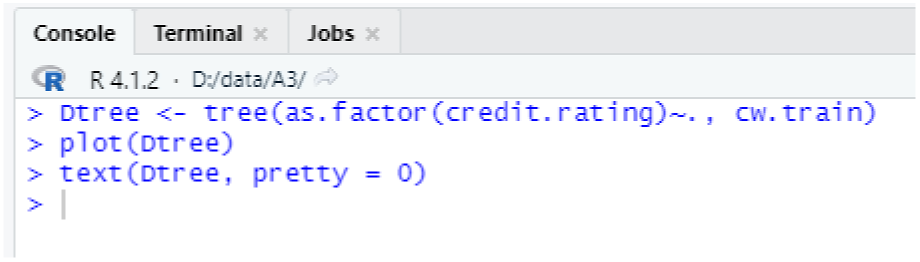
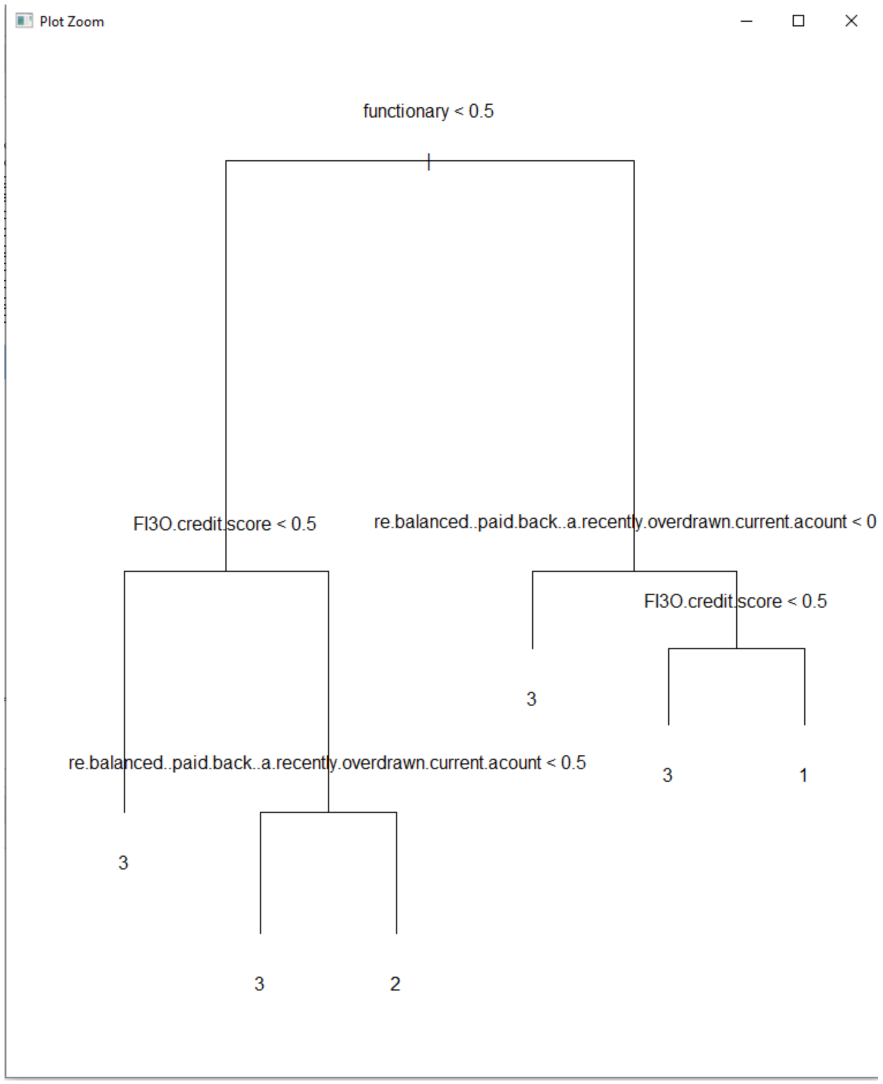
Question 2

(a) Report the resulting tree.



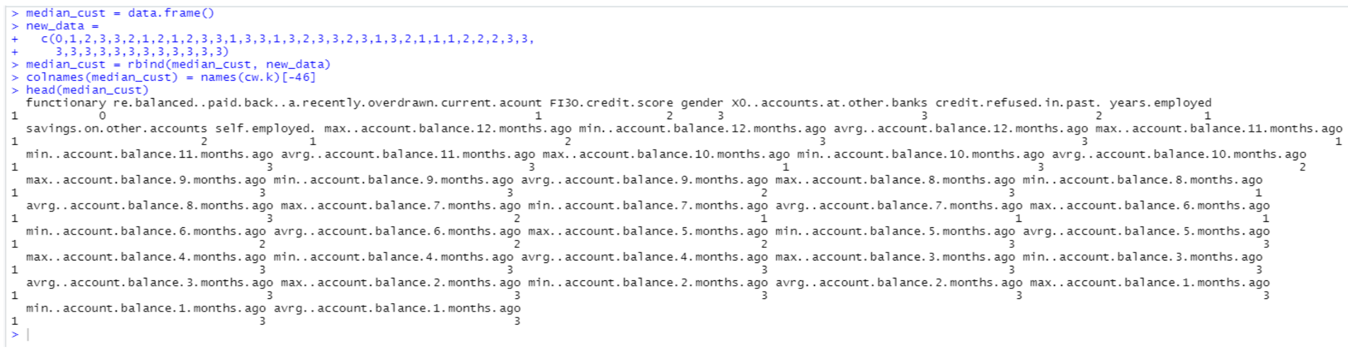
```
> Dtree <- tree(as.factor(credit.rating)~., cw.train)
> plot(Dtree)
> text(Dtree, pretty = 0)
> 
```


Plot Zoom
functionary < 0.5
FI3O.credit.score < 0.5
re.balanced..paid.back..a.recently.overdrawn.current.acount < 0
FI3O.credit.score < 0.5
3
re.balanced..paid.back..a.recently.overdrawn.current.acount < 0.5
3
1
3
3
2

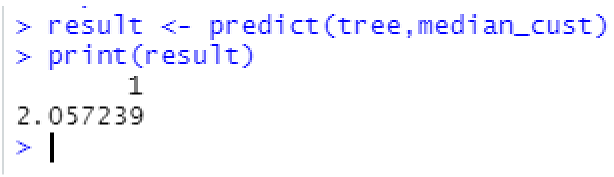
The resulting tree has 5 conditions ending with 6 leaf nodes. It’s first split is functionary < 0.5.

