Question 10.1

Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model. In R, you can use the tree package or the rpart package, and the randomForest package. For each model, describe one or two qualitative takeaways you get from analyzing the results (i.e., don't just stop when you have a good model, but interpret it too).

(a)

I decided to use the tree package to build a regression tree for the first part of this question. After loading my data, I used the tree function in the tree package to build a regression tree. I wanted to predict "Crime", so I used all the data points but the crime column in the crime data set. I then took a summary to see the regression tree model that I just built.

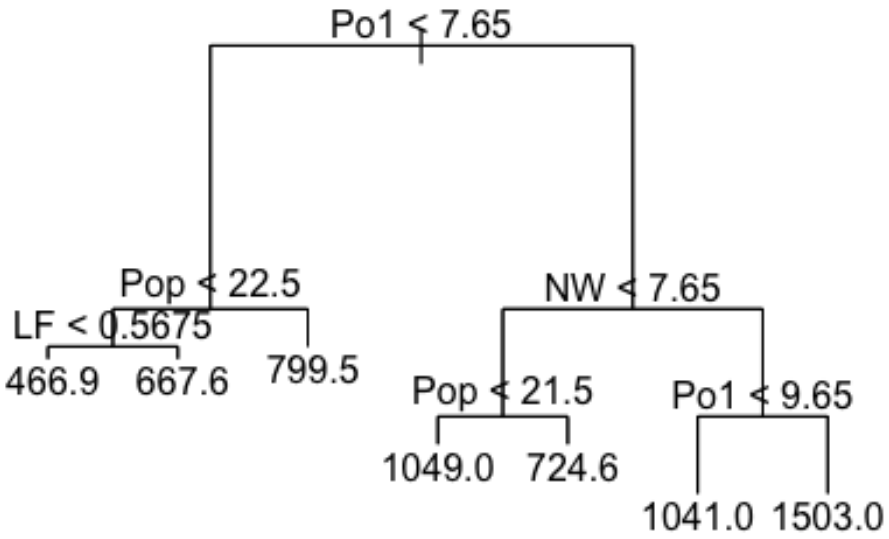
```
library(DAAG)
library(tree)
set.seed(123)
crimedata <- read.table('uscrime.txt', header = TRUE)

tree_model <- tree(Crime ~., data = crimedata)
summary(tree_model)

##
## Regression tree:
## tree(formula = Crime ~ ., data = crimedata)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100
```

What I could see is that the model has used only four variables for the purpose of prediction, "Po1", "Pop", "LF", and "NW," out of 15 of them. It also shows that there are 7 leaves in the regression tree. To get a visual of this, I plotted the model that I just created.

```
plot(tree_model)
text(tree_model)
```



So, when a new data point is introduced to this model, it will first check if PO1 < 7.65 and if it is yes, it will move to the left branch, and if no, it will move to the right branch and will continue to branch until it reaches a final estimation value. I wanted to see how well this model predicted these values, so I wanted to find the R squared value for it.

```
pr <- predict(tree_model, crimedata)
SSE = sum((pr - crimedata$Crime)^2)
Stotal = sum((crimedata$Crime - mean(crimedata$Crime))^2)

R2_tree = 1 - SSE/Stotal
R2_tree

## [1] 0.7244962
```

The R squared value I got for this model is 0.7244962, which is pretty good. This means that the tree accounts for ~72% of the variance in the data. Next, I wanted to see if it made sense to prune any of the branches – meaning if removing any of the nodes I was able to improve the R squared value. Currently, I currently had 7 terminal nodes, so I reduced the number of nodes by 1 each time and checked what the deviation of each amount was.

```
prune.tree(tree_model)$size
```

```
## [1] 7 6 5 4 3 2 1
```

```
prune.tree(tree_model)$dev
```

```
## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928
```

