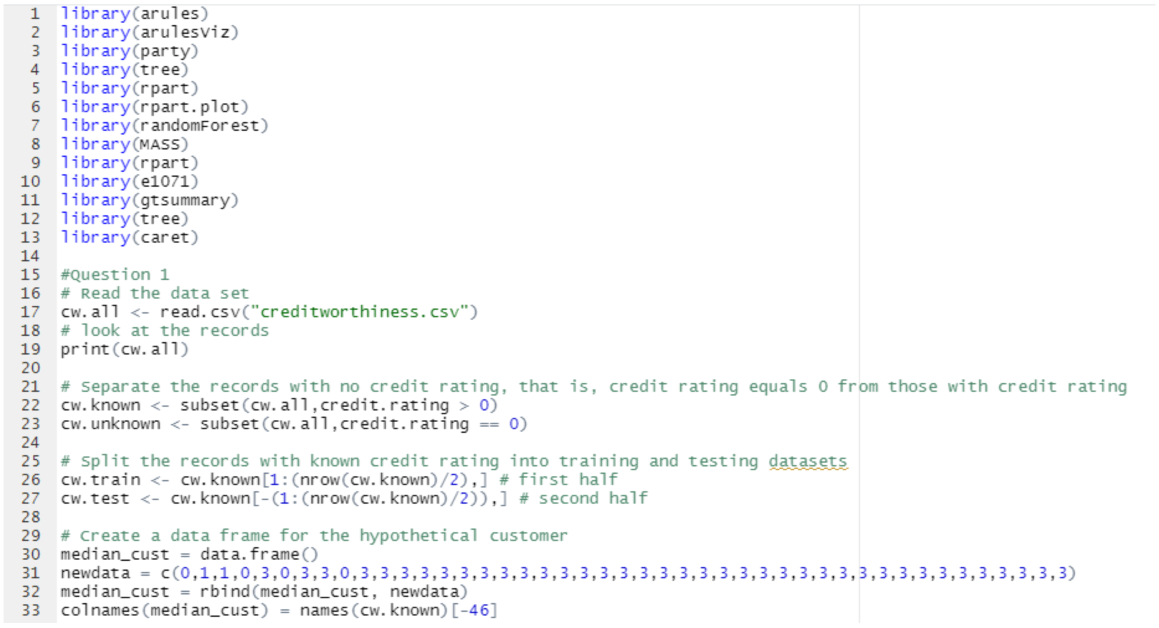
Individual Assignment 2

Name: Zhang XueCheng

Student ID: 7020739

Question 1:



```
library(arules)
library(arulesviz)
library(party)
library(tree)
library(rpart)
library(rpart.plot)
library(randomForest)
library(MASS)
library(rpart)
library(e1071)
library(gtsummary)
library(tree)
library(caret)

#Question 1
# Read the data set
cw.all <- read.csv("creditworthiness.csv")
# look at the records
print(cw.all)

# Separate the records with no credit rating, that is, credit rating equals 0 from those with credit rating
cw.known <- subset(cw.all,credit.rating > 0)
cw.unknown <- subset(cw.all,credit.rating == 0)

# Split the records with known credit rating into training and testing datasets
cw.train <- cw.known[1:(nrow(cw.known)/2),] # first half
cw.test <- cw.known[-(1:(nrow(cw.known)/2)),] # second half

# Create a data frame for the hypothetical customer
median_cust = data.frame()
newdata = 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)
median_cust = rbind(median_cust, newdata)
colnames(median_cust) = names(cw.known)[-46]
```

For question 1, load the needed libraries and read in the data set “creditworthiness.csv”.

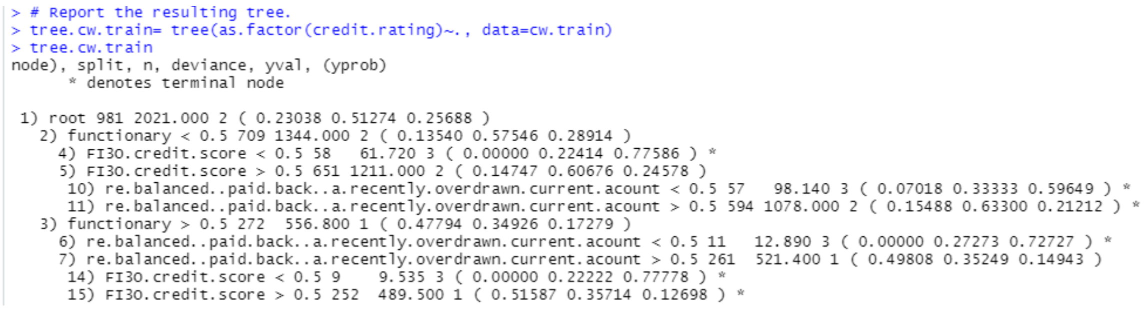
To predict the credit rating that would be assigned to each individual, we need to first separate the records and use those records with credit rating > 0. I have name it as cw.known.

After splitting the records with cw.known, we will split the dataset into a training and a test set.

Question 2:

a)

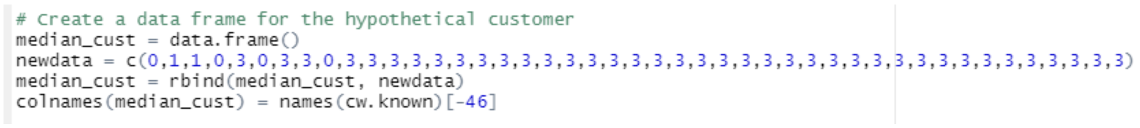
Make use of the tree package **library(tree)** to report the resulting tree



```
> # Report the resulting 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) FI30.credit.score < 0.5 58   61.720 3 ( 0.00000 0.22414 0.77586 ) *
     5) FI30.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 > 0.5 261  521.400 1 ( 0.49808 0.35249 0.14943 )
      14) FI30.credit.score < 0.5 9    9.535 3 ( 0.00000 0.22222 0.77778 ) *
      15) FI30.credit.score > 0.5 252  489.500 1 ( 0.51587 0.35714 0.12698 ) *
```

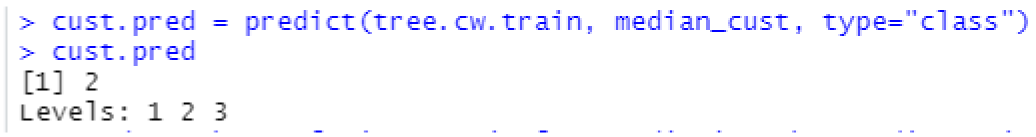
b)

Based on the resulting tree output “tree.cw.train”, we can predict the credit rating of a hypothetical “median” customer



```
# Create a data frame for the hypothetical customer
median_cust = data.frame()
newdata = 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)
median_cust = rbind(median_cust, newdata)
colnames(median_cust) = names(cw.known)[-46]
```

We first create a data frame for the hypothetical customer, and then insert some data inside, then use the row-bind function rbind to combine the data frame and new data together, and the column name will be the same as cw.known.

