##  $ V81  : num  0.00323 1.41289 -0.1052 -0.53894 -0.32207 ...
##  $ V82  : num  0.17 -0.8 -0.107 -0.662 0.17 ...
##  $ V83  : num  0.289 -0.742 -0.124 1.526 0.289 ...
##  $ V84  : num  -0.5 1.242 1.067 0.371 -0.675 ...
##  $ V85  : num  -0.154 0.484 -0.154 0.165 -0.154 ...
##  $ V86  : num  -0.192 0.517 -0.192 0.163 -0.192 ...
##  $ V87  : num  0.0865 -0.6796 0.0865 0.0865 0.0865 ...
##  $ V88  : num  1.35 -1.18 -1.18 -0.62 1.07 ...
##  $ V89  : num  0.125 5.478 -0.321 -0.321 0.125 ...
##  $ V90  : num  0.6591 -1.0954 -0.7195 -0.3435 0.0325 ...
##  $ V91  : num  1.212 -0.436 -0.436 -0.436 1.212 ...
##  $ V92  : num  1.425 -0.419 -0.419 -0.419 -0.419 ...
##  $ V93  : num  0.668 -1.263 -0.539 -0.78 0.427 ...
##  $ V94  : num  0.374 -0.509 -0.332 -0.332 0.374 ...
##  $ V95  : num  -0.0565 -0.4449 -0.4449 -0.0565 -0.4449 ...
##  $ V96  : num  0.255 -0.735 -0.405 -0.735 0.255 ...
##  $ V97  : num  -0.184 -0.184 -0.184 0.664 -0.184 ...
##  $ V98  : num  0.213 -0.498 -0.498 0.213 -0.498 ...
##  $ V99  : num  0.999 -1.492 -0.247 -0.247 0.999 ...
##   [list output truncated]
```

```r
dim(norm_test)
```

```
## [1] 1878  335
```

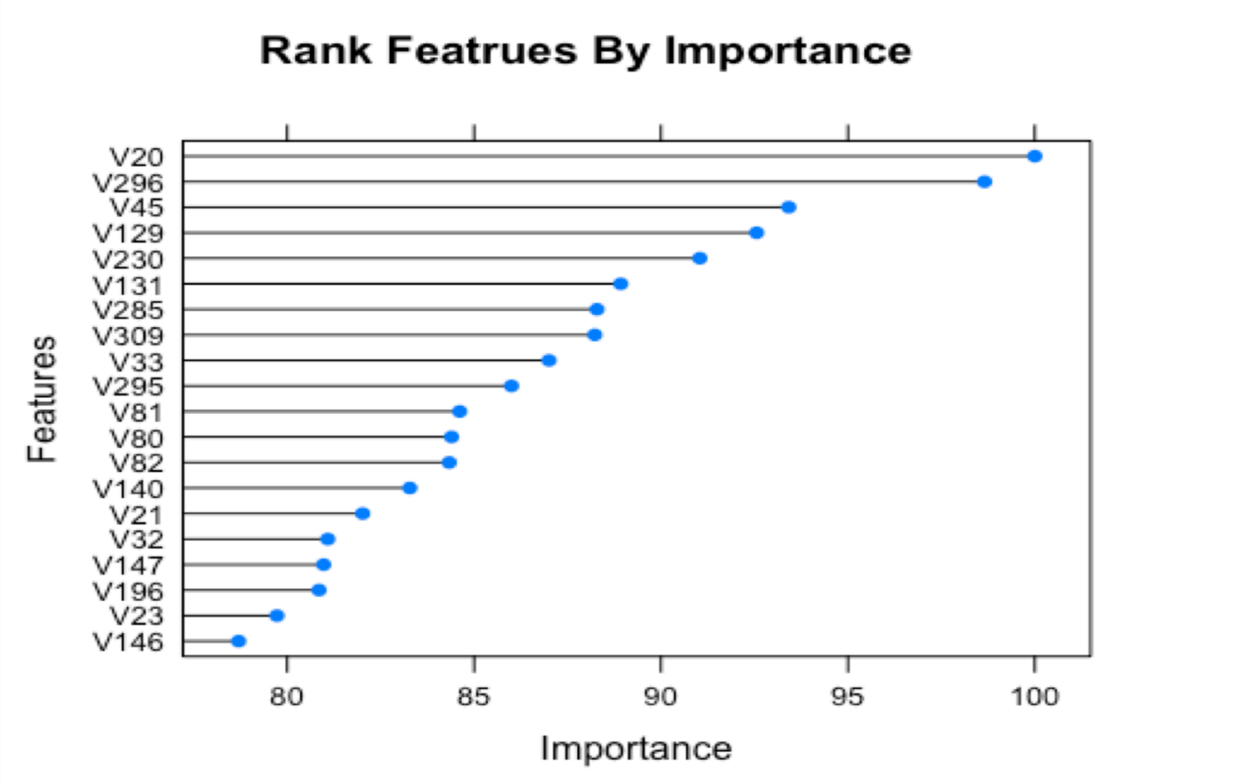
Feature selection

I selected feature by rank by importance method. library(caret) has the functions for this steps. It used all feature ans rank them by importance and gave me 20 top important variables. Thus, I used all these variables for my final model.