As shown, the deviation keeps on increasing as the number of nodes decreasing – meaning the number of nodes that the model created is the optimal amount. To see what records in the data fall into each given node, I did the below:

```
str(tree_model)
```

```
## List of 6
##  $ frame  :'data.frame': 13 obs. of  5 variables:
##   ..$ var   : Factor w/ 16 levels "<leaf>","M","So",..: 5 9 7 1 1 1 10 9 1
1 ...
##   ..$ n     : num [1:13] 47 23 12 7 5 11 24 10 5 5 ...
##   ..$ dev   : num [1:13] 6880928 779243 243811 48519 77757 ...
##   ..$ yval  : num [1:13] 905 670 550 467 668 ...
##   ..$ splits: chr [1:13, 1:2] "<7.65" "<22.5" "<0.5675" "" ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "cutleft" "cutright"
##  $ where  : Named int [1:47] 6 13 4 13 9 10 12 13 6 5 ...
##   ..- attr(*, "names")= chr [1:47] "1" "2" "3" "4" ...
##  $ terms  :Classes 'terms', 'formula'  language Crime ~ M + So + Ed + Po1
+ Po2 + LF + M.F + Pop + NW + U1 + U2 + Wealth +      Ineq + Prob + Time
##   .. ..- attr(*, "variables")= language list(Crime, M, So, Ed, Po1, Po2, L
F, M.F, Pop, NW, U1, U2, Wealth, Ineq,      Prob, Time)
##   .. ..- attr(*, "factors")= int [1:16, 1:15] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:16] "Crime" "M" "So" "Ed" ...
##   .. .. .. ..$ : chr [1:15] "M" "So" "Ed" "Po1" ...
##   .. ..- attr(*, "term.labels")= chr [1:15] "M" "So" "Ed" "Po1" ...
##   .. ..- attr(*, "order")= int [1:15] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(Crime, M, So, Ed, Po1, Po2, LF
, M.F, Pop, NW, U1, U2, Wealth, Ineq,      Prob, Time)
##   .. ..- attr(*, "dataClasses")= Named chr [1:16] "numeric" "numeric" "num
eric" "numeric" ...
##   .. .. ..- attr(*, "names")= chr [1:16] "Crime" "M" "So" "Ed" ...
##  $ call   : language tree(formula = Crime ~ ., data = crimedata)
##  $ y      : Named int [1:47] 791 1635 578 1969 1234 682 963 1555 856 705 .
..
##   ..- attr(*, "names")= chr [1:47] "1" "2" "3" "4" ...
##  $ weights: num [1:47] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "class")= chr "tree"
##  - attr(*, "xlevels")=List of 15
```

```
##   ..$ M     : NULL
##   ..$ So    : NULL
##   ..$ Ed    : NULL
##   ..$ Po1   : NULL
##   ..$ Po2   : NULL
##   ..$ LF    : NULL
##   ..$ M.F   : NULL
##   ..$ Pop   : NULL
##   ..$ NW    : NULL
##   ..$ U1    : NULL
##   ..$ U2    : NULL
##   ..$ Wealth: NULL
##   ..$ Ineq  : NULL
##   ..$ Prob  : NULL
##   ..$ Time  : NULL
```

```
leaf <- crimedata[which(tree_model$where == 6),]
leaf
```

```
##       M So   Ed Po1 Po2    LF  M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1  15.1  1  9.1 5.8 5.6 0.510 95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 9  15.7  1  9.0 6.5 6.2 0.553 95.5  39 28.6 0.081 2.8   4210 23.9 0.071697
## 12 13.4  0 10.8 7.5 7.1 0.595 97.2  47  5.9 0.083 3.1   5800 17.2 0.031201
## 13 12.8  0 11.3 6.7 6.0 0.624 97.2  28  1.0 0.077 2.5   5070 20.6 0.045302
## 15 15.2  1  8.7 5.7 5.3 0.530 98.6  30  7.2 0.092 4.3   4050 26.4 0.069100
## 21 12.6  0 10.8 7.4 6.7 0.602 98.4  34  1.2 0.102 3.3   5570 19.5 0.022800
## 30 16.6  1  8.9 5.8 5.4 0.521 97.3  46 25.4 0.072 2.6   3960 23.7 0.075298
## 33 14.7  1 10.4 6.3 6.4 0.560 97.2  23  9.5 0.076 2.4   4620 23.3 0.049499
## 37 17.7  1  8.7 5.8 5.6 0.638 97.4  24 34.9 0.076 2.8   3820 25.4 0.045198
## 39 14.9  1  8.8 6.1 5.4 0.515 95.3  36 16.5 0.086 3.5   3950 25.1 0.047099
## 43 16.2  1  9.9 7.5 7.0 0.522 99.6  40 20.8 0.073 2.7   4960 22.4 0.054902
##       Time Crime
## 1  26.2011   791
## 9  29.4001   856
## 12 34.2984   849
## 13 36.2993   511
## 15 22.7008   798
## 21 37.5998   742
## 30 28.3011   696
## 33 25.5005  1072
## 37 31.6995   831
## 39 27.3004   826
## 43 31.9989   823
```

So, all these records are what came up with the regression model for the 6th node in the regression tree. I thought it would be interesting to see how each node equation was derived to calculate the expected crime. A qualitative takeaway from analyzing the results of this regression tree model would be grouping each crime rate into their respective characteristics. The purpose of a regression tree model is to put respective data points with

other similar ones and run a regression based on those similarities - which this one does with each characteristic of what accounts towards a crime rate.

(b)

After loading my data, I wanted to look at the structure of it. Because I wanted to run a regression model and not a classifier, I wanted to make sure the response column, Crime, was not a factor (meaning R would think I want to do classification).

```
set.seed(123)
crimedata <- read.table("uscrime.txt", header = TRUE)

str(crimedata)
```

```
## 'data.frame':    47 obs. of  16 variables:
##  $ M     : num  15.1 14.3 14.2 13.6 14.1 12.1 12.7 13.1 15.7 14 ...
##  $ So    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Ed    : num  9.1 11.3 8.9 12.1 12.1 11 11.1 10.9 9 11.8 ...
##  $ Po1   : num  5.8 10.3 4.5 14.9 10.9 11.8 8.2 11.5 6.5 7.1 ...
##  $ Po2   : num  5.6 9.5 4.4 14.1 10.1 11.5 7.9 10.9 6.2 6.8 ...
##  $ LF    : num  0.51 0.583 0.533 0.577 0.591 0.547 0.519 0.542 0.553 0.632
...
##  $ M.F   : num  95 101.2 96.9 99.4 98.5 ...
##  $ Pop   : int  33 13 18 157 18 25 4 50 39 7 ...
##  $ NW    : num  30.1 10.2 21.9 8 3 4.4 13.9 17.9 28.6 1.5 ...
##  $ U1    : num  0.108 0.096 0.094 0.102 0.091 0.084 0.097 0.079 0.081 0.1
...
##  $ U2    : num  4.1 3.6 3.3 3.9 2 2.9 3.8 3.5 2.8 2.4 ...
##  $ Wealth: int  3940 5570 3180 6730 5780 6890 6200 4720 4210 5260 ...
##  $ Ineq  : num  26.1 19.4 25 16.7 17.4 12.6 16.8 20.6 23.9 17.4 ...
##  $ Prob  : num  0.0846 0.0296 0.0834 0.0158 0.0414 ...
##  $ Time  : num  26.2 25.3 24.3 29.9 21.3 ...
##  $ Crime : int  791 1635 578 1969 1234 682 963 1555 856 705 ...
```

