

1. Data Preprocessing and Train Test Split

```
1 # read dataset
2 data <- read.csv("creditworthiness.csv")
3 # filter customers who have been rating
4 filtered_data <- data[data$credit.rating !=0, ]
5 # view the dimension of data structure in filtered data
6 dim(filtered_data)
7 ### split dataset into 50% training set and 50% test set based on filtered dataset
8 # assign variable n
9 n <- nrow(filtered_data)
10 # random sample 50% as train index
11 train_index <- sample(1:n, size = floor(0.5*n))
12 # set training and test set
13 train_set <- filtered_data[train_index, ]
14 test_set <- filtered_data[-train_index, ]
15 # view the dimension of data structure
16 dim(train_set)
17 dim(test_set)
```

17:14 (Top Level) ↕

R Script ↕

R 4.4.3 · ~/Desktop/911_A2/ ↗

```
> # read dataset
> data <- read.csv("creditworthiness.csv")
> # filter customers who have been rating
> filtered_data <- data[data$credit.rating !=0, ]
> # view the dimension of data structure in filtered data
> dim(filtered_data)
[1] 1962 46
> ### split dataset into 50% training set and 50% test set based on filtered dataset
> # assign variable n
> n <- nrow(filtered_data)
> # random sample 50% as train index
> train_index <- sample(1:n, size = floor(0.5*n))
> # set training and test set
> train_set <- filtered_data[train_index, ]
> test_set <- filtered_data[-train_index, ]
> # view the dimension of data structure
> dim(train_set)
[1] 981 46
> dim(test_set)
[1] 981 46
>
```

In this task, entries with unknown credit ratings were excluded, resulting in a filtered dataset with 1962 rows and 46 variables. The dataset was randomly split into a training set and a test set by selecting 50% of the rows using randomly sampled indices. Each subset contains 981 rows and 46 columns.

1. Random Forest Optimization and Tree Analysis

(a) Fit the decision tree model:

```
1 # Q2.a
2 # install packages for decision tree
3 install.packages("tree")
4 install.packages("rpart.plot")
5 # loading resources
6 library(tree)
7 library(rpart)
8 library(randomForest)
9 library(rpart.plot)
10
11 # create a decision tree with rpart
12 tree_model <- rpart(credit.rating~., data = train_set, method = "class")
13 # view the result of this decision tree
14 print(tree_model)
15 # plot the tree
16 rpart.plot(tree_model,
17             main = "Decision Tree for Credit Rating",
18             type = 3,
19             extra = 101
20 )
```

```

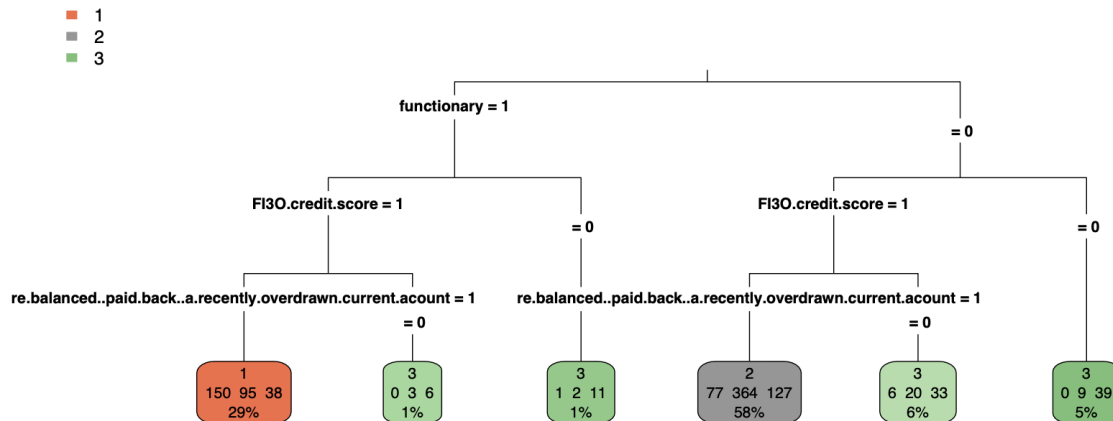
> # view the result of this decision tree
> print(tree_model)
n= 981

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

1) root 981 488 2 (0.23853211 0.50254842 0.25891947)
2) functionary>=0.5 306 155 1 (0.49346405 0.32679739 0.17973856)
4) FI30.credit.score>=0.5 292 142 1 (0.51369863 0.33561644 0.15068493)
8) re.balanced..paid.back..a.recently.overdrawn.current.account>=0.5 283 133 1 (0.53003534 0.33568905 0.13427562) *
9) re.balanced..paid.back..a.recently.overdrawn.current.account< 0.5 9 3 3 (0.00000000 0.33333333 0.66666667) *
5) FI30.credit.score< 0.5 14 3 3 (0.07142857 0.14285714 0.78571429) *
3) functionary< 0.5 675 282 2 (0.12296296 0.58222222 0.29481481)
6) FI30.credit.score>=0.5 627 243 2 (0.13237640 0.61244019 0.25518341)
12) re.balanced..paid.back..a.recently.overdrawn.current.account>=0.5 568 204 2 (0.13556338 0.64084507 0.22359155) *
13) re.balanced..paid.back..a.recently.overdrawn.current.account< 0.5 59 26 3 (0.10169492 0.33898305 0.55932203) *
7) FI30.credit.score< 0.5 48 9 3 (0.00000000 0.18750000 0.81250000) *

```

Decision Tree for Credit Rating



The root node contains 981 instances in the training set, among which 488 are misclassified. The majority class is **Class 2**, and the class distribution at the root node is 23.85% for Class 1, 50.25% for Class 2, and 25.89% for Class 3.

The first split occurs on the binary variable **functionary**, where:

- If **functionary = 1**, the left branch contains **306 instances**, the majority class is **Class 1** (49.35%). The next split is on **FI30.credit.score = 1**:
 - If **yes**, we get 283 instances, which are further split based on **re.balanced..account = 1**.
 - If **yes**, **150 instances** belong to Class 1, majority class = 1.
 - If **no**, **6 instances** belong to Class 3, majority class = 3.
 - If **FI30.credit.score = 0**, the path leads to **14 instances**, majority class = 3.
- If **functionary = 0**, the right branch has **675 instances**, the majority class is **Class 2** (58.22%). The next split is again on **FI30.credit.score = 1**.
 - If **yes**, we obtain **568 instances**, which are further split on **re.balanced..account**:
 - If = 1 -> majority class = 2.
 - If = 0 -> majority class = 3.

- If FI30.credit.score= 0, the path leads to **48 instances**, and Class 3 is the majority.

Asterisk (*) represent leaf nodes, where the decision path terminates.