```
library(mlbench)
control1 <- trainControl(method = "cv", number = 10)
model <- train(Class~., data = norm_train, method = "lvq", preProcess =
'scale',trControl = control1)
importance <- varImp(model)
print(importance)
```

```
## ROC curve variable importance
##
##   only 20 most important variables shown (out of 334)
##
##       Importance
## V20      100.00
## V296      98.65
## V45       93.42
## V129      92.56
## V230      91.04
## V131      88.92
## V285      88.29
## V309      88.23
## V33       87.00
## V295      86.00
## V81       84.62
## V80       84.40
## V82       84.34
## V140      83.28
## V21       82.02
## V32       81.09
## V147      80.98
## V196      80.85
## V23       79.73
## V146      78.70
```

```
plot(importance, top = 20, main = "Rank Featrues By Importance", ylab =
"Features")
```




```
# feature_selection_train
sel_train <- norm_train[c('V20','V296','V45','V129','V230','V131','V285',

                          'V309','V33','V295','V81','V80','V82','V140',

                          'V21','V32','V147','V196','V23','V147',

                          'Class')]
dim(sel_train)

## [1] 1818   21

#feature_selection_test
sel_test <- norm_test[c('V20','V296','V45','V129','V230','V131','V285',

                        'V309','V33','V295','V81','V80','V82','V140',

                        'V21','V32','V147','V196','V23','V147',

                        'Class')]
dim(sel_test)

## [1] 1878   21
```

Thus, my final model for both Dection tree and Random forest are comination of the columns of the top 20 important variables using library(caret)

<<ANALYSIS & CODES for Problem 2>>

```r
library(caret)
library(ggplot2)
library(lattice)
library(mlbench)

# load the dataset
setwd('/Users/jaygkay/Desktop/CSC529/HW3')
x_train = read.table('train10000.csv', header=F, sep = ',')
x_test = read.table("test10000.csv", header=F, sep = ',')
y_train = read.table('train10000_Label.csv', header=F, sep = ',')
y_test = read.table("test10000_label.csv", header=F, sep = ',')
# combine with x and y variables
colnames(y_train) <- "Class"
colnames(y_test) <- "Class"



(((Balanced, normalized, feature_selection parts are shown above)))
```

2-1. Decision Tree with Raw data

```r
#### Train
raw_dt <- train(Class~., data = train, method = 'rpart',
                parms = list(split = "infomation"), tuneLength = 10)
raw_dt

## CART
##
## 10000 samples
##   334 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, 10000, ...
## Resampling results across tuning parameters:
##
##   cp            Accuracy   Kappa
##   0.0003667033  0.8622374  0.03843126
##   0.0005500550  0.8646833  0.03946004
##   0.0007334067  0.8663081  0.03961708
##   0.0009900990  0.8681146  0.04025329
##   0.0011001100  0.8717984  0.04040332
##   0.0017287443  0.8822992  0.04188736
##   0.0017601760  0.8822992  0.04188736
##   0.0033003300  0.8969292  0.04332955
##   0.0036670334  0.8985591  0.04347630
##   0.0044004400  0.9027842  0.04112559
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.00440044.
```

```
#### confusion on train
rawdt_train_pred <- predict(raw_dt, newdata = train)
confusionMatrix(rawdt_train_pred, train$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9091  909
##          1    0    0
##
##                Accuracy : 0.9091
##                  95% CI : (0.9033, 0.9147)
##     No Information Rate : 0.9091
##     P-Value [Acc > NIR] : 0.5088
##
##                   Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9091
##          Neg Pred Value :    NaN
##              Prevalence : 0.9091
##          Detection Rate : 0.9091
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
rawdt_test_pred <- predict(raw_dt, newdata = test)
confusionMatrix(rawdt_test_pred, test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9061  939
##          1    0    0
##
##                Accuracy : 0.9061
##                  95% CI : (0.9002, 0.9117)
##     No Information Rate : 0.9061
##     P-Value [Acc > NIR] : 0.5087
##
```