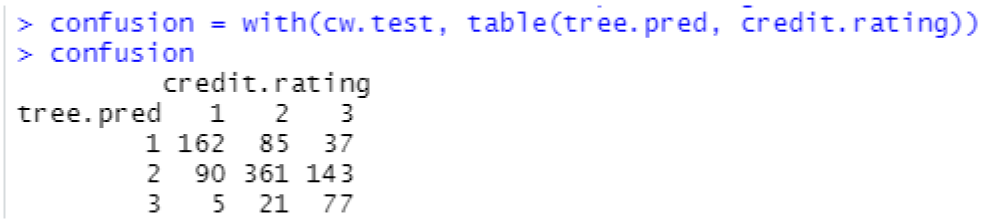


```
> cust.pred = predict(tree.cw.train, median_cust, type="class")
> cust.pred
[1] 2
Levels: 1 2 3
```

This will give the levels result output

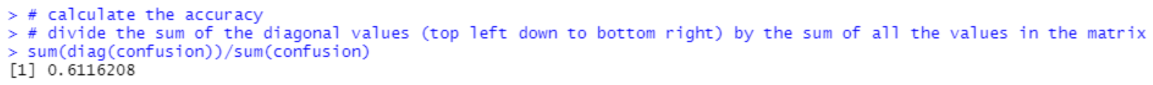
c)

After that, we can start to produce the confusion matrix for predicting the credit rating from the tree produce earlier



```
> 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
```

After we generated the confusion matrix, we can calculate the accuracy



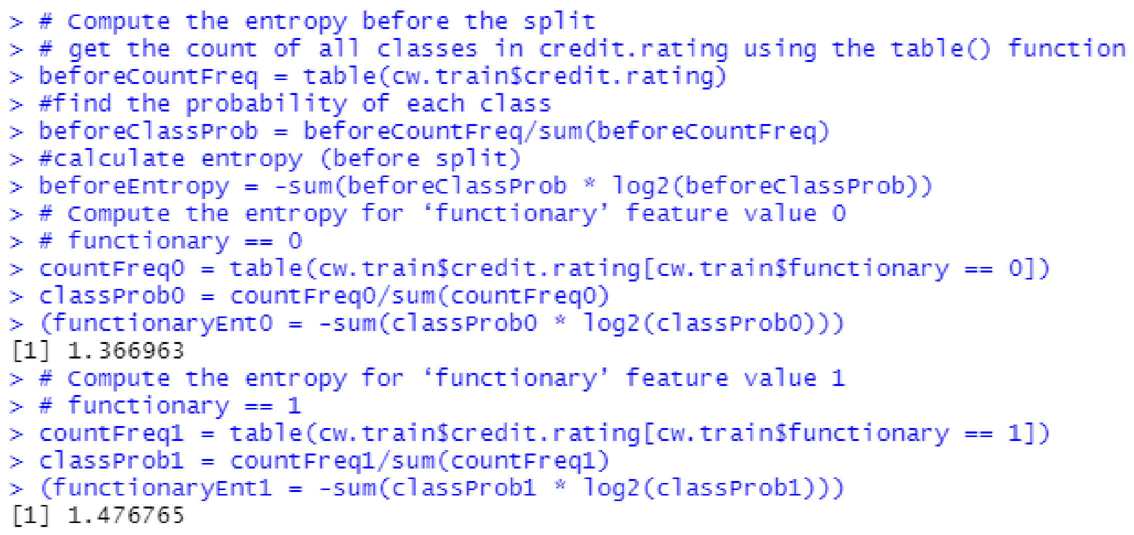
```
> # calculate the accuracy
> # divide the sum of the diagonal values (top left down to bottom right) by the sum of all the values in the matrix
> sum(diag(confusion))/sum(confusion)
[1] 0.6116208
```

The accuracy rate is 61.2%

d)

To calculate the numerical value of the gain in entropy corresponding to the first split at the top of the tree, we first need to compute the entropy before the split by using the below formula:

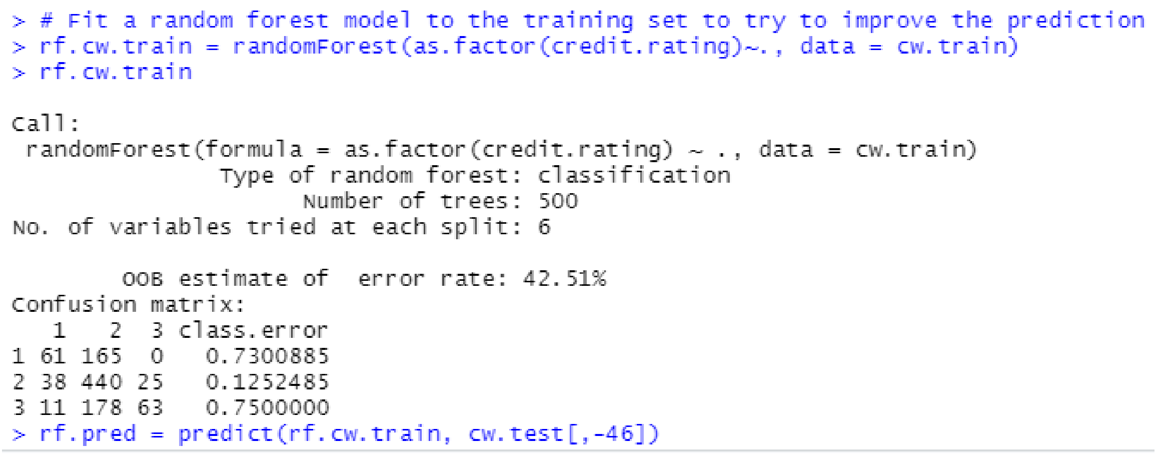
$$\boxed{Entropy = -\sum_{i=1}^{n} p_i \times \log_2 p_i}$$



```
> # Compute the entropy before the split
> # get the count of all classes in credit.rating using the table() function
> beforeCountFreq = table(cw.train$credit.rating)
> #find the probability of each class
> beforeClassProb = beforeCountFreq/sum(beforeCountFreq)
> #calculate entropy (before split)
> beforeEntropy = -sum(beforeClassProb * log2(beforeClassProb))
> # Compute the entropy for ‘functionary’ feature value 0
> # functionary == 0
> countFreq0 = table(cw.train$credit.rating[cw.train$functionary == 0])
> classProb0 = countFreq0/sum(countFreq0)
> (functionaryEnt0 = -sum(classProb0 * log2(classProb0)))
[1] 1.366963
> # Compute the entropy for ‘functionary’ feature value 1
> # functionary == 1
> countFreq1 = table(cw.train$credit.rating[cw.train$functionary == 1])
> classProb1 = countFreq1/sum(countFreq1)
> (functionaryEnt1 = -sum(classProb1 * log2(classProb1)))
[1] 1.476765
```