(b) Create “median” customer and prediction :

```

22 # Q2.b
23 # create median customer
24 median_customer <- data.frame(
25   functionary = 0,
26   re.balanced..paid.back...a.recently.overdrawn.current.account = 1,
27   FI30.credit.score = 1,
28   gender = 0,
29   X0..accounts.at.other.banks = 3,
30   credit.refused.in.past. = 0,
31   years.employed = 3,
32   savings.on.other.accounts = 3,
33   self.employed. = 0,
34   max..account.balance.12.months.ago = 3,
35   min..account.balance.12.months.ago = 3,
36   avrg..account.balance.12.months.ago = 3,
37   max..account.balance.11.months.ago = 3,
38   min..account.balance.11.months.ago = 3,
39   avrg..account.balance.11.months.ago = 3,
40   max..account.balance.10.months.ago = 3,
41   min..account.balance.10.months.ago = 3,
42   avrg..account.balance.10.months.ago = 3,
43   max..account.balance.9.months.ago = 3,
44   min..account.balance.9.months.ago = 3,
45   avrg..account.balance.9.months.ago = 3,
46   max..account.balance.8.months.ago = 3,
47   min..account.balance.8.months.ago = 3,
48   avrg..account.balance.8.months.ago = 3,
49   max..account.balance.7.months.ago = 3,
50   min..account.balance.7.months.ago = 3,
51   avrg..account.balance.7.months.ago = 3,
52   max..account.balance.6.months.ago = 3,
53   min..account.balance.6.months.ago = 3,
54   avrg..account.balance.6.months.ago = 3,
55   max..account.balance.5.months.ago = 3,
56   min..account.balance.5.months.ago = 3,
57   avrg..account.balance.5.months.ago = 3,
58   max..account.balance.4.months.ago = 3,
59   min..account.balance.4.months.ago = 3,
60   avrg..account.balance.4.months.ago = 3,
61   max..account.balance.3.months.ago = 3,
62   min..account.balance.3.months.ago = 3,
63   avrg..account.balance.3.months.ago = 3,
64   max..account.balance.2.months.ago = 3,
65   min..account.balance.2.months.ago = 3,
66   avrg..account.balance.2.months.ago = 3,
67   max..account.balance.1.months.ago = 3,
68   min..account.balance.1.months.ago = 3,
69   avrg..account.balance.1.months.ago = 3
70 )
71 # use tree_model to predict
72 predicted_rating <- predict(tree_model, newdata = median_customer, type = "class")
73 # print predicted result
74 cat("Predicted result for the median customer is:", as.character(predicted_rating), "\n")

```

Verification results: The predicted credit rating of the median customer is **Class 2**.

```

> # use tree_model to predict
> predicted_rating <- predict(tree_model, newdata = median_customer, type = "class")
> # print predicted result
> cat("Predicted result for the median customer is:", as.character(predicted_rating), "\n")
Predicted result for the median customer is: 2
>

```

Based on the output of (a), the process is clearly presented:

functionary = 0 leads to the right child node (functionary < 0.5), FI30.credit.score = 1 leads to the left child node (FI30.credit.score >= 0.5), re.balanced.paid.back...account = 1 leads to the left child node (>= 0.5). And a terminal node that predicts **Class 2** as the credit rating. The results of code verification are consistent with this process.

(c) Create confusion matrix and prediction code:

```

76 # Q2.c
77 # predict the credit rating for the test set
78 predict_result <- predict(tree_model, newdata = test_set, type = "class")
79 # create the confusion matrix
80 confusion_matrix <- table(test_set$credit.rating, predict_result)
81 # print the confusion matrix
82 print(confusion_matrix)
83 # calculate the overall accuracy rate
84 accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
85 # print the result of the overall accuracy
86 cat(accuracy)

> # Q2.c
> # predict the credit rating for the test set
> predict_result <- predict(tree_model, newdata = test_set, type = "class")
> # create the confusion matrix
> confusion_matrix <- table(test_set$credit.rating, predict_result)
> # print the confusion matrix
> print(confusion_matrix)
      predict_result
      1    2    3
1 142 105    2
2   80 373   24
3   31 142   82
> # calculate the overall accuracy rate
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> # print the result of the overall accuracy
> cat(accuracy)
0.6085627
>

```

For the confusion matrix, the rows are the actual classification and the columns are the predicted classifications.

- Total number of correct predictions for each class = $142 + 373 + 82 = 597$
- Total samples = $142 + 105 + 2 + 80 + 373 + 24 + 31 + 142 + 82 = 981$
- Accuracy = $597 / 981 = 60.86\%$

(d)

1) Calculate the entropy of the root node

root 981 488 2 (0.23853211 0.50254842 0.25891947)

Step1: define class probabilities:

```

88 # Q2.d
89 # define class probabilities
90 p1 <- 0.23853211
91 p2 <- 0.50254842
92 p3 <- 0.25891947
93 # calculate the entropy H(D) of the root node
94 H_root <- - (p1 * log2(p1) + p2 * log2(p2) + p3 * log2(p3))
95 H_root

```

Step2: According to the entropy formula to calculate:

$H(D) = -(p_1 \log_2 p_1 + p_2 \log_2 p_2 + p_3 \log_2 p_3)$

$H(D) = -(0.23853211 * \log_2 0.23853211 + 0.50254842 * \log_2 0.50254842 + 0.25891947 * \log_2 0.25891947) \approx 1.49683$

```

> # define class probabilities
> p1 <- 0.23853211
> p2 <- 0.50254842
> p3 <- 0.25891947
> # calculate the entropy H(D) of the root node
> H_root <- - (p1 * log2(p1) + p2 * log2(p2) + p3 * log2(p3))
> H_root
[1] 1.49683

```

2) Entropy gain after the first split at the top of the tree:

Step 1: Left child node:

functionary>=0.5 306 155 1 (0.49346405 0.32679739 0.17973856)

```
114 ## Entropy gain after the first split at the top of the tree:
115 # Step 1: Left child node
116 # set class probabilities
117 p1_left <- 0.49346405
118 p2_left <- 0.32679739
119 p3_left <- 0.17973856
120 # calculate the entropy H(D) of left
121 H_left <- - (p1_left * log2(p1_left) + p2_left * log2(p2_left) + p3_left * log2(p3_left))
122 cat(H_left)
```

The result of the left child node is 1.475167.

```
> # Step 1: Left child node
> # set class probabilities
> p1_left <- 0.49346405
> p2_left <- 0.32679739
> p3_left <- 0.17973856
> # calculate the entropy H(D) of left
> H_left <- - (p1_left * log2(p1_left) + p2_left * log2(p2_left) + p3_left * log2(p3_left))
> cat(H_left)
1.475167
```

Step 2: Right child node:

functionary<0.5 675 282 2 (0.12296296 0.58222222 0.29481481)

```
124 # Step 2: Right child node
125 p1_right <- 0.12296296
126 p2_right <- 0.58222222
127 p3_right <- 0.29481481
128 # calculate the entropy H(D) of right
129 H_right <- - (p1_right * log2(p1_right) + p2_right * log2(p2_right) + p3_right * log2(p3_right))
130 cat(H_right)
131
```

The result of the right child node is 1.345644.

```
> # Step 2: Right child node
> p1_right <- 0.12296296
> p2_right <- 0.58222222
> p3_right <- 0.29481481
> # calculate the entropy H(D) of right
> H_right <- - (p1_right * log2(p1_right) + p2_right * log2(p2_right) + p3_right * log2(p3_right))
> cat(H_right)
1.345644
```

Step 3: Total first split total entropy:

```
132 # Step 3: Calculate weighted average entropy
133 total_samples = 981
134 left_sample <- 306
135 right_sample <- 675
136 weighted_entropy <- (left_sample/total_samples) * H_left + (right_sample/total_samples) * H_right
137 cat(weighted_entropy)
138
```