(b) Based on this output, predict the credit rating of a hypothetical "median" customer, i.e., one with the attributes listed in Table 1 (in the last page), showing the steps involved.

Data frame for the hypothetical customer



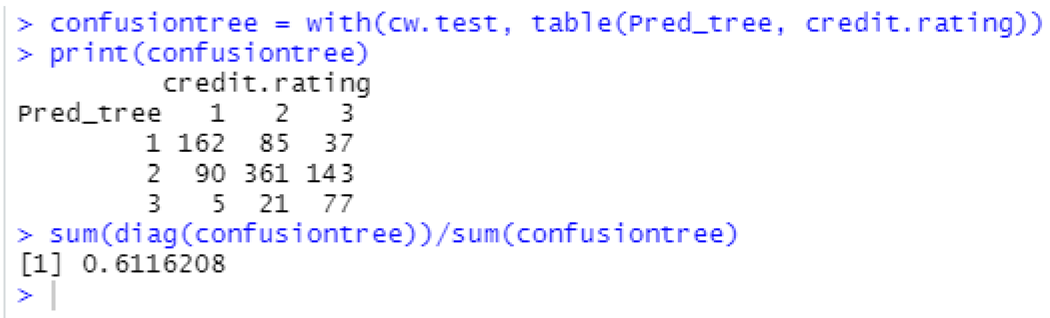
```
> median_cust = data.frame()
> new_data =
+   c(0,1,2,3,3,2,1,2,1,2,3,3,1,3,3,1,3,2,3,3,2,3,1,3,2,1,1,1,2,2,2,3,3,
+     3,3,3,3,3,3,3,3,3,3,3,3)
> median_cust = rbind(median_cust, new_data)
> colnames(median_cust) = names(cw.k)[-46]
> head(median_cust)
  functionary re.balanced..paid.back..a.recently.overdrawn.current.acount FI30.credit.score gender X0..accounts.at.other.banks credit.refused.in.past. years.employed
1           0                                                           1                 2      3                            3                       2              1
  savings.on.other.accounts self.employed. max..account.balance.12.months.ago min..account.balance.12.months.ago avrg..account.balance.12.months.ago max..account.balance.11.months.ago
1                         2              1                                  2                                  3                                   3                                  1
  min..account.balance.11.months.ago avrg..account.balance.11.months.ago max..account.balance.10.months.ago min..account.balance.10.months.ago avrg..account.balance.10.months.ago
1                                  3                                   3                                  1                                  3                                   2
  max..account.balance.9.months.ago min..account.balance.9.months.ago avrg..account.balance.9.months.ago max..account.balance.8.months.ago min..account.balance.8.months.ago
1                                 3                                 3                                  2                                 3                                 1
  avrg..account.balance.8.months.ago max..account.balance.7.months.ago min..account.balance.7.months.ago avrg..account.balance.7.months.ago max..account.balance.6.months.ago
1                                  3                                 2                                 1                                  1                                 1
  min..account.balance.6.months.ago avrg..account.balance.6.months.ago max..account.balance.5.months.ago min..account.balance.5.months.ago avrg..account.balance.5.months.ago
1                                 2                                  2                                 2                                 3                                  3
  max..account.balance.4.months.ago min..account.balance.4.months.ago avrg..account.balance.4.months.ago max..account.balance.3.months.ago min..account.balance.3.months.ago
1                                 3                                 3                                  3                                 3                                 3
  avrg..account.balance.3.months.ago max..account.balance.2.months.ago min..account.balance.2.months.ago avrg..account.balance.2.months.ago max..account.balance.1.months.ago
1                                  3                                 3                                 3                                  3                                 3
  min..account.balance.1.months.ago avrg..account.balance.1.months.ago
1                                 3                                  3
> |
```



```
> result <- predict(tree,median_cust)
> print(result)
       1
2.057239
> |
```

Using the library, the prediction for the credit rating median_cust is 2. If we follow through with the conditions plotted on the decision tree, it also ends up at leaf node 2.

(c) (0.5 mark) Produce the confusion matrix for predicting the credit rating from this tree on the test set, and also report the overall accuracy rate.

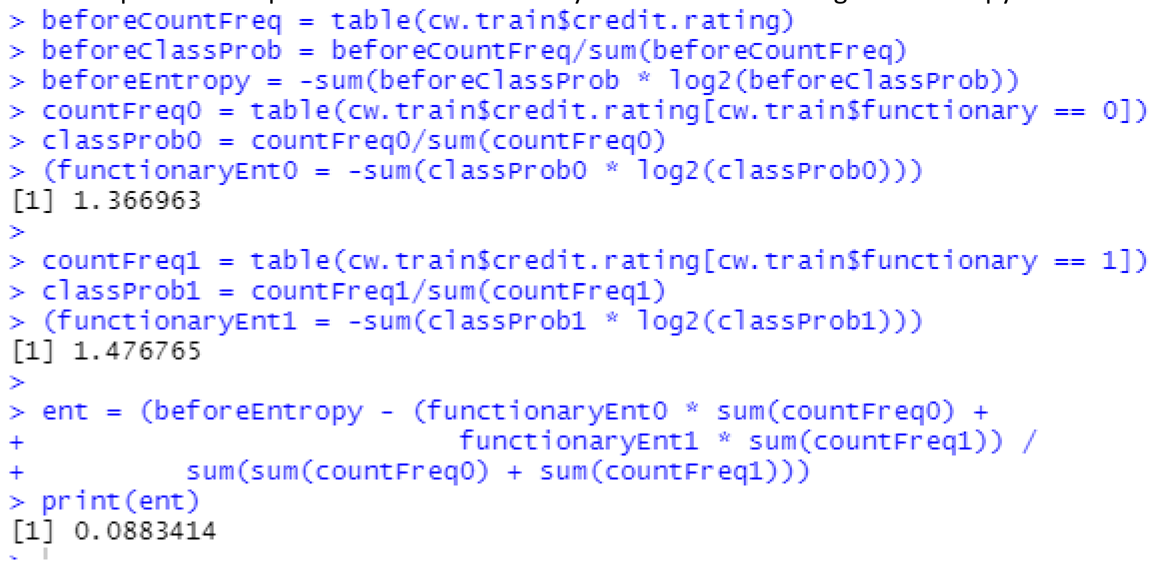


```
> confusiontree = with(cw.test, table(Pred_tree, credit.rating))
> print(confusiontree)
         credit.rating
Pred_tree   1   2   3
        1 162  85  37
        2  90 361 143
        3   5  21  77
> sum(diag(confusiontree))/sum(confusiontree)
[1] 0.6116208
> |
```

The confusion matrix seems to be very accurate with predicting results for credit rating 1 and 2 but not 3. The overall accuracy is 0.6116208, roughly 60%.