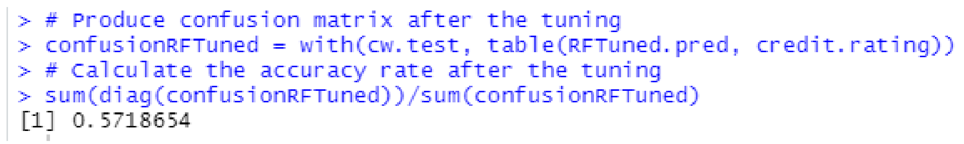
e)

Fit a random forest model to the training set to try to improve prediction



```
> # Fit a random forest model to the training set to try to improve the prediction
> 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: 42.51%
Confusion matrix:
   1   2  3 class.error
1 61 165  0   0.7300885
2 38 440 25   0.1252485
3 11 178 63   0.7500000
> rf.pred = predict(rf.cw.train, cw.test[,-46])
```

f)



```
> # Produce confusion matrix after the tuning
> confusionRFTuned = with(cw.test, table(RFTuned.pred, credit.rating))
> # Calculate the accuracy rate after the tuning
> sum(diag(confusionRFTuned))/sum(confusionRFTuned)
[1] 0.5718654
```

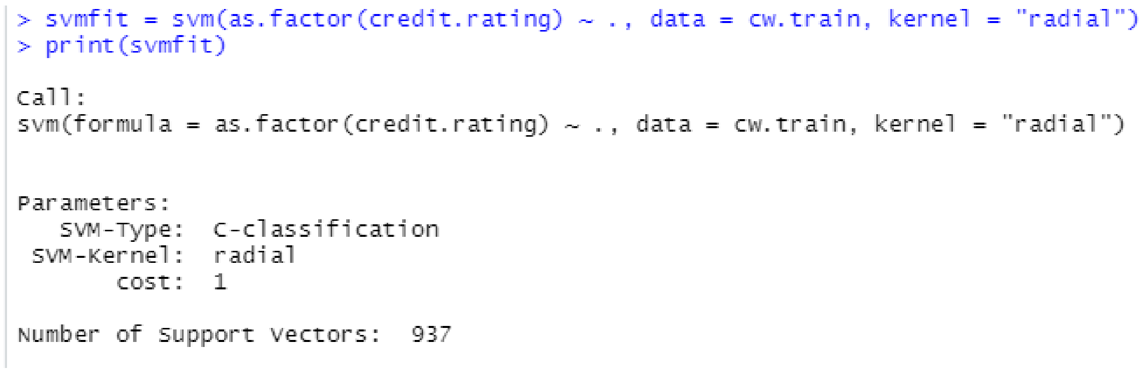
With the prediction value, we can produce the confusion matrix between the test dataset and predicted value. Since the overall accuracy is 57.2% and only decreased by 4.0% compared to the accuracy without tunning, hence overfitting does not occur.

Question 3:

By using default settings for svm() from the e1071 package, we can fit a support vector machine to predict the credit ratings of customers using all of the other variables in the dataset.

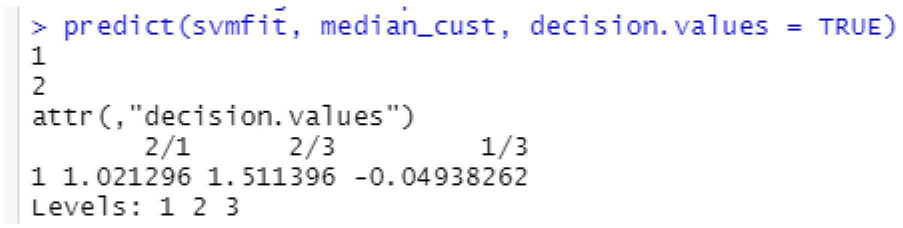
a)

Predict the credit rating of a hypothetical “median” customer



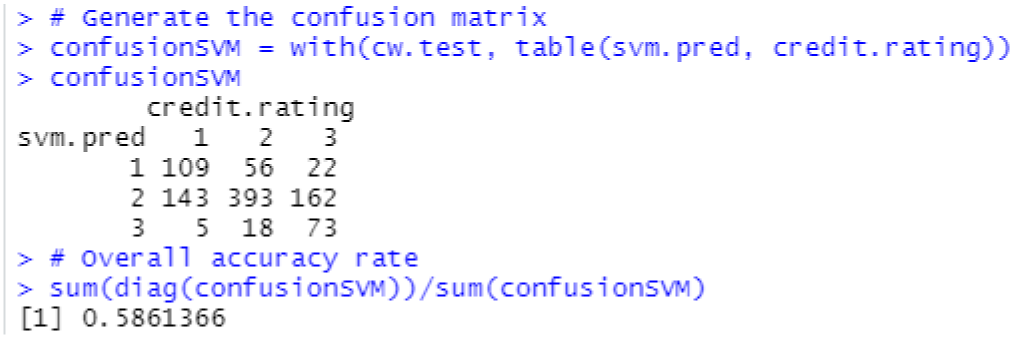
```
> svmfit = svm(as.factor(credit.rating) ~ ., data = cw.train, kernel = "radial")
> print(svmfit)

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
```