The result of total first split total entropy is 1.386046.

```
> # Step 3: Calculate weighted average entropy
> total_samples = 981
> left_sample <- 306
> right_sample <- 675
> weighted_entropy <- (left_sample/total_samples) * H_left + (right_sample/total_samples) * H_right
> cat(weighted_entropy)
1.386046
>
```

3) **Information gain** = The entropy of the root node - Total first split total entropy = 1.49683 - 1.386046 = 0.110784

```

139 # calculate information gain
140 Infor_gain = H_root - weighted_entropy
141 cat(Infor_gain)
142 |

```

The result of information gain is **0.110784**.

```

> # calculate information gain
> Infor_gain = H_root - weighted_entropy
> cat(Infor_gain)
0.110784
> |

```

(e) The code of my random forest model:

```

146 # make sure CR as factor
147 train_set$credit.rating <- as.factor(train_set$credit.rating)
148 #create the random forest model
149 rf_model <- randomForest(credit.rating ~., data = train_set)
150 #print the model summary
151 print(rf_model)
152 # generate confusion matrix from the model object
153 cm <- rf_model$confusion
154 # calculate training accuracy from confusion matrix
155 total_correct <- sum(diag(cm))
156 total_instances <- sum(cm[, "class.error"]*rowSums(cm)) + total_correct
157 accuracy <- total_correct/total_instances
158 # print the training accuracy
159 print(paste("Training Accuracy Rate: ", round(accuracy,4)))

```

> #print the model summary
> print(rf_model)

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

OOB estimate of error rate: 43.63%

Confusion matrix:

| | 1 | 2 | 3 | class.error |
|---|----|-----|----|-------------|
| 1 | 74 | 155 | 1 | 0.6782609 |
| 2 | 52 | 403 | 23 | 0.1569038 |
| 3 | 23 | 174 | 76 | 0.7216117 |

```

> # generate confusion matrix from the model object
> cm <- rf_model$confusion
> # calculate training accuracy from confusion matrix
> total_correct <- sum(diag(cm))
> total_instances <- sum(cm[, "class.error"]*rowSums(cm)) + total_correct
> accuracy <- total_correct/total_instances
> # print the training accuracy
> print(paste("Training Accuracy Rate: ", round(accuracy,4)))
[1] "Training Accuracy Rate: 0.5631"

```

The default value of Random Forest Model, ntree = 500, mtry = 6, and the OOB error rate in the training set is **43.63%**, the train set accuracy is **56.31%**.

To optimized code with automated tuning:

```

168 ### to optimized
169 # set exploring space
170 ntree_vals <- c(500, 700, 900, 1000)
171 mtry_vals <- seq(6, 20, 2)
172 # create a table to record
173 results <- data.frame(ntree = integer(), mtry = integer(), train_acc = numeric(), oob_error=numeric())
174 # exploring grid
175 for (nt in ntree_vals) {
176   for (mt in mtry_vals) {
177     cat("Training RF with ntree =", nt, ", mtry =", mt, "\n")
178
179     rf_model <- randomForest(
180       credit.rating ~ ., data = train_set,
181       ntree = nt, mtry = mt, importance = FALSE
182     )
183     # TEST ACC based on train set
184     cm <- rf_model$confusion
185     train_acc <- sum(diag(cm)) / sum(cm[, 1:3])
186     # OOB error at final tree
187     oob <- rf_model$err.rate[nt, "OOB"]
188
189     # inserting results
190     results <- rbind(results,
191                       data.frame(ntree = nt, mtry = mt,
192                                 train_acc = train_acc, oob_error = oob))
193   }
194 }
195 # print results
196 print(results)

```

```

> print(results)
      ntree mtry train_acc oob_error
OOB      500   6 0.5647299 0.4352701
OOB1     500   8 0.5739042 0.4260958
OOB2     500  10 0.5861366 0.4138634
OOB3     500  12 0.5830785 0.4169215
OOB4     500  14 0.5728848 0.4271152
OOB5     500  16 0.5912334 0.4087666
OOB6     500  18 0.5800204 0.4199796
OOB7     500  20 0.5881753 0.4118247
OOB8     700   6 0.5626911 0.4373089
OOB9     700   8 0.5718654 0.4281346
OOB10    700  10 0.5800204 0.4199796
OOB11    700  12 0.5739042 0.4260958
OOB12    700  14 0.5942915 0.4057085
OOB13    700  16 0.5840979 0.4159021
OOB14    700  18 0.5912334 0.4087666
OOB15    700  20 0.5861366 0.4138634
OOB16    900   6 0.5555556 0.4444444
OOB17    900   8 0.5739042 0.4260958
OOB18    900  10 0.5891947 0.4108053
OOB19    900  12 0.5820591 0.4179409
OOB20    900  14 0.5932722 0.4067278
OOB21    900  16 0.5800204 0.4199796
OOB22    900  18 0.5973496 0.4026504
OOB23    900  20 0.5963303 0.4036697
OOB24   1000   6 0.5535168 0.4464832
OOB25   1000   8 0.5810398 0.4189602
OOB26   1000  10 0.5820591 0.4179409
OOB27   1000  12 0.5861366 0.4138634
OOB28   1000  14 0.5881753 0.4118247
OOB29   1000  16 0.5902141 0.4097859
OOB30   1000  18 0.5922528 0.4077472
OOB31   1000  20 0.5891947 0.4108053

```

The best parameter combination is **ntree = 900, mtry = 18**, with the highest accuracy of **59.73%** based on train set, and the lowest OOB error rate of **40.26%**.

(f) Predicting the credit rating using the random forest model on the test set


```

199 # set SEED TO ENSURE RESULT CANBE REPEAT
200 set.seed(123)
201 # create RF model
202 rf_model <- randomForest(credit.rating ~., data = train_set, ntree = 900, mtry = 18)
203 # print the model summary
204 print(rf_model)
205 # extract cm from the model
206 cm <- rf_model$confusion
207 # test based on test set
208 test_pred <- predict(rf_model, newdata = test_set)
209 #create cm for the test set
210 rf_test_cm <- table(test_set$credit.rating, test_pred)
211 # print the cm for test set
212 print(rf_test_cm)
213 # calculate the overall acc for the teat set
214 rf_acc_test <- sum(diag(rf_test_cm) / sum(rf_test_cm))
215 # print the overall acc of the test set
216 print(paste("Overall test accuracy: ", round(rf_acc_test, 4)))

> # print the cm for test set
> print(rf_test_cm)
      test_pred
      1  2  3
1 136 114  3
2  77 382 33
3  28 124 84
> # calculate the overall acc for the teat set
> rf_acc_test <- sum(diag(rf_test_cm) / sum(rf_test_cm))
> # print the overall acc of the test set
> print(paste("Overall test accuracy: ", round(rf_acc_test, 4)))
[1] "Overall test accuracy:  0.6137"

```

For the parameters `ntree = 900`, `mtry = 18`, the confusion matrix for predicting the credit rating from the random forest on the test set as follows, the overall accuracy is **61.37%**. Compared to the decision tree model used in (c), it is slightly higher than **60.86%** of the decision tree. This suggests that the random forest model had a slight edge in overall classification performance.

- For Class 1, the decision tree correctly classified 142 instances, outperforming the random forest's 136.
- For Class 2, the random forest model performed better, correctly classifying 382 instances, compared to 373 by the decision tree.
- For Class 3, the random forest also showed improved performance, predicting 84 instances correctly versus 82 by the decision tree.

