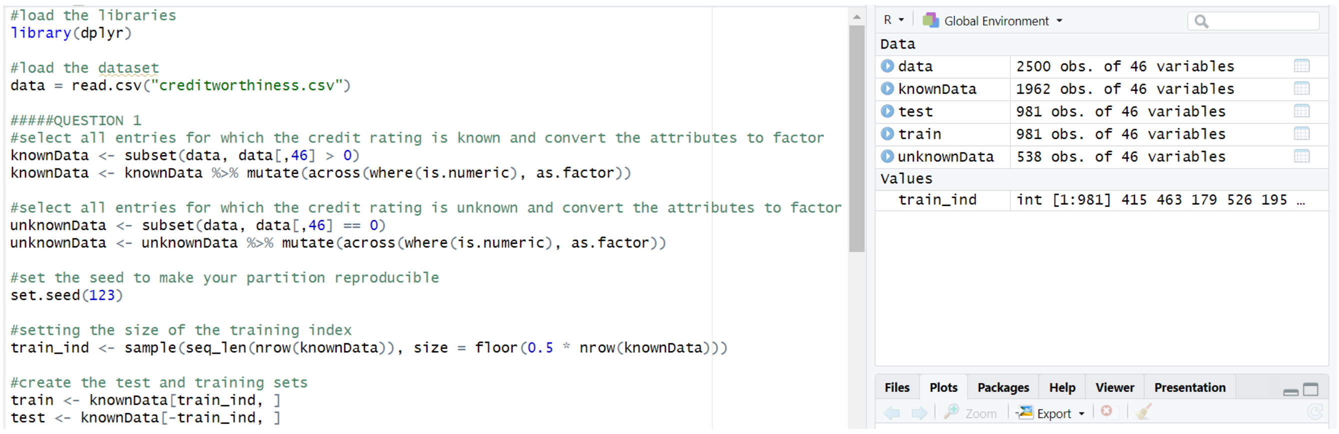
Question 1

The code for splitting the dataset into two sets, half for training and half for testing with only the data with known ratings is shown below as well as in Figure 1 where the number of rows for each set is shown as well.

```r
#load the libraries
library(dplyr)

#load the dataset
data = read.csv("creditworthiness.csv")

#select all entries for which the credit rating is known and convert the attributes to factor
knownData <- subset(data, data[,46] > 0)
knownData <- knownData %>% mutate(across(where(is.numeric), as.factor))

#select all entries for which the credit rating is unknown and convert the attributes to factor
unknownData <- subset(data, data[,46] == 0)
unknownData <- unknownData %>% mutate(across(where(is.numeric), as.factor))

#set the seed to make your partition reproducible
set.seed(123)

#setting the size of the training index
train_ind <- sample(seq_len(nrow(knownData)), size = floor(0.5 * nrow(knownData)))

#create the test and training sets
train <- knownData[train_ind, ]
test <- knownData[-train_ind, ]
```



```r
#load the libraries
library(dplyr)

#load the dataset
data = read.csv("creditworthiness.csv")

#####QUESTION 1
#select all entries for which the credit rating is known and convert the attributes to factor
knownData <- subset(data, data[,46] > 0)
knownData <- knownData %>% mutate(across(where(is.numeric), as.factor))

#select all entries for which the credit rating is unknown and convert the attributes to factor
unknownData <- subset(data, data[,46] == 0)
unknownData <- unknownData %>% mutate(across(where(is.numeric), as.factor))

#set the seed to make your partition reproducible
set.seed(123)

#setting the size of the training index
train_ind <- sample(seq_len(nrow(knownData)), size = floor(0.5 * nrow(knownData)))

#create the test and training sets
train <- knownData[train_ind, ]
test <- knownData[-train_ind, ]
```

```
R ▾   Global Environment ▾
Data
  data          2500 obs. of 46 variables
  knownData     1962 obs. of 46 variables
  test          981 obs. of 46 variables
  train         981 obs. of 46 variables
  unknownData   538 obs. of 46 variables
Values
  train_ind     int [1:981] 415 463 179 526 195 …
```

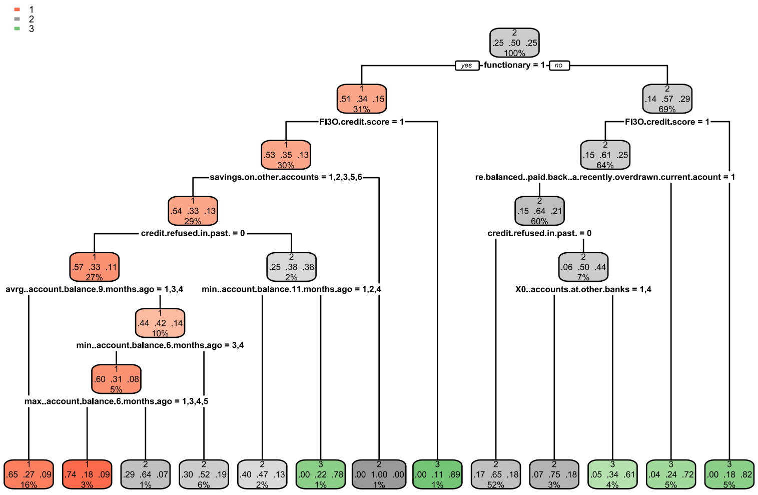
Figure 1: *Code and Result for Data Split*

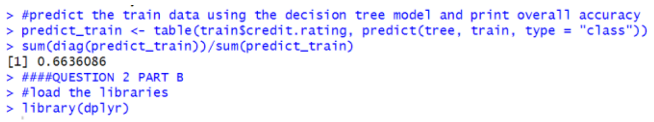
Since there are 1962 data with known credit rating, half of them is 981. Therefore, the training and test sets have 981 data each as shown in Figure 1.

QUESTION 2

a) The code for the resulting tree is shown below. To visualize the resulted tree, the rpart.plot() function was used to print out the tree and the tree is shown in Figure 2. The overall accuracy for predicting the training set is shown in Figure 2 as well.

```r
#load the libraries
library(caret)
library(rpart)
library(rpart.plot)

#create a decision tree with rpart and print it
tree <- rpart(credit.rating ~., data = train)
rpart.plot(tree)

#predict the train data using the decision tree model and print overall accuracy
predict_train <- table(train$credit.rating, predict(tree, train, type = "class"))
sum(diag(predict_train))/sum(predict_train)
```





```
> #predict the train data using the decision tree model and print overall accuracy
> predict_train <- table(train$credit.rating, predict(tree, train, type = "class"))
> sum(diag(predict_train))/sum(predict_train)
[1] 0.6636086
> ####QUESTION 2 PART B
> #load the libraries
> library(dplyr)
```

Figure 2: *Resulting Decision Tree and Overall Accuracy for Predicting the Training Set*

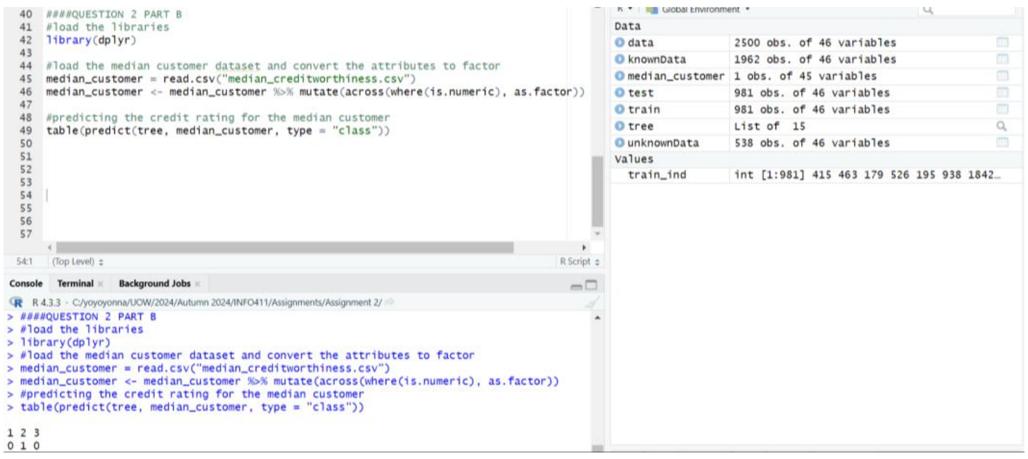
Figure 2 illustrates the resulted decision tree. As shown in the figure, there are only 10 attributes shown in the tree. This is because the tree considers them the most important attributes that have the most significant effects on the classification of the credit rating decision. Each node is plotted in the shape of a box and each node has a predicted class (1, 2, or 3), predicted probabilities for the classes, and an observation percentage. Each predicted class has its own colour. From the legend on the top left, we can deduce that the nodes with the red, blue, and green colours have classes 1, 2, and 3 as their predicted classes respectively. For example, the box of the root node (the first box at the top), is blue indicating that the predicted class is 2. This is because class 2 (0.5) has the highest predicted percentage at that node compared to class 1 (0.25) and class 2 (0.25). Hence, the root node is based on clients with class 2. It also has 100% observations since it is the root node. Splitting this node is based on functionary = 1 where the clients with functionary equal to 1 (31% of the observations) are placed in the left side of the tree and those that do not have functionary equal to 1 (69% of the observations) are placed on the right side of the tree. Same process occurs for every other node that follows.

By definition, a decision tree is balanced if the difference between the left and right trees' heights for any node is no more than 1, and if the right and left subtrees are balanced (GeeksforGeeks 2022). Hence, we can deduce that this decision tree is unbalanced by 2 as the left subtree of the root node has a height of 8 while that of the right subtree is 6. This indicates that there is a class imbalance (Farinda 2019). The dataset itself is also small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance.

When using the decision tree to predict the credit rating on the training set, the overall accuracy rate as shown in the Figure 2 is around 0.6636 (66.36% (0. 6636 * 100)) which is a reasonable accuracy considering the class imbalance in the dataset.

b) The code for predicting the credit rating of the median customer is shown below and the output is shown in Figure 3.

```r
#load the libraries
library(dplyr)

#load the median customer dataset and convert the attributes to factor
median_customer = read.csv("median_creditworthiness.csv")
median_customer <- median_customer %>% mutate(across(where(is.numeric), as.factor))

#predicting the credit rating for the median customer
table(predict(tree, median_customer, type = "class"))
```



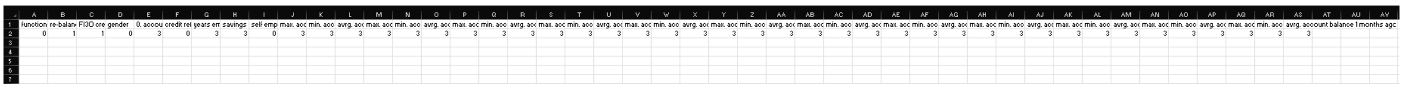
```r
####QUESTION 2 PART B
#load the libraries
library(dplyr)

#load the median customer dataset and convert the attributes to factor
median_customer = read.csv("median_creditworthiness.csv")
median_customer <- median_customer %>% mutate(across(where(is.numeric), as.factor))

#predicting the credit rating for the median customer
table(predict(tree, median_customer, type = "class"))
```

```
R 4.3.3 · C:/yoyoyonna/UOW/2024/Autumn 2024/INFO411/Assignments/Assignment 2/
> ####QUESTION 2 PART B
> #load the libraries
> library(dplyr)
> #load the median customer dataset and convert the attributes to factor
> median_customer = read.csv("median_creditworthiness.csv")
> median_customer <- median_customer %>% mutate(across(where(is.numeric), as.factor))
> #predicting the credit rating for the median customer
> table(predict(tree, median_customer, type = "class"))

1 2 3
0 1 0
```

```
Data
data              2500 obs. of 46 variables
knownData         1962 obs. of 46 variables
median_customer   1 obs. of 45 variables
test              981 obs. of 46 variables
train             981 obs. of 46 variables
tree              List of  15
unknownData       538 obs. of 46 variables
Values
  train_ind       int [1:981] 415 463 179 526 195 938 1842…
```

Figure 3: *Code and Result for Predicting the Credit Rating for the Median Customer*

To predict the credit rating of the median customer, I put the data of the client provided in Table 1 in the assignment specification into a .CSV file that I named as median_creditworthiness as shown in Figure 4. Then, I loaded the file and converted the data into factor as shown in the above code and in Figure 3 where only 1 observation was recorded as shown in Figure 3. Then, I used the predict() and table() functions to output the predicted credit rating as shown in Figure 3. The output shows that the median customer has a credit rating 2.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	function	re-balan	FI3O cre	gender	0. accou	credit rel	years em	savings	self emp	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc
2	0	1	1	0	3	0	3	3	0	3	3	3	3	3	3
3															
4															
5															
6															
7															

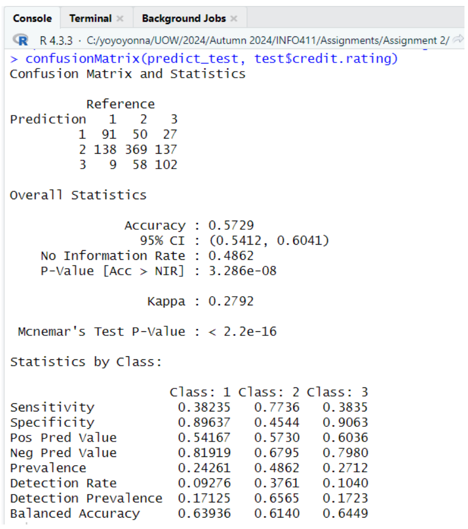
	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc
2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
3															
4															
5															
6															
7															