Report the decision value:



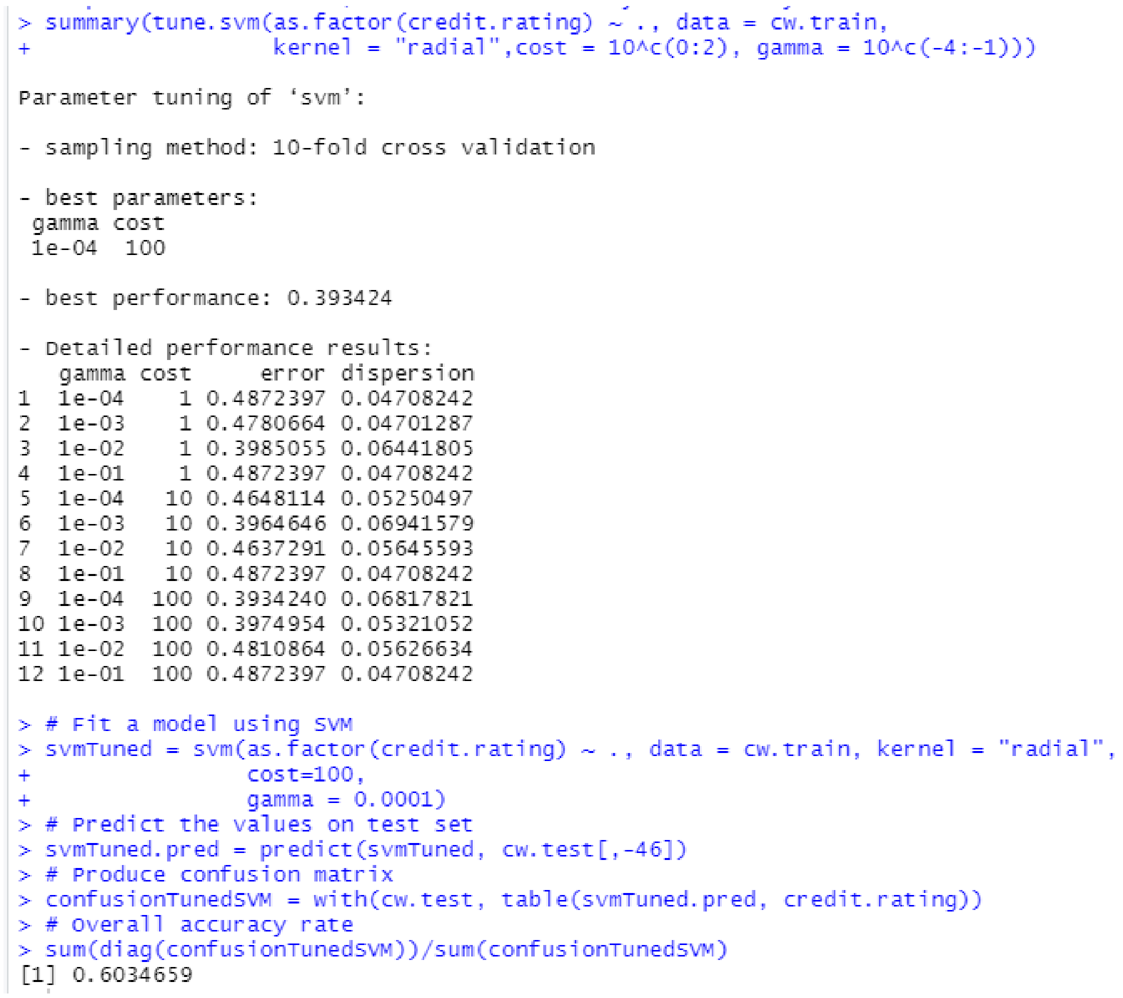
```
> predict(svmfit, median_cust, decision.values = TRUE)
1
2
attr(,"decision.values")
        2/1      2/3         1/3
1 1.021296 1.511396 -0.04938262
Levels: 1 2 3
```

b)



```
> # Generate the confusion matrix
> confusionSVM = with(cw.test, table(svm.pred, credit.rating))
> confusionSVM
        credit.rating
svm.pred   1   2   3
       1 109  56  22
       2 143 393 162
       3   5  18  73
> # Overall accuracy rate
> sum(diag(confusionSVM))/sum(confusionSVM)
[1] 0.5861366
```

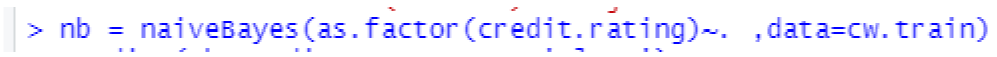
c)



```
> 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
 1e-04  100

- best performance: 0.393424

- Detailed performance results:
   gamma cost     error dispersion
1  1e-04    1 0.4872397 0.04708242
2  1e-03    1 0.4780664 0.04701287
3  1e-02    1 0.3985055 0.06441805
4  1e-01    1 0.4872397 0.04708242
5  1e-04   10 0.4648114 0.05250497
6  1e-03   10 0.3964646 0.06941579
7  1e-02   10 0.4637291 0.05645593
8  1e-01   10 0.4872397 0.04708242
9  1e-04  100 0.3934240 0.06817821
10 1e-03  100 0.3974954 0.05321052
11 1e-02  100 0.4810864 0.05626634
12 1e-01  100 0.4872397 0.04708242

> # Fit a model using SVM
> svmTuned = svm(as.factor(credit.rating) ~ ., data = cw.train, kernel = "radial",
+                 cost=100,
+                 gamma = 0.0001)
> # Predict the values on test set
> svmTuned.pred = predict(svmTuned, cw.test[,-46])
> # Produce confusion matrix
> confusionTunedSVM = with(cw.test, table(svmTuned.pred, credit.rating))
> # Overall accuracy rate
> sum(diag(confusionTunedSVM))/sum(confusionTunedSVM)
[1] 0.6034659
```