(d) (1.5 marks) What is the numerical value of the gain in entropy corresponding to the first split at the top of the tree? (Use logarithms to base 2, and show the details of the calculation rather than just providing a final answer.)

The first split at the top of the tree is functionary so lets calculate the gain in entropy.



```
> beforeCountFreq = table(cw.train$credit.rating)
> beforeClassProb = beforeCountFreq/sum(beforeCountFreq)
> beforeEntropy = -sum(beforeClassProb * log2(beforeClassProb))
> countFreq0 = table(cw.train$credit.rating[cw.train$functionary == 0])
> classProb0 = countFreq0/sum(countFreq0)
> (functionaryEnt0 = -sum(classProb0 * log2(classProb0)))
[1] 1.366963
>
> countFreq1 = table(cw.train$credit.rating[cw.train$functionary == 1])
> classProb1 = countFreq1/sum(countFreq1)
> (functionaryEnt1 = -sum(classProb1 * log2(classProb1)))
[1] 1.476765
>
> ent = (beforeEntropy - (functionaryEnt0 * sum(countFreq0) +
+                           functionaryEnt1 * sum(countFreq1)) /
+          sum(sum(countFreq0) + sum(countFreq1)))
> print(ent)
[1] 0.0883414
```

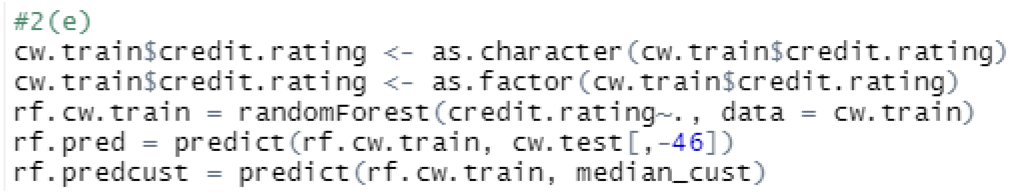
Firstly, we calculate the entropy before the split. This can be gotten by applying this formula to the probability of each class.
-sum(beforeClassProb * log2(beforeClassProb))
This will give us 1.485749.

Next, we compute the entropy for functionary == 0.
(functionaryEnt0 = -sum(classProb0 * log2(classProb0)))
This will give us 1.366963.

Next, we compute the entropy for functionary == 1.
(functionaryEnt1 = -sum(classProb1 * log2(classProb1)))
This will give us 1.476765.

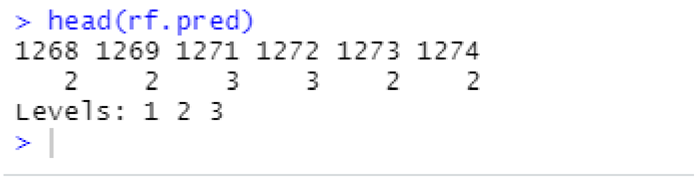
Next with we compute for the information gain on functionary.
ent = (beforeEntropy - (functionaryEnt0 * sum(countFreq0) +
functionaryEnt1 * sum(countFreq1)) /
sum(sum(countFreq0) + sum(countFreq1)))
This will give us 0.00883414.

(e) (0.5 mark) Fit a random forest model to the training set to try to improve prediction, and report the R output.



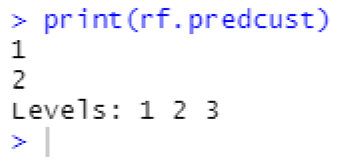
```
#2(e)
cw.train$credit.rating <- as.character(cw.train$credit.rating)
cw.train$credit.rating <- as.factor(cw.train$credit.rating)
rf.cw.train = randomForest(credit.rating~., data = cw.train)
rf.pred = predict(rf.cw.train, cw.test[,-46])
rf.predcust = predict(rf.cw.train, median_cust)
```

Apply random forest model to training set. The output of the prediction credit rating on the testing set is below



```
> head(rf.pred)
1268 1269 1271 1272 1273 1274
   2    2    3    3    2    2
Levels: 1 2 3
>
```

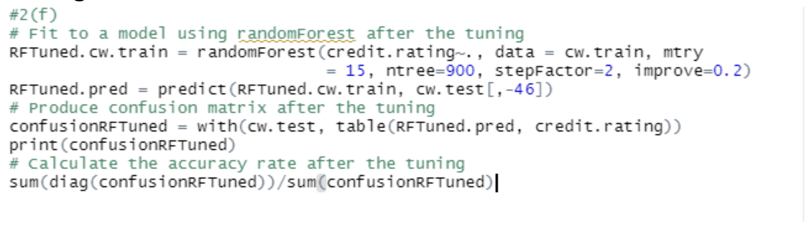
The output of the prediction credit rating on our hypothetical customer is still 2.



```
> print(rf.predcust)
1
2
Levels: 1 2 3
>
```

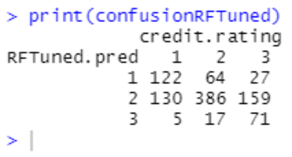
(f) (0.5 mark) Produce the confusion matrix for predicting the credit rating from this forest on the test set, and also report the overall accuracy rate.

Tuning with 900 trees.



