INFO411 Assignment 2 | Heng Li En Shaun 6858144

1. Reading in creditworthiness.csv

```
> cw <- read.csv("creditworthiness.csv")
> # cw.k <- cw %>% filter(credit.rating > 0)
> cw.k <- subset(cw, credit.rating > 0)
> cw.uk <- subset(cw, credit.rating == 0)
> cw.train <- cw.k[1:(nrow(cw.k)/2), ]
> cw.test <- cw.k[-(1:(nrow(cw.k)/2)), ]
```

2. Decision tree

Reporting the tree

```
> tree.cw.train = tree(as.factor(credit.rating)~., data=cw.train)
> tree.cw.train
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 981 2021.000 2 ( 0.23038 0.51274 0.25688 )
   2) functionary < 0.5 709 1344.000 2 ( 0.13540 0.57546 0.28914 )
     4) FI3O.credit.score < 0.5 58   61.720 3 ( 0.00000 0.22414 0.77586 ) *
     5) FI3O.credit.score > 0.5 651 1211.000 2 ( 0.14747 0.60676 0.24578 )
      10) re.balanced..paid.back..a.recently.overdrawn.current.acount < 0.5
 57   98.140 3 ( 0.07018 0.33333 0.59649 ) *
      11) re.balanced..paid.back..a.recently.overdrawn.current.acount > 0.5
 594 1078.000 2 ( 0.15488 0.63300 0.21212 ) *
   3) functionary > 0.5 272  556.800 1 ( 0.47794 0.34926 0.17279 )
     6) re.balanced..paid.back..a.recently.overdrawn.current.acount < 0.5 11
   12.890 3 ( 0.00000 0.27273 0.72727 ) *
     7) re.balanced..paid.back..a.recently.overdrawn.current.acount > 0.5 26
1  521.400 1 ( 0.49808 0.35249 0.14943 )
      14) FI3O.credit.score < 0.5 9    9.535 3 ( 0.00000 0.22222 0.77778 ) *
      15) FI3O.credit.score > 0.5 252  489.500 1 ( 0.51587 0.35714 0.12698 )
 *
```

Generating the median customer

```
> median.cust = data.frame()
> median.cust.data =
+   c(0,1,1,0,3,0,3,3,0,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
+     3,3,3,3,3,3,3,3,3,3,3,3)
> median.cust = rbind(median.cust, median.cust.data)
> colnames(median.cust) = names(cw.k)[-46]
```

Predicting the median customer's credit rating

```
> cust.pred = predict(tree.cw.train, median.cust, type="class")
> cust.pred
[1] 2
Levels: 1 2 3
```

Confusion Matrix for predicting

```
> tree.pred = predict(tree.cw.train, cw.test, type="class")
> confusion = with(cw.test, table(tree.pred, credit.rating))
> confusion
         credit.rating
tree.pred   1   2   3
        1 162  85  37
        2  90 361 143
        3   5  21  77
```

Accuracy rate for this decision tree

```
> tree.acc = sum(diag(confusion))/sum(confusion)
> tree.acc
[1] 0.6116208
```

Entropy gains for the first split:

Before split

```
> # count of all classes in credit.rating
> before.count = table(cw.train$credit.rating)
> # probability of each class
> before.prob = before.count/sum(before.count)
> # entropy before split
> before.ent = -sum(before.prob * log2(before.prob))
> before.ent
[1] 1.485749
```

Taking functionary == 0

```
> # functionary == 0
> func0.count = table(cw.train$credit.rating[cw.train$functionary == 0])
> func0.prob = func0.count/sum(func0.count)
> func0.ent = -sum(func0.prob * log2(func0.prob))
> func0.ent
[1] 1.366963
```

functionary == 1

```
> # functionary == 1
> func1.count = table(cw.train$credit.rating[cw.train$functionary == 1])
> func1.prob = func1.count/sum(func1.count)
> func1.ent = -sum(func1.prob * log2(func1.prob))
> func1.ent
[1] 1.476765
```

Final entropy

```
> ent = (before.ent - (func0.ent * sum(func0.count) +
+                        func1.ent * sum(func1.count)) /
+          sum(sum(func0.count) + sum(func1.count)))
> ent
[1] 0.0883414
```