```
##                  Kappa : 0
##  Mcnemar's Test P-Value : <2e-16
##
##            Sensitivity : 1.0000
##            Specificity : 0.0000
##         Pos Pred Value : 0.9061
##         Neg Pred Value :    NaN
##             Prevalence : 0.9061
##         Detection Rate : 0.9061
##   Detection Prevalence : 1.0000
##      Balanced Accuracy : 0.5000
##
##       'Positive' Class : 0
##
```

2-1. Decision Tree with balanced data

```
#### Train
bal_dt <- train(Class~., data = bal_train, method = 'rpart',
                parms = list(split = "infomation"), tuneLength = 10)
bal_dt
```

```
## CART
##
## 1818 samples
##  334 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.005867253  0.5472579  0.09496204
##   0.006050605  0.5479107  0.09630828
##   0.007334067  0.5507427  0.10207364
##   0.007700770  0.5502637  0.10128148
##   0.008250825  0.5516999  0.10404660
##   0.008800880  0.5509779  0.10257545
##   0.009900990  0.5541839  0.10929261
##   0.022002200  0.5621550  0.12436483
##   0.041804180  0.5590969  0.11900942
##   0.143014301  0.5157921  0.03956831
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.0220022.
```

```
#### confusion on train
baldt_train_pred <- predict(bal_dt, newdata = bal_train)
confusionMatrix(baldt_train_pred, bal_train$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 696 528
##          1 213 381
##
##                Accuracy : 0.5924
##                  95% CI : (0.5694, 0.6151)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.65e-15
##
##                   Kappa : 0.1848
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7657
##             Specificity : 0.4191
##          Pos Pred Value : 0.5686
##          Neg Pred Value : 0.6414
##              Prevalence : 0.5000
##          Detection Rate : 0.3828
##    Detection Prevalence : 0.6733
##       Balanced Accuracy : 0.5924
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
baldt_test_pred <- predict(bal_dt, newdata = bal_test)
confusionMatrix(baldt_test_pred, bal_test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 715 536
##          1 224 403
##
##                Accuracy : 0.5953
##                  95% CI : (0.5727, 0.6176)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1906
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7614
##             Specificity : 0.4292
##          Pos Pred Value : 0.5715
##          Neg Pred Value : 0.6427
```

```
##               Prevalence : 0.5000
##           Detection Rate : 0.3807
##     Detection Prevalence : 0.6661
##        Balanced Accuracy : 0.5953
##
##         'Positive' Class : 0
##
```

2-1. Decision Tree with normalized data

```
#### Train
norm_dt <- train(Class~., data = norm_train, method = 'rpart',
                 parms = list(split = "infomation"), tuneLength = 10)
norm_dt

## CART
##
## 1818 samples
##  334 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.005867253  0.5391503  0.07858453
##   0.006050605  0.5396896  0.07945514
##   0.007334067  0.5472859  0.09548960
##   0.007700770  0.5472360  0.09548685
##   0.008250825  0.5486860  0.09806169
##   0.008800880  0.5487285  0.09810150
##   0.009900990  0.5498931  0.10102989
##   0.022002200  0.5597186  0.11968605
##   0.041804180  0.5544749  0.10847708
##   0.143014301  0.5232128  0.05302736
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.0220022.

#### confusion on train
normdt_train_pred <- predict(norm_dt, newdata = norm_train)
confusionMatrix(normdt_train_pred, norm_train$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 696 528
##          1 213 381
```

```
##
##                Accuracy : 0.5924
##                  95% CI : (0.5694, 0.6151)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.65e-15
##
##                   Kappa : 0.1848
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7657
##             Specificity : 0.4191
##          Pos Pred Value : 0.5686
##          Neg Pred Value : 0.6414
##              Prevalence : 0.5000
##          Detection Rate : 0.3828
##    Detection Prevalence : 0.6733
##       Balanced Accuracy : 0.5924
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
normdt_test_pred <- predict(norm_dt, newdata = norm_test)
confusionMatrix(normdt_test_pred, norm_test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 715 536
##          1 224 403
##
##                Accuracy : 0.5953
##                  95% CI : (0.5727, 0.6176)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1906
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7614
##             Specificity : 0.4292
##          Pos Pred Value : 0.5715
##          Neg Pred Value : 0.6427
##              Prevalence : 0.5000
##          Detection Rate : 0.3807
##    Detection Prevalence : 0.6661
##       Balanced Accuracy : 0.5953
##
```

```
##        'Positive' Class : 0
##
```

2-1. Decision Tree with feature-selected data

```
#### Train
sel_dt <- train(Class~., data = sel_train, method = 'rpart',
                parms = list(split = "infomation"), tuneLength = 10)
sel_dt

## CART
##
## 1818 samples
##   20 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   cp           Accuracy   Kappa
##   0.003667033  0.5455982  0.09224139
##   0.003850385  0.5457188  0.09249184
##   0.004400440  0.5481799  0.09685824
##   0.005500550  0.5534287  0.10725610
##   0.006600660  0.5587382  0.11858099
##   0.007700770  0.5563954  0.11383659
##   0.008525853  0.5561206  0.11415733
##   0.016501650  0.5593424  0.12142227
##   0.041804180  0.5542303  0.11354194
##   0.143014301  0.5085772  0.03524018
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01650165.