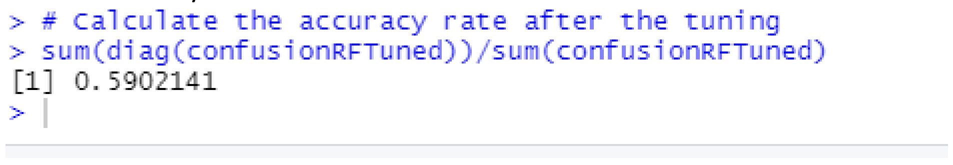
```
#2(f)
# Fit to a model using randomForest after the tuning
RFTuned.cw.train = randomForest(credit.rating~., data = cw.train, mtry
                               = 15, ntree=900, stepFactor=2, improve=0.2)
RFTuned.pred = predict(RFTuned.cw.train, cw.test[,-46])
# Produce confusion matrix after the tuning
confusionRFTuned = with(cw.test, table(RFTuned.pred, credit.rating))
print(confusionRFTuned)
# Calculate the accuracy rate after the tuning
sum(diag(confusionRFTuned))/sum(confusionRFTuned)
```

Confusion matrix



```
> print(confusionRFTuned)
            credit.rating
RFTuned.pred   1   2   3
           1 122  64  27
           2 130 386 159
           3   5  17  71
>
```

Overall accuracy rate

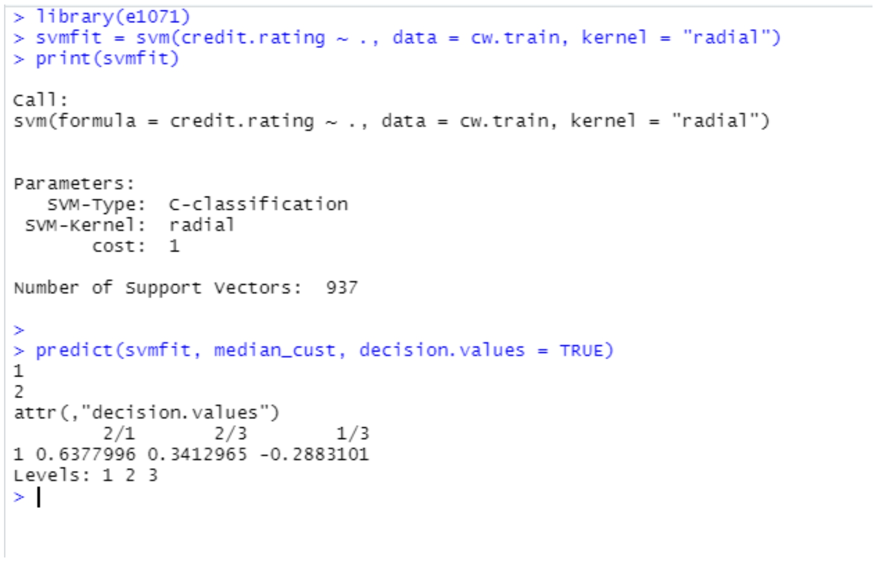


```
> # Calculate the accuracy rate after the tuning
> sum(diag(confusionRFTuned))/sum(confusionRFTuned)
[1] 0.5902141
>
```

Question 3

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

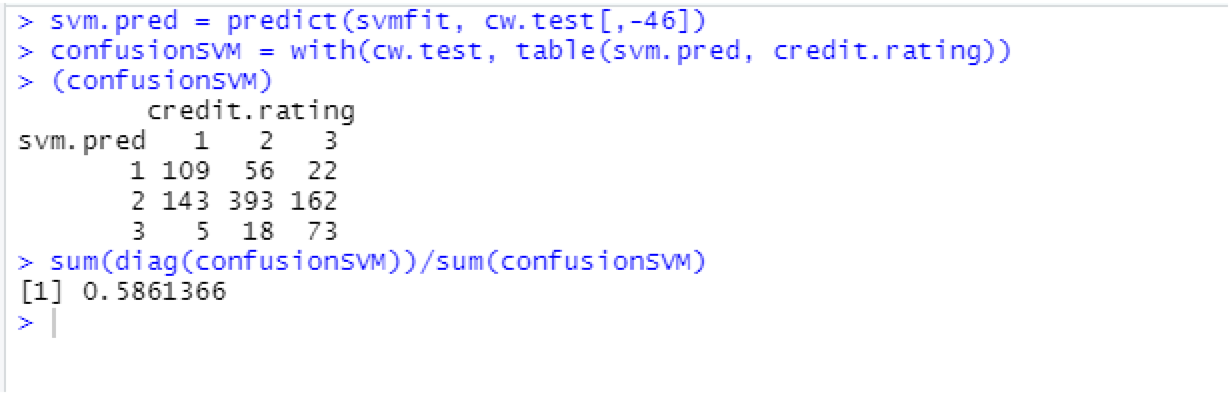
(a) (1 mark) Predict the credit rating of a hypothetical "median" customer, i.e., one with the attributes listed in Table 1 (in the last page). Report decision values as well.



```
> library(e1071)
> svmfit = svm(credit.rating ~ ., data = cw.train, kernel = "radial")
> print(svmfit)

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


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

Number of Support Vectors:  937

>
> predict(svmfit, median_cust, decision.values = TRUE)
1
2
attr(,"decision.values")
        2/1       2/3        1/3
1 0.6377996 0.3412965 -0.2883101
Levels: 1 2 3
> |
```

Using svm, the prediction of the credit rating is also 2 .

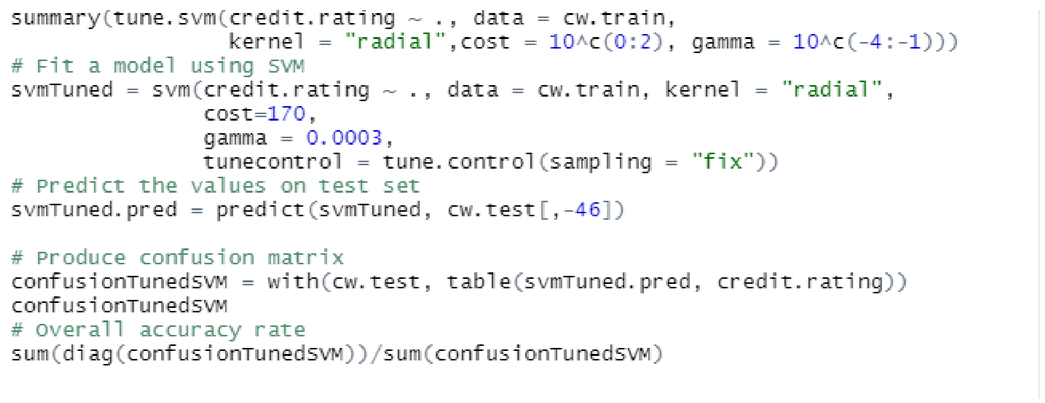
(b) (0.5 mark) Produce the confusion matrix for predicting the credit rating from this SVM on the test set, and also report the overall accuracy rate.



```
> 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
> sum(diag(confusionSVM))/sum(confusionSVM)
[1] 0.5861366
> |
```