Random Forest model

```
> rf.cw.train = randomForest(as.factor(credit.rating)~., data = cw.train)
> rf.cw.train

Call:
 randomForest(formula = as.factor(credit.rating) ~ ., data = cw.train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 6

        OOB estimate of  error rate: 43.02%
Confusion matrix:
   1   2  3 class.error
1 57 169  0   0.7477876
2 39 441 23   0.1232604
3 18 173 61   0.7579365
> rf.pred = predict(rf.cw.train, cw.test[,-46])
> confusion.rf = with(cw.test, table(rf.pred, credit.rating))
> confusion.rf
       credit.rating
rf.pred   1   2   3
      1  59  36  15
      2 196 419 195
      3   2  12  47
> rf.acc = sum(diag(confusion.rf))/sum(confusion.rf)
> rf.acc
[1] 0.5351682
```

Attempting at tuning using different parameters

```
> rftuning.cw.train = randomForest(as.factor(credit.rating)~., data = cw.train, mtry
+                                  = 15, ntree=500, stepFactor=2, improve=0.2)
> rftuning.pred = predict(rftuning.cw.train, cw.test[,-46])
> confusion.rftuning = with(cw.test, table(rftuning.pred, credit.rating))
> confusion.rftuning
             credit.rating
rftuning.pred   1   2   3
            1 116  64  28
            2 136 384 159
            3   5  19  70
> acc = sum(diag(confusion.rftuning))/sum(confusion.rftuning)
> acc
[1] 0.5810398
```

Accuracy did improve by about 5%

3. Using svm() from e1071 package

```
> svmfit = svm(as.factor(credit.rating)~., data = cw.train, kernel = "radial")
> svmfit

Call:
svm(formula = as.factor(credit.rating) ~ ., data = cw.train, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  937
> svm.med.pred = predict(svmfit, median.cust, decision.values = TRUE)
> svm.med.pred
1
2
attr(,"decision.values")
       2/1      2/3         1/3
1 1.021296 1.511396 -0.04938262
Levels: 1 2 3

> svm.pred = predict(svmfit, cw.test[,-46])
>
> confusion.svm = with(cw.test, table(svm.pred, credit.rating))
> confusion.svm
        credit.rating
svm.pred   1   2   3
       1 109  56  22
       2 143 393 162
       3   5  18  73
>
> acc = sum(diag(confusion.svm))/sum(confusion.svm)
> acc
[1] 0.5861366
```

Accuracy of 58%

```
> summary(tune.svm(as.factor(credit.rating) ~ ., data = cw.train,
+                  kernel = "radial",cost = 10^c(0:2), gamma = 10^c(-4:-1)))

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
 0.001   10

- best performance: 0.3945475
```

Attempting to use the above tuning to find a better accuracy

```
> svmtuning = svm(as.factor(credit.rating) ~ ., data = cw.train, kernel = "radial",
+                cost=10,
+                gamma = 0.001)
>
> svmtuning.pred = predict(svmtuning, cw.test[,-46])
>
> confusion.svmtuning = with(cw.test, table(svmtuning.pred, credit.rating))
> confusion.svmtuning
              credit.rating
svmtuning.pred   1   2   3
             1 160  87  39
             2  92 361 147
             3   5  19  71
>
> acc = sum(diag(confusion.svmtuning))/sum(confusion.svmtuning)
> acc
[1] 0.6034659
```

The accuracy did indeed improve by about 2%

4. Naïve Bayes

Predict median customer and probabilities

```
> nb = naiveBayes(as.factor(credit.rating)~. ,data=cw.train)
> nb.class.pred = predict(nb, median.cust, type='class')
> nb.class.pred
[1] 1
Levels: 1 2 3
> nb.class.raw = predict(nb, median.cust, type='raw')
> nb.class.raw
             1          2            3
[1,] 0.9850729 0.01393277 0.0009942948
```

NB confusion matrix & accuracy

```
> nb.pred = predict(nb, cw.test[,-46])
> confusion.nb = with(cw.test, table(nb.pred, credit.rating))
> confusion.nb
       credit.rating
nb.pred   1   2   3
      1 252 439 173
      2   0   4   6
      3   5  24  78
> acc = sum(diag(confusion.nb))/sum(confusion.nb)
> acc
[1] 0.3404689
```

```
> nb

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        1         2         3 
0.2303772 0.5127421 0.2568807 

Conditional probabilities:
   functionary
Y        [,1]      [,2]
  1 0.5752212 0.4954066
  2 0.1888668 0.3917924
  3 0.1865079 0.3902912
```

In this dataset that contains 45 variables, excluding credit rating:

$P(Y = 1 | X_1 = 0, \ldots , X_{45} = 3) \propto P(Y = 1) \ldots P(X_1 = 0, \ldots , X_{45} = 5)$

$= ((0.2303772) \times \phi((0 - 0.5752212)/0.4954066)/0.4954066 \times \ldots = m$

Using the same formula for Y = 2 and Y = 3, variables n and o are calculated, and obtaining the probabilities with m/(m + n + o) along with the other 3 should attain a total of 1 i.e., 100%.

