INFO411 Data Mining and Knowledge Discovery

Assignment 2

Audric Ng Si Kai

7431168

2a)

```
> 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
261  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
) *
```

Based on the tree constructed, the first node is split at the root, the number of values is 981, and the deviance is 2021.000. The second node is split by whether the functionary score is less than 0.5, the number of values is 709, and the deviance is 1344.000. The third node is split by whether the functionary score is more than 0.5, the number of values is 272, and the deviance is 556.800. The fourth node is split by whether the FI3o credit score is less than 0.5, the number of values is 58, and the deviance is 61.720. The fifth node is split by whether the FI3o credit score is more than 0.5, the number of values is 651, and the deviance is 1211.000. Therefore, the splitting criteria of the tree can be determined to be values that are more or less than 0.5.

b)

```
> # Create a dataframe for the hypothetical "median" customer
> # Create an empty data frame
> medianCust = data.frame()
> theData = 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)
> # Bind the data (hypothetical "median" customer to the dataframe)
> medianCust = rbind(medianCust , theData)
> # copy the name of the column (feature)
> colnames(medianCust) = names(cw.known)[-46]
```

I created a new dataframe to represent the data of a hypothetical customer. First, I created a new dataframe for the customer called medianCust and created a new dataset with random values to simulate the records of a new customer. Then, I bind the new dataset with the new dataframe created. And lastly, I added the names of the columns of the original dataset to the new dataset for medianCust.

This new dataframe medianCust will be the hypothetical customer used for predicting and testing all of the models.

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

```
Levels: 1 2 3
```

c)

```
> 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


> sum(diag(confusion))/sum(confusion)*100
[1] 61.16208
```

Total value = 162 + 85 + 37 + 90 + 361 + 143 + 5 + 21 + 77 = 981

Diag = 162 + 361 + 77 = 600

Accuracy rate = 600/981 x 100 = 61.16%

From the confusion matrix, the decision tree model has a moderately high accuracy for its prediction.

d)

```
> beforeCountFreq = table(cw.train$credit.rating)
> beforeCountFreq

  1   2   3 
226 503 252 
> # Find the probability of each class
> beforeClassProb = beforeCountFreq/sum(beforeCountFreq)
> beforeClassProb

        1         2         3 
0.2303772 0.5127421 0.2568807 
> # calculate entropy (before split)
> beforeEntropy = -sum(beforeClassProb * log2(beforeClassProb))
> beforeEntropy
[1] 1.485749
> # First split on functionary > 0.5
> # functionary == 0
> countFreq0 = table(cw.train$credit.rating[cw.train$functionary == 0])
> countFreq0

  1   2   3 
 96 408 205 
> # Find the probability of each class
> classProb0 = countFreq0/sum(countFreq0)
> (functionaryEnt0 = -sum(classProb0 * log2(classProb0)))
[1] 1.366963
> functionaryEnt0
[1] 1.366963
> # functionary == 1
> countFreq1 = table(cw.train$credit.rating[cw.train$functionary == 0])
> countFreq1

  1   2   3 
 96 408 205 
> # Find the probability of each class
> classProb1 = countFreq1/sum(countFreq1)
> classProb1
```

```
        1         2         3
0.1354020 0.5754584 0.2891396
> (functionaryEnt1 = -sum(classProb1 * log2(classProb1)))
[1] 1.366963
> functionaryEnt1
[1] 1.366963
> # Numerical value of the gain in entropy
> ent = (beforeEntropy - (functionaryEnt0 * sum(countFreq0) + functionaryE
nt1 * sum(countFreq1))/sum(sum(countFreq0) + sum(countFreq1)))
> ent
[1] 0.1187861
```

e)

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

Call:
 randomForest(formula = 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: 42.61%
Confusion matrix:
   1   2  3 class.error
1 51 175  0   0.7743363
2 32 448 23   0.1093439
3 14 174 64   0.7460317
> rf.pred = predict(rf.cw.train, cw.test[,-46])
```

Total value = 51 + 175 + 0 + 32 + 448 + 23 + 14 + 174 + 64 = 981

Diag = 51 + 448 + 64 = 508

Accuracy rate = 508/981 x 100 = 51.78%

Using the random forest model to train the training dataset, the accuracy rate is lower than the accuracy rate of the decision tree test set. Therefore, there is no improvement in the accuracy rate.

f)