2. SVM Hyperparameter Tuning and Modeling

Fit a support vector machine with default settings:

```

1 # Q3
2 library(e1071)
3 # fit the SVM model with default settings
4 svm_model <- svm(credit.rating ~ ., data = train_set, probability = TRUE)
5 # print the model summary
6 summary(svm_model)
7 |

```



```

> # print the model summary
> summary(svm_model)

Call:
svm(formula = credit.rating ~ ., data = train_set, probability = TRUE)

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

Number of Support Vectors: 944

( 449 266 229 )

Number of Classes: 3

Levels:
1 2 3

```

The SVM model was fitted using an RBF (Radial Basis Function) kernel with default parameters. The penalty parameter was set to cost ($c = 1$), and a total of 944 support vectors were identified: 449 in Class 1, 266 in Class 2, and 229 in Class 3.

(a) Predict the credit rating of the median customer

```

8 # Q3.a
9 # predict the credit rating for the median customer using the SVM model
10 median_customer_prediction <- predict(svm_model, newdata = median_customer, decision.values = TRUE)
11 # print the result of the SVM model predicted
12 print(median_customer_prediction)
13 print(paste("Predicted for median customer: ", median_customer_prediction))
14 |

> # print the result of the SVM model predicted
> print(median_customer_prediction)
1
2
attr(,"decision.values")
      2/3      2/1      3/1
1 1.429468 0.9517357 0.08365773
Levels: 1 2 3
> print(paste("Predicted for median customer: ", median_customer_prediction))
[1] "Predicted for median customer: 2"

```

The predicted credit rating for the median customer is **Class 2**. The decision.values indicate a stronger preference for Class 2 compared to Class 1 and Class 3, as reflected in the higher margins: $2/3 = 1.43$, $2/1 = 0.95$.

(b) produce the confusion matrix for predicting the credit rating on test set

```

15 # Q3.b
16 # predict on test set
17 svm_test_prediction <- predict(svm_model, newdata = test_set, probability = TRUE)
18 # create the confusion matrix for the test set
19 svm_test_confusion_matrix <- table(test_set$credit.rating, svm_test_prediction)
20 # print the result for the confusion matrix on the test set
21 print(svm_test_confusion_matrix)
22 # calculate the overall accuracy for the test set
23 svm_test_acc <- sum(diag(svm_test_confusion_matrix)) / sum(svm_test_confusion_matrix)
24 # print the overall acc for test set
25 print(paste("the overall test accuracy (SVM): ", round(svm_test_acc, 4)))

```

```

> # print the result for the confusion matrix on the test set
> print(svm_test_confusion_matrix)
  svm_test_prediction
    1  2  3
1 129 116  8
2  73 383 36
3  26 125 85
> # calculate the overall accuracy for thr test set
> svm_test_acc <- sum(diag(svm_test_confusion_matrix)) / sum(svm_test_confusion_matrix)
> # print the overall acc for test set
> print(paste("the overall test accuracy (SVM): ", round(svm_test_acc, 4)))
[1] "the overall test accuracy (SVM): 0.6086"

```

The overall test accuracy of the SVM model is **60.86%**, based on the confusion matrix with 129, 383, and 85 correct predictions for Classes 1, 2, and 3 respectively.
Calculation: $(129 + 383 + 85) / 981 = 60.86\%$

(c) Automatically tune the SVM to improve prediction

```

28 train_set$credit.rating <- as.factor(train_set$credit.rating)
29 # automatically parameters adjustment by adjusting cost and gamma
30 tune_SVM <- tune(svm, credit.rating ~ .,
31                 data = train_set,
32                 kernel = "radial",
33                 type = "C-classification",
34                 ranges = list(cost = c(0.1, 1, 10, 50), gamma = c(0.01, 0.1, 1, 5))
35 )
36 # retrieve the best model
37 best_SVM_model <- tune_SVM$best.model
38 best_SVM_model
39 best_SVM_model$cost
40 best_SVM_model$gamma
41 # using tuned model to test the test set
42 tuned_prediction <- predict(best_SVM_model, newdata = test_set)
43 # create the confusion matrix
44 tuned_cm <- table(test_set$credit.rating, tuned_prediction)
45 print(tuned_cm)
46 # calculate accuracy
47 tuned_acc <- mean(tuned_prediction == test_set$credit.rating)
48 cat("Tuned SVM accuracy: ", round(tuned_acc * 100, 2), "%")
..

> # retrieve the best model
> best_SVM_model <- tune_SVM$best.model
> best_SVM_model

Call:
best.tune(METHOD = svm, train.x = credit.rating ~ ., data = train_set, ranges = list(cost = c(0.1, 1, 10, 50), gamma = c(0.01, 0.1, 1, 5)), kernel = "radial", type = "C-classification")

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

Number of Support Vectors: 913

> best_SVM_model$cost
[1] 1
> best_SVM_model$gamma
[1] 0.01
> # using tuned model to test the test set
> tuned_prediction <- predict(best_SVM_model, newdata = test_set)
> # create the confusion matrix
> tuned_cm <- table(test_set$credit.rating, tuned_prediction)
> print(tuned_cm)
  tuned_prediction
    1  2  3
1 150  98  5
2  84 383 25
3  32 130 74
> # calculate accuracy
> tuned_acc <- mean(tuned_prediction == test_set$credit.rating)
> cat("Tuned SVM accuracy: ", round(tuned_acc * 100, 2), "%")
Tuned SVM accuracy: 61.88 %

```

Overall test accuracy of the tuned model is **61.88%**, which improves upon the default model's **60.86%** accuracy reported in Q3(b). To improve the performance of the default SVM model, I performed parameter tuning by adjusting the cost and gamma values, and found the **best parameter combination** was: **cost = 1, gamma = 0.01**. The tuning process led to a slight but meaningful improvement in classification performance. In particular:

- Class 1 prediction improved (150 vs. 129 correct in default model), indicating significant improvement.
- Class 2 accuracy remained strong (383 vs. 383), indicating reaching stability.
- Class 3 prediction slightly decreased (74 vs. 85), indicating potential trade-offs.

3. Naive Bayes Classification Analysis

(a) Predict the credit rating of the median customer on the Naive Bayes model

```

30 # Q4.a
31 # predict the credit rating and probabilities for the median customer
32 predicted_rating <- predict(nb_model, newdata = median_customer)
33 warnings(predicted_rating)
34 predicted_probabilities <- predict(nb_model, newdata = median_customer, type = "raw")
35 # print the predicted results
36 print(paste("Predicted credit rating for median customer: ", predicted_rating))
37 print("Predicted probabilities for median customer: ")
38 print(predicted_probabilities)
--

> # print the predicted results
> print(paste("Predicted credit rating for median customer: ", predicted_rating))
[1] "Predicted credit rating for median customer: 2"
> print("Predicted probabilities for median customer: ")
[1] "Predicted probabilities for median customer: "
> print(predicted_probabilities)
      1 2      3
[1,] 2.768196e-37 1 2.753874e-42
>

```

The Naive Bayes model confirms that the median customer belongs to **Class 2** with extremely high confidence (1), while probabilities for Class 1 (= 2.768196e-37) and Class 3 (= 2.753874e-42) are effectively 0.