Seeing that it was not a factor and is seen as an integer, I was good to go with building a regression model. I then want to split the data into training and testing. I decided to put 80% of the data into training and 20% into testing.

```
cd <- sample(2, nrow(crimedata), replace = TRUE, prob = c(0.8, 0.2))
train <- crimedata[cd == 1, ]
test <- crimedata[cd == 2, ]
```

Using the training data, I used the random forest function.

```
randomForest_train <- randomForest(Crime ~., data = train, ntree = 1000, mtry
= 6, importance = TRUE)
randomForest_train
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = train, ntree = 1000,   mtry = 6,
importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 81987.69
##                     % Var explained: 24.77
```

By default, "ntree" will = 500 and "mtry" will = number of values / 3. Playing around with different values of the two, I found that creating 1000 trees and using 6 variables at each split to create the best results. I did not create a for loop or anything, so I am sure that this is not the "optimal" values for both, but by just hard inputting values for it was the best I found. The variance this random forest explains is 24.77%, which is not very good. Like I said, by creating a for loop for both the "mtry" and "ntree" I'm sure I could have found a combination that explains a higher percentage, but this was the highest I found. I then wanted to test how well the model performs on the test data set.

```
pr <- predict(randomForest_train, test)

table1 <- table(pr, test$Crime)
table1
```

```
##
## pr                 373 742 754 946 968 1225 1234 1555 1674 1969
##   725.798883333333   1   0   0   0   0    0    0    0    0    0
##   872.156116666667   0   1   0   0   0    0    0    0    0    0
##   918.338416666667   0   0   0   0   0    0    1    0    0    0
##   926.298683333334   0   0   0   0   1    0    0    0    0    0
##   958.561450000001   0   0   1   0   0    0    0    0    0    0
##   977.297516666668   0   0   0   0   0    1    0    0    0    0
##   977.881300000001   0   0   0   1   0    0    0    0    0    0
##   991.398266666667   0   0   0   0   0    0    0    1    0    0
##   1035.88586666667   0   0   0   0   0    0    0    0    1    0
##   1210.71471666667   0   0   0   0   0    0    0    0    0    1
```

I created a confusion matrix using the training data that has been tested with the test data set as the vertical numbers and the crime data from the test data set as the horizontal numbers. To test for the accuracy of this confusion matrix, I took the sum of the diagonals divided by the total number of records in the table.

```
Acc1 <- sum(diag(table1))/sum(table1)
Acc1
```

```
## [1] 0.6
```

Accuracy came out to be 60%, which is not that great, but I am sure could be improved by those optimal “mtry” and “ntree” values. 60% was the highest I could find, with others being in the 40%’s and lower. From analyzing the results on this random forest model, I can see that there is a lot of room for improvement. We learned in class that random forest can usually give better numbers than a singular regression tree can, but that is not this case. I will be interested to see if anybody managed to get higher numbers for % of variance explained and accuracy when doing peer reviews, and what those numbers will be.

Question 10.2

Describe a situation or problem from your job, everyday life, current events, etc., for which a logistic regression model would be appropriate. List some (up to 5) predictors that you might use.

As someone who works in financial advising, predicting whether a client will default on a loan or not (exactly like we have been talking about in class), would be a great logistic regression model to have. This would be a binary outcome, where the model estimates the probability of default (yes/no) based on various predictors related to the client's financial behavior and profile. These predictors would be:

1. Credit score: A numerical representation of the client's creditworthiness.
2. Income level: The client's annual income, which reflects their ability to repay debt.
3. Debt-to-income ratio: The proportion of a client's monthly income that goes towards debt payments.
4. Employment status: Whether the client is employed, self-employed, or unemployed.
5. Loan amount: The size of the loan being applied for, which could affect the risk of default.

Using these, I could help clients determine whether it makes sense for them to attempt to take a loan out or not.

Question 10.3

1. Using the GermanCredit data set germancredit.txt from http://archive.ics.uci.edu/ml/machine- learning-databases/statlog/german / (description at http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29), use logistic regression to find a good predictive model for whether credit applicants are good credit risks or not. Show your model (factors used and their coefficients), the software output, and the quality of fit. You can use the glm function in R. To get a logistic regression (logit) model on data where the response is either zero or one, use family=binomial(link="logit") in your glm function call.