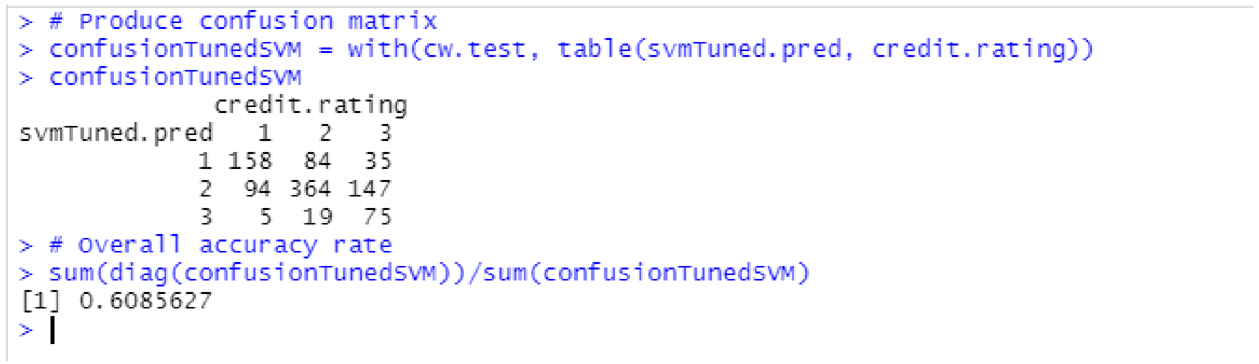
However, when used on the test set, it seems to have poor accuracy on predicting credit.rating when the true value is 1 or 3. The SVM seems to predict 2 even when the true value is 1 or 3. The overall accuracy is 0.5861366 before tuning.

(c) (0.5 mark) Automatically or manually tune the SVM to improve prediction over that found in question 3(b). Report the resulting SVM settings and the resulting confusion matrix for predicting the test set. (Any amount of improvement is acceptable.)



```
summary(tune.svm(credit.rating ~ ., data = cw.train,
                 kernel = "radial",cost = 10^c(0:2), gamma = 10^c(-4:-1)))
# Fit a model using SVM
svmTuned = svm(credit.rating ~ ., data = cw.train, kernel = "radial",
               cost=170,
               gamma = 0.0003,
               tunecontrol = tune.control(sampling = "fix"))
# 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))
confusionTunedSVM
# Overall accuracy rate
sum(diag(confusionTunedSVM))/sum(confusionTunedSVM)
```

I set the cost to 170 and gamma to 0.0003.



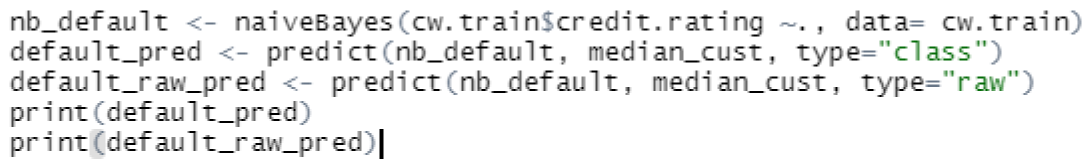
```
> # Produce confusion matrix
> confusionTunedSVM = with(cw.test, table(svmTuned.pred, credit.rating))
> confusionTunedSVM
             credit.rating
svmTuned.pred   1   2   3
            1 158  84  35
            2  94 364 147
            3   5  19  75
> # Overall accuracy rate
> sum(diag(confusionTunedSVM))/sum(confusionTunedSVM)
[1] 0.6085627
> 
```

There seems to be improvement from 0.5861366 to 0.6085627 in accuracy.
According to the confusion matrix still seems to be bad at predicting credit.rating when the true value is 3. It seems to predict 2 even when the true value is 3.

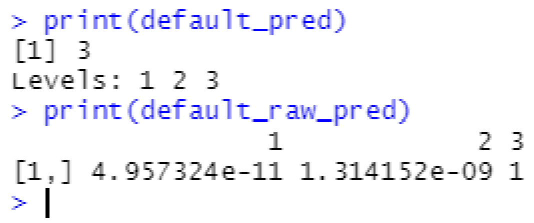
Question 4

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

(a) (1 mark) Predict the credit rating of a hypothetical “median” customer, i.e., one with the attributes listed in Table 1 (in the last page). Report predicted probabilities as well.

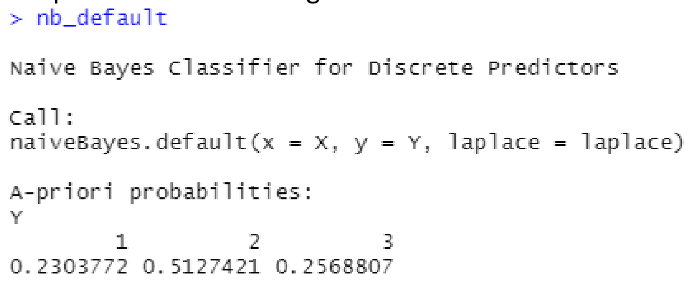


```
nb_default <- naiveBayes(cw.train$credit.rating ~., data= cw.train)
default_pred <- predict(nb_default, median_cust, type="class")
default_raw_pred <- predict(nb_default, median_cust, type="raw")
print(default_pred)
print(default_raw_pred)
```



```
> print(default_pred)
[1] 3
Levels: 1 2 3
> print(default_raw_pred)
                1            2 3
[1,] 4.957324e-11 1.314152e-09 1
> 
```