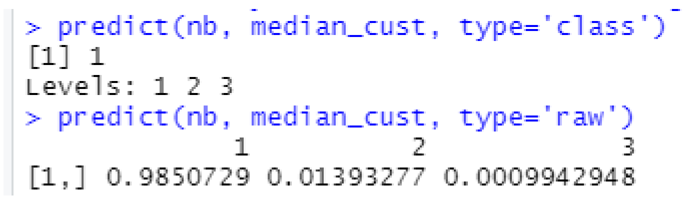
Question 4:

Fit the Naive Bayes model to predict the credit ratings of customers using all of the other variables in the dataset.



```
> nb = naiveBayes(as.factor(credit.rating)~. ,data=cw.train)
```

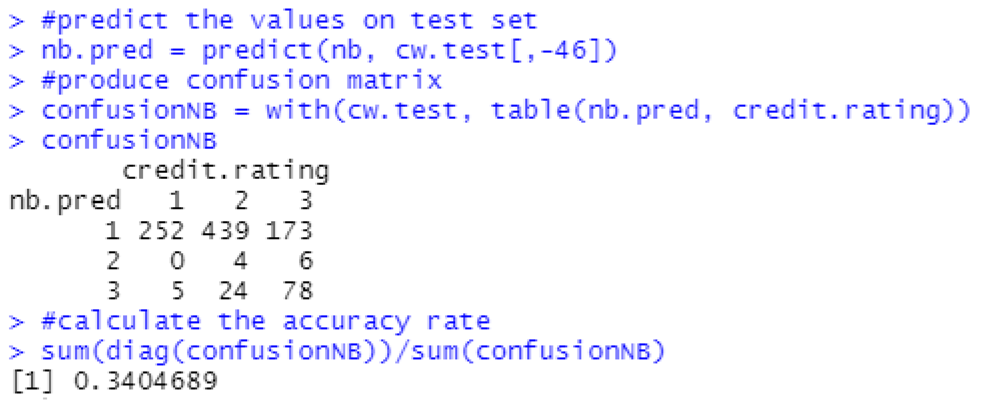
a)



```
> predict(nb, median_cust, type='class')
[1] 1
Levels: 1 2 3
> predict(nb, median_cust, type='raw')
             1          2            3
[1,] 0.9850729 0.01393277 0.0009942948
```

b)

We can use the Naïve Bayes model to predict, produce the confusion matrix, and then compute the accuracy rate.



```
> #predict the values on test set
> nb.pred = predict(nb, cw.test[,-46])
> #produce confusion matrix
> confusionNB = with(cw.test, table(nb.pred, credit.rating))
> confusionNB
       credit.rating
nb.pred   1   2   3
      1 252 439 173
      2   0   4   6
      3   5  24  78
> #calculate the accuracy rate
> sum(diag(confusionNB))/sum(confusionNB)
[1] 0.3404689
```

Question 5:

a) Which of the classifiers look to be the best?

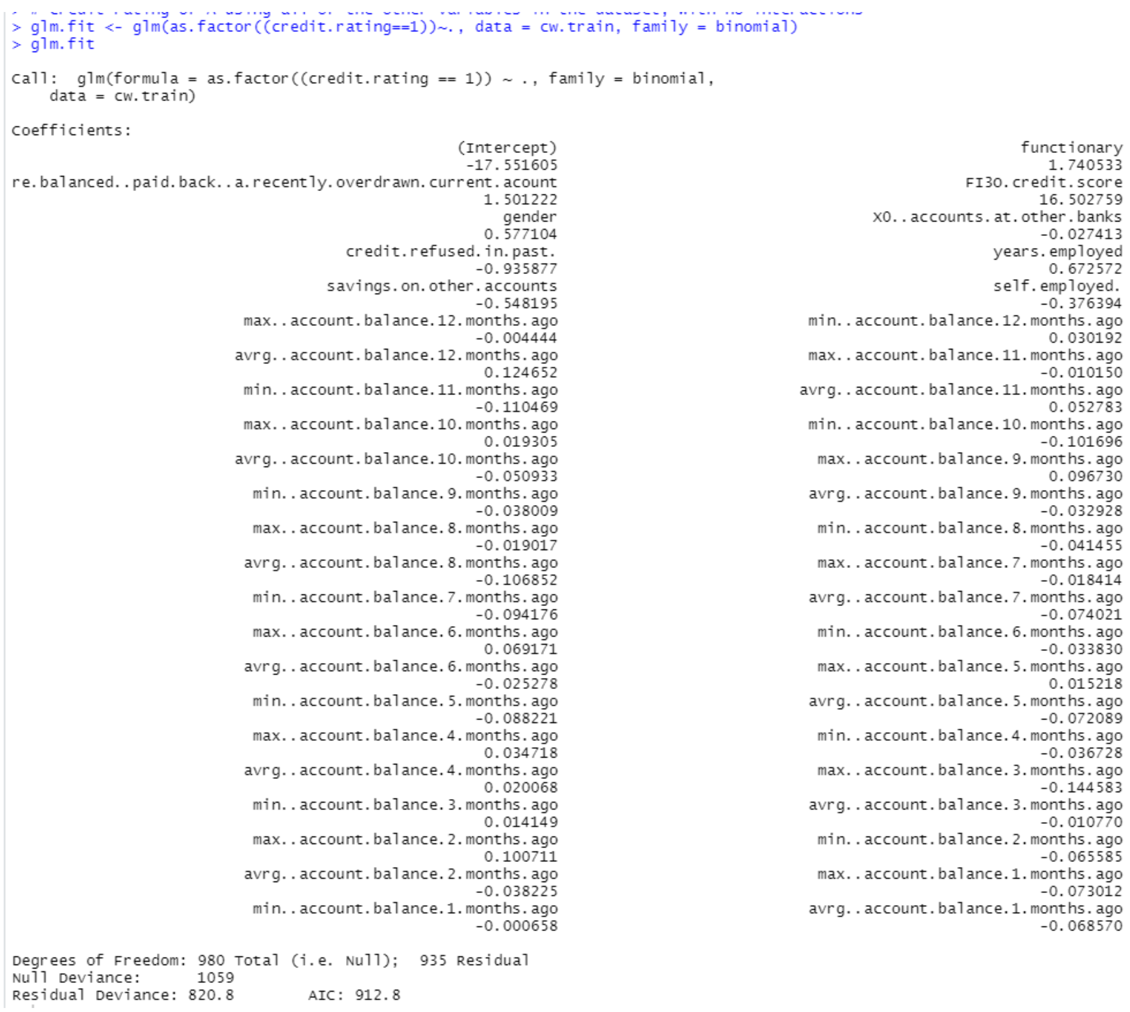
As we can see from the above results, decision tree classifier gives an overall accuracy of 61.2% and 57.2% after tunning. SVM classifier gives an overall accuracy of 58.6% and 60.3% after tunning. Naïve Bayes classifier gives an overall accuracy of 34.0% which is too low compared to decision tree and SVM classifier. Since both decision tree and SVM classifier have only a small difference between the accuracy before and after tuning, the process maximized the model's performance without overfitting and therefore SVM classifier looks the best.