2. Because the model gives a result between 0 and 1, it requires setting a threshold probability to separate between "good" and "bad" answers. In this data set, they estimate that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad. Determine a good threshold probability based on your model.

After loading in the data, I looked through it to see the response variable – it was in 1's and 2's. Since this was going to be a logistic regression, I needed to get them into 0's and 1's. I did this by simply subtracting the column V21 by 1. So now, 0 means good credit and 1 means bad credit.

```
set.seed(123)
creditdata <- read.table("germancredit.txt", header = FALSE)
creditdata$V21 <- creditdata$V21 - 1
```

Next, I split my data into training and testing. I used 70% of the data into training and 30% into testing. Using the glm function, I created a logistic regression of the training data using family = "binomial' and link = "logit."

```
cd = sample(2, nrow(creditdata), replace = TRUE, prob = c(0.7, 0.3))
train <- creditdata[cd ==1, ]
test <- creditdata[cd ==2, ]

model.glm <- glm(V21 ~., data = train, family = 'binomial'(link = 'logit'))
summary(model.glm)

##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
```

```
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.346e+00  1.321e+00   1.019 0.308304
## V1A12       -6.757e-01  2.682e-01  -2.519 0.011775 *
## V1A13       -1.093e+00  4.317e-01  -2.532 0.011346 *
## V1A14       -2.029e+00  2.918e-01  -6.954 3.54e-12 ***
## V2           1.967e-02  1.138e-02   1.728 0.084024 .
## V3A31       -5.376e-01  6.611e-01  -0.813 0.416068
## V3A32       -1.067e+00  4.900e-01  -2.177 0.029514 *
## V3A33       -9.093e-01  5.406e-01  -1.682 0.092527 .
## V3A34       -1.859e+00  5.073e-01  -3.665 0.000247 ***
## V4A41       -1.802e+00  4.620e-01  -3.901 9.58e-05 ***
## V4A410      -1.745e+00  9.924e-01  -1.758 0.078731 .
## V4A42       -7.686e-01  3.208e-01  -2.396 0.016589 *
## V4A43       -8.801e-01  2.999e-01  -2.934 0.003343 **
## V4A44       -1.303e+00  9.777e-01  -1.333 0.182670
## V4A45        1.054e-01  6.687e-01   0.158 0.874787
## V4A46       -4.204e-02  5.183e-01  -0.081 0.935342
## V4A48       -2.149e+00  1.273e+00  -1.688 0.091347 .
## V4A49       -1.015e+00  4.036e-01  -2.515 0.011900 *
## V5           1.163e-04  5.284e-05   2.200 0.027786 *
## V6A62       -3.356e-01  3.640e-01  -0.922 0.356483
## V6A63       -5.177e-01  4.775e-01  -1.084 0.278264
## V6A64       -8.912e-01  5.809e-01  -1.534 0.124969
## V6A65       -9.149e-01  3.138e-01  -2.915 0.003555 **
## V7A72        5.262e-02  5.452e-01   0.097 0.923118
## V7A73        4.242e-02  5.376e-01   0.079 0.937108
## V7A74       -5.485e-01  5.671e-01  -0.967 0.333466
## V7A75        7.413e-02  5.220e-01   0.142 0.887084
## V8           3.148e-01  1.081e-01   2.911 0.003607 **
## V9A92       -3.023e-01  4.849e-01  -0.624 0.532920
## V9A93       -7.749e-01  4.762e-01  -1.627 0.103726
## V9A94       -2.404e-01  5.606e-01  -0.429 0.668082
## V10A102     -3.240e-02  5.227e-01  -0.062 0.950574
## V10A103     -1.285e+00  5.818e-01  -2.209 0.027191 *
## V11         -7.169e-02  1.037e-01  -0.691 0.489495
## V12A122      1.237e-01  3.029e-01   0.408 0.682998
## V12A123      7.141e-02  2.875e-01   0.248 0.803875
## V12A124      7.512e-01  5.107e-01   1.471 0.141267
## V13         -2.546e-02  1.145e-02  -2.224 0.026164 *
## V14A142      5.335e-02  5.161e-01   0.103 0.917658
## V14A143     -7.322e-01  3.045e-01  -2.404 0.016212 *
## V15A152     -8.709e-02  2.938e-01  -0.296 0.766869
## V15A153     -2.876e-01  5.831e-01  -0.493 0.621813
## V16          2.735e-01  2.179e-01   1.255 0.209426
## V17A172      3.759e-01  7.719e-01   0.487 0.626269
## V17A173      4.243e-01  7.417e-01   0.572 0.567288
## V17A174      5.765e-01  7.484e-01   0.770 0.441097
## V18          4.816e-01  3.189e-01   1.510 0.130938
## V19A192     -2.451e-01  2.432e-01  -1.008 0.313603
```

```
## V20A202     -1.694e+00  8.253e-01  -2.052 0.040164 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 853.56  on 704  degrees of freedom
## Residual deviance: 618.21  on 656  degrees of freedom
## AIC: 716.21
##
## Number of Fisher Scoring iterations: 5
```