	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ	AR	AS	AT	AU	AV
1	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. acc	max. acc	min. acc	avrg. account balance 1months agc			
2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3			
3																		
4																		
5																		
6																		
7																		

Figure 4: *median_creditworthiness.csv's Contents*

c) The code for the confusion matrix is shown below and the output as well as the overall accuracy of the confusion matrix are shown in Figure 5.

```r
#load the libraries
library(caret)

#predict on the test data
predict_test <- predict(tree, test, type = "class")

#print the confusion matrix and statistics (including overall accuracy)
confusionMatrix(predict_test, test$credit.rating)
```


Console Terminal Background Jobs

R 4.3.3 · C:/yoyoyonna/UOW/2024/Autumn 2024/INFO411/Assignments/Assignment 2/

```
> confusionMatrix(predict_test, test$credit.rating)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1  91  50  27
         2 138 369 137
         3   9  58 102

Overall Statistics

               Accuracy : 0.5729
                 95% CI : (0.5412, 0.6041)
    No Information Rate : 0.4862
    P-Value [Acc > NIR] : 3.286e-08

                  Kappa : 0.2792

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity           0.38235   0.7736   0.3835
Specificity           0.89637   0.4544   0.9063
Pos Pred Value        0.54167   0.5730   0.6036
Neg Pred Value        0.81919   0.6795   0.7980
Prevalence            0.24261   0.4862   0.2712
Detection Rate        0.09276   0.3761   0.1040
Detection Prevalence  0.17125   0.6565   0.1723
Balanced Accuracy     0.63936   0.6140   0.6449
```

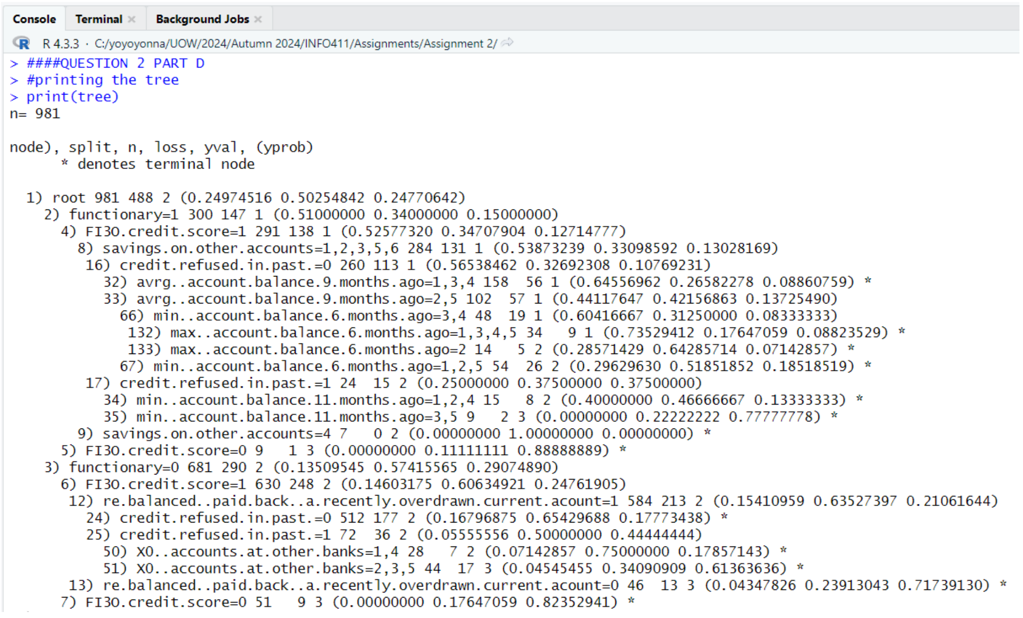
Figure 5: *Code, Output, and Overall Accuracy of the Confusion Matrix*

The confusion matrix for predicting the test set is shown in Figure 5. As illustrated, the decision tree was able to correctly classify 562 (diagonal sum of the true positive data (data that is positive and was predicted as such (Mitrani 2019)) (91 + 369 + 102)) out of the 981 data. However, it incorrectly classified 138 and 9 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 1 as class 2 and class 3 respectively. In addition, it also incorrectly classified 50 and 58 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 2 as class 1 and class 3 respectively. Furthermore, it also incorrectly classified 27 and 137 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 3 as class 1 and class 2 respectively.

Because of all of these inaccuracies, the overall accuracy rate of the decision tree is shown in the Figure 5 as 0.5729 (57.29%). This accuracy rate is calculated as the diagonal sum (which we have already calculated as 562) divided by the total number of observations which is 981. It is also almost 10% lower than when the decision tree was used to predict the training set (66.36%) which indicates that the model may not be good at predicting data it has not seen before. It is also pretty low indicating that the error rate is 42.71% (100% - 57.29%). This is because the decision tree had a hard time classifying classes 1 and 3 especially class 1 as it had only 91 data classified correctly out of 238 (91 + 138 + 9) indicating that 0.38235 (91 / 238) or 38.235% were classified correctly, while class 3 had 102 data classified correctly out of 266 (102 + 27 + 137) indicating that 0.3835 (102 / 266) or 38.35% were classified correctly. Class 2, on the other hand, had 369 data classified correctly out of 477 (50 + 369 + 58) indicating that 0.7736 (369 / 477) or 77.36% were classified correctly. These values represent the sensitivity of the model to each class. The results again indicates that the model is biased towards class 2 indicating that there is class imbalance (Farinda 2019) and as mentioned before, the dataset itself is small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance.

d) The code for printing the decision tree illustrated in Figure 2 is shown below and the output is shown in Figure 6. The creation of the tree is shown above in the code for Question 2a.

```
#printing the tree
print(tree)
```


Console
Terminal
Background Jobs
R 4.3.3 · C:/yoyoyonna/UOW/2024/Autumn 2024/INFO411/Assignments/Assignment 2/

```
> ####QUESTION 2 PART D
> #printing the tree
> print(tree)
n= 981

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 981 488 2 (0.24974516 0.50254842 0.24770642)
    2) functionary=1 300 147 1 (0.51000000 0.34000000 0.15000000)
      4) FI30.credit.score=1 291 138 1 (0.52577320 0.34707904 0.12714777)
        8) savings.on.other.accounts=1,2,3,5,6 284 131 1 (0.53873239 0.33098592 0.13028169)
         16) credit.refused.in.past.=0 260 113 1 (0.56538462 0.32692308 0.10769231)
           32) avrg..account.balance.9.months.ago=1,3,4 158  56 1 (0.64556962 0.26582278 0.08860759) *
           33) avrg..account.balance.9.months.ago=2,5 102  57 1 (0.44117647 0.42156863 0.13725490)
             66) min..account.balance.6.months.ago=3,4 48  19 1 (0.60416667 0.31250000 0.08333333)
              132) max..account.balance.6.months.ago=1,3,4,5 34   9 1 (0.73529412 0.17647059 0.08823529) *
              133) max..account.balance.6.months.ago=2 14   5 2 (0.28571429 0.64285714 0.07142857) *
             67) min..account.balance.6.months.ago=1,2,5 54  26 2 (0.29629630 0.51851852 0.18518519) *
         17) credit.refused.in.past.=1 24  15 2 (0.25000000 0.37500000 0.37500000)
           34) min..account.balance.11.months.ago=1,2,4 15   8 2 (0.40000000 0.46666667 0.13333333) *
           35) min..account.balance.11.months.ago=3,5 9   2 3 (0.00000000 0.22222222 0.77777778) *
        9) savings.on.other.accounts=4 7   0 2 (0.00000000 1.00000000 0.00000000) *
      5) FI30.credit.score=0 9   1 3 (0.00000000 0.11111111 0.88888889) *
    3) functionary=0 681 290 2 (0.13509545 0.57415565 0.29074890)
      6) FI30.credit.score=1 630 248 2 (0.14603175 0.60634921 0.24761905)
       12) re.balanced..paid.back..a.recently.overdrawn.current.acount=1 584 213 2 (0.15410959 0.63527397 0.21061644)
         24) credit.refused.in.past.=0 512 177 2 (0.16796875 0.65429688 0.17773438) *
         25) credit.refused.in.past.=1 72  36 2 (0.05555556 0.50000000 0.44444444)
           50) X0..accounts.at.other.banks=1,4 28   7 2 (0.07142857 0.75000000 0.17857143) *
           51) X0..accounts.at.other.banks=2,3,5 44  17 3 (0.04545455 0.34090909 0.61363636) *
       13) re.balanced..paid.back..a.recently.overdrawn.current.acount=0 46  13 3 (0.04347826 0.23913043 0.71739130) *
      7) FI30.credit.score=0 51   9 3 (0.00000000 0.17647059 0.82352941) *
```

Figure 6: *Code and Output for Printing the Decision Tree*

$D_{ent} \;=\; -\sum_{k=1}^{K} \hat{p}_{mk}\, \log_2(\hat{p}_{mk})$ where $\hat{p}_{mk}$ is the proportion of class k observations from the node m.

$$
\begin{aligned}
\textit{Total Entropy in Root Node} &= \left(-\frac{981-488}{981} \times \log_2\left(\frac{981-488}{981}\right)\right) - \left(\frac{488}{981} \times \log_2\left(\frac{488}{981}\right)\right)\\
&= \left(-\frac{493}{981} \times \log_2\left(\frac{493}{981}\right)\right) - \left(\frac{488}{981} \times \log_2\left(\frac{488}{981}\right)\right)\\
&\approx (-0.50254842 \times -0.9926654898) - (0.49745158 \;\times\; -1.0073719887)\\
&\approx 0.498862473487516116 + 0.501118787426557146\\
&\approx 0.999981260914\; bits
\end{aligned}
$$

$$
\begin{aligned}
\textit{Entropy in Left Daughter Node After First Split} &= \left(-\frac{300-147}{300} \times \log_2\left(\frac{300-147}{300}\right)\right) - \left(\frac{147}{300} \times \log_2\left(\frac{147}{300}\right)\right)\\
&= \left(-\frac{153}{300} \times \log_2\left(\frac{153}{300}\right)\right) - \left(\frac{147}{300} \times \log_2\left(\frac{147}{300}\right)\right)\\
&\approx (-0.51 \times -0.9714308478) - (0.49 \;\times\; -1.0291463457)\\
&\approx 0.495429732378 + 0.504281709393\\
&\approx 0.999711441771\; bits
\end{aligned}
$$

$$
\begin{aligned}
\textit{Entropy in Right Daughter Node After First Split} &= \left(-\frac{681-290}{681} \times \log_2\left(\frac{681-290}{681}\right)\right) - \left(\frac{290}{681} \times \log_2\left(\frac{290}{681}\right)\right)\\
&= \left(-\frac{391}{681} \times \log_2\left(\frac{391}{681}\right)\right) - \left(\frac{290}{681} \times \log_2\left(\frac{290}{681}\right)\right)\\
&\approx (-0.5741556535 \times -0.8004861906) - (0.4258443465 \;\times\; -1.2316018982)\\
&\approx 0.459603671882 + 0.524470705487\\
&\approx 0.984074377369\; bits
\end{aligned}
$$

The total entropy after first split is calculated using the number of instances down each branch as weight factor:

$$
\begin{aligned}
\textit{Total Entropy After First Split} \;&= \left(\frac{300}{981} \times 0.999711441771\right) + \left(\frac{681}{981} \times 0.984074377369\right)\\
&\approx (0.305810397554 \times 0.999711441771) + (0.694189602446 \;\times\; 0.984074377369)\\
&\approx 0.305722153447 + 0.683134200803\\
&\approx 0.98885635425\; bits
\end{aligned}
$$

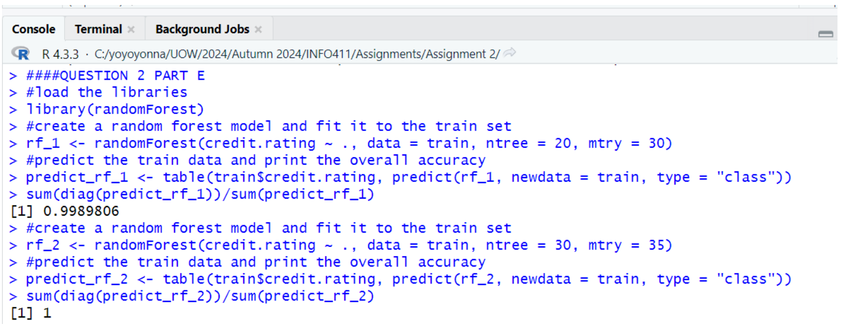
This value (0.98885635425) is smaller than the total entropy in the root node (0.999981260914). Hence, the information gain following the split is:

$$
\begin{aligned}
Gain &\approx 0.999981260914 - 0.98885635425 \\
&\approx 0.011124906664\ bits
\end{aligned}
$$