b) Are there any categories that all classifiers seem to have trouble with?

Since I have calculated the entropy before and after split, I realized that the entropy gained from 1.37 to 1.48 after the entropy split, which means by splitting the "functionary" column, our entropy increases, hence the functionary category has a high level of disorder and it is not suitable to use it as a category for any classifier for training and testing the data set.

Question 6:

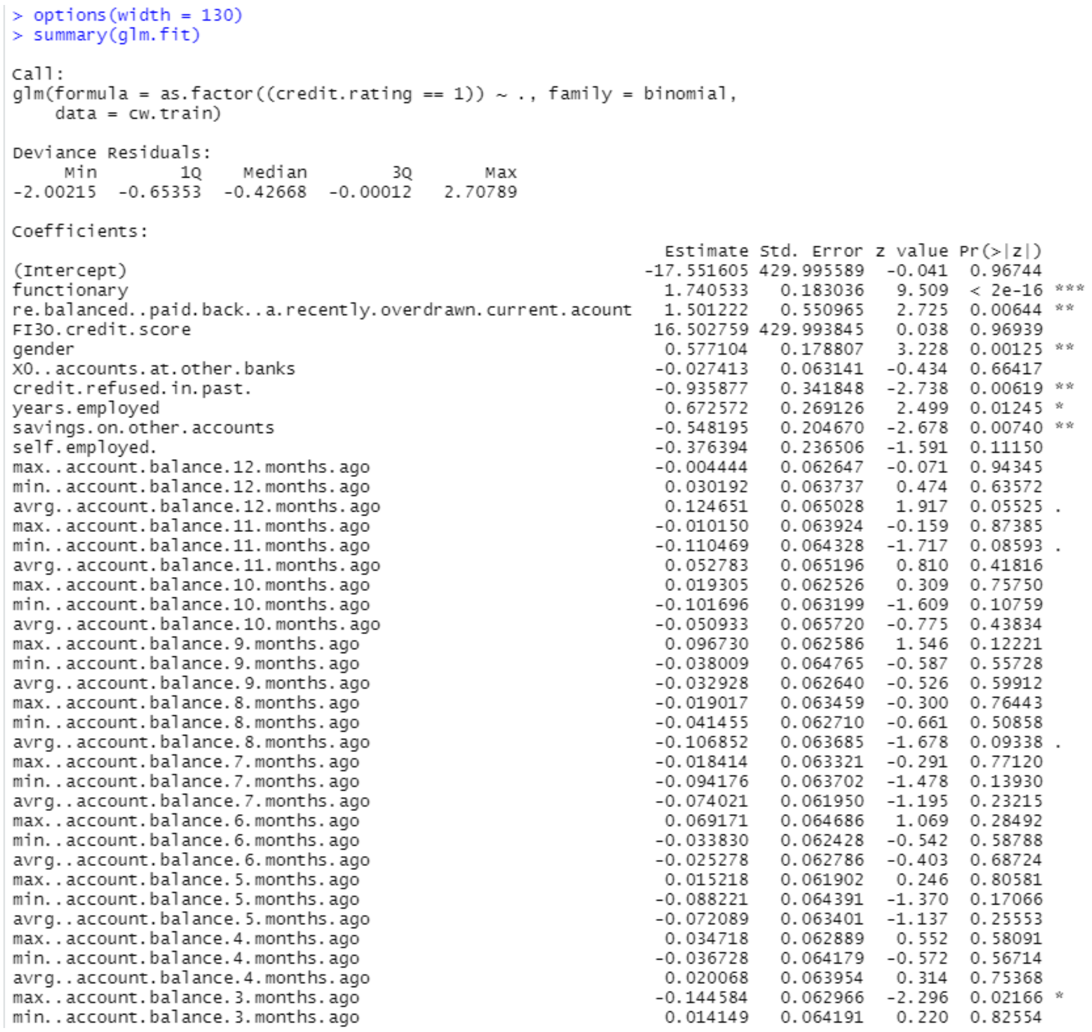
a)