From the summary, I can see that not all the coefficients are significant. Noticing this, I created a second model using only the coefficients that had a symbol next to them (<= 0.05).

```
model.glm2 <- glm(V21 ~ V1+V2+V3+V4+V5+V6+V8+V10+V13+V14+V20, data = train, f
amily = 'binomial'(link = 'logit'))
summary(model.glm2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 +
##     V13 + V14 + V20, family = binomial(link = "logit"), data = train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.186e+00  7.476e-01   2.925 0.003449 **
## V1A12       -6.865e-01  2.571e-01  -2.670 0.007580 **
## V1A13       -1.114e+00  4.212e-01  -2.645 0.008175 **
## V1A14       -2.066e+00  2.781e-01  -7.432 1.07e-13 ***
## V2           2.063e-02  1.063e-02   1.942 0.052154 .
## V3A31       -7.429e-01  6.265e-01  -1.186 0.235730
## V3A32       -1.295e+00  4.604e-01  -2.813 0.004906 **
## V3A33       -1.034e+00  5.244e-01  -1.971 0.048725 *
## V3A34       -1.943e+00  4.896e-01  -3.969 7.23e-05 ***
## V4A41       -1.675e+00  4.313e-01  -3.884 0.000103 ***
## V4A410      -1.640e+00  9.042e-01  -1.813 0.069761 .
## V4A42       -6.818e-01  3.053e-01  -2.233 0.025536 *
## V4A43       -9.197e-01  2.901e-01  -3.170 0.001525 **
## V4A44       -1.155e+00  9.086e-01  -1.271 0.203575
## V4A45        1.065e-01  6.772e-01   0.157 0.874974
## V4A46        1.079e-01  5.099e-01   0.212 0.832463
## V4A48       -2.198e+00  1.221e+00  -1.800 0.071866 .
## V4A49       -1.095e+00  3.886e-01  -2.819 0.004821 **
## V5           9.669e-05  4.671e-05   2.070 0.038473 *
## V6A62       -3.061e-01  3.440e-01  -0.890 0.373498
## V6A63       -5.642e-01  4.670e-01  -1.208 0.226992
```

```
## V6A64       -7.806e-01  5.607e-01  -1.392 0.163820
## V6A65       -9.759e-01  3.031e-01  -3.220 0.001283 **
## V8           2.805e-01  1.013e-01   2.770 0.005608 **
## V10A102      4.232e-02  5.087e-01   0.083 0.933698
## V10A103     -1.192e+00  5.497e-01  -2.169 0.030109 *
## V13         -2.302e-02  9.399e-03  -2.450 0.014301 *
## V14A142      1.250e-02  4.923e-01   0.025 0.979748
## V14A143     -7.712e-01  2.930e-01  -2.632 0.008492 **
## V20A202     -1.606e+00  8.071e-01  -1.990 0.046627 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 853.56  on 704  degrees of freedom
## Residual deviance: 635.83  on 675  degrees of freedom
## AIC: 695.83
##
## Number of Fisher Scoring iterations: 5
```

To see which model is better, I looked at the AIC values. We learned that a model with a lower AIC value is the better model, and in this case the second model has an AIC value of 695.83 and the first model has an AIC value of 716.21, indicating that the second model is the better model overall. I then used models to see how well they can predict on the test data.

```
pr1 <- predict(model.glm, test, type = 'response')
t = 0.50
for (i in 1:length(pr1)){
  if (pr1[[i]] >= t){
    pr1[[i]] <- 1
  }else{
    pr1[[i]] <- 0
  }
  }
table1 <- table(pr1, test$V21)
table1
```

```
##
## pr1   0   1
##   0 166  46
##   1  36  47
```

```
Acc1 <- sum(diag(table1))/sum(table1)
Acc1
```

```
## [1] 0.7220339
```

```
testc = test%>%select('V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V8', 'V10', 'V13',
'V14', 'V20')
```

```
pr2 <- predict(model.glm2, testc, type = 'response')
t = 0.50
for (i in 1:length(pr2)){
  if (pr2[[i]] >= t){
    pr2[[i]] <- 1
  }else{
    pr2[[i]] <- 0
  }
}
table2 <- table(pr2, test$V21)
table2

##
## pr2   0   1
##   0 172  47
##   1  30  46

Acc2 <- sum(diag(table2))/sum(table2)
Acc2

## [1] 0.7389831
```