The entropies (impurities in the dataset) before and after splitting are almost 1 and the information gain or IG (which is a value between 0 and 1 with 0 as the lowest and 1 as the highest) is almost 0.01 indicating that the splitting of the nodes is not good as low information gain indicates less entropies removed and vice versa (Amelia 2019; Krishnan 2021; Zhou 2022).

e) The code for the random forest models is shown below and the overall accuracies are shown in Figure 7.

```
#load the libraries
library(randomForest)

#create a random forest model and fit it to the train set
rf_1 <- randomForest(credit.rating ~ ., data = train, ntree = 20, mtry = 30)

#predict the train data and print the overall accuracy
predict_rf_1 <- table(train$credit.rating, predict(rf_1, newdata = train, type = "class"))
sum(diag(predict_rf_1))/sum(predict_rf_1)

#create a random forest model and fit it to the train set
rf_2 <- randomForest(credit.rating ~ ., data = train, ntree = 40, mtry = 40)

#predict the train data and print the overall accuracy
predict_rf_2 <- table(train$credit.rating, predict(rf_2, newdata = train, type = "class"))
sum(diag(predict_rf_2))/sum(predict_rf_2)
```


Console
Terminal
Background Jobs
R 4.3.3 · C:/yoyoyonna/UOW/2024/Autumn 2024/INFO411/Assignments/Assignment 2/

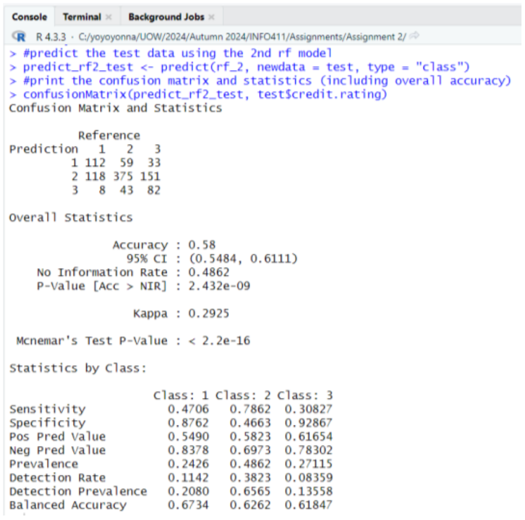
```
> ####QUESTION 2 PART E
> #load the libraries
> library(randomForest)
> #create a random forest model and fit it to the train set
> rf_1 <- randomForest(credit.rating ~ ., data = train, ntree = 20, mtry = 30)
> #predict the train data and print the overall accuracy
> predict_rf_1 <- table(train$credit.rating, predict(rf_1, newdata = train, type = "class"))
> sum(diag(predict_rf_1))/sum(predict_rf_1)
[1] 0.9989806
> #create a random forest model and fit it to the train set
> rf_2 <- randomForest(credit.rating ~ ., data = train, ntree = 30, mtry = 35)
> #predict the train data and print the overall accuracy
> predict_rf_2 <- table(train$credit.rating, predict(rf_2, newdata = train, type = "class"))
> sum(diag(predict_rf_2))/sum(predict_rf_2)
[1] 1
```

***Figure 7:** Random Forest Models' Outputs + Overall Accuracies*

I have created the above random forest models to predict the credit rating in the training set and inspected the models' overall accuracies in order to find the one with the highest accuracy. This was a trial-and-error process that involved changing the ntree and mtry values. The code provided above and in Figure 7, shows the models I created along with their accuracies. The first model (rf_1) had 20 and 30 as the ntree and mtry values respectively. However, it had the lowest accuracy reaching almost 0.999 or 99.9%. For the second model (rf_2), I tried increasing the ntree value to 40 and the mtry value to 40. This model achieved a 100% accuracy because the model is trained on this dataset and has seen it before. Hence, rf_2 will be used in the upcoming questions where required.

f) The code for the confusion matrices and the outputs are shown in below and Figure 8:

```
#load the libraries
library(caret)

#predict the test data using the 2nd rf model
predict_rf2_test <- predict(rf_2, newdata = test, type = "class")

#print the confusion matrix and statistics (including overall accuracy)
confusionMatrix(predict_rf2_test, test$credit.rating)
```



Console
Terminal
Background Jobs
R 4.3.3 · C:/yoyoyonna/UOW/2024/Autumn 2024/INFO411/Assignments/Assignment 2/

```
> #predict the test data using the 2nd rf model
> predict_rf2_test <- predict(rf_2, newdata = test, type = "class")
> #print the confusion matrix and statistics (including overall accuracy)
> confusionMatrix(predict_rf2_test, test$credit.rating)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 112  59  33
         2 118 375 151
         3   8  43  82

Overall Statistics

               Accuracy : 0.58
                 95% CI : (0.5484, 0.6111)
    No Information Rate : 0.4862
    P-Value [Acc > NIR] : 2.432e-09

                  Kappa : 0.2925

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.4706   0.7862  0.30827
Specificity            0.8762   0.4663  0.92867
Pos Pred Value         0.5490   0.5823  0.61654
Neg Pred Value         0.8378   0.6973  0.78302
Prevalence             0.2426   0.4862  0.27115
Detection Rate         0.1142   0.3823  0.08359
Detection Prevalence   0.2080   0.6565  0.13558
Balanced Accuracy      0.6734   0.6262  0.61847
```

Figure 8: *Confusion Matrices for Random Forest Model rf_2*

The confusion matrix for predicting the test set is shown in Figure 8. As illustrated, the random forest model (rf_2) was able to correctly classify 569 (diagonal sum of the true positive data (data that is positive and was predicted as such (Mitrani 2019)) (112 + 375 + 82)) out of the 981 data. However, it incorrectly classified 118 and 9 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 1 as class 2 and class 3 respectively. In addition, it also incorrectly classified 59 and 43 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 2 as class 1 and class 3 respectively. Furthermore, it also incorrectly classified 33 and 151 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 3 as class 1 and class 2 respectively.

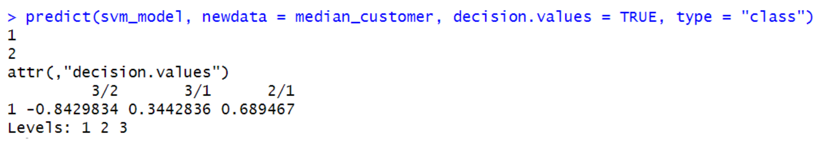
Because of all of these inaccuracies, the overall accuracy rate of the random forest is shown in the Figure 8 as 0.58 (58%). This accuracy rate is calculated as the diagonal sum (which we have already calculated as 569) divided by the total number of observations which is 981. It is also 42% lower than when the random forest model was used to predict the training set (100%) which indicates that the model may not be good at predicting data it has not seen before and that the model overfitted. It is also pretty low indicating that the error rate is 42% (100% - 58%). This is because the random forest had a hard time classifying classes 1 and 3 especially class 3 as it had only 82 data classified correctly out of 266 (33 + 151 + 82) indicating that 0.3083 (82 / 266) or 30.83% were classified correctly, while class 1 had 112 data classified correctly out of 238 (112 + 118 + 8) indicating that 0.4706 (112 / 238) or 47.06% were classified correctly. Class 2, on the other hand, had 375 data classified correctly out of 477 (59 + 75 + 43) indicating that 0.7862 (375 / 477) or 78.62% were classified correctly. The results again indicates that the model is biased towards class 2 indicating that there is class imbalance (Farinda 2019) and as mentioned before, the dataset itself is small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance.

Compared to the standard decision tree, the overall accuracy of the random forest model (100%) when it was fit to the training set is much higher than that of the standard decision tree (66.36%). On the other hand, when the decision tree and random forest models were fit to the test set, the accuracies dropped to 0.5729 and 0.58 respectively (refer to Figures 5 and 8) with the accuracy of the random forest model higher than that of the decision tree model by about 1% indicating that the models overfitted. When it comes to the sensitivity (ability to predict true positives) and specificity (ability to predict true negatives) of the models to classes 1, 2, and 3 (refer to Figures 5 and 8), both models have very high sensitivities but low specificities to class 2 compared to classes 1 and 3. The random forest model has higher sensitivity to class (0.4706) and class 2 (0.7862) compared to the decision tree model that has 0.38235 and 0.7736 sensitivity for classes 1 and 2 respectively. However, the decision tree model is more sensitive to class 3 (0.3835) compared to the random forest model (0.30827). This again indicates that the random forest model has a hard time classifying class 3 compared to the other classes while the decision tree model has a hard time classifying class 1 compared to the other classes. On the other hand, the specificities of the random forest model for classes 1, 2, and 3 are 0.8762, 0.4663, and 92867 respectively. Whereas those of the decision tree are 0.89637, 0.4544, 0.9063 for classes 1, 2, and 3 respectively. This indicates that the random forest model has higher specificities to classes 2 and 3 but lower specificity to class 1 compared to the decision tree model which is expected as the opposite happens with the sensitivity.

QUESTION 3

a) The code for predicting the credit rating of the median customer is shown below and the output is shown in Figure 9.

```
#load the libraries
library(e1071)

#creating the svm model
svm_model <- svm(credit.rating ~ ., data = train, probability = TRUE)

#predicting the credit rating for the median customer
predict(svm_model, newdata = median_customer, decision.values = TRUE, type = "class")
```



```
> predict(svm_model, newdata = median_customer, decision.values = TRUE, type = "class")
1
2
attr(,"decision.values")
         3/2       3/1      2/1
1 -0.8429834 0.3442836 0.689467
Levels: 1 2 3
```

Figure 9: *Predicting Median Customer using SVM*

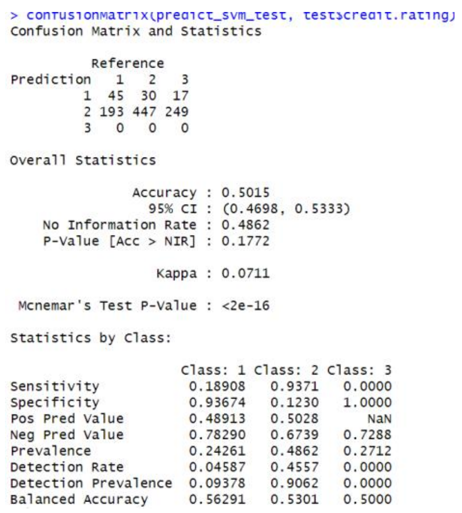
I used the dataset I created in question 1b to predict the credit rating of the median customer. As it is shown in Figure 9 and as the decision tree model predicted, the median customer has a credit rating 2.

In a svm model, decision values represent the distance of a data point from the decision boundary \ hyperplane . These decision values can help understand the confidence of the svm model in its predictions. From the output shown Figure 9, we can deduce that the svm model created 3 hyperplanes:

- **3/2 Decision Value:** represents the distance of the data point from the hyperplane between classes 3 and 2. The negative value “-0.8429834” indicates that the data point lies on the side of class 2.
- **3/1 Decision Value:** represents the distance of the data point from the hyperplane between classes 3 and 1. The positive value “0.3442836” indicates that the data point lies on the side of class 3.
- **2/1 Decision Value:** represents the distance of the data point from the hyperplane between classes 2 and 1. The positive value “0.689467” indicates that the data point is on the side of class 2. Since the “0.689467” is the highest, and it’s on class 2’s side, the model predicted that the median customer has therefore a credit rating of 2.

b) The code for the confusion matrix and the accuracy is shown below and the output is shown in Figure 10.

```
#load the libraries
library(caret)

#predict on the test data
predict_svm_test <- predict(svm_model, test, type = "class")

#print the confusion matrix and statistics (including overall accuracy)
confusionMatrix(predict_svm_test, test$credit.rating)
```



```
> confusionMatrix(predict_svm_test, test$credit.rating)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1  45  30  17
         2 193 447 249
         3   0   0   0

Overall Statistics

               Accuracy : 0.5015
                 95% CI : (0.4698, 0.5333)
    No Information Rate : 0.4862
    P-Value [Acc > NIR] : 0.1772

                  Kappa : 0.0711

 Mcnemar's Test P-Value : <2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity           0.18908   0.9371   0.0000
Specificity           0.93674   0.1230   1.0000
Pos Pred Value        0.48913   0.5028      NaN
Neg Pred Value        0.78290   0.6739   0.7288
Prevalence            0.24261   0.4862   0.2712
Detection Rate        0.04587   0.4557   0.0000
Detection Prevalence  0.09378   0.9062   0.0000
Balanced Accuracy     0.56291   0.5301   0.5000
```

Figure 10: *Confusion Matrix + Overall Accuracy for Predicting the Credit Rating from the Test Set Using SVM*