#### confusion on train
seldt_train_pred <- predict(sel_dt, newdata = sel_train)
confusionMatrix(seldt_train_pred, sel_train$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 696 528
##          1 213 381
##
##                Accuracy : 0.5924
##                  95% CI : (0.5694, 0.6151)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.65e-15
```

```
##
##                   Kappa : 0.1848
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7657
##             Specificity : 0.4191
##          Pos Pred Value : 0.5686
##          Neg Pred Value : 0.6414
##              Prevalence : 0.5000
##          Detection Rate : 0.3828
##    Detection Prevalence : 0.6733
##       Balanced Accuracy : 0.5924
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
seldt_test_pred <- predict(sel_dt, newdata = sel_test)
confusionMatrix(seldt_test_pred, sel_test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 715 536
##          1 224 403
##
##                Accuracy : 0.5953
##                  95% CI : (0.5727, 0.6176)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1906
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7614
##             Specificity : 0.4292
##          Pos Pred Value : 0.5715
##          Neg Pred Value : 0.6427
##              Prevalence : 0.5000
##          Detection Rate : 0.3807
##    Detection Prevalence : 0.6661
##       Balanced Accuracy : 0.5953
##
##        'Positive' Class : 0
##
```

2-2. Random Forest with Raw data

```
raw_rf <- train(Class~., data = train, method = 'rf', ntree = 20)
raw_rf
```

```
## Random Forest
##
## 10000 samples
##   334 predictor
##     2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 10000, 10000, 10000, 10000, 10000, 10000, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   0.8888814  -0.004704299
##   168   0.8903263   0.021878977
##   334   0.8898402   0.018565556
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 168.
```

```
#### confusion on train
rawrf_train_pred <- predict(raw_rf, newdata = train)
confusionMatrix(rawrf_train_pred, train$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 9087   73
##          1    4  836
##
##                Accuracy : 0.9923
##                  95% CI : (0.9904, 0.9939)
##     No Information Rate : 0.9091
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9518
##  Mcnemar's Test P-Value : 9.239e-15
##
##             Sensitivity : 0.9996
##             Specificity : 0.9197
##          Pos Pred Value : 0.9920
##          Neg Pred Value : 0.9952
##              Prevalence : 0.9091
##          Detection Rate : 0.9087
##    Detection Prevalence : 0.9160
##       Balanced Accuracy : 0.9596
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
rawrf_test_pred <- predict(raw_rf, newdata = test)
confusionMatrix(rawrf_test_pred, test$Class)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction    0    1
##          0 8863  902
##          1  198   37
##
##                Accuracy : 0.89
##                  95% CI : (0.8837, 0.8961)
##     No Information Rate : 0.9061
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0264
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9781
##             Specificity : 0.0394
##          Pos Pred Value : 0.9076
##          Neg Pred Value : 0.1574
##              Prevalence : 0.9061
##          Detection Rate : 0.8863
##    Detection Prevalence : 0.9765
##       Balanced Accuracy : 0.5088
##
##        'Positive' Class : 0
##
```

2-2. Random Forest with balanced data

```
#### Train
bal_rf <- train(Class~., data = bal_train, method = 'rf', ntree = 20)
bal_rf
```

```
## Random Forest
##
## 1818 samples
##  334 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   0.5481957  0.09726745
##   168   0.5478148  0.09654157
```

```
##   334   0.5480263  0.09698425
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
#### confusion on train
balrf_train_pred <- predict(bal_rf, newdata = bal_train)
confusionMatrix(balrf_train_pred, bal_train$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 892  45
##          1  17 864
##
##                Accuracy : 0.9659
##                  95% CI : (0.9565, 0.9738)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9318
##  Mcnemar's Test P-Value : 0.0006058
##
##             Sensitivity : 0.9813
##             Specificity : 0.9505
##          Pos Pred Value : 0.9520
##          Neg Pred Value : 0.9807
##              Prevalence : 0.5000
##          Detection Rate : 0.4906
##    Detection Prevalence : 0.5154
##       Balanced Accuracy : 0.9659
##
##        'Positive' Class : 0
##
```