When using the predict function, it does not really give values in terms of only 0 and only 1. It gives values from 0 to 1 (0.3, .0746, etc.). However, for logistic regression I needed the values to be either 0 or 1. So, I picked a threshold (0.5) and said that anything above the threshold means to make it a 1 and anything below means to make it a 0. After getting the predictions, I put them into a confusion matrix and checked the accuracy of each model. Model 1 had an accuracy of 0.7220339 and model 2 had an accuracy of 0.7389831. So, based on the AIC values and the accuracy values, it makes sense to say that model 2 is the better one. But when looking at any data, it is important to understand what exactly the data signifies. In this case we also have to look at the true positives, the model predicts that it is a good credit risk, and it actually is a good credit risk, and false positives, the model predicts that it is a good credit risk but it is actually a bad credit risk. With estimating that incorrectly identifying a bad customer as good, is 5 times worse than incorrectly classifying a good customer as bad, the number of false positives should be very low. To investigate this, I want to vary the threshold from 0 to 1 and check the false positives and the accuracy for that threshold. The code for this is:

```
l <- c(seq(0,1, by = 0.05))
accuracy_model2 <- c()
false_positives <- c()
j = 0
for(k in seq(0,1, by = 0.05)){
  pr2 <- predict(model.glm2, testc, type = 'response')

  for(i in 1:length(pr2)){
```

```
    if(pr2[[i]] >= k){
      pr2[[i]] <- 1
    } else {
      pr2[[i]] <- 0
    }
  }

  table2 <- table(pr2, test$V21)
  Acc2 <- sum(diag(table2))/sum(table2)
  accuracy_model2[j] = Acc2
  if(j < length(l)-2 && j > 2){
    false_positives[j] = table2[1,2]/sum(table2)
  } else {
    false_positives[j] = NA
  }
  j = j + 1
}
cbind("Threshold" = l, accuracy_model2, false_positives)
```

```
## Warning in cbind(Threshold = l, accuracy_model2, false_positives): number
of
## rows of result is not a multiple of vector length (arg 2)
```

```
##       Threshold accuracy_model2 false_positives
##  [1,]      0.00       0.4372881              NA
##  [2,]      0.05       0.5627119              NA
##  [3,]      0.10       0.6474576      0.02711864
##  [4,]      0.15       0.6610169      0.05084746
##  [5,]      0.20       0.6813559      0.06779661
##  [6,]      0.25       0.6915254      0.08474576
##  [7,]      0.30       0.6949153      0.11186441
##  [8,]      0.35       0.7084746      0.13220339
##  [9,]      0.40       0.7288136      0.13559322
## [10,]      0.45       0.7389831      0.15932203
## [11,]      0.50       0.7322034      0.17966102
## [12,]      0.55       0.7288136      0.20338983
## [13,]      0.60       0.7322034      0.22033898
## [14,]      0.65       0.7288136      0.23050847
## [15,]      0.70       0.7254237      0.25762712
## [16,]      0.75       0.7084746      0.27457627
## [17,]      0.80       0.6949153      0.30169492
## [18,]      0.85       0.6915254      0.30847458
## [19,]      0.90       0.6881356              NA
## [20,]      0.95       0.6847458              NA
## [21,]      1.00       0.4372881              NA
```

The accuracy of model 2 increases up to a threshold of 0.45, then decreases after that. Looking at the false positives, it keeps on increasing as the threshold increases. This indicates that it is probably better to pick a model that has a lower threshold. Even if it has

a lower accuracy, it is okay because having many false positives is more worrisome in this case. There is a tradeoff between the two, but when a false positive is 5 times as bad as turning a good credit risk away, a low false positive should be selected. In this case, I would select a threshold of 0.10. It is the lowest possible false positive amount and the accuracy tradeoff between all the very small numbers is not that large compared to the false positive tradeoff.

Just for fun I decided to plot the ROC of each model and check the area under the curve of each.

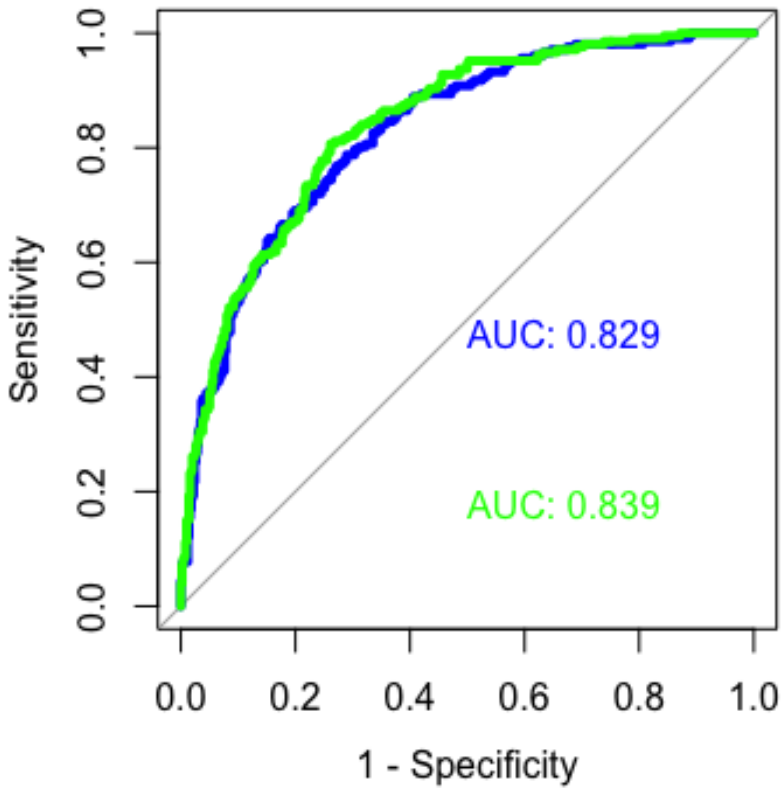
```
par(pty = 's')
roc_data <- roc(train$V21 , model.glm2$fitted.values, plot = TRUE, legacy.axe
s = TRUE, col = "blue", lwd = 4, print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

plot.roc(train$V21 , model.glm$fitted.values,legacy.axes = TRUE, col = "green
", lwd = 4, print.auc = TRUE, add = TRUE, print.auc.y = 0.2)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



What is interesting is that the first model has a larger value for its area under the curve, which is the probability that the model estimates a random “yes” point higher than a random “no” point. This graph indicates that model 1 is the better model of the two because of its larger area under the curve.

CODE:

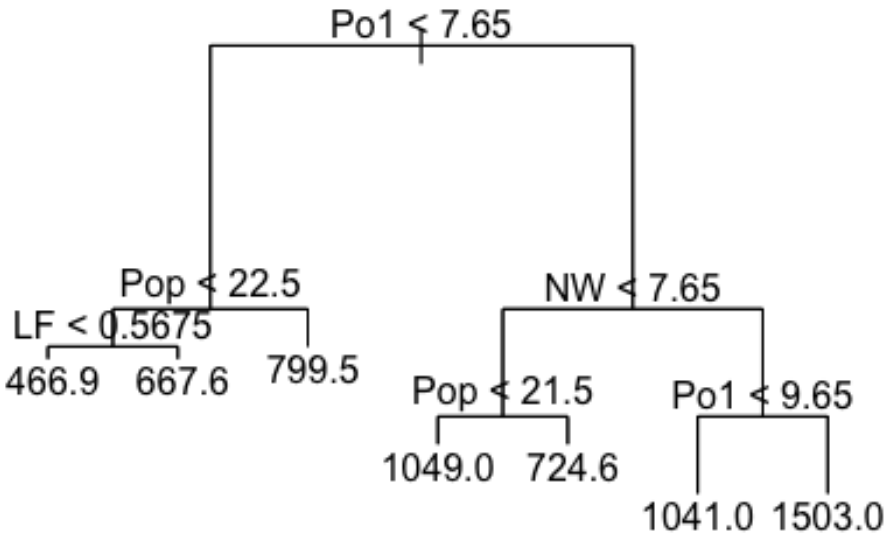
Regression Tree