(b) Fit the naive bayes model

```
1 # fit the naive bayes model
2 library(e1071)
3 library(caret)
4 # read csv
5 data <- read.csv("creditworthiness.csv")
6 # convert all cols to factors
7 category_cols <- names(data)
8 data[category_cols] <- lapply(data[category_cols], as.factor)
9 # filter ppl with credit rating
10 filtered_data <- data[data$credit.rating != 0,]
11 filtered_data$credit.rating <- droplevels(filtered_data$credit.rating)
12 # set seed to make sure results can repeat
13 set.seed(123)
14 # assign variable n
15 n <- nrow(filtered_data)
16 # random sample 50% as train index
17 train_index <- sample(1:n, size = floor(0.5*n))
18 # set training and test set
19 train_set <- filtered_data[train_index, ]
20 test_set <- filtered_data[-train_index, ]
21 # ensure credit.rating as a factor for classification
22 train_set$credit.rating <- as.factor(train_set$credit.rating)
23 test_set$credit.rating <- as.factor(test_set$credit.rating)
24 # fit the naive bayes model
25 nb_model <- naiveBayes(credit.rating ~ ., data = train_set)
26 # print the model summary
27 summary(nb_model)
28 print(nb_model)
29
> print(nb_model)
```

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 |

Step 1: A-priori probabilities showed that of the probabilities of each class in the training data :

- $P(Y = A) = P(Y = 1) = 0.2497452$
- $P(Y = B) = P(Y = 2) = 0.5025484$
- $P(Y = C) = P(Y = 3) = 0.2477064$

Conditional probabilities:

functionary

```
Y      0      1
1 0.3755102 0.6244898
2 0.7931034 0.2068966
3 0.8148148 0.1851852
```

re.balanced..paid.back..a.recently.overdrawn.current.account

```
Y      0      1
1 0.008163265 0.991836735
2 0.030425963 0.969574037
3 0.168724280 0.831275720
```

FI30.credit.score

```
Y      0      1
1 0.000000000 1.000000000
2 0.02028398 0.97971602
3 0.20576132 0.79423868
```

gender

```
Y      0      1
1 0.4530612 0.5469388
2 0.5070994 0.4929006
3 0.5308642 0.4691358
```

X0..accounts.at.other.banks

```
Y      1      2      3      4      5
1 0.2204082 0.1714286 0.2163265 0.2040816 0.1877551
2 0.2089249 0.1784990 0.1784990 0.2210953 0.2129817
3 0.1728395 0.1769547 0.1687243 0.2057613 0.2757202
```

credit.refused.in.past.

```
Y      0      1
1 0.95918367 0.04081633
2 0.90872211 0.09127789
3 0.76543210 0.23456790
```

years.employed

```
Y      1      2      3      4      5
1 0.2040816 0.1877551 0.1918367 0.2285714 0.1877551
2 0.1906694 0.1947262 0.2150101 0.1886410 0.2109533
3 0.1769547 0.2098765 0.2057613 0.2386831 0.1687243
```

savings.on.other.accounts

```
Y      1      2      3      4      5      6
1 0.20408163 0.18775510 0.19183673 0.00000000 0.22857143 0.18775510
2 0.16024341 0.15821501 0.19878296 0.03651116 0.23529412 0.21095335
3 0.17695473 0.20987654 0.20576132 0.00000000 0.23868313 0.16872428
```

self.employed.

```
Y      0      1
1 0.8040816 0.1959184
2 0.8093306 0.1906694
3 0.7818930 0.2181070
```

max..account.balance.12.months.ago

```
Y      1      2      3      4      5
1 0.2326531 0.2081633 0.2204082 0.1755102 0.1632653
2 0.2089249 0.2170385 0.1764706 0.2210953 0.1764706
3 0.1769547 0.2263374 0.2139918 0.1687243 0.2139918
```

min..account.balance.12.months.ago

```
Y      1      2      3      4      5
1 0.2244898 0.1877551 0.2081633 0.2204082 0.1591837
2 0.1866126 0.2170385 0.2271805 0.1744422 0.1947262
3 0.1975309 0.1810700 0.1769547 0.2139918 0.2304527
```

avrg..account.balance.12.months.ago

```
Y      1      2      3      4      5
1 0.2326531 0.2204082 0.2000000 0.1714286 0.1755102
2 0.1967546 0.1967546 0.1926978 0.2271805 0.1866126
3 0.1934156 0.1358025 0.2345679 0.2016461 0.2345679
```

max..account.balance.11.months.ago

```
Y      1      2      3      4      5
1 0.2040816 0.1959184 0.2244898 0.2081633 0.1673469
2 0.1967546 0.2150101 0.1967546 0.1825558 0.2089249
3 0.2181070 0.1316872 0.1893004 0.2551440 0.2057613
```

min..account.balance.11.months.ago

```
Y      1      2      3      4      5
1 0.2571429 0.1510204 0.2040816 0.1877551 0.2000000
2 0.2028398 0.2109533 0.2068966 0.1987830 0.1805274
3 0.2139918 0.2181070 0.1769547 0.1851852 0.2057613
```

avrg..account.balance.11.months.ago

```
Y      1      2      3      4      5
1 0.2081633 0.1551020 0.2285714 0.2408163 0.1673469
2 0.1886410 0.1947262 0.2170385 0.2109533 0.1886410
3 0.2222222 0.2016461 0.1934156 0.1893004 0.1934156
```

max..account.balance.10.months.ago

```
Y      1      2      3      4      5
1 0.1959184 0.2122449 0.1714286 0.2326531 0.1877551
2 0.2292089 0.1886410 0.1825558 0.1845842 0.2150101
3 0.2181070 0.1728395 0.2222222 0.2016461 0.1851852
```

min..account.balance.10.months.ago

```
Y      1      2      3      4      5
1 0.2244898 0.2326531 0.1673469 0.1755102 0.2000000
2 0.1724138 0.2210953 0.1643002 0.2231237 0.2190669
3 0.1604938 0.2757202 0.1893004 0.1893004 0.1851852
```

avrg..account.balance.10.months.ago

```
Y      1      2      3      4      5
1 0.1795918 0.2244898 0.2122449 0.1959184 0.1877551
2 0.1703854 0.2109533 0.1967546 0.2109533 0.2109533
3 0.1769547 0.2139918 0.1893004 0.1728395 0.2469136
```

max..account.balance.9.months.ago

```
Y      1      2      3      4      5
1 0.2000000 0.2204082 0.1877551 0.1918367 0.2000000
2 0.2231237 0.2109533 0.2028398 0.1582150 0.2048682
3 0.2139918 0.1893004 0.1934156 0.2016461 0.2016461
```

min..account.balance.9.months.ago

```
Y      1      2      3      4      5
1 0.2163265 0.2163265 0.1755102 0.2367347 0.1551020
2 0.2292089 0.2150101 0.2109533 0.1724138 0.1724138
3 0.1687243 0.2633745 0.1481481 0.1975309 0.2222222
```

Step 2: Conditional probabilities of the Top 20 attributes from R output. For example, the variable functionary as $x = 0$, so we look at:

- $P(x = 0 \mid Y = 1) = 0.3755102$
- $P(x = 0 \mid Y = 2) = 0.7931034$
- $P(x = 0 \mid Y = 3) = 0.8148148$

Step 3: Apply the Naive Bayes formula using the priori and conditional probabilities for the 20 attributes based on data.frame of the median customer.