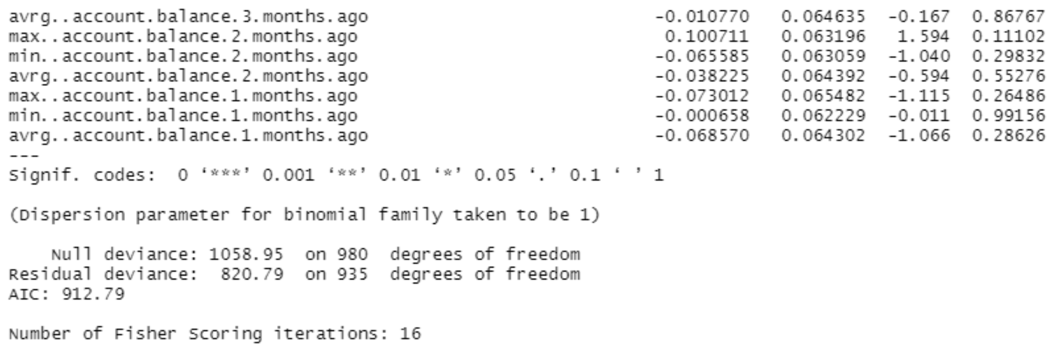
```
> # credit rating of A using all of the other variables in the dataset, with no interactions
> glm.fit <- glm(as.factor((credit.rating==1))~., data = cw.train, family = binomial)
> glm.fit

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

Coefficients:
                                                (Intercept)                                                 functionary  
                                                 -17.551605                                                    1.740533  
re.balanced..paid.back..a.recently.overdrawn.current.acount                                           FI3O.credit.score  
                                                   1.501222                                                   16.502759  
                                                     gender                                 X0..accounts.at.other.banks  
                                                   0.577104                                                   -0.027413  
                                    credit.refused.in.past.                                              years.employed  
                                                  -0.935877                                                    0.672572  
                                  savings.on.other.accounts                                              self.employed.  
                                                  -0.548195                                                   -0.376394  
                         max..account.balance.12.months.ago                          min..account.balance.12.months.ago  
                                                  -0.004444                                                    0.030192  
                        avrg..account.balance.12.months.ago                          max..account.balance.11.months.ago  
                                                   0.124652                                                   -0.010150  
                         min..account.balance.11.months.ago                         avrg..account.balance.11.months.ago  
                                                  -0.110469                                                    0.052783  
                         max..account.balance.10.months.ago                          min..account.balance.10.months.ago  
                                                   0.019305                                                   -0.101696  
                        avrg..account.balance.10.months.ago                           max..account.balance.9.months.ago  
                                                  -0.050933                                                    0.096730  
                          min..account.balance.9.months.ago                          avrg..account.balance.9.months.ago  
                                                  -0.038009                                                   -0.032928  
                          max..account.balance.8.months.ago                           min..account.balance.8.months.ago  
                                                  -0.019017                                                   -0.041455  
                         avrg..account.balance.8.months.ago                           max..account.balance.7.months.ago  
                                                  -0.106852                                                   -0.018414  
                          min..account.balance.7.months.ago                          avrg..account.balance.7.months.ago  
                                                  -0.094176                                                   -0.074021  
                          max..account.balance.6.months.ago                           min..account.balance.6.months.ago  
                                                   0.069171                                                   -0.033830  
                         avrg..account.balance.6.months.ago                           max..account.balance.5.months.ago  
                                                  -0.025278                                                    0.015218  
                          min..account.balance.5.months.ago                          avrg..account.balance.5.months.ago  
                                                  -0.088221                                                   -0.072089  
                          max..account.balance.4.months.ago                           min..account.balance.4.months.ago  
                                                   0.034718                                                   -0.036728  
                         avrg..account.balance.4.months.ago                           max..account.balance.3.months.ago  
                                                   0.020068                                                   -0.144583  
                          min..account.balance.3.months.ago                          avrg..account.balance.3.months.ago  
                                                   0.014149                                                   -0.010770  
                          max..account.balance.2.months.ago                           min..account.balance.2.months.ago  
                                                   0.100711                                                   -0.065585  
                         avrg..account.balance.2.months.ago                           max..account.balance.1.months.ago  
                                                  -0.038225                                                   -0.073012  
                          min..account.balance.1.months.ago                          avrg..account.balance.1.months.ago  
                                                  -0.000658                                                   -0.068570  

Degrees of Freedom: 980 Total (i.e. Null);  935 Residual
Null Deviance:      1059 
Residual Deviance: 820.8        AIC: 912.8
```

b)

Report the summary of the model



```
> options(width = 130)
> summary(glm.fit)

Call:
glm(formula = as.factor((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 **
FI3O.credit.score                                          16.502759 429.993845   0.038  0.96939
gender                                                      0.577104   0.178807   3.228  0.00125 **
X0..accounts.at.other.banks                                -0.027413   0.063141  -0.434  0.66417
credit.refused.in.past.                                    -0.935877   0.341848  -2.738  0.00619 **
years.employed                                              0.672572   0.269126   2.499  0.01245 *
savings.on.other.accounts                                  -0.548195   0.204670  -2.678  0.00740 **
self.employed.                                             -0.376394   0.236506  -1.591  0.11150
max..account.balance.12.months.ago                         -0.004444   0.062647  -0.071  0.94345
min..account.balance.12.months.ago                          0.030192   0.063737   0.474  0.63572
avrg..account.balance.12.months.ago                         0.124651   0.065028   1.917  0.05525 .
max..account.balance.11.months.ago                         -0.010150   0.063924  -0.159  0.87385
min..account.balance.11.months.ago                         -0.110469   0.064328  -1.717  0.08593 .
avrg..account.balance.11.months.ago                         0.052783   0.065196   0.810  0.41816
max..account.balance.10.months.ago                          0.019305   0.062526   0.309  0.75750
min..account.balance.10.months.ago                         -0.101696   0.063199  -1.609  0.10759
avrg..account.balance.10.months.ago                        -0.050933   0.065720  -0.775  0.43834
max..account.balance.9.months.ago                           0.096730   0.062586   1.546  0.12221
min..account.balance.9.months.ago                          -0.038009   0.064765  -0.587  0.55728
avrg..account.balance.9.months.ago                         -0.032928   0.062640  -0.526  0.59912
max..account.balance.8.months.ago                          -0.019017   0.063459  -0.300  0.76443
min..account.balance.8.months.ago                          -0.041455   0.062710  -0.661  0.50858
avrg..account.balance.8.months.ago                         -0.106852   0.063685  -1.678  0.09338 .
max..account.balance.7.months.ago                          -0.018414   0.063321  -0.291  0.77120
min..account.balance.7.months.ago                          -0.094176   0.063702  -1.478  0.13930
avrg..account.balance.7.months.ago                         -0.074021   0.061950  -1.195  0.23215
max..account.balance.6.months.ago                           0.069171   0.064686   1.069  0.28492
min..account.balance.6.months.ago                          -0.033830   0.062428  -0.542  0.58788
avrg..account.balance.6.months.ago                         -0.025278   0.062786  -0.403  0.68724
max..account.balance.5.months.ago                           0.015218   0.061902   0.246  0.80581
min..account.balance.5.months.ago                          -0.088221   0.064391  -1.370  0.17066
avrg..account.balance.5.months.ago                         -0.072089   0.063401  -1.137  0.25553
max..account.balance.4.months.ago                           0.034718   0.062889   0.552  0.58091
min..account.balance.4.months.ago                          -0.036728   0.064179  -0.572  0.56714
avrg..account.balance.4.months.ago                          0.020068   0.063954   0.314  0.75368
max..account.balance.3.months.ago                          -0.144584   0.062966  -2.296  0.02166 *
min..account.balance.3.months.ago                           0.014149   0.064191   0.220  0.82554
```