```
library(DAAG)
library(tree)
set.seed(123)
crimedata <- read.table('uscrime.txt', header = TRUE)

tree_model <- tree(Crime ~., data = crimedata)
summary(tree_model)

##
## Regression tree:
## tree(formula = Crime ~ ., data = crimedata)
## Variables actually used in tree construction:
## [1] "Po1" "Pop" "LF"  "NW"
## Number of terminal nodes:  7
## Residual mean deviance:  47390 = 1896000 / 40
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -573.900  -98.300   -1.545    0.000  110.600  490.100

plot(tree_model)
text(tree_model)
```




```
pr <- predict(tree_model, crimedata)
SSE = sum((pr - crimedata$Crime)^2)
Stotal = sum((crimedata$Crime - mean(crimedata$Crime))^2)

R2_tree = 1 - SSE/Stotal
R2_tree

## [1] 0.7244962

prune.tree(tree_model)$size

## [1] 7 6 5 4 3 2 1

prune.tree(tree_model)$dev

## [1] 1895722 2013257 2276670 2632631 3364043 4383406 6880928

str(tree_model)

## List of 6
##  $ frame  :'data.frame': 13 obs. of  5 variables:
##   ..$ var   : Factor w/ 16 levels "<leaf>","M","So",..: 5 9 7 1 1 1 10 9 1
1 ...
##   ..$ n     : num [1:13] 47 23 12 7 5 11 24 10 5 5 ...
##   ..$ dev   : num [1:13] 6880928 779243 243811 48519 77757 ...
```

```
##   ..$ yval  : num [1:13] 905 670 550 467 668 ...
##   ..$ splits: chr [1:13, 1:2] "<7.65" "<22.5" "<0.5675" "" ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "cutleft" "cutright"
##  $ where  : Named int [1:47] 6 13 4 13 9 10 12 13 6 5 ...
##   ..- attr(*, "names")= chr [1:47] "1" "2" "3" "4" ...
##  $ terms  :Classes 'terms', 'formula'  language Crime ~ M + So + Ed + Po1
+ Po2 + LF + M.F + Pop + NW + U1 + U2 + Wealth +      Ineq + Prob + Time
##   .. ..- attr(*, "variables")= language list(Crime, M, So, Ed, Po1, Po2, L
F, M.F, Pop, NW, U1, U2, Wealth, Ineq,      Prob, Time)
##   .. ..- attr(*, "factors")= int [1:16, 1:15] 0 1 0 0 0 0 0 0 0 0 ...
##   .. .. ..- attr(*, "dimnames")=List of 2
##   .. .. .. ..$ : chr [1:16] "Crime" "M" "So" "Ed" ...
##   .. .. .. ..$ : chr [1:15] "M" "So" "Ed" "Po1" ...
##   .. ..- attr(*, "term.labels")= chr [1:15] "M" "So" "Ed" "Po1" ...
##   .. ..- attr(*, "order")= int [1:15] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "intercept")= int 1
##   .. ..- attr(*, "response")= int 1
##   .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
##   .. ..- attr(*, "predvars")= language list(Crime, M, So, Ed, Po1, Po2, LF
, M.F, Pop, NW, U1, U2, Wealth, Ineq,      Prob, Time)
##   .. ..- attr(*, "dataClasses")= Named chr [1:16] "numeric" "numeric" "num
eric" "numeric" ...
##   .. .. ..- attr(*, "names")= chr [1:16] "Crime" "M" "So" "Ed" ...
##  $ call   : language tree(formula = Crime ~ ., data = crimedata)
##  $ y      : Named int [1:47] 791 1635 578 1969 1234 682 963 1555 856 705 .
..
##   ..- attr(*, "names")= chr [1:47] "1" "2" "3" "4" ...
##  $ weights: num [1:47] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "class")= chr "tree"
##  - attr(*, "xlevels")=List of 15
##   ..$ M     : NULL
##   ..$ So    : NULL
##   ..$ Ed    : NULL
##   ..$ Po1   : NULL
##   ..$ Po2   : NULL
##   ..$ LF    : NULL
##   ..$ M.F   : NULL
##   ..$ Pop   : NULL
##   ..$ NW    : NULL
##   ..$ U1    : NULL
##   ..$ U2    : NULL
##   ..$ Wealth: NULL
##   ..$ Ineq  : NULL
##   ..$ Prob  : NULL
##   ..$ Time  : NULL
```