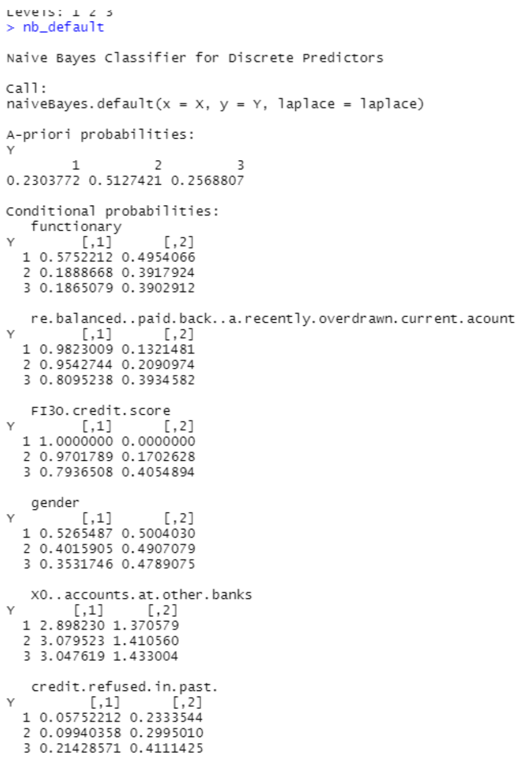
The predicted credit rating for the median customer is 3.



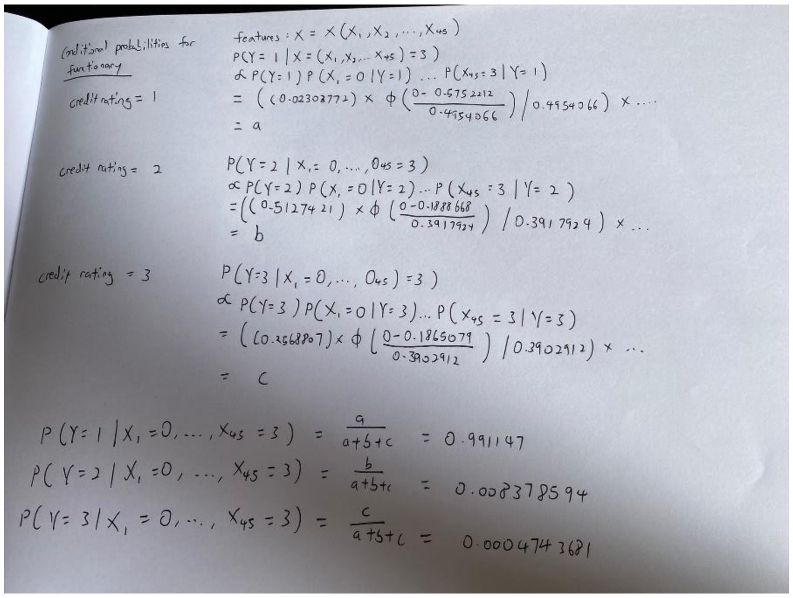
```
> nb_default

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 
```

The reported probabilities for 1 ,2 and 3 is shown above.

(b) (1 mark) Reproduce the first 20 or so lines of the R output for the Naive Bayes fit, and use them to explain the steps involved in making this prediction



```
Levels: 1 2 3
> nb_default

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
```



Conditional probabilities for functionary

features : $X = X(X_1, X_2, \dots, X_{45})$

credit rating = 1

$P(Y=1 \mid X = (X_1, X_2, \dots X_{45}) = 3)$

$\propto P(Y=1)\, P(X_1 = 0 \mid Y=1) \dots P(X_{45} = 3 \mid Y=1)$

$= \left((0.02303772) \times \phi\left(\frac{0 - 0.5752212}{0.4954066}\right) / 0.4954066 \right) \times \dots$

$= a$

credit rating = 2

$P(Y=2 \mid X_1 = 0, \dots, 0_{45} = 3)$

$\propto P(Y=2)\, P(X_1 = 0 \mid Y=2) \dots P(X_{45} = 3 \mid Y=2)$

$= \left((0.5127421) \times \phi\left(\frac{0 - 0.1888668}{0.3917924}\right) / 0.3917924 \right) \times \dots$

$= b$

credit rating = 3

$P(Y=3 \mid X_1 = 0, \dots, 0_{45}) = 3)$

$\propto P(Y=3)\, P(X_1 = 0 \mid Y=3) \dots P(X_{45} = 3 \mid Y=3)$

$= \left((0.2568807) \times \phi\left(\frac{0 - 0.1865079}{0.3902912}\right) / 0.3902912 \right) \times \dots$

$= c$

$P(Y=1 \mid X_1 = 0, \dots, X_{43} = 3) = \frac{a}{a+b+c} = 0.991147$

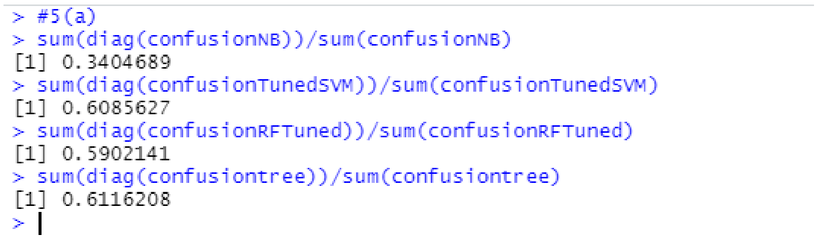
$P(Y=2 \mid X_1 = 0, \dots, X_{45} = 3) = \frac{b}{a+b+c} = 0.008378594$

$P(Y=3 \mid X_1 = 0, \dots, X_{45} = 3) = \frac{c}{a+b+c} = 0.0004743681$

Question 5

Based on the confusion matrices reported in the preceding parts,

(a) (0.5 mark) Which of the classifiers look to be the best? (Be specific, and specify the figures you used to answer this question.)



```
> #5(a)
> sum(diag(confusionNB))/sum(confusionNB)
[1] 0.3404689
> sum(diag(confusionTunedSVM))/sum(confusionTunedSVM)
[1] 0.6085627
> sum(diag(confusionRFTuned))/sum(confusionRFTuned)
[1] 0.5902141
> sum(diag(confusiontree))/sum(confusiontree)
[1] 0.6116208
> |
```

Decision tree classifiers seems to be the most accurate at 0.6116208.