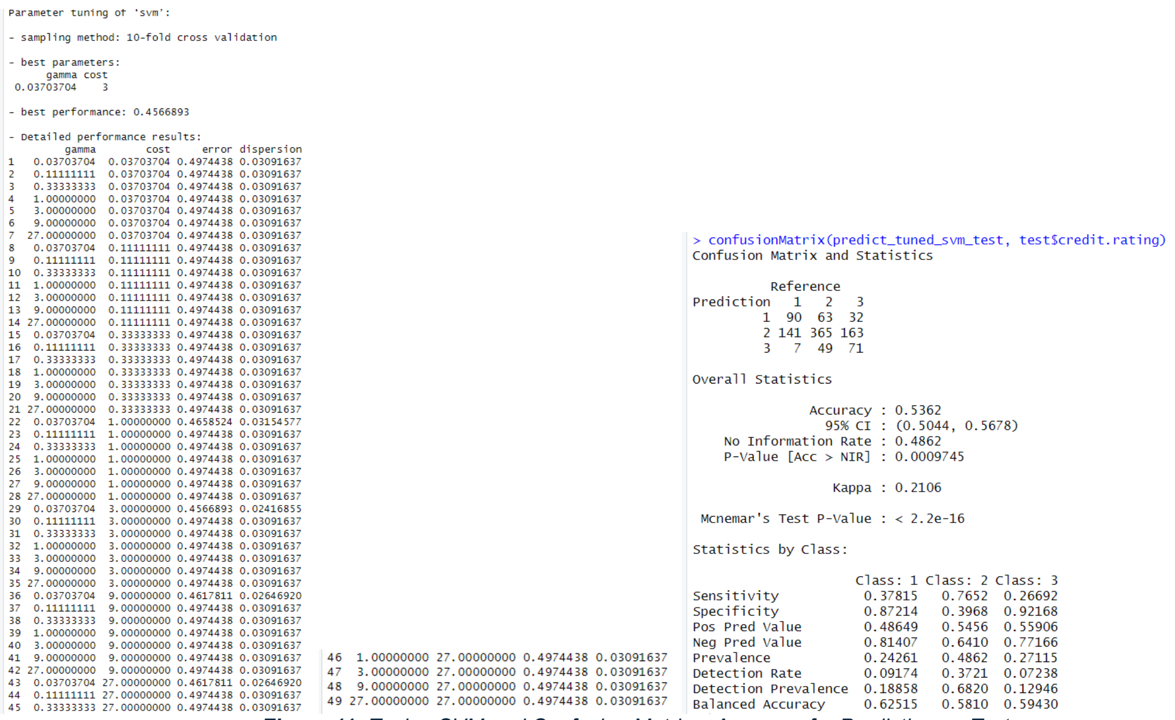
The confusion matrix for predicting the credit rating from test set is shown in Figure 10. As illustrated, the svm model was able to correctly classify 492 (diagonal sum of the true positive data (data that is positive and was predicted as such (Mitrani 2019)) (45 + 447)) out of the 981 data which is less than the decision tree's 562 and the random forest's 569. The svm model incorrectly classified 193 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 1 as class 2. In addition, it also incorrectly classified 30 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 2 as class 1. Furthermore, it also incorrectly classified 17 and 249 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 3 as class 1 and class 2 respectively. An interesting observation is that the svm model did not incorrectly classify any class 1 or 2 data as class 3. However, it wrongly classified all of class 3's data as class 1 and 2 especially class 2. This indicates that the model may need to be tuned in order to work better with the imbalance in the data.

Because of all of these inaccuracies, the overall accuracy rate of the svm is shown in the Figure 10 as 0.5015 (50.15%) which again is less than those of the decision tree model and the random forest model. This accuracy rate is calculated as the diagonal sum (which we have already calculated as 492) divided by the total number of observations which is 981. It is also pretty low indicating that the error rate is around 50% (100% - 50.15%). This is because, like the previous models, the svm had a hard time classifying classes 1 and 3 especially class 3 (like the random forest) as it had 0 data classified correctly out of 266 (17 + 249) indicating that 0% (0 / 266) were classified correctly, while class 1 had 45 data classified correctly out of 238 (45 + 193) indicating that 0.233 (45 / 238) or 23.3% were classified correctly. Class 2, on the other hand, had 447 data classified correctly out of 477 (30 + 447) indicating that 0.937 (447 / 477) or 93.7% were classified correctly. These values represent the sensitivity of the model to each class. The results again indicates that the model is biased towards class 2 indicating that there is class imbalance (Farinda 2019) and as mentioned before, the dataset itself is small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance. They also indicate that the svm model had a very hard time classifying classes 1 and 3 especially 3 compared to the random forest and the decision tree, but it performed better in classifying class 2 compared to them.

c) The code for tuning the svm is shown below and the output is shown in Figure 11

```
#automatic tuning
#defining the value of the gamma and cost
gamma_vals <- 3^seq(-3, 3)
cost_vals <- 3^seq(-3, 3)

#auto tuning
```

```r
auto_tune <- tune.svm(credit.rating ~ ., data = train, kernel = "radial", gamma = gamma_vals, cost =
cost_vals)
#printing summary of the model
summary(auto_tune)
#predicting on test set using the tuned svm model and printing accuracy and confusion matrix
svm.tuned <- auto_tune$best.model
predict_tuned_svm_test <- predict(svm.tuned, newdata = test)
confusionMatrix(predict_tuned_svm_test, test$credit.rating)
```



```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
      gamma cost
 0.03703704    3

- best performance: 0.4566893

- Detailed performance results:
        gamma        cost     error dispersion
1   0.03703704  0.03703704 0.4974438 0.03091637
2   0.11111111  0.03703704 0.4974438 0.03091637
3   0.33333333  0.03703704 0.4974438 0.03091637
4   1.00000000  0.03703704 0.4974438 0.03091637
5   3.00000000  0.03703704 0.4974438 0.03091637
6   9.00000000  0.03703704 0.4974438 0.03091637
7  27.00000000  0.03703704 0.4974438 0.03091637
8   0.03703704  0.11111111 0.4974438 0.03091637
9   0.11111111  0.11111111 0.4974438 0.03091637
10  0.33333333  0.11111111 0.4974438 0.03091637
11  1.00000000  0.11111111 0.4974438 0.03091637
12  3.00000000  0.11111111 0.4974438 0.03091637
13  9.00000000  0.11111111 0.4974438 0.03091637
14 27.00000000  0.11111111 0.4974438 0.03091637
15  0.03703704  0.33333333 0.4974438 0.03091637
16  0.11111111  0.33333333 0.4974438 0.03091637
17  0.33333333  0.33333333 0.4974438 0.03091637
18  1.00000000  0.33333333 0.4974438 0.03091637
19  3.00000000  0.33333333 0.4974438 0.03091637
20  9.00000000  0.33333333 0.4974438 0.03091637
21 27.00000000  0.33333333 0.4974438 0.03091637
22  0.03703704  1.00000000 0.4658524 0.03154577
23  0.11111111  1.00000000 0.4974438 0.03091637
24  0.33333333  1.00000000 0.4974438 0.03091637
25  1.00000000  1.00000000 0.4974438 0.03091637
26  3.00000000  1.00000000 0.4974438 0.03091637
27  9.00000000  1.00000000 0.4974438 0.03091637
28 27.00000000  1.00000000 0.4974438 0.03091637
29  0.03703704  3.00000000 0.4566893 0.02416855
30  0.11111111  3.00000000 0.4974438 0.03091637
31  0.33333333  3.00000000 0.4974438 0.03091637
32  1.00000000  3.00000000 0.4974438 0.03091637
33  3.00000000  3.00000000 0.4974438 0.03091637
34  9.00000000  3.00000000 0.4974438 0.03091637
35 27.00000000  3.00000000 0.4974438 0.03091637
36  0.03703704  9.00000000 0.4617811 0.02646920
37  0.11111111  9.00000000 0.4974438 0.03091637
38  0.33333333  9.00000000 0.4974438 0.03091637
39  1.00000000  9.00000000 0.4974438 0.03091637
40  3.00000000  9.00000000 0.4974438 0.03091637
41  9.00000000  9.00000000 0.4974438 0.03091637
42 27.00000000  9.00000000 0.4974438 0.03091637
43  0.03703704 27.00000000 0.4617811 0.02646920
44  0.11111111 27.00000000 0.4974438 0.03091637
45  0.33333333 27.00000000 0.4974438 0.03091637
```

```
46  1.00000000 27.00000000 0.4974438 0.03091637
47  3.00000000 27.00000000 0.4974438 0.03091637
48  9.00000000 27.00000000 0.4974438 0.03091637
49 27.00000000 27.00000000 0.4974438 0.03091637
```

```
> confusionMatrix(predict_tuned_svm_test, test$credit.rating)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1  90  63  32
         2 141 365 163
         3   7  49  71

Overall Statistics

               Accuracy : 0.5362
                 95% CI : (0.5044, 0.5678)
    No Information Rate : 0.4862
    P-Value [Acc > NIR] : 0.0009745

                  Kappa : 0.2106

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity           0.37815   0.7652  0.26692
Specificity           0.87214   0.3968  0.92168
Pos Pred Value        0.48649   0.5456  0.55906
Neg Pred Value        0.81407   0.6410  0.77166
Prevalence            0.24261   0.4862  0.27115
Detection Rate        0.09174   0.3721  0.07238
Detection Prevalence  0.18858   0.6820  0.12946
Balanced Accuracy     0.62515   0.5810  0.59430
```

Figure 11: *Tuning SVM and Confusion Matrix + Accuracy for Predicting on Test*

I have used the radial kernel and the gamma and cost values ranging from 3^{-3} to 3^{3} to automatically tune the svm. In the end, the best tuning parameters turned out to be gamma = 0.03703704 and cost = 3. I then used that best performing model to predict on the test set. The produced confusion matrix indicates that the tuned svm model was able to correctly classify 526 (diagonal sum of the true positive data (data that is positive and was predicted as such (Mitrani 2019)) (90 + 365 + 71)) out of the 981 data which is higher than the untuned svm model's 492. The svm model incorrectly classified 63 and 32 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 1 as classes 2 and 3 respectively. In addition, it also incorrectly classified 141 and 163 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 2 as classes 1 and 3 respectively. Furthermore, it also incorrectly classified 7 and 49 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 3 as class 1 and class 2 respectively.

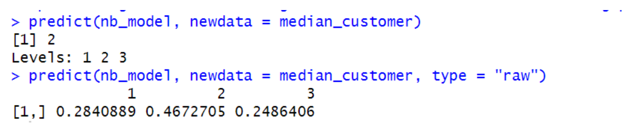
Because of all of these inaccuracies, the overall accuracy rate of the tuned svm is shown in the Figure 11 as 0.5362 (53.62%). This accuracy rate is calculated as the diagonal sum (which we have already calculated as 526) divided by the total number of observations which is 981. It is also pretty low indicating that the error rate is around 46% (100% - 53.61876%). This accuracy rate is higher than the untuned svm model by almost 3%. The model itself performed much better at predicting classes 1 and 3 compared to the untuned svm model but its performance with class 2 was a little bit less. But it still had a hard time classifying classes 1 and 3 especially class 3 as it had only 71 data classified correctly out of 266 (32 + 163 + 71) indicating that 0.26692 (71 / 266) or 26.692% were classified correctly, while class 1 had 90 data classified correctly out of 238 (90 + 141 + 7) indicating that 0.37815 (90 / 238) or 37.815% were classified correctly. Class 2, on the other hand, had 365 data classified correctly out of 477 (63 + 365 + 49) indicating that 0.7652 (365 / 477) or

76.52% were classified correctly. These values represent the sensitivity of the model to each class. The results again indicates that the model is biased towards class 2 indicating that there is class imbalance (Farinda 2019) and as mentioned before, the dataset itself is small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance.

QUESTION 4

a) The code for predicting the median customer is shown below and the output along with the probabilities are shown in Figure 12.

```
#load the libraries
library(e1071)
#creating the naive bayes model
nb_model <- naiveBayes(credit.rating ~ ., data = train)
#predicting the credit rating for the median customer and showing probabilities
predict(nb_model, newdata = median_customer)
predict(nb_model, newdata = median_customer, type = "raw")
```