Let $X = (x_1 = 0, x_2 = 1, x_3 = 1, \dots, x_{20} = 3)$ be the features of the median customer data frame, the corresponding values are:

- $x_1 = 0$ (functionary = 0)
- $x_2 = 1$ (rebalance.paid.back = 1)
- $x_3 = 1$ (FI30.credit.score = 1)

- ...
- $x_{20} = 3$ (min.balance.9.months.ago = 3)

$$P_1 : P(Y=1) * P(x_1 | Y=1) * P(x_2 | Y=1) * P(x_3 | Y=1) * P(x_4 | Y=1) * P(x_5 | Y=1) * P(x_6 | Y=1) * P(x_7 | Y=1) * P(x_8 | Y=1) * P(x_9 | Y=1) * P(x_{10} | Y=1) * P(x_{11} | Y=1) * P(x_{12} | Y=1) * P(x_{13} | Y=1) * P(x_{14} | Y=1) * P(x_{15} | Y=1) * P(x_{16} | Y=1) * P(x_{17} | Y=1) * P(x_{18} | Y=1) * P(x_{19} | Y=1) * P(x_{20} | Y=1)$$

$$P_2 : P(Y=2) * P(x_1 | Y=2) * P(x_2 | Y=2) * P(x_3 | Y=2) * P(x_4 | Y=2) * P(x_5 | Y=2) * P(x_6 | Y=2) * P(x_7 | Y=2) * P(x_8 | Y=2) * P(x_9 | Y=2) * P(x_{10} | Y=2) * P(x_{11} | Y=2) * P(x_{12} | Y=2) * P(x_{13} | Y=2) * P(x_{14} | Y=2) * P(x_{15} | Y=2) * P(x_{16} | Y=2) * P(x_{17} | Y=2) * P(x_{18} | Y=2) * P(x_{19} | Y=2) * P(x_{20} | Y=2)$$

$$P_3 : P(Y=3) * P(x_1 | Y=3) * P(x_2 | Y=3) * P(x_3 | Y=3) * P(x_4 | Y=3) * P(x_5 | Y=3) * P(x_6 | Y=3) * P(x_7 | Y=3) * P(x_8 | Y=3) * P(x_9 | Y=3) * P(x_{10} | Y=3) * P(x_{11} | Y=3) * P(x_{12} | Y=3) * P(x_{13} | Y=3) * P(x_{14} | Y=3) * P(x_{15} | Y=3) * P(x_{16} | Y=3) * P(x_{17} | Y=3) * P(x_{18} | Y=3) * P(x_{19} | Y=3) * P(x_{20} | Y=3)$$

Then, we normalize to obtain the posterior probabilities:

$P(Y=k | X) = P_k / (P_1 + P_2 + P_3)$, with $k = 1, 2, 3$

And assign the class with the highest posterior as the predicted label.

(c)Predicting the credit rating using naive bayes on the test set

```
42 # predict on the test set with the naive bayes model
43 nb_prediction <- predict(nb_model, newdata = test_set)
44 # create confusion matrix for the test
45 nb_cm <- table(test_set$credit.rating, nb_prediction)
46 # print the confusion matrix for the test
47 print("Confusion mactrix for the test set (NB): ")
48 print(nb_cm)
49 # calculate overall accuracy for th test
50 nb_acc <- sum(diag(nb_cm)) / sum(nb_cm)
51 # print the overall accuracy for test set
52 print(paste("Overall accuracy for the test set(NB): ", round(nb_acc, 4)))
53 confusion_stats <- confusionMatrix(nb_cm)
54 print(confusion_stats)
```

```
> # print the confusion matrix for the test
> print("Confusion mactrix for the test set (NB): ")
[1] "Confusion mactrix for the test set (NB): "
> print(nb_cm)
  nb_prediction
    1    2    3
1  94 133  11
2  66 341  70
3  36 137  93
> # calculate overall accuracy for th test
> nb_acc <- sum(diag(nb_cm)) / sum(nb_cm)
> # print the overall accuracy for test set
> print(paste("Overall accuracy for the test set(NB): ", round(nb_acc, 4)))
[1] "Overall accuracy for the test set(NB):  0.5382"
```

The overall accuracy rate using the Naive Bayes model on the test set is **53.82%**, calculated as $(94 + 341 + 93) / (94 + 133 + 11 + 66 + 341 + 70 + 36 + 137 + 93) = 528 / 981 = 0.5382263$.

- For Class 1 (total: 238 samples), 94 were correctly classified, while 133 were misclassified as Class 2 and 11 as Class 3.
- For Class 2 (total: 477 samples), 341 were correctly classified, with 66 misclassified as Class 1 and 70 as Class 3.
- For Class 3 (total: 266 samples), 93 were correctly classified, while 36 were misclassified as Class 1 and 137 as Class 2.

4. Model Benchmarking ROC AUC Evaluation

| Test Set Prediction | | Decision Tree | | | Random Forest | | | SVM tuned | | | Naive Bayes | | |
|---------------------|---|---------------|-----|----|---------------|-----|----|-----------|-----|----|-------------|-----|----|
| Confusion Matrix | | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 |
| | 1 | 142 | 105 | 2 | 136 | 114 | 3 | 150 | 98 | 5 | 94 | 133 | 11 |
| | 2 | 80 | 373 | 24 | 77 | 382 | 22 | 84 | 383 | 25 | 66 | 341 | 70 |
| | 3 | 31 | 142 | 82 | 28 | 124 | 84 | 32 | 130 | 74 | 36 | 137 | 93 |
| Accuracy | | 60.86% | | | 61.37% | | | 61.88% | | | 53.82% | | |

The best classifier is SVM tuned with the highest test set accuracy of **61.88%**.

The worst classifier is Naive Bayes with the lowest test set accuracy of **53.82%**.

| Decision Tree | TP | FP | FN | TN | Precision | Recall |
|---------------|-----|-----|-----|-----|--------------|--------------|
| Class 1 | 142 | 111 | 107 | 621 | 0.5612648221 | 0.5702811245 |
| Class 2 | 373 | 247 | 104 | 257 | 0.6016129032 | 0.7819706499 |
| Class 3 | 82 | 26 | 173 | 700 | 0.7592592593 | 0.3215686275 |

In the Decision Tree model, the easiest to recognize is Class 2, with recall 78.2% and precision 60.2%. In contrast, Class 3 is the most challenging, with the lowest recall of 32.15%.

| Random Forest | TP | FP | FN | TN | Precision | Recall |
|---------------|-----|-----|-----|-----|--------------|--------------|
| Class 1 | 136 | 105 | 117 | 612 | 0.5643153527 | 0.5375494071 |
| Class 2 | 382 | 238 | 99 | 251 | 0.6161290323 | 0.7941787942 |
| Class 3 | 84 | 25 | 152 | 709 | 0.7706422018 | 0.3559322034 |

For the Random Forest model, Class 2 achieves the highest recall 79.4% and precision 61.61%, while Class 3 remains the lowest recall of 35.6%.

| SVM | TP | FP | FN | TN | Precision | Recall |
|---------|-----|-----|-----|-----|--------------|--------------|
| Class 1 | 150 | 116 | 103 | 612 | 0.5639097744 | 0.5928853755 |
| Class 2 | 383 | 228 | 109 | 261 | 0.6268412439 | 0.7784552846 |
| Class 3 | 74 | 30 | 162 | 715 | 0.7115384615 | 0.313559322 |

The SVM model performs best on Class 2 (precision 62.7%, recall 77.8%), while Class 3 again has the lowest recall at 31.36%.

| Naive Bayes | TP | FP | FN | TN | Precision | Recall |
|-------------|-----|-----|-----|-----|--------------|--------------|
| Class 1 | 94 | 102 | 144 | 641 | 0.4795918367 | 0.3949579832 |
| Class 2 | 341 | 270 | 136 | 234 | 0.558101473 | 0.714884696 |
| Class 3 | 93 | 81 | 173 | 634 | 0.5344827586 | 0.3496240602 |

In the Naive Bayes model, Class 1 and Class 3 both show relatively low recall, with Class 3 being the weakest of 34.96%. However, Class 2 is still the best classification in terms of comprehensive precision (55.81%) and recall (71.49%).