```
> confusionRF = with(cw.test, table(rf.pred, credit.rating))
> confusionRF
       credit.rating
rf.pred   1   2   3
      1  55  42  17
      2 200 411 194
      3   2  14  46
> # Question 2f: Overall accuracy
> sum(diag(confusionRF))/sum(confusionRF)*100
[1] 52.19164
```

Total value = 55 + 42 + 17 + 200 + 411 + 194 + 2 + 14 + 46 = 981

Diag = 55 + 411 + 46 = 512

Accuracy rate = 512/981 x 100 = 52.19%

From the confusion matrix, the random forest model has a moderately high accuracy, but it is lower than the accuracy of the decision tree model.

3a)

```
> predict(svmfit, medianCust, 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)

```
> svm.pred = predict(svmfit, cw.test[,-46])
> 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
```

Total value = 109 + 56 + 22 + 143 + 393 + 162 + 5 + 18 + 73 = 981

Diag = 109 + 393 + 73 = 575

Accuracy rate = 575/981 x 100 = 58.60%

From the confusion matrix, the SVM(support vector machine) model has a moderately high accuracy, but it is lower than the accuracy of the decision tree model.

c)

```
> summary(tune.svm(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  100

- best performance: 0.3965986

- Detailed performance results:
   gamma cost     error dispersion
1  1e-04    1 0.4872501 0.04769583
2  1e-03    1 0.4811276 0.04639319
3  1e-02    1 0.3986807 0.06673036
4  1e-01    1 0.4872501 0.04769583
5  1e-04   10 0.4658627 0.04869084
6  1e-03   10 0.3966296 0.05072210
7  1e-02   10 0.4659658 0.06918890
8  1e-01   10 0.4872501 0.04769583
9  1e-04  100 0.3976500 0.05108128
10 1e-03  100 0.3965986 0.05235591
11 1e-02  100 0.4781591 0.06583392
12 1e-01  100 0.4872501 0.04769583

> # Fit a model using SVM
> svmTuned = svm(credit.rating~., data = cw.train,
+                kernel = "radial", cost = 10,
+                gamma = 0.001)
> print(svmTuned)

Call:
svm(formula = credit.rating ~ ., data = cw.train, kernel = "radial", cost 
= 10,
    gamma = 0.001)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  834
> # Predict the values on test set
> svmTuned.pred = predict(svmTuned, cw.test[,-46])
> # Produce confusion matrix
> confusionTunedSVM = with(cw.test, table(credit.rating, svmTuned.pred))
> confusionTunedSVM
             svmTuned.pred
credit.rating   1   2   3
            1 160  92   5
            2  87 361  19
            3  39 147  71
> # Overall accuracy rate
> sum(diag(confusionTunedSVM))/sum(confusionTunedSVM)*100
[1] 60.34659
```

Total value = 160 + 92 + 5 + 87 + 361 + 19 + 39 + 147 + 71 = 981

Diag = 160 + 361 + 71 = 592

Accuracy rate = 592/981 x 100 = 60.35%

From the confusion matrix, the SVM model accuracy after tuning increased by approximately 1.75%, but it is still lower than the accuracy of the decision tree model.

4a)

```
> nb = naiveBayes(credit.rating~., data = cw.train)
> predict(nb, medianCust, type = 'class')
[1] 1
Levels: 1 2 3
> predict(nb, medianCust, type = 'raw')
             1          2            3
[1,] 0.9850729 0.01393277 0.0009942948
> # 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)*100
[1] 34.04689
```