RF is tuned with 900 ntrees

```
RFTuned.cw.train = randomForest(credit.rating~., data = cw.train, mtry
                    = 12, ntree=900, stepFactor=2, improve=0.2)
```

SVM is tuned with 200 cost and gamma set at 0.0200 with tune control sampling at fix.

```
svmTuned = svm(credit.rating ~ ., data = cw.train, kernel = "radial",
        cost=170,
        gamma = 0.0003,
        tunecontrol = tune.control(sampling = "fix"))
```

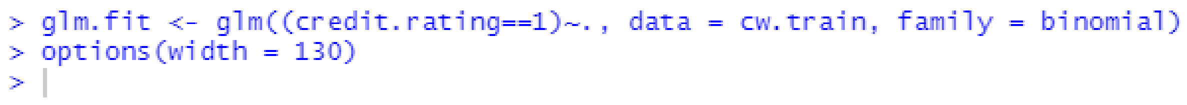
(b) (0.5 mark) Are there any categories that all classifiers seem to have trouble with?

All classifiers seem to have problem with ordinal categories. As we can seen with this assignment, the classifiers have very poor accuracy on the ordinal features and our nominal target. I believe it could be because of the inherent order on my target, credit rating. When I perform these classification algorithms on these ordinal data, I am assigning the same penalty whenever my classifier predicts a wrong class, no matter which one. Example, I want to predict credit rating: 1, 2, 3. My prediction is 2 when the true label is 1. I got it wrong however I didn't get it as wrong as if I predicted a 3.

Question 6

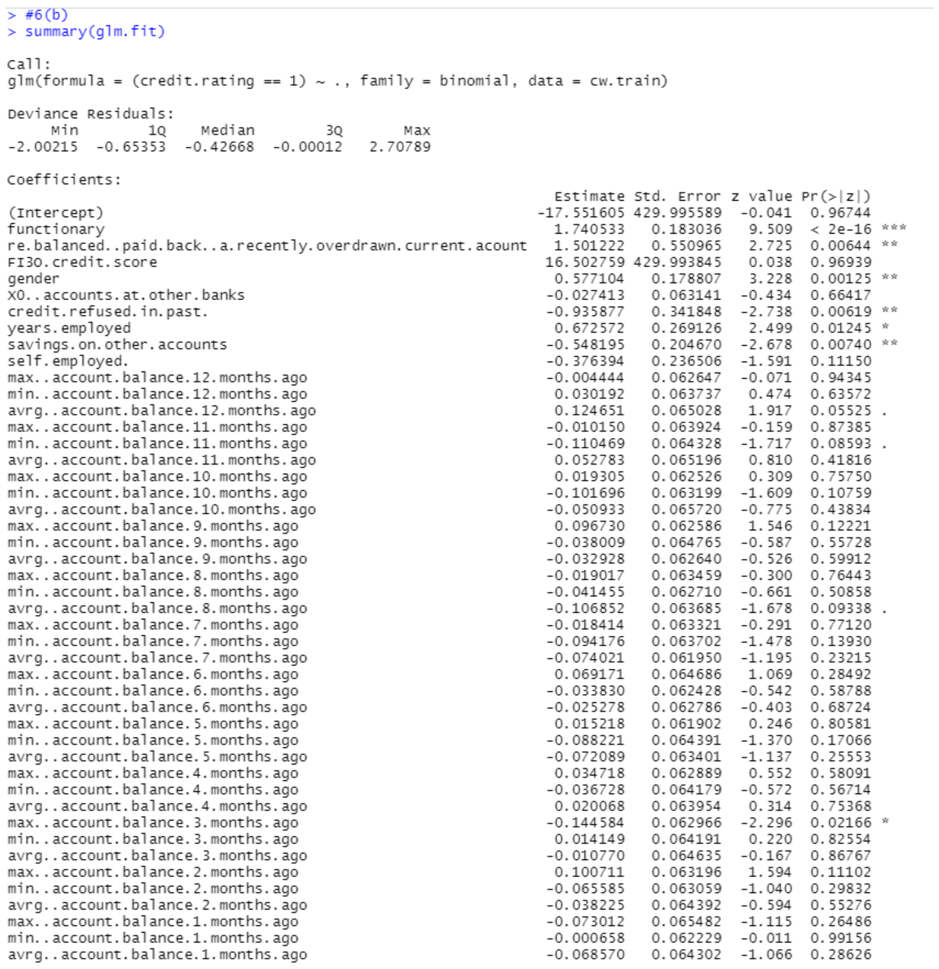
(2.5 marks) Consider a simpler problem of predicting whether a customer gets a credit rating of A or not.

(a) (0 mark) Fit a logistic regression model to predict whether a customer gets a credit rating of A using all of the other variables in the dataset, with no interactions.

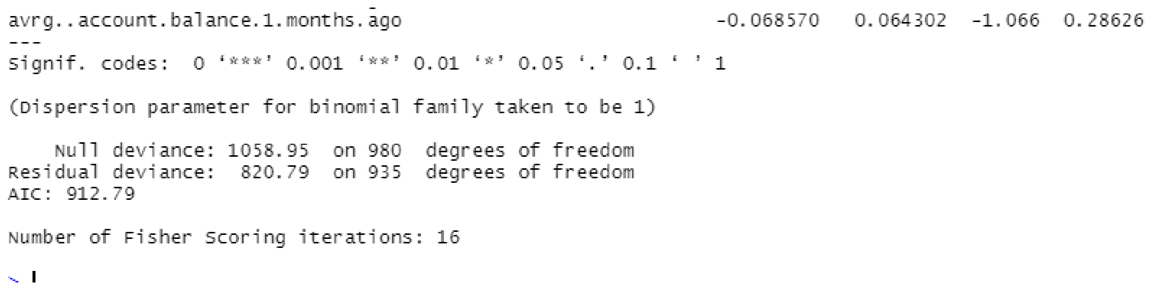


```
> glm.fit <- glm((credit.rating==1)~., data = cw.train, family = binomial)
> options(width = 130)
> 
```