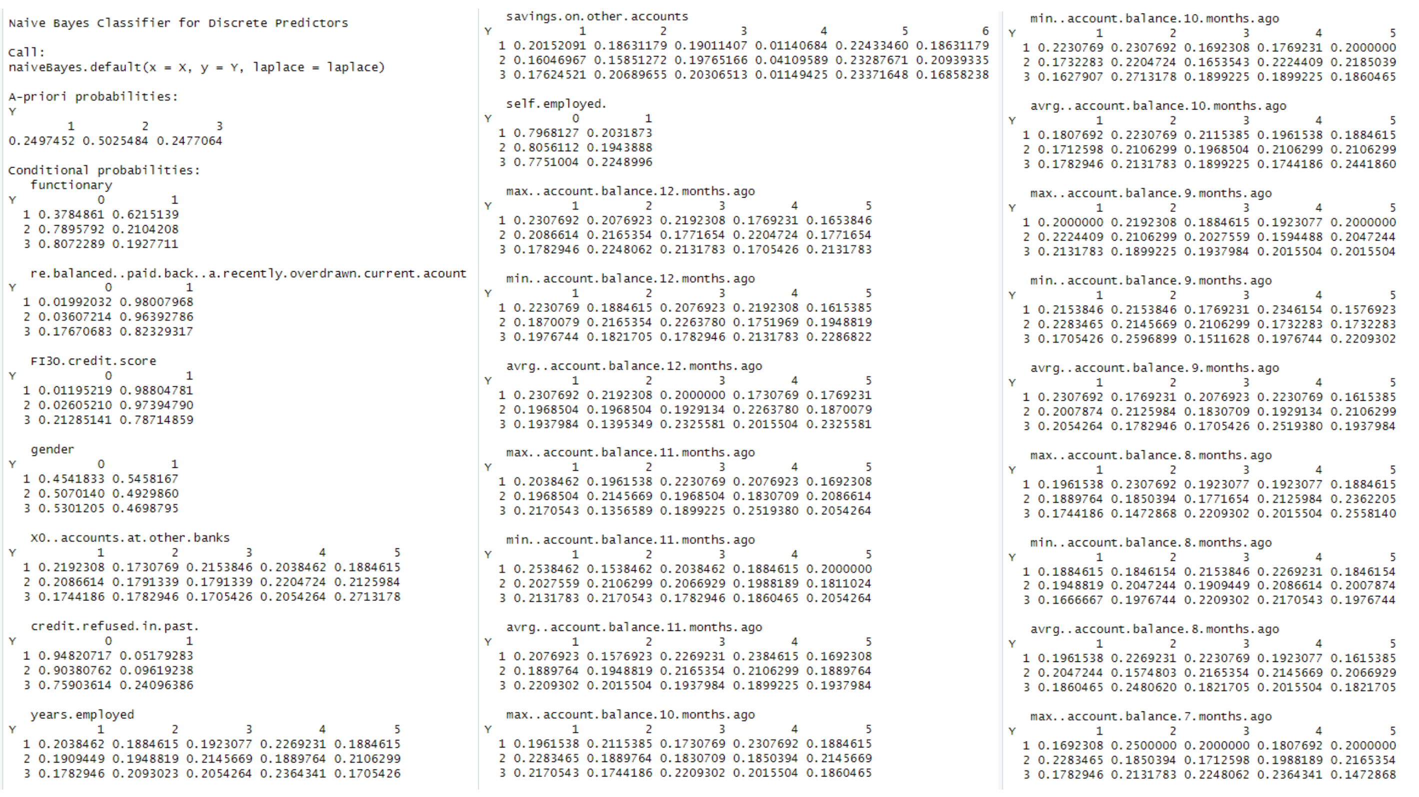
```
> predict(nb_model, newdata = median_customer)
[1] 2
Levels: 1 2 3
> predict(nb_model, newdata = median_customer, type = "raw")
             1         2         3
[1,] 0.2840889 0.4672705 0.2486406
```

Figure 12: *Predicting Median Customer Using Naive Bayes and Showing Probabilities*

I used the dataset I created in question 1b and 2a to predict the credit rating of the median customer. As it is shown in Figure 12 and as the decision tree model and the untuned svm model predicted, the median customer has a credit rating 2. The probabilities are also shown in Figure 12 with the median customer having a probability of 0.2840889 to be class 1, 0.4672705 to be class 2, and 0.2486406 to be class 3. Since class 2's probability is the highest the customer is therefore a class 2.

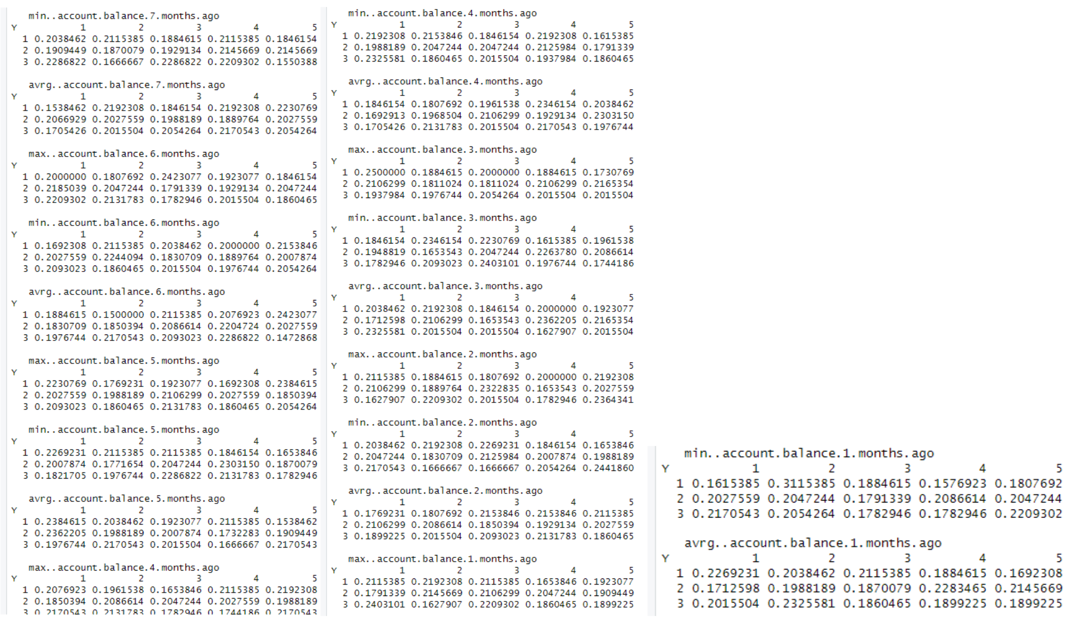
b) The code for reproducing the first 20 or so lines of R output is shown below and the output is shown in Figure 13.

```
####QUESTION 4 PART B
#reproducing the first 20 or so lines
naiveBayes(credit.rating ~ ., data = train, laplace = 3)
```



```
Naive Bayes Classifier for Discrete Predictors

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

A-priori probabilities:
Y
        1         2         3 
0.2497452 0.5025484 0.2477064 

Conditional probabilities:
   functionary
Y           0         1
  1 0.3784861 0.6215139
  2 0.7895792 0.2104208
  3 0.8072289 0.1927711

   re.balanced..paid.back..a.recently.overdrawn.current.acount
Y            0          1
  1 0.01992032 0.98007968
  2 0.03607214 0.96392786
  3 0.17670683 0.82329317

   FI3O.credit.score
Y            0          1
  1 0.01195219 0.98804781
  2 0.02605210 0.97394790
  3 0.21285141 0.78714859

   gender
Y           0         1
  1 0.4541833 0.5458167
  2 0.5070140 0.4929860
  3 0.5301205 0.4698795

   X0..accounts.at.other.banks
Y           1         2         3         4         5
  1 0.2192308 0.1730769 0.2153846 0.2038462 0.1884615
  2 0.2086614 0.1791339 0.1791339 0.2204724 0.2125984
  3 0.1744186 0.1782946 0.1705426 0.2054264 0.2713178

   credit.refused.in.past.
Y            0          1
  1 0.94820717 0.05179283
  2 0.90380762 0.09619238
  3 0.75903614 0.24096386

   years.employed
Y           1         2         3         4         5
  1 0.2038462 0.1884615 0.1923077 0.2269231 0.1884615
  2 0.1909449 0.1948819 0.2145669 0.1889764 0.2106299
  3 0.1782946 0.2093023 0.2054264 0.2364341 0.1705426

   savings.on.other.accounts
Y            1          2          3          4          5          6
  1 0.20152091 0.18631179 0.19011407 0.01140684 0.22433460 0.18631179
  2 0.16046967 0.15851272 0.19765166 0.04109589 0.23287671 0.20939335
  3 0.17624521 0.20689655 0.20306513 0.01149425 0.23371648 0.16858238

   self.employed.
Y           0         1
  1 0.7968127 0.2031873
  2 0.8056112 0.1943888
  3 0.7751004 0.2248996

   max..account.balance.12.months.ago
Y           1         2         3         4         5
  1 0.2307692 0.2076923 0.2192308 0.1769231 0.1653846
  2 0.2086614 0.2165354 0.1771654 0.2204724 0.1771654
  3 0.1782946 0.2248062 0.2131783 0.1705426 0.2131783

   min..account.balance.12.months.ago
Y           1         2         3         4         5
  1 0.2230769 0.1884615 0.2076923 0.2192308 0.1615385
  2 0.1870079 0.2165354 0.2263780 0.1751969 0.1948819
  3 0.1976744 0.1821705 0.1782946 0.2131783 0.2286822

   avrg..account.balance.12.months.ago
Y           1         2         3         4         5
  1 0.2307692 0.2192308 0.2000000 0.1730769 0.1769231
  2 0.1968504 0.1968504 0.1929134 0.2263780 0.1870079
  3 0.1937984 0.1395349 0.2325581 0.2015504 0.2325581

   max..account.balance.11.months.ago
Y           1         2         3         4         5
  1 0.2038462 0.1961538 0.2230769 0.2076923 0.1692308
  2 0.1968504 0.2145669 0.1968504 0.1830709 0.2086614
  3 0.2170543 0.1356589 0.1899225 0.2519380 0.2054264

   min..account.balance.11.months.ago
Y           1         2         3         4         5
  1 0.2538462 0.1538462 0.2038462 0.1884615 0.2000000
  2 0.2027559 0.2106299 0.2066929 0.1988189 0.1811024
  3 0.2131783 0.2170543 0.1782946 0.1860465 0.2054264

   avrg..account.balance.11.months.ago
Y           1         2         3         4         5
  1 0.2076923 0.1576923 0.2269231 0.2384615 0.1692308
  2 0.1889764 0.1948819 0.2165354 0.2106299 0.1889764
  3 0.2209302 0.2015504 0.1937984 0.1899225 0.1937984

   max..account.balance.10.months.ago
Y           1         2         3         4         5
  1 0.1961538 0.2115385 0.1730769 0.2307692 0.1884615
  2 0.2283465 0.1889764 0.1830709 0.1850394 0.2145669
  3 0.2170543 0.1744186 0.2209302 0.2015504 0.1860465

   min..account.balance.10.months.ago
Y           1         2         3         4         5
  1 0.2230769 0.2307692 0.1692308 0.1769231 0.2000000
  2 0.1732283 0.2204724 0.1653543 0.2224409 0.2185039
  3 0.1627907 0.2713178 0.1899225 0.1899225 0.1860465

   avrg..account.balance.10.months.ago
Y           1         2         3         4         5
  1 0.1807692 0.2230769 0.2115385 0.1961538 0.1884615
  2 0.1712598 0.2106299 0.1968504 0.2106299 0.2106299
  3 0.1782946 0.2131783 0.1899225 0.1744186 0.2441860

   max..account.balance.9.months.ago
Y           1         2         3         4         5
  1 0.2000000 0.2192308 0.1884615 0.1923077 0.2000000
  2 0.2224409 0.2106299 0.2027559 0.1594488 0.2047244
  3 0.2131783 0.1899225 0.1937984 0.2015504 0.2015504

   min..account.balance.9.months.ago
Y           1         2         3         4         5
  1 0.2153846 0.2153846 0.1769231 0.2346154 0.1576923
  2 0.2283465 0.2145669 0.2106299 0.1732283 0.1732283
  3 0.1705426 0.2596899 0.1511628 0.1976744 0.2209302

   avrg..account.balance.9.months.ago
Y           1         2         3         4         5
  1 0.2307692 0.1769231 0.2076923 0.2230769 0.1615385
  2 0.2007874 0.2125984 0.1830709 0.1929134 0.2106299
  3 0.2054264 0.1782946 0.1705426 0.2519380 0.1937984

   max..account.balance.8.months.ago
Y           1         2         3         4         5
  1 0.1961538 0.2307692 0.1923077 0.1923077 0.1884615
  2 0.1889764 0.1850394 0.1771654 0.2125984 0.2362205
  3 0.1744186 0.1472868 0.2209302 0.2015504 0.2558140

   min..account.balance.8.months.ago
Y           1         2         3         4         5
  1 0.1884615 0.1846154 0.2153846 0.2269231 0.1846154
  2 0.1948819 0.2047244 0.1909449 0.2086614 0.2007874
  3 0.1666667 0.1976744 0.2209302 0.2170543 0.1976744

   avrg..account.balance.8.months.ago
Y           1         2         3         4         5
  1 0.1961538 0.2269231 0.2230769 0.1923077 0.1615385
  2 0.2047244 0.1574803 0.2165354 0.2145669 0.2066929
  3 0.1860465 0.2480620 0.1821705 0.2015504 0.1821705

   max..account.balance.7.months.ago
Y           1         2         3         4         5
  1 0.1692308 0.2500000 0.2000000 0.1807692 0.2000000
  2 0.2283465 0.1850394 0.1712598 0.1988189 0.2165354
  3 0.1782946 0.2131783 0.2248062 0.2364341 0.1472868
```