Overall, **Class 3 is the hardest to classify for all models**, while Class 2 is consistently the most accurately identified, Class 1 indicates moderate difficulty in accurate prediction.

Question 6

(a) Prepare data for fitting logistic regression model

```
1 # Q6 predicting A rating
2 library(e1071)
3 library(pROC)
4 # load and prepare data
5 data <- read.csv("creditworthiness.csv")
6 # Filter out samples without credit rating
7 filtered_data <- subset(data, credit.rating != 0)
8 # 50/50 train-test split
9 set.seed(123)
10 n <- nrow(filtered_data)
11 train_indices <- sample(1:n, size = floor(0.5 * n))
12 train_set <- filtered_data[train_indices, ]
13 test_set <- filtered_data[-train_indices, ]
14 # automatically identify predictor types
15 predictor_names <- setdiff(names(train_set), "credit.rating")
16 numeric_columns <- predictor_names[sapply(train_set[predictor_names], is.numeric)]
17 categorical_columns <- setdiff(predictor_names, numeric_columns)
18 # convert categorical predictors to factor
19 train_set[categorical_columns] <- lapply(train_set[categorical_columns], as.factor)
20 test_set[categorical_columns] <- lapply(test_set[categorical_columns], as.factor)
21 # convert target variable to binary factor: 1 = A, 0 = not A
22 train_set$credit.rating <- factor(ifelse(train_set$credit.rating == 1, 1, 0), levels = c(0, 1))
23 test_set$credit.rating <- factor(ifelse(test_set$credit.rating == 1, 1, 0), levels = c(0, 1))
24
```

Fit logistic regression model:

```
26 # Q6.a fit logistic regression
27 log_model <- glm(credit.rating ~ ., data = train_set, family = binomial("logit"))
28
```

In the logistic regression model, the response variable is defined as $I(\text{credit.rating} == 1)$, which evaluates to TRUE for customers with a credit rating of 1 (representing an “A” rating), and FALSE otherwise. This effectively creates a binary classification problem where the target is 1 for “A” ratings and 0 for all other ratings.

The **family = binomial("logit")** argument specifies that the model should use the logistic (logit) link function, making this a logistic regression model. And the model is trained using the train_data dataset to learn the relationship between the predictors and the likelihood of a customer receiving an “A” rating.

(b) summary table of the logistic regression model lift:

```
29 # Q6.b report model summary
30 summary(log_model)
31
```

```

> summary(log_model)

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

Coefficients:

(Intercept)                                Estimate Std. Error z value Pr(>|z|)
functionary                                -16.967890 452.776996  -0.037 0.970106
re.balanced..paid.back..a.recently.overdrawn.current.account  1.951383   0.180109   10.834 < 2e-16 ***
FI30.credit.score                        2.007671   0.756807    2.653 0.007982 **
gender                                   16.204181 452.774921    0.036 0.971451
X0..accounts.at.other.banks              0.110848   0.176689    0.627 0.530421
credit.refused.in.past.                  -0.077863   0.062209   -1.252 0.210705
years.employed                           -1.297887   0.378854   -3.426 0.000613 ***
savings.on.other.accounts                0.797981   0.280181    2.848 0.004398 **
self.employed.                           -0.678957   0.213424   -3.181 0.001466 **
max..account.balance.12.months.ago       0.013329   0.223498    0.060 0.952444
min..account.balance.12.months.ago      -0.098037   0.063136   -1.553 0.120473
avrg..account.balance.12.months.ago     -0.072718   0.063532   -1.145 0.252379
max..account.balance.11.months.ago      -0.154475   0.063310   -2.440 0.014688 *
min..account.balance.11.months.ago      -0.013238   0.063115   -0.210 0.833862
avrg..account.balance.11.months.ago     -0.053227   0.061936   -0.859 0.390120
max..account.balance.10.months.ago      -0.088947   0.063774   -1.400 0.888431
min..account.balance.10.months.ago      -0.005284   0.062196   -0.085 0.932292
avrg..account.balance.10.months.ago     -0.170413   0.062551   -2.724 0.006442 **
max..account.balance.9.months.ago       0.079087   0.063812   -1.239 0.215206
min..account.balance.9.months.ago       0.007368   0.061552    0.120 0.904719
avrg..account.balance.9.months.ago      -0.013907   0.062167   -0.224 0.822983
max..account.balance.8.months.ago       -0.061895   0.061610   -1.005 0.315075
min..account.balance.8.months.ago       -0.078902   0.061707   -1.279 0.201020
avrg..account.balance.8.months.ago      0.056640   0.063255    0.895 0.370559
max..account.balance.7.months.ago       -0.084367   0.064338   -1.311 0.189755
min..account.balance.7.months.ago       0.007968   0.062428    0.128 0.898433
avrg..account.balance.7.months.ago     -0.009398   0.062448   -0.150 0.880376
max..account.balance.6.months.ago       0.064803   0.062188    1.042 0.297393
min..account.balance.6.months.ago      -0.004701   0.062816   -0.075 0.940349
avrg..account.balance.6.months.ago      0.032692   0.062226    0.525 0.599323
max..account.balance.5.months.ago       0.136193   0.064343    2.117 0.034288 *
min..account.balance.5.months.ago      -0.050134   0.060200   -0.833 0.404963
avrg..account.balance.5.months.ago     -0.090495   0.063159   -1.433 0.151908
max..account.balance.4.months.ago       0.001910   0.062203    0.031 0.975503
min..account.balance.4.months.ago      -0.005650   0.061640   -0.092 0.926968
avrg..account.balance.4.months.ago     -0.047362   0.063166   -0.750 0.453374
max..account.balance.3.months.ago      -0.013403   0.062120   -0.216 0.829179
min..account.balance.3.months.ago      -0.099156   0.060656   -1.635 0.102108
avrg..account.balance.3.months.ago     -0.052394   0.063438   -0.826 0.408857
max..account.balance.2.months.ago       0.059194   0.062988    0.940 0.347332
min..account.balance.2.months.ago      0.026477   0.061595    0.430 0.667299
avrg..account.balance.2.months.ago     -0.073250   0.062410   -1.174 0.240518
max..account.balance.1.months.ago       0.031456   0.062606    0.502 0.615352
min..account.balance.1.months.ago      -0.022554   0.064211   -0.351 0.725398
avrg..account.balance.1.months.ago     -0.033762   0.062144   -0.543 0.586937
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1102.75  on 980  degrees of freedom
Residual deviance:  840.54  on 935  degrees of freedom
AIC: 932.54

Number of Fisher Scoring iterations: 16

```

Estimate indicates the estimated coefficient for each predictor:

- A **positive value** means an increase in the predictor increases the log-odds of being in Class 1 that receiving credit rating A.
- A **negative value** means an increase in the predictor decreases the log-odds.