```
#### confusion on test
balrf_test_pred <- predict(bal_rf, newdata = bal_test)
confusionMatrix(balrf_test_pred, bal_test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 549 402
##          1 390 537
##
##                Accuracy : 0.5783
##                  95% CI : (0.5556, 0.6007)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 6.234e-12
```

```
##
##                   Kappa : 0.1565
##  Mcnemar's Test P-Value : 0.6959
##
##             Sensitivity : 0.5847
##             Specificity : 0.5719
##          Pos Pred Value : 0.5773
##          Neg Pred Value : 0.5793
##              Prevalence : 0.5000
##          Detection Rate : 0.2923
##    Detection Prevalence : 0.5064
##       Balanced Accuracy : 0.5783
##
##        'Positive' Class : 0
##
```

2-2. Random Forest with normalized data

```
#### Train
norm_rf <- train(Class~., data = norm_train, method = 'rf', ntree = 20)
norm_rf
```

```
## Random Forest
##
## 1818 samples
##  334 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##     2   0.5469271  0.09547107
##   168   0.5515742  0.10453057
##   334   0.5464952  0.09438067
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 168.
```

```
#### confusion on train
normrf_train_pred <- predict(norm_rf, newdata = norm_train)
confusionMatrix(normrf_train_pred, norm_train$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 901  17
##          1   8 892
```

```
##
##                Accuracy : 0.9862
##                  95% CI : (0.9798, 0.9911)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9725
##  Mcnemar's Test P-Value : 0.1096
##
##             Sensitivity : 0.9912
##             Specificity : 0.9813
##          Pos Pred Value : 0.9815
##          Neg Pred Value : 0.9911
##              Prevalence : 0.5000
##          Detection Rate : 0.4956
##    Detection Prevalence : 0.5050
##       Balanced Accuracy : 0.9862
##
##        'Positive' Class : 0
##

#### confusion on test
normrf_test_pred <- predict(norm_rf, newdata = norm_test)
confusionMatrix(normrf_test_pred, norm_test$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 506 370
##          1 433 569
##
##                Accuracy : 0.5724
##                  95% CI : (0.5497, 0.5949)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.875e-10
##
##                   Kappa : 0.1448
##  Mcnemar's Test P-Value : 0.02867
##
##             Sensitivity : 0.5389
##             Specificity : 0.6060
##          Pos Pred Value : 0.5776
##          Neg Pred Value : 0.5679
##              Prevalence : 0.5000
##          Detection Rate : 0.2694
##    Detection Prevalence : 0.4665
##       Balanced Accuracy : 0.5724
##
```

```
##        'Positive' Class : 0
##
```

2-2. Random Forest with feature-selected data

```
#### Train
sel_rf <- train(Class~., data = sel_train, method = 'rf', ntree = 20)
sel_rf

## Random Forest
##
## 1818 samples
##   20 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1818, 1818, 1818, 1818, 1818, 1818, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.5544335  0.10879439
##   11    0.5511317  0.10248864
##   20    0.5454218  0.09091433
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.

#### confusion on train
selrf_train_pred <- predict(sel_rf, newdata = sel_train)
confusionMatrix(selrf_train_pred, sel_train$Class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 901  26
##          1   8 883
##
##                Accuracy : 0.9813
##                  95% CI : (0.974, 0.987)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9626
##  Mcnemar's Test P-Value : 0.003551
##
##             Sensitivity : 0.9912
##             Specificity : 0.9714
##          Pos Pred Value : 0.9720
##          Neg Pred Value : 0.9910
```

```
##               Prevalence : 0.5000
##           Detection Rate : 0.4956
##     Detection Prevalence : 0.5099
##        Balanced Accuracy : 0.9813
##
##         'Positive' Class : 0
##
```

```r
#### confusion on test
selrf_test_pred <- predict(sel_rf, newdata = sel_test)
confusionMatrix(selrf_test_pred, sel_test$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 530 397
##          1 409 542
##
##                Accuracy : 0.5708
##                  95% CI : (0.5481, 0.5933)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 4.535e-10
##
##                   Kappa : 0.1416
##  Mcnemar's Test P-Value : 0.6984
##
##             Sensitivity : 0.5644
##             Specificity : 0.5772
##          Pos Pred Value : 0.5717
##          Neg Pred Value : 0.5699
##              Prevalence : 0.5000
##          Detection Rate : 0.2822
##    Detection Prevalence : 0.4936
##       Balanced Accuracy : 0.5708
##
##        'Positive' Class : 0
##
```