5. Looking at confusion matrices again

a). It appears that decision tree is the best classifier since it has the highest accuracy of 61% among the others.

b). All classifiers seem to predict more ratings of 2 when the actual rating is 3, otherwise it would be pretty accurate.

6. Logistic Regression

```
> # Logistic Regression
> glm.fit <- glm((credit.rating==1)~., data = cw.train, family = binomial)
> summary(glm.fit)

Call:
glm(formula = (credit.rating == 1) ~ ., family = binomial, data = cw.train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.00215  -0.65353  -0.42668  -0.00012   2.70789

Coefficients:
                                                             Estimate Std. Error z
 value Pr(>|z|)
(Intercept)                                                -17.551605 429.995589  -
0.041  0.96744
functionary                                                  1.740533   0.183036
 9.509  < 2e-16 ***
re.balanced..paid.back..a.recently.overdrawn.current.acount  1.501222   0.550965
 2.725  0.00644 **
FI30.credit.score                                           16.502759 429.993845
 0.038  0.96939
```

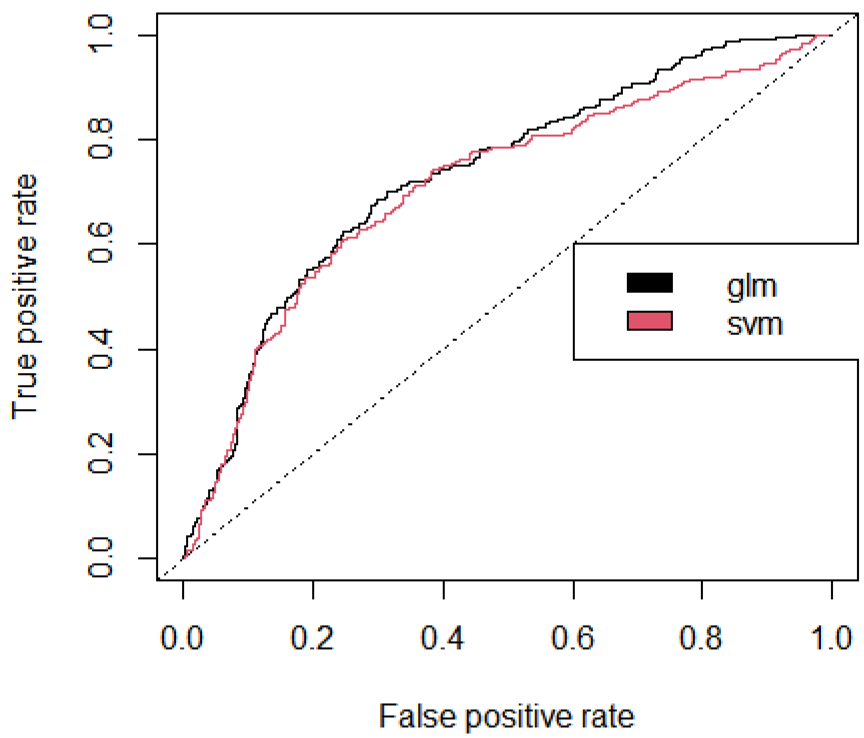
FICO credit score having an estimate and standard error that high seems very suspicious as logically thinking, credit score should be directly linked to credit rating.

Recently paid back data also has the closest standard error to 1, thus possibly making it the most significant attribute.

svm model

Using the same “best” parameters from earlier in question 3:

```
> # Predict the values on test set[SVM]
> svm.fit.pred = predict(svm2, cw.test[,-46], decision.values =TRUE)
> # Predict the values on test set[GLM]
> glm.fit.pred = predict(glm.fit, cw.test[,-46])
> # Make prediction using SVM
> confusionSVM = prediction(-attr(svm.fit.pred, "decision.values"),
+                           cw.test$credit.rating == 1)
> # Create rocs curve based on prediction
> rocsSVM <- performance(confusionSVM, "tpr", "fpr")
> #make prediction using Logistic Regression
> confusionGLM = prediction(glm.fit.pred, cw.test$credit.rating == 1)
> #create rocs curve based on prediction
> rocsGLM <- performance(confusionGLM, "tpr", "fpr")
> # Plot the graph
> plot(rocsGLM, col=1)
> plot(rocsSVM, col= 2 ,add=TRUE)
> abline(0, 1, lty = 3)
> # Add the legend to the graphs
> legend(0.6, 0.6, c('glm','svm'), 1:2)
```



GLM appears to perform better than SVM seeing as there are more true positives.