b)

```
> 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

   re.balanced..paid.back..a.recently.overdrawn.current.acount
Y        [,1]      [,2]
  1 0.9823009 0.1321481
  2 0.9542744 0.2090974
  3 0.8095238 0.3934582

   FI3O.credit.score
Y        [,1]      [,2]
  1 1.0000000 0.0000000
  2 0.9701789 0.1702628
  3 0.7936508 0.4054894

   gender
Y        [,1]      [,2]
  1 0.5265487 0.5004030
  2 0.4015905 0.4907079
  3 0.3531746 0.4789075

   X0..accounts.at.other.banks
Y       [,1]     [,2]
  1 2.898230 1.370579
  2 3.079523 1.410560
  3 3.047619 1.433004

   credit.refused.in.past.
Y         [,1]      [,2]
  1 0.05752212 0.2333544
  2 0.09940358 0.2995010
  3 0.21428571 0.4111425

   years.employed
Y       [,1]     [,2]
  1 3.013274 1.409429
  2 2.972167 1.412530
  3 3.039683 1.314345

   savings.on.other.accounts
Y       [,1]     [,2]
  1 3.442478 1.854427
  2 3.626243 1.802905
  3 3.420635 1.748548

   self.employed.
Y        [,1]      [,2]
  1 0.1637168 0.3708398
  2 0.2206759 0.4151152
  3 0.2142857 0.4111425

   max..account.balance.12.months.ago
Y       [,1]     [,2]
  1 2.955752 1.453819
  2 2.978131 1.425968
  3 2.940476 1.408719

   min..account.balance.12.months.ago
Y       [,1]     [,2]
  1 2.973451 1.391787
  2 2.956262 1.412126
  3 3.115079 1.393872
```

```
    avrg..account.balance.12.months.ago
Y         [,1]     [,2]
  1 3.225664 1.450657
  2 2.982107 1.366094
  3 3.015873 1.414124

    max..account.balance.11.months.ago
Y         [,1]     [,2]
  1 2.884956 1.390458
  2 2.992048 1.412782
  3 3.059524 1.436723

    min..account.balance.11.months.ago
Y         [,1]     [,2]
  1 2.827434 1.442636
  2 2.990060 1.370543
  3 2.984127 1.405647

    avrg..account.balance.11.months.ago
Y         [,1]     [,2]
  1 3.017699 1.391928
  2 2.978131 1.361654
  3 2.996032 1.407147

    max..account.balance.10.months.ago
Y         [,1]     [,2]
  1 3.026549 1.454252
  2 3.001988 1.445558
  3 2.968254 1.413856

    min..account.balance.10.months.ago
Y         [,1]     [,2]
  1 2.792035 1.419273
  2 3.031809 1.434833
  3 2.904762 1.416626

    avrg..account.balance.10.months.ago
Y         [,1]     [,2]
  1 2.946903 1.325582
  2 2.998012 1.410686
  3 3.027778 1.470569

    max..account.balance.9.months.ago
Y         [,1]     [,2]
  1 3.110619 1.467022
  2 2.912525 1.414320
  3 2.980159 1.440591

    min..account.balance.9.months.ago
Y         [,1]     [,2]
  1 2.920354 1.363926
  2 2.914513 1.419362
  3 3.067460 1.385545
```

5a)

The decision tree model looks to be the best as the test set has the highest accuracy rate of 61.16%. This is as compared to the other models, random forest with 52.19%, SVM with 60.35%, and naive bayes with 34.05%.

b)

The Naïve Bayes classifier predicted the conditional probability of the first A-priori probability using the FI3o credit score as a zero. This means that the classifier may have encountered some issues in predicting the A-priori probability using the FI3o credit score.

6a)

```
> glm.fit <- glm((credit.rating==1)~., data = cw.train, family = binomial)
> options(width = 130)
```