Figure 13*: Naive Bayes - Reproducing the First 20 or so Lines of R Output - 1*


```
   min..account.balance.7.months.ago
Y           1         2         3         4         5
  1 0.2038462 0.2115385 0.1884615 0.2115385 0.1846154
  2 0.1909449 0.1870079 0.1929134 0.2145669 0.2145669
  3 0.2286822 0.1666667 0.2286822 0.2209302 0.1550388

   avrg..account.balance.7.months.ago
Y           1         2         3         4         5
  1 0.1538462 0.2192308 0.1846154 0.2192308 0.2230769
  2 0.2066929 0.2027559 0.1988189 0.1889764 0.2027559
  3 0.1705426 0.2015504 0.2054264 0.2170543 0.2054264

   max..account.balance.6.months.ago
Y           1         2         3         4         5
  1 0.2000000 0.1807692 0.2423077 0.1923077 0.1846154
  2 0.2185039 0.2047244 0.1791339 0.1929134 0.2047244
  3 0.2209302 0.2131783 0.1782946 0.2015504 0.1860465

   min..account.balance.6.months.ago
Y           1         2         3         4         5
  1 0.1692308 0.2115385 0.2038462 0.2000000 0.2153846
  2 0.2027559 0.2244094 0.1830709 0.1889764 0.2007874
  3 0.2093023 0.1860465 0.2015504 0.1976744 0.2054264

   avrg..account.balance.6.months.ago
Y           1         2         3         4         5
  1 0.1884615 0.1500000 0.2115385 0.2076923 0.2423077
  2 0.1830709 0.1850394 0.2086614 0.2204724 0.2027559
  3 0.1976744 0.2170543 0.2093023 0.2286822 0.1472868

   max..account.balance.5.months.ago
Y           1         2         3         4         5
  1 0.2230769 0.1769231 0.1923077 0.1692308 0.2384615
  2 0.2027559 0.1988189 0.2106299 0.2027559 0.1850394
  3 0.2093023 0.1860465 0.2131783 0.1860465 0.2054264

   min..account.balance.5.months.ago
Y           1         2         3         4         5
  1 0.2269231 0.2115385 0.2115385 0.1846154 0.1653846
  2 0.2007874 0.1771654 0.2047244 0.2303150 0.1870079
  3 0.1821705 0.1976744 0.2286822 0.2131783 0.1782946

   avrg..account.balance.5.months.ago
Y           1         2         3         4         5
  1 0.2384615 0.2038462 0.1923077 0.2115385 0.1538462
  2 0.2362205 0.1988189 0.2007874 0.1732283 0.1909449
  3 0.1976744 0.2170543 0.2015504 0.1666667 0.2170543

   max..account.balance.4.months.ago
Y           1         2         3         4         5
  1 0.2076923 0.1961538 0.1653846 0.2115385 0.2192308
  2 0.1850394 0.2086614 0.2047244 0.2027559 0.1988189
  3 0.2170543 0.2131783 0.1782946 0.1744186 0.2170543
```

```
   min..account.balance.4.months.ago
Y           1         2         3         4         5
  1 0.2192308 0.2153846 0.1846154 0.2192308 0.1615385
  2 0.1988189 0.2047244 0.2047244 0.2125984 0.1791339
  3 0.2325581 0.1860465 0.2015504 0.1937984 0.1860465

   avrg..account.balance.4.months.ago
Y           1         2         3         4         5
  1 0.1846154 0.1807692 0.1961538 0.2346154 0.2038462
  2 0.1692913 0.1968504 0.2106299 0.1929134 0.2303150
  3 0.1705426 0.2131783 0.2015504 0.2170543 0.1976744

   max..account.balance.3.months.ago
Y           1         2         3         4         5
  1 0.2500000 0.1884615 0.2000000 0.1884615 0.1730769
  2 0.2106299 0.1811024 0.1811024 0.2106299 0.2165354
  3 0.1937984 0.1976744 0.2054264 0.2015504 0.2015504

   min..account.balance.3.months.ago
Y           1         2         3         4         5
  1 0.1846154 0.2346154 0.2230769 0.1615385 0.1961538
  2 0.1948819 0.1653543 0.2047244 0.2263780 0.2086614
  3 0.1782946 0.2093023 0.2403101 0.1976744 0.1744186

   avrg..account.balance.3.months.ago
Y           1         2         3         4         5
  1 0.2038462 0.2192308 0.1846154 0.2000000 0.1923077
  2 0.1712598 0.2106299 0.1653543 0.2362205 0.2165354
  3 0.2325581 0.2015504 0.2015504 0.1627907 0.2015504

   max..account.balance.2.months.ago
Y           1         2         3         4         5
  1 0.2115385 0.1884615 0.1807692 0.2000000 0.2192308
  2 0.2106299 0.1889764 0.2322835 0.1653543 0.2027559
  3 0.1627907 0.2209302 0.2015504 0.1782946 0.2364341

   min..account.balance.2.months.ago
Y           1         2         3         4         5
  1 0.2038462 0.2192308 0.2269231 0.1846154 0.1653846
  2 0.2047244 0.1830709 0.2125984 0.2007874 0.1988189
  3 0.2170543 0.1666667 0.1666667 0.2054264 0.2441860

   avrg..account.balance.2.months.ago
Y           1         2         3         4         5
  1 0.1769231 0.1807692 0.2153846 0.2153846 0.2115385
  2 0.2106299 0.2086614 0.1850394 0.1929134 0.2027559
  3 0.1899225 0.2015504 0.2093023 0.2131783 0.1860465

   max..account.balance.1.months.ago
Y           1         2         3         4         5
  1 0.2115385 0.2192308 0.2115385 0.1653846 0.1923077
  2 0.1791339 0.2145669 0.2106299 0.2047244 0.1909449
  3 0.2403101 0.1627907 0.2209302 0.1860465 0.1899225
```

```
   min..account.balance.1.months.ago
Y           1         2         3         4         5
  1 0.1615385 0.3115385 0.1884615 0.1576923 0.1807692
  2 0.2027559 0.2047244 0.1791339 0.2086614 0.2047244
  3 0.2170543 0.2054264 0.1782946 0.1782946 0.2209302

   avrg..account.balance.1.months.ago
Y           1         2         3         4         5
  1 0.2269231 0.2038462 0.2115385 0.1884615 0.1692308
  2 0.1712598 0.1988189 0.1870079 0.2283465 0.2145669
  3 0.2015504 0.2325581 0.1860465 0.1899225 0.1899225
```

***Figure 14**: Naive Bayes - Reproducing the First 20 or so Lines of R Output - Continued*

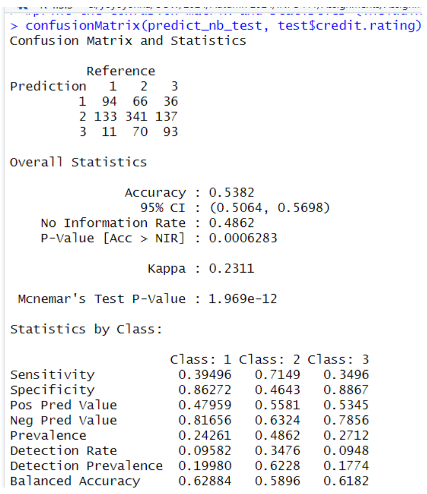
In Figure 13, two kinds of probabilities are shown:

1) **A-priori probabilities:** These are the probabilities of each class occurring in the dataset where the probability of class 1 occurring is 0.2497452, class 2 is 0.5025484, and class 3 is 0.2477064. Class 2's probability's is the highest followed by class 1 and then class 3 indicating again that there is data imbalance. The median customer's probabilities were also in the same order (class 2, then 1, then 3) indicating that class 3 is the one with the least occurrence in the dataset. Hence, it is not as easily classified.
2) **Conditional probabilities:** For each predictor variable (e.g., functionary, etc.), there are conditional probabilities provided for each class that represent the likelihood of observing a specific value of the predictor variable in a specific class. For example, functionary's conditional probabilities for class 1 are 0.3784861 and 0.6215139 for values 0 and 1 respectively which indicates that the likelihood of a customer with functionary = 0 being labelled as class 1 is 0.3784861 and the probability that a customer with functionary = 1 being labelled as class 1 is 0.6215139, etc.

Considering that the median customer has a functionary = 0, and re.balanced..paid.back..a.recently.overdrawn.current.acount = 1, the model will find the highest values for functionary (0.8072289) and re.balanced..paid.back..a.recently.overdrawn.current.acount (0.98007968) that correspond with the 0 and 1 then multiply them with each other and the same occurs with the rest of the variables. After all the conditional probabilities for all the variables are multiplied with one another, the resulting value will then be multiplied by each of the A-priori probabilities and divided by the sum of the A-priori probabilities. The one with the highest result is the class of the customer.

c) The code for predicting on the test set is shown below and the confusion matrix along with the overall accuracy is shown in Figure 14.

```
#load the libraries
library(caret)

#predict on the test data
predict_nb_test <- predict(nb_model, test, type = "class")

#print the confusion matrix and statistics (including overall accuracy)
confusionMatrix(predict_nb_test, test$credit.rating)
```



```
> confusionMatrix(predict_nb_test, test$credit.rating)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1  94  66  36
         2 133 341 137
         3  11  70  93

Overall Statistics

               Accuracy : 0.5382
                 95% CI : (0.5064, 0.5698)
    No Information Rate : 0.4862
    P-Value [Acc > NIR] : 0.0006283

                  Kappa : 0.2311

 Mcnemar's Test P-Value : 1.969e-12

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity           0.39496   0.7149   0.3496
Specificity           0.86272   0.4643   0.8867
Pos Pred Value        0.47959   0.5581   0.5345
Neg Pred Value        0.81656   0.6324   0.7856
Prevalence            0.24261   0.4862   0.2712
Detection Rate        0.09582   0.3476   0.0948
Detection Prevalence  0.19980   0.6228   0.1774
Balanced Accuracy     0.62884   0.5896   0.6182
```

Figure 15: *Confusion Matrix + Overall Accuracy for Predicting on Test Set Using Naive Bayes*

The confusion matrix for predicting the credit rating from test set is shown in Figure 14. As illustrated, the naïve bayes model was able to correctly classify 528 (diagonal sum of the true positive data (data that is positive and was predicted as such (Mitrani 2019)) (94 + 341 + 93)) out of the 981 data. The naïve bayes model incorrectly classified 133 and 11 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 1 as classes 2 and 3 respectively. In addition, it also incorrectly classified 66 and 70 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 2 as classes 1 and 3 respectively. Furthermore, it also incorrectly classified 36 and 137 (false negatives (data that was predicted as negative but is actually positive (Mitrani 2019))) data that should be class 3 as class 1 and class 2 respectively.

Because of all of these inaccuracies, the overall accuracy rate of the naïve bayes model is shown in the Figure 14 as 0.5382 (53.82%). This accuracy rate is calculated as the diagonal sum (which we have already calculated as 528) divided by the total number of observations which is 981. It is also pretty low indicating that the error rate is around 46% (100% - 53.82%). This is because, the naïve bayes model had a hard time classifying classes 1 and 3 especially class 3 as it had only 93 data classified correctly out of 266 (36 + 137 + 93) indicating that 0.3496 (93 / 266) or 34.96% were classified correctly, while class 1 had 94 data classified correctly out of 238 (94 + 133 + 11) indicating that 0.395 (94 / 238) or 39.5% were classified correctly. Class 2, on the other hand, had 341 data classified correctly out of 477 (66 + 341 + 70) indicating that 0.7149 (341 / 477) or 71.49% were classified correctly. These values represent the sensitivity of the model to each class. The results again indicates that the model is biased towards class 2 indicating that there is class imbalance (Farinda 2019) and as mentioned before, the dataset itself is small hence the data is homogeneous (Statistics How To n.d.) which also leads to class imbalance.

QUESTION 5

Table 1 shows a comparison between the classifiers when they were used to predict on the test sets. The overall accuracies of each classifier have been presented before in the report and are stated in Table 1 as well. Based on the accuracies, we can deduce that random forest has the best performance as it achieved a 0.58 accuracy. This is followed by the decision tree (0.5729) then the naïve bayes model (0.5382), then the tuned svm model (0.5362), and lastly the untuned svm model (0.5015). As discussed before, because of the class imbalance, the models had a hard time classifying classes 1 and 3 especially class 3 for most models (except for the decision tree model that struggled a little bit more with class 1) with less than 0.1 differences in their sensitivities for all the models. There is also the fact that most of the predictions that should be for classes 1 and 3 are being classified as class 2 as was shown in the previous confusion matrices. Hence, I assume that the difference between class 2 and classes 1 and 3 might not be that much but because of the size of the dataset and the imbalance in it (half of the data is class 2 as was calculated in the A-priori probabilities by the naïve bayes model and about 25% is class 1 and 25% is class 3 with class 3 being about 0.002 less frequent than class 1), there is not enough variation in the data for the models to be trained on and hence, they classify them as class 2.

The sensitivity of each model to each class has been discussed before. The decision tree model had a sensitivity of 0.38235 to class 1, 0.7736 to class 2, and 0.3835 to class 3. This model had the highest sensitivity to class 3. The random forest model, specifically rf_2, had a sensitivity of 0.4706 to class 1, 0.7862 to class 2, and 0.30827 to class 3. It had the highest sensitivity to class 1 and the second highest sensitivity to class 2 after the untuned svm model. The untuned svm model had a sensitivity of 0.18908 to class 1, 0.9371 to class 2, and 0 to class 3. It had the highest sensitivity to class 2 and the lowest sensitivities to classes 1 and 3. The tuned svm model had a sensitivity of 0.37815 to class 1, 0.7652 to class 2, and 0.26692 to class 3. It had the second lowest sensitivities to classes 1 and 3 after the untuned svm model. Lastly, the naïve bayes model had a sensitivity of 0.39496 to class 1, 0.7149 to class 2, and 0.3496 to class 3. It had the second highest sensitivity to class 1 after the random forest model and the second highest sensitivity to class 3 after the decision tree model.

So, to summarize:

- **Best performing classifier:** the random forest model
- **Worst performing classifier:** the untuned svm model
- **Categories that classifiers struggle with:** classes 1 and 3 especially class 3 for most models except the decision tree model that struggled a little bit more with class 1.