Std. Error measures the variability (uncertainly) in the coefficient estimate, larger values indicate less accuracy.

Z value: The standardized test statistic for the hypothesis test of each coefficient (Estimate / Std. Error).

Pr(>|z|) also known as p-value, is the most critical statistical significance test result is that value < 0.05 is a significant feature, the smaller the value, the better.

Significance codes: Asterisks (*) indicate which attributes have a statistically significant effect on the response after controlling for other predictors in the model, such as functionary and credit.refused.in. past etc.

Model fit statistics:

The model converged after 16 Fisher scoring iterations. The Null Deviance was 1102.75 (on 980 degrees of freedom), and the Residual Deviance was 840.54 (on 935 degrees of freedom). The Akaike Information Criterion (AIC) was 932.54.

(c) Significant predictors at 5% level

```
32 # Q6.c identify significant predictors at 5% level
33 p_values <- summary(log_model)$coefficients[, 4]
34 significant_predictors <- names(p_values[p_values < 0.05])
35 print("Significant predictors at 5% level:")
36 print(significant_predictors)
```

If the **p-value** of a variable is **less than 0.05**, that is, the variable is a significant predictor at 5% level in the logistic regression model. The results are 9 significant predictors at 5% level as below shown:

```
> print("Significant predictors at 5% level:")
[1] "Significant predictors at 5% level:"
> print(significant_predictors)
[1] "functionary"
[3] "credit.refused.in.past."
[5] "savings.on.other.accounts"
[7] "min..account.balance.10.months.ago"
[9] "avrg..account.balance.1.months.ago"
"re.balanced..paid.back..a.recently.overdrawn.current.account"
"years.employed"
"avrg..account.balance.12.months.ago"
"avrg..account.balance.6.months.ago"
```

(d) Fit an SVM model

```
39 # Q6.d fit SVM model
40 svm_model <- svm(credit.rating ~ ., data = train_set, kernel = "linear", probability = TRUE)
41 summary(svm_model)
42
```

```
> summary(svm_model)

Call:
svm(formula = credit.rating ~ ., data = train_set, kernel = "linear", probability = TRUE)

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

Number of Support Vectors:  549

( 313 236 )

Number of Classes:  2

Levels:
0 1
```

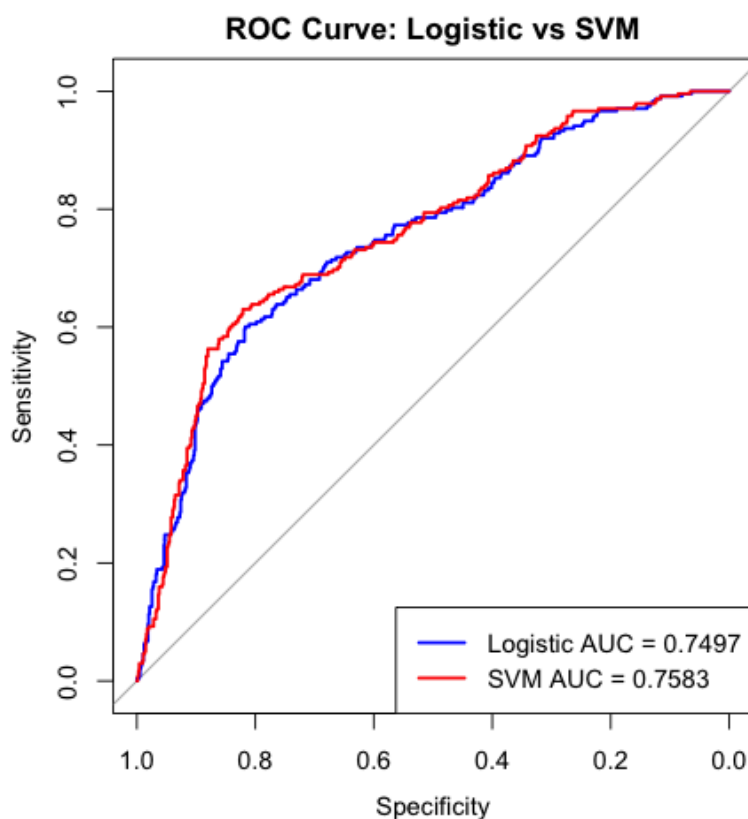
A linear SVM model was fitted using a linear kernel and cost = 1, with 549 support vectors (313 for Class A and 236 for Class “not A”).

(e) ROC comparison between Logistic Regression and SVM

```

42 # Q6.e compare ROC curves
43 # logistic regression predictions
44 log_probs <- predict(log_model, newdata = test_set, type = "response")
45 # SVM predictions with probabilities
46 svm_pred <- predict(svm_model, newdata = test_set, probability = TRUE)
47 svm_probs <- attr(svm_pred, "probabilities")[, "1"] # Probability of class 1 = A rating
48 # ROC and AUC calculation
49 roc_log <- roc(test_set$credit.rating, log_probs)
50 roc_svm <- roc(test_set$credit.rating, svm_probs)
51
52 auc_log <- auc(roc_log)
53 auc_svm <- auc(roc_svm)
54 # plot ROC curves
55 plot(roc_log, col = "blue", main = "ROC Curve: Logistic vs SVM")
56 plot(roc_svm, col = "red", add = TRUE)
57 legend("bottomright",
58       legend = c(paste("Logistic AUC =", round(auc_log, 4)),
59                 paste("SVM AUC =", round(auc_svm, 4))),
60       col = c("blue", "red"), lwd = 2)

```



The ROC chart above compares the binary classification performance of the logistic regression model and the SVM model on the test set. The logistic regression model achieved an AUC of 0.75, whereas the SVM model achieved an AUC of 0.76, indicating that both models perform reasonably well. However, the SVM model shows a slightly better trade-off between sensitivity and specificity, particularly at lower false positive rates.

```

62 # accuracy comparison at threshold 0.5
63 log_class <- ifelse(log_probs > 0.5, 1, 0)
64 svm_class <- ifelse(svm_probs > 0.5, 1, 0)
65
66 log_acc <- mean(log_class == as.numeric(as.character(test_set$credit.rating)))
67 svm_acc <- mean(svm_class == as.numeric(as.character(test_set$credit.rating)))
68
69 cat("Logistic Regression Accuracy:", round(log_acc, 4), "\n")
70 cat("SVM Accuracy:", round(svm_acc, 4), "\n")
71
72 table(filtered_data$credit.rating)
73 table(train_set$credit.rating)
74 table(test_set$credit.rating)

```

```

> svm_acc <- mean(svm_class == as.numeric(as.character(test_set$credit.rating)))
> cat("Logistic Regression Accuracy:", round(log_acc, 4), "\n")
Logistic Regression Accuracy: 0.7778
> cat("SVM Accuracy:", round(svm_acc, 4), "\n")
SVM Accuracy: 0.7931

```

At a 0.5 threshold, SVM also achieved higher accuracy 79.31% compared to logistic regression 77.78%, consistent with its slightly superior AUC.

This result suggests that the SVM classifier has better discriminative ability in this dataset, which is consistent with its strength in finding optimal margins and handling borderline cases. However, the logistic model provides higher interpretability, in particular to understand each predictor variable.

The final model choice between these models would depend on whether predictive performance or model transparency is prioritized in practice.