b)

```
> 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 ** 
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)

The coefficient for FI3o credit score is abnormally high at 16.50 as compared to the intercept value of -17.55. The coefficient also does not fall in the same range as the other attributes of the dataset from -1 to 1.

d)

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

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
 0.001  100

- best performance: 0.2171408

- Detailed performance results:
   gamma  cost     error dispersion
1  1e-04 1e-02 0.2303649 0.02447423
2  1e-03 1e-02 0.2303649 0.02447423
3  1e-02 1e-02 0.2303649 0.02447423
4  1e-01 1e-02 0.2303649 0.02447423
5  1e+00 1e-02 0.2303649 0.02447423
6  1e+01 1e-02 0.2303649 0.02447423
7  1e-04 1e-01 0.2303649 0.02447423
8  1e-03 1e-01 0.2303649 0.02447423
9  1e-02 1e-01 0.2303649 0.02447423
10 1e-01 1e-01 0.2303649 0.02447423
11 1e+00 1e-01 0.2303649 0.02447423
12 1e+01 1e-01 0.2303649 0.02447423
13 1e-04 1e+00 0.2303649 0.02447423
14 1e-03 1e+00 0.2303649 0.02447423
15 1e-02 1e+00 0.2303649 0.02250408
16 1e-01 1e+00 0.2303649 0.02447423
17 1e+00 1e+00 0.2303649 0.02447423
18 1e+01 1e+00 0.2303649 0.02447423
19 1e-04 1e+01 0.2303649 0.02447423
20 1e-03 1e+01 0.2293445 0.03000640
21 1e-02 1e+01 0.2303649 0.03628059
22 1e-01 1e+01 0.2303649 0.02447423
23 1e+00 1e+01 0.2303649 0.02447423
24 1e+01 1e+01 0.2303649 0.02447423
25 1e-04 1e+02 0.2293445 0.03000640
26 1e-03 1e+02 0.2171408 0.02372726
27 1e-02 1e+02 0.2395382 0.05200855
28 1e-01 1e+02 0.2303649 0.02447423
29 1e+00 1e+02 0.2303649 0.02447423
30 1e+01 1e+02 0.2303649 0.02447423

> (svm2 = svm((credit.rating==1)~., data = cw.train, type = "C"))

Call:
svm(formula = (credit.rating == 1) ~ ., data = cw.train, type = "C")


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

Number of Support Vectors:  664

> #Fit a model using svm
> svm.fit = svm((credit.rating==1) ~ .,data = cw.train, type = 'C',
+               gamma = 0.001, cost = 100, kernel = "radial")
> print(svm.fit)

Call:
```

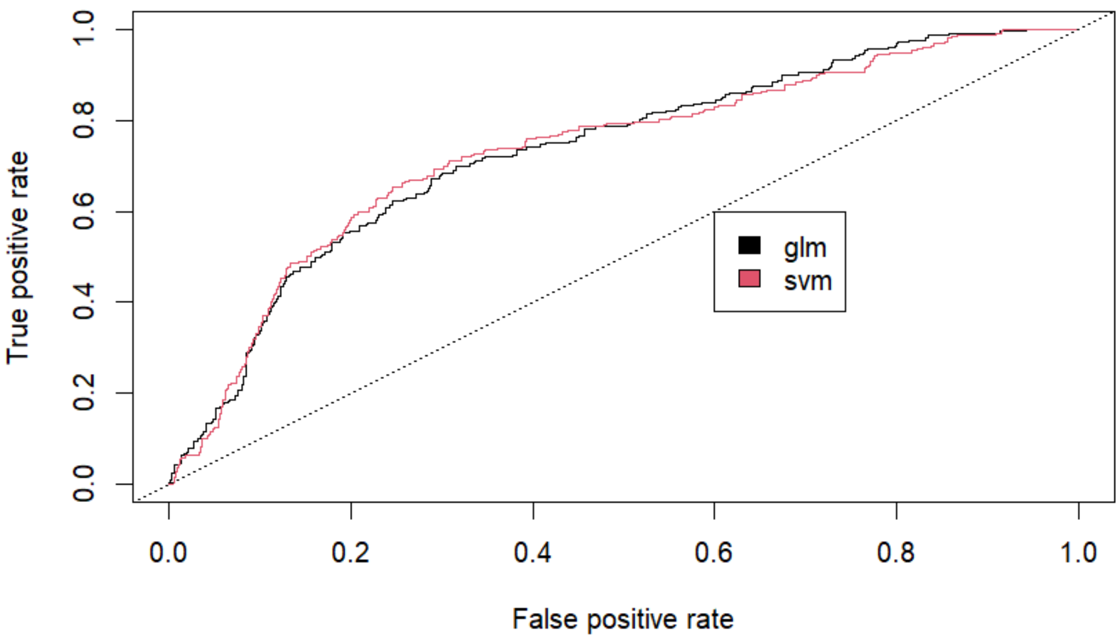
```
svm(formula = (credit.rating == 1) ~ ., data = cw.train, type = "C", gamma
= 0.001, cost = 100, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  100

Number of Support Vectors:  522
```

e)

```
> # Predict the values on test set[SVM]
> svm.fit.pred = predict(svm.fit, 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 graph
> legend(0.6, 0.6, c('glm','svm'), 1:2)
```



Even though both models started at the same false positive rate, the SVM model has a higher false positive rate than the GLM model. Whereas, both models have the same true positive rate at the end of the prediction.