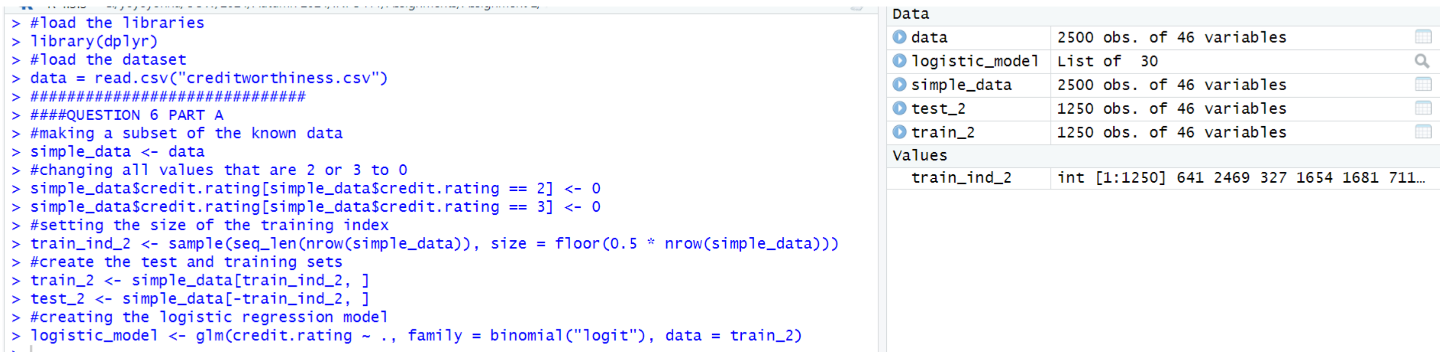
Comparison Points	Decision Tree	Random Forest	SVM Untuned	SVM Tuned	Naïve Bayes
Overall Accuracy	0.5729	0.58	0.5015	0.5362	0.5382
Class 1 Sensitivity	0.38235	0.4706	0.18908	0.37815	0.39496
Class 2 Sensitivity	0.7736	0.7862	0.9371	0.7652	0.7149
Class 3 Sensitivity	0.3835	0.30827	0	0.26692	0.3496

Table 1: *Comparison Between Classifiers*

QUESTION 6

a) The code for fitting a logistic regression model to predict whether a customer gets a credit rating of A or not is shown below and the output is shown in Figure 15.

```r
#making a subset of the known data
simple_data <- data
#changing all values that are 2 or 3 to 0
simple_data$credit.rating[simple_data$credit.rating == 2] <- 0
simple_data$credit.rating[simple_data$credit.rating == 3] <- 0

#setting the size of the training index
train_ind_2 <- sample(seq_len(nrow(simple_data)), size = floor(0.5 * nrow(simple_data)))

#create the test and training sets
train_2 <- simple_data[train_ind_2, ]
test_2 <- simple_data[-train_ind_2, ]

#creating the logistic regression model
logistic_model <- glm(credit.rating ~ ., family = binomial("logit"), data = train_2)
```



```
> #load the libraries
> library(dplyr)
> #load the dataset
> data = read.csv("creditworthiness.csv")
> ###############################
> ####QUESTION 6 PART A
> #making a subset of the known data
> simple_data <- data
> #changing all values that are 2 or 3 to 0
> simple_data$credit.rating[simple_data$credit.rating == 2] <- 0
> simple_data$credit.rating[simple_data$credit.rating == 3] <- 0
> #setting the size of the training index
> train_ind_2 <- sample(seq_len(nrow(simple_data)), size = floor(0.5 * nrow(simple_data)))
> #create the test and training sets
> train_2 <- simple_data[train_ind_2, ]
> test_2 <- simple_data[-train_ind_2, ]
> #creating the logistic regression model
> logistic_model <- glm(credit.rating ~ ., family = binomial("logit"), data = train_2)
```

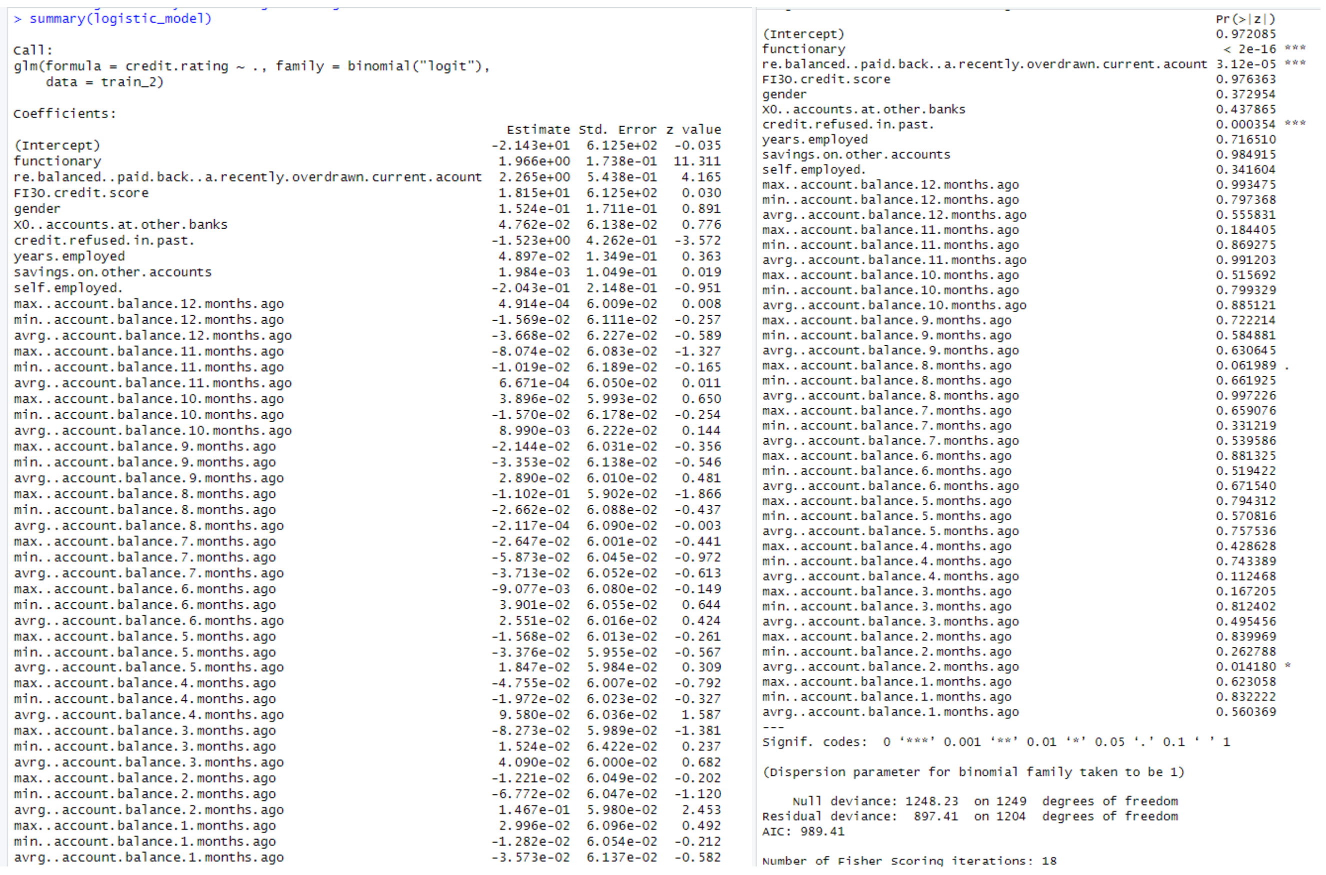
```
Data
  data             2500 obs. of 46 variables
  logistic_model   List of  30
  simple_data      2500 obs. of 46 variables
  test_2           1250 obs. of 46 variables
  train_2          1250 obs. of 46 variables
Values
  train_ind_2      int [1:1250] 641 2469 327 1654 1681 711…
```

Figure 16: *Logistic Regression Model Output*

In order to only predict whether a customer has a credit rating A (class 1) or not, I made a copy of the original dataset and changed all customers with classes 2 (credit rating B) or 3 (credit rating C) to 0 and divided half into training (1250 observations) and half into testing (1250 observations) as shown in the code above and in Figure 15. After completing this process, I created the logistic regression model.

b) The code for the summary is shown below and the output is shown in Figure 16.

```r
#creating a summary of the logistic regression model
summary(logistic_model)
```



```
> summary(logistic_model)

Call:
glm(formula = credit.rating ~ ., family = binomial("logit"),
    data = train_2)

Coefficients:
                                                     Estimate Std. Error z value
(Intercept)                                        -2.143e+01  6.125e+02  -0.035
functionary                                         1.966e+00  1.738e-01  11.311
re.balanced..paid.back..a.recently.overdrawn.current.acount  2.265e+00  5.438e-01   4.165
FI3O.credit.score                                   1.815e+01  6.125e+02   0.030
gender                                              1.524e-01  1.711e-01   0.891
X0..accounts.at.other.banks                         4.762e-02  6.138e-02   0.776
credit.refused.in.past.                            -1.523e+00  4.262e-01  -3.572
years.employed                                      4.897e-02  1.349e-01   0.363
savings.on.other.accounts                           1.984e-03  1.049e-01   0.019
self.employed.                                     -2.043e-01  2.148e-01  -0.951
max..account.balance.12.months.ago                  4.914e-04  6.009e-02   0.008
min..account.balance.12.months.ago                 -1.569e-02  6.111e-02  -0.257
avrg..account.balance.12.months.ago                -3.668e-02  6.227e-02  -0.589
max..account.balance.11.months.ago                 -8.074e-02  6.083e-02  -1.327
min..account.balance.11.months.ago                 -1.019e-02  6.189e-02  -0.165
avrg..account.balance.11.months.ago                 6.671e-04  6.050e-02   0.011
max..account.balance.10.months.ago                  3.896e-02  5.993e-02   0.650
min..account.balance.10.months.ago                 -1.570e-02  6.178e-02  -0.254
avrg..account.balance.10.months.ago                 8.990e-03  6.222e-02   0.144
max..account.balance.9.months.ago                  -2.144e-02  6.031e-02  -0.356
min..account.balance.9.months.ago                  -3.353e-02  6.138e-02  -0.546
avrg..account.balance.9.months.ago                  2.890e-02  6.010e-02   0.481
max..account.balance.8.months.ago                  -1.102e-01  5.902e-02  -1.866
min..account.balance.8.months.ago                  -2.662e-02  6.088e-02  -0.437
avrg..account.balance.8.months.ago                 -2.117e-04  6.090e-02  -0.003
max..account.balance.7.months.ago                  -2.647e-02  6.001e-02  -0.441
min..account.balance.7.months.ago                  -5.873e-02  6.045e-02  -0.972
avrg..account.balance.7.months.ago                 -3.713e-02  6.052e-02  -0.613
max..account.balance.6.months.ago                  -9.077e-03  6.080e-02  -0.149
min..account.balance.6.months.ago                   3.901e-02  6.055e-02   0.644
avrg..account.balance.6.months.ago                  2.551e-02  6.016e-02   0.424
max..account.balance.5.months.ago                  -1.568e-02  6.013e-02  -0.261
min..account.balance.5.months.ago                  -3.376e-02  5.955e-02  -0.567
avrg..account.balance.5.months.ago                  1.847e-02  5.984e-02   0.309
max..account.balance.4.months.ago                  -4.755e-02  6.007e-02  -0.792
min..account.balance.4.months.ago                  -1.972e-02  6.023e-02  -0.327
avrg..account.balance.4.months.ago                  9.580e-02  6.036e-02   1.587
max..account.balance.3.months.ago                  -8.273e-02  5.989e-02  -1.381
min..account.balance.3.months.ago                   1.524e-02  6.422e-02   0.237
avrg..account.balance.3.months.ago                  4.090e-02  6.000e-02   0.682
max..account.balance.2.months.ago                  -1.221e-02  6.049e-02  -0.202
min..account.balance.2.months.ago                  -6.772e-02  6.047e-02  -1.120
avrg..account.balance.2.months.ago                  1.467e-01  5.980e-02   2.453
max..account.balance.1.months.ago                   2.996e-02  6.096e-02   0.492
min..account.balance.1.months.ago                  -1.282e-02  6.054e-02  -0.212
avrg..account.balance.1.months.ago                 -3.573e-02  6.137e-02  -0.582
```

```
                                                            Pr(>|z|)
(Intercept)                                                 0.972085
functionary                                                  < 2e-16 ***
re.balanced..paid.back..a.recently.overdrawn.current.acount 3.12e-05 ***
FI3O.credit.score                                           0.976363
gender                                                      0.372954
X0..accounts.at.other.banks                                 0.437865
credit.refused.in.past.                                     0.000354 ***
years.employed                                              0.716510
savings.on.other.accounts                                   0.984915
self.employed.                                              0.341604
max..account.balance.12.months.ago                          0.993475
min..account.balance.12.months.ago                          0.797368
avrg..account.balance.12.months.ago                         0.555831
max..account.balance.11.months.ago                          0.184405
min..account.balance.11.months.ago                          0.869275
avrg..account.balance.11.months.ago                         0.991203
max..account.balance.10.months.ago                          0.515692
min..account.balance.10.months.ago                          0.799329
avrg..account.balance.10.months.ago                         0.885121
max..account.balance.9.months.ago                           0.722214
min..account.balance.9.months.ago                           0.584881
avrg..account.balance.9.months.ago                          0.630645
max..account.balance.8.months.ago                           0.061989 .
min..account.balance.8.months.ago                           0.661925
avrg..account.balance.8.months.ago                          0.997226
max..account.balance.7.months.ago                           0.659076
min..account.balance.7.months.ago                           0.331219
avrg..account.balance.7.months.ago                          0.539586
max..account.balance.6.months.ago                           0.881325
min..account.balance.6.months.ago                           0.519422
avrg..account.balance.6.months.ago                          0.671540
max..account.balance.5.months.ago                           0.794312
min..account.balance.5.months.ago                           0.570816
avrg..account.balance.5.months.ago                          0.757536
max..account.balance.4.months.ago                           0.428628
min..account.balance.4.months.ago                           0.743389
avrg..account.balance.4.months.ago                          0.112468
max..account.balance.3.months.ago                           0.167205
min..account.balance.3.months.ago                           0.812402
avrg..account.balance.3.months.ago                          0.495456
max..account.balance.2.months.ago                           0.839969
min..account.balance.2.months.ago                           0.262788
avrg..account.balance.2.months.ago                          0.014180 *
max..account.balance.1.months.ago                           0.623058
min..account.balance.1.months.ago                           0.832222
avrg..account.balance.1.months.ago                          0.560369
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1248.23  on 1249  degrees of freedom
Residual deviance:  897.41  on 1204  degrees of freedom
AIC: 989.41

Number of Fisher Scoring iterations: 18
```