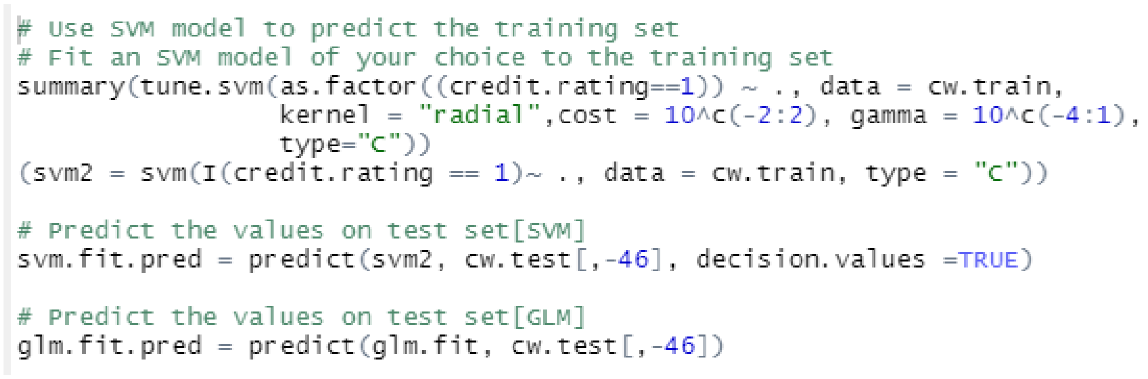
```
avrg..account.balance.3.months.ago                         -0.010770   0.064635  -0.167  0.86767
max..account.balance.2.months.ago                           0.100711   0.063196   1.594  0.11102
min..account.balance.2.months.ago                          -0.065585   0.063059  -1.040  0.29832
avrg..account.balance.2.months.ago                         -0.038225   0.064392  -0.594  0.55276
max..account.balance.1.months.ago                          -0.073012   0.065482  -1.115  0.26486
min..account.balance.1.months.ago                          -0.000658   0.062229  -0.011  0.99156
avrg..account.balance.1.months.ago                         -0.068570   0.064302  -1.066  0.28626
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1058.95  on 980  degrees of freedom
Residual deviance:  820.79  on 935  degrees of freedom
AIC: 912.79

Number of Fisher Scoring iterations: 16
```

c)

Based on the summary above, we can see “functionary” and “re.balanced..paid.back..a.recently.overdrawn.current.acount” are appear to be significant since they have the highest positive estimate rate.

However, “FI3O.credit.score” is likely to be spuriously since the estimate rate and std.error is times higher than other predictors.

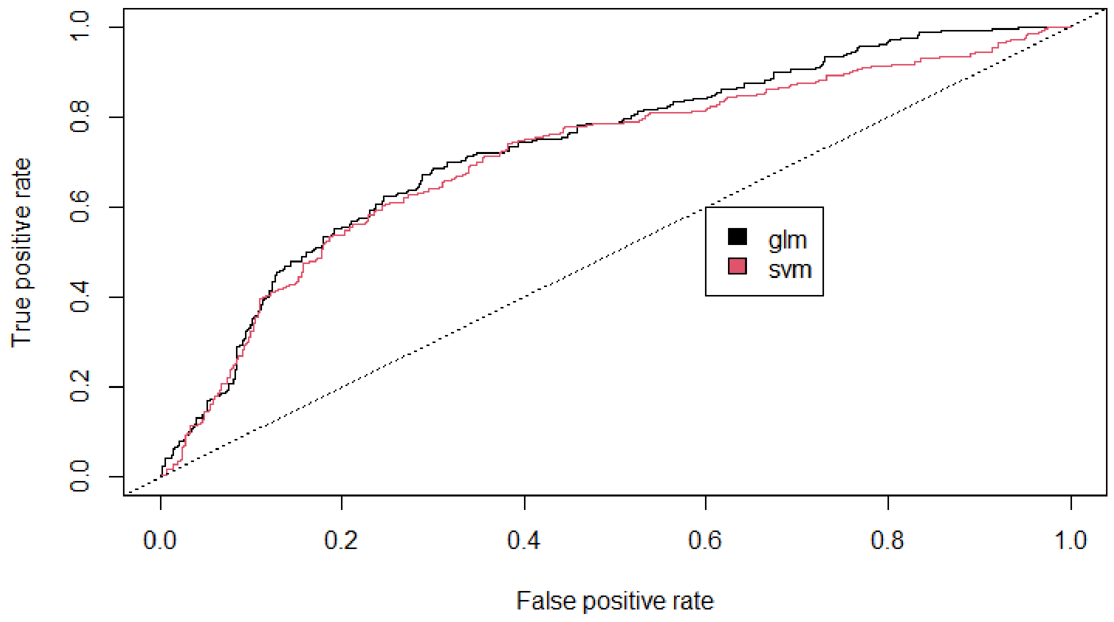
d)



```
# Use SVM model to predict the training set
# Fit an SVM model of your choice to the training set
summary(tune.svm(as.factor((credit.rating==1)) ~ ., data = cw.train,
                 kernel = "radial",cost = 10^c(-2:2), gamma = 10^c(-4:1),
                 type="C"))
(svm2 = svm(I(credit.rating == 1)~ ., data = cw.train, type = "C"))

# 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])
```

e)

```
library(ROCR)
# 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 Logidtic Regression
confusionGLM = prediction(glm.fit.pred, cw.test$credit.rating == 1)

#create rocs curve based on prediction
rocsGLM <- performance(confusionGLM, "tpr", "fpr")

# Produce ROC chart
# Plot the graph
plot(rocsGLM, col=1)
plot(rocsSVM, col= 2 ,add=TRUE)
abline(0, 1, lty = 3)
# Add the legend to the graph
legend(0.6, 0.6, c('glm','svm'), 1:2)
```


True positive rate
False positive rate
glm
svm