(b)(0.5 mark) Report the summary table of the logistic regression model fit.



```
> #6(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
```



```
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) (0.5 mark) Which predictors of credit rating appear to be significant? Which of them are likely to be spuriously so?

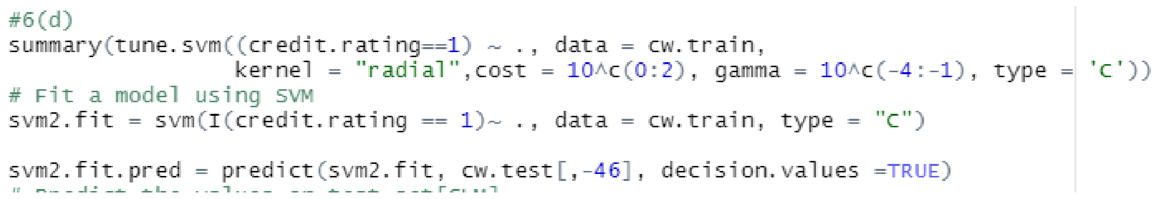
We can look at the Pr(>|z|) . The p-value Pr(>|z|) tells us the probability associated with a particular z value. This essentially tells us how well each predictor variable is able to predict the value of the response variable in the model.

The predictors with p-values that are worth mentioning,
FI3O.credit.score , 0.96939
max..account.balance.12.months.ago, 0.94345
max..account.balance.11.months.ago, 0.87385
max..account.balance.10.months.ago ,0.75750
max..account.balance.8.months.ago ,0.76443
max..account.balance.7.months.ago ,0.77120
max..account.balance.5.months.ago ,0.80581
min..account.balance.3.months.ago 0.82554
avrg..account.balance.3.months.ago 0.86767
min..account.balance.1.months.ago 0.99156

The predictos with p-values that are spurious.
credit.refused.in.past at 0.00619
years.employed at 0.01245
savings.on.other.accounts at 0.00740

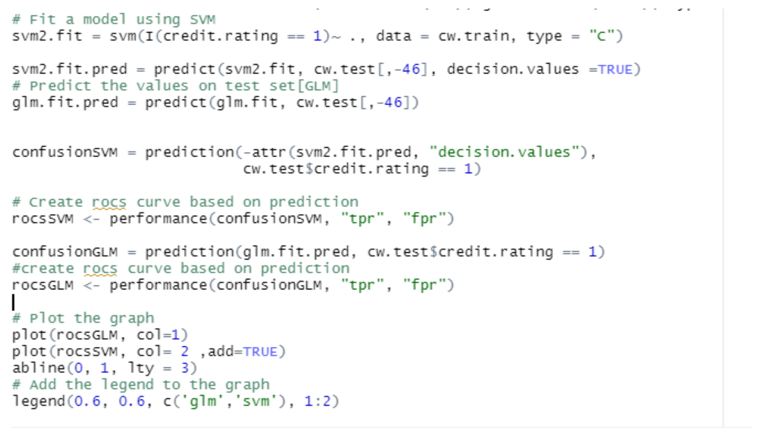
I believe these 3 predictors should be more significant than reported by the p-value.

(d)(0 mark) Fit an SVM model of your choice to the training set.

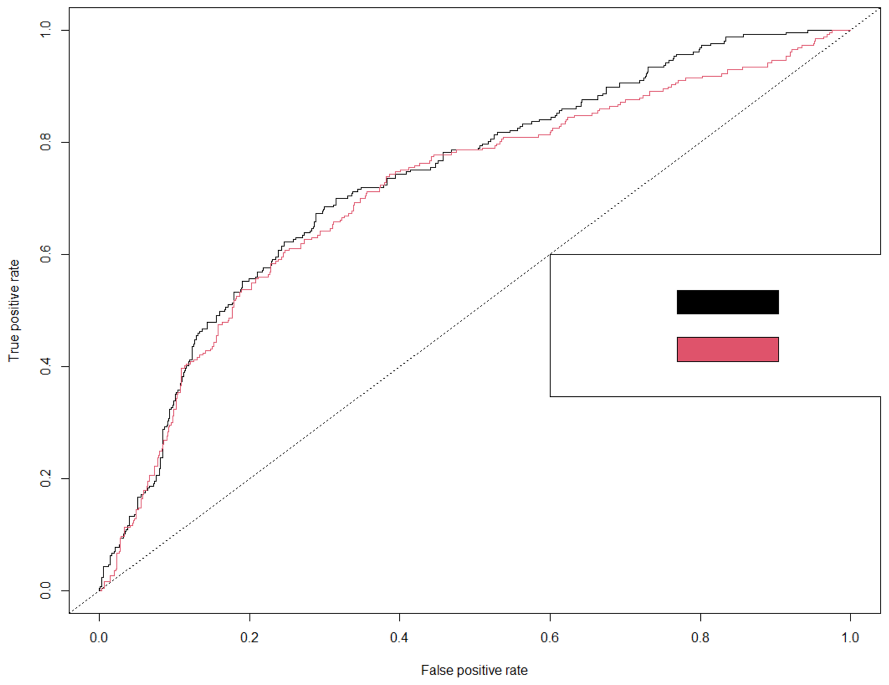


```
#6(d)
summary(tune.svm((credit.rating==1) ~ ., data = cw.train,
                 kernel = "radial",cost = 10^c(0:2), gamma = 10^c(-4:-1), type = 'C'))
# Fit a model using SVM
svm2.fit = svm(I(credit.rating == 1)~ ., data = cw.train, type = "C")

svm2.fit.pred = predict(svm2.fit, cw.test[,-46], decision.values =TRUE)
```

(e)(1.5 marks) Produce an ROC chart comparing the logistic regression and the SVM results of predicting the test set. Comment on any differences in their performance.



```
# Fit a model using SVM
svm2.fit = svm(I(credit.rating == 1)~ ., data = cw.train, type = "C")

svm2.fit.pred = predict(svm2.fit, cw.test[,-46], decision.values =TRUE)
# Predict the values on test set[GLM]
glm.fit.pred = predict(glm.fit, cw.test[,-46])


confusionSVM = prediction(-attr(svm2.fit.pred, "decision.values"),
                          cw.test$credit.rating == 1)

# Create rocs curve based on prediction
rocsSVM <- performance(confusionSVM, "tpr", "fpr")

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)
```



Logistic regression is represented by the black line while red line is SVM. We can see that logistic regression performs better as it curves closer to the top-left corner. SVM gets closer to the baseline ROC space as FPR and TPR increases where else Logistic regression does not. Both did not turn out to be perfect classifiers.