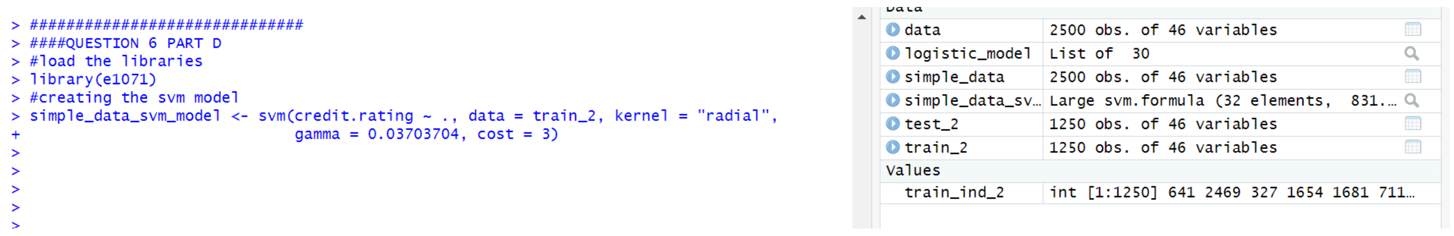
Figure 17: *Summary of Logistic Regression Model*

As is indicated by Figure 16, 18 iterations were performed when the logistic regression model was being fitted. Under the "coefficients" heading, estimated coefficients for each attribute is given along with the standard error and z-value. It also shows the significance codes for each attribute varying from 0 to 1.

c) The significance codes shown at the end of the logistic regression summary output in Figure 16 show asterisks (*). These asterisks indicate the level of significance of each coefficient. As shown in Figure 16, three asterisks (***) indicate a 0.001 (0.1% (0.001 * 100)) level of significance, two asterisks (**) indicate a 0.01 (1% (0.01 * 100)) level of significance, one asterisk (*) indicates a 0.05 (5% (0.05 * 100)) level of significance, a dot (.) indicates a 0.1 (10% (0.1 * 100)) level of significance, and nothing indicates no significance. On the right side of Figure 16, some attributes have those symbols next to them. "functionary", "re.balanced..paid.back..a.recently.overdrawn.current.acount", and "credit.refused.in.past" have three asterisks (***) indicating a 0.1% level of significance, "avrg..account.balance.2.months.ago" has one asterisk (*) indicates a 5% level of significance, and "max..account.balance.8.months.ago" has a dot (.) indicating a 10% level of significance.

d) The code for fitting the svm model is shown below and the output is shown in Figure 17.

```r
#load the libraries
library(e1071)

#creating the svm model
simple_data_svm_model <- svm(credit.rating ~ ., data = train_2, kernel = "radial", gamma = 0.03703704, cost
= 3)
```



```
> ##############################
> ####QUESTION 6 PART D
> #load the libraries
> library(e1071)
> #creating the svm model
> simple_data_svm_model <- svm(credit.rating ~ ., data = train_2, kernel = "radial",
+                              gamma = 0.03703704, cost = 3)
>
>
>
>
>
```

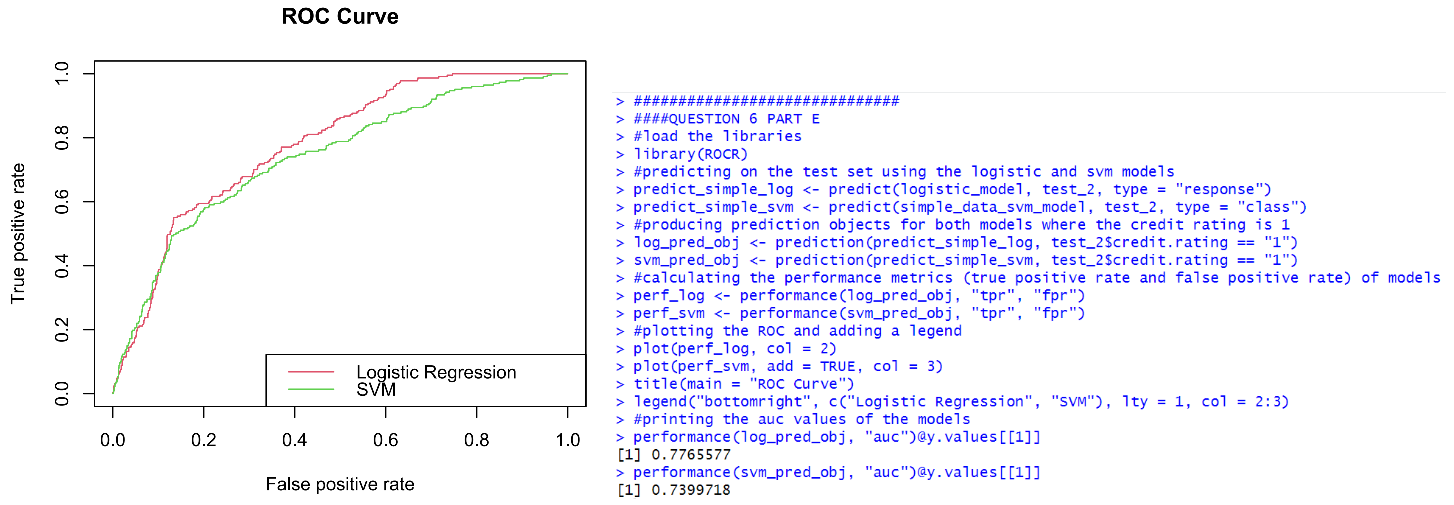
```
Data
 data                 2500 obs. of 46 variables
 logistic_model       List of  30
 simple_data          2500 obs. of 46 variables
 simple_data_sv…      Large svm.formula (32 elements,  831.…
 test_2               1250 obs. of 46 variables
 train_2              1250 obs. of 46 variables
Values
 train_ind_2          int [1:1250] 641 2469 327 1654 1681 711…
```

Figure 18: *SVM Model*

I used the best performing svm model in question 3c with the radial kernel and gamma = 0.03703704 and cost = 3 and fitted it into my training set.

e) The code for the ROC and the AUC is shown below and the output is shown in Figure 18.

```
#load the libraries
library(ROCR)
#predicting on the test set using the logistic and svm models
predict_simple_log <- predict(logistic_model, test_2, type = "response")
predict_simple_svm <- predict(simple_data_svm_model, test_2, type = "class")
#producing prediction objects for both models where the credit rating is 1
log_pred_obj <- prediction(predict_simple_log, test_2$credit.rating == "1")
svm_pred_obj <- prediction(predict_simple_svm, test_2$credit.rating == "1")
#calculating the performance metrics (true positive rate and false positive rate) of models
perf_log <- performance(log_pred_obj, "tpr", "fpr")
perf_svm <- performance(svm_pred_obj, "tpr", "fpr")
#plotting the ROC and adding a legend
plot(perf_log, col = 2)
plot(perf_svm, add = TRUE, col = 3)
title(main = "ROC Curve")
legend("bottomright", c("Logistic Regression", "SVM"), lty = 1, col = 2:3)
#printing the auc values of the models
performance(log_pred_obj, "auc")@y.values[[1]]
performance(svm_pred_obj, "auc")@y.values[[1]]
```



```
> ##############################
> ####QUESTION 6 PART E
> #load the libraries
> library(ROCR)
> #predicting on the test set using the logistic and svm models
> predict_simple_log <- predict(logistic_model, test_2, type = "response")
> predict_simple_svm <- predict(simple_data_svm_model, test_2, type = "class")
> #producing prediction objects for both models where the credit rating is 1
> log_pred_obj <- prediction(predict_simple_log, test_2$credit.rating == "1")
> svm_pred_obj <- prediction(predict_simple_svm, test_2$credit.rating == "1")
> #calculating the performance metrics (true positive rate and false positive rate) of models
> perf_log <- performance(log_pred_obj, "tpr", "fpr")
> perf_svm <- performance(svm_pred_obj, "tpr", "fpr")
> #plotting the ROC and adding a legend
> plot(perf_log, col = 2)
> plot(perf_svm, add = TRUE, col = 3)
> title(main = "ROC Curve")
> legend("bottomright", c("Logistic Regression", "SVM"), lty = 1, col = 2:3)
> #printing the auc values of the models
> performance(log_pred_obj, "auc")@y.values[[1]]
[1] 0.7765577
> performance(svm_pred_obj, "auc")@y.values[[1]]
[1] 0.7399718
```

Figure 19: *ROC + AUC*

In order to output the ROC and the AUC, I first used the models to predict on the test set before producing prediction objects for both models where the credit rating is 1 and calculating the performance metrics, the true positive and false positive rates, of the models. The output is shown in Figure 18.

The ROC (Receiver Operating Characteristic) curve that illustrates the performances of the svm (green line) and the logistics regression (red line) models at all classification thresholds. It has two parameters, the true positive rate (tpr) and the false positive rate (fpr). As we can observe from the ROC curve that both models initially had similar tpr and fpr in the beginning, but later on the svm model started to have lower fpr than the logistic model before they became similar again. Then around 0.4 tpr, the logistic regression model started to have higher tpr than the svm model. The lower the fpr value and the higher tpr value indicate that the false positive predictions are decreasing, and the true positive predictions which correctly belong to the class (class 1) are increasing. So, since the logistic regression model continued to achieve high tpr values with low fpr values compared to the svm model, it is deemed to be the better model for this classification. It also indicates that it has higher overall prediction accuracy. Higher tpr and lower fpr also indicates higher AUC (Area Under the Curve) and this demonstrated in Figure 18 where the logistic regression model has an AUC of 0.7765577 while that of the svm model was 0.7399718, about 0.04 difference between them. These results indicate that the logistic regression model learned better than the svm model and therefore, was able to gain a better discrimination ability in order to distinguish between which clients should be class 1 and which should not even though it was not as good in the beginning. Considering the fact that I have used the full dataset and changed every credit rating that was not 1 to 0, there would have been a class imbalance with 0 being much frequent than 1. The results of the ROC and the AUC indicate that the logistic regression model was able to deal with this imbalance better than the svm model.

REFERENCES

Amelia, A 2019, *Decision tree: Part 2/2*, Towards Data Science, viewed 16 May 2024, <https://towardsdatascience.com/decision-tree-part-2-34b31b1dc328#:~:text=The%20next%20step%20is%20to,every%20feature%2C%20one%20by%20one>.

Farinda, R 2019, *Decision Tree on Imbalanced Dataset*, Medium, viewed 16 May 2024, <https://medium.com/@160shelf/decision-tree-on-imbalanced-dataset-f1575414a6c2#:~:text=It%20is%20a%20condition%20where,as%20the%20performance%20metrics%2C%20etc>.

GeeksforGeeks 2022, *Balanced Binary Tree*, GeeksforGeeks, viewed 16 May 2024, <https://www.geeksforgeeks.org/balanced-binary-tree/>.

Krishnan, S 2021, *Decision Tree for Classification, Entropy, and Information Gain*, Medium, viewed 16 May 2024, <https://medium.com/codex/decision-tree-for-classification-entropy-and-information-gain-cd9f99a26e0d#:~:text=Entropy%20is%20a%20measure%20of,which%20will%20decrease%20the%20entropy>.

Mitrani, A 2019, *Evaluating Categorical Models*, Towards Data Science, viewed 16 May 2024, <https://towardsdatascience.com/evaluating-categorical-models-e667e17987fd>.

Statistics How To n.d., *Homogeneity, Homogeneous Data & Homogeneous Sampling*, Statistics How To, viewed 16 May 2024, <https://www.statisticshowto.com/homogeneity-homogeneous/>.

Zhou, V 2022, *'What Information Gain and Information Entropy are and how they're used to train Decision Trees.', A Simple Explanation of Information Gain and Entropy*, weblog post, 16 September, viewed 16 May 2024, <https://victorzhou.com/blog/information-gain/>.