```
leaf <- crimedata[which(tree_model$where == 6),]
leaf
```

```
##       M So   Ed Po1 Po2    LF  M.F Pop   NW    U1  U2 Wealth Ineq     Prob
## 1  15.1  1  9.1 5.8 5.6 0.510 95.0  33 30.1 0.108 4.1   3940 26.1 0.084602
## 9  15.7  1  9.0 6.5 6.2 0.553 95.5  39 28.6 0.081 2.8   4210 23.9 0.071697
## 12 13.4  0 10.8 7.5 7.1 0.595 97.2  47  5.9 0.083 3.1   5800 17.2 0.031201
## 13 12.8  0 11.3 6.7 6.0 0.624 97.2  28  1.0 0.077 2.5   5070 20.6 0.045302
## 15 15.2  1  8.7 5.7 5.3 0.530 98.6  30  7.2 0.092 4.3   4050 26.4 0.069100
## 21 12.6  0 10.8 7.4 6.7 0.602 98.4  34  1.2 0.102 3.3   5570 19.5 0.022800
## 30 16.6  1  8.9 5.8 5.4 0.521 97.3  46 25.4 0.072 2.6   3960 23.7 0.075298
## 33 14.7  1 10.4 6.3 6.4 0.560 97.2  23  9.5 0.076 2.4   4620 23.3 0.049499
## 37 17.7  1  8.7 5.8 5.6 0.638 97.4  24 34.9 0.076 2.8   3820 25.4 0.045198
## 39 14.9  1  8.8 6.1 5.4 0.515 95.3  36 16.5 0.086 3.5   3950 25.1 0.047099
## 43 16.2  1  9.9 7.5 7.0 0.522 99.6  40 20.8 0.073 2.7   4960 22.4 0.054902
##       Time Crime
## 1  26.2011   791
## 9  29.4001   856
## 12 34.2984   849
## 13 36.2993   511
## 15 22.7008   798
## 21 37.5998   742
## 30 28.3011   696
## 33 25.5005  1072
## 37 31.6995   831
## 39 27.3004   826
## 43 31.9989   823
```

Random Forest

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed(123)
crimedata <- read.table("uscrime.txt", header = TRUE)

str(crimedata)
```

```
## 'data.frame':    47 obs. of  16 variables:
##  $ M     : num  15.1 14.3 14.2 13.6 14.1 12.1 12.7 13.1 15.7 14 ...
##  $ So    : int  1 0 1 0 0 0 1 1 1 0 ...
##  $ Ed    : num  9.1 11.3 8.9 12.1 12.1 11 11.1 10.9 9 11.8 ...
##  $ Po1   : num  5.8 10.3 4.5 14.9 10.9 11.8 8.2 11.5 6.5 7.1 ...
##  $ Po2   : num  5.6 9.5 4.4 14.1 10.1 11.5 7.9 10.9 6.2 6.8 ...
##  $ LF    : num  0.51 0.583 0.533 0.577 0.591 0.547 0.519 0.542 0.553 0.632
...
##  $ M.F   : num  95 101.2 96.9 99.4 98.5 ...
##  $ Pop   : int  33 13 18 157 18 25 4 50 39 7 ...
##  $ NW    : num  30.1 10.2 21.9 8 3 4.4 13.9 17.9 28.6 1.5 ...
##  $ U1    : num  0.108 0.096 0.094 0.102 0.091 0.084 0.097 0.079 0.081 0.1
...
##  $ U2    : num  4.1 3.6 3.3 3.9 2 2.9 3.8 3.5 2.8 2.4 ...
##  $ Wealth: int  3940 5570 3180 6730 5780 6890 6200 4720 4210 5260 ...
##  $ Ineq  : num  26.1 19.4 25 16.7 17.4 12.6 16.8 20.6 23.9 17.4 ...
##  $ Prob  : num  0.0846 0.0296 0.0834 0.0158 0.0414 ...
##  $ Time  : num  26.2 25.3 24.3 29.9 21.3 ...
##  $ Crime : int  791 1635 578 1969 1234 682 963 1555 856 705 ...
```

```r
cd <- sample(2, nrow(crimedata), replace = TRUE, prob = c(0.8, 0.2))
train <- crimedata[cd == 1, ]
test <- crimedata[cd == 2, ]

randomForest_train <- randomForest(Crime ~., data = train, ntree = 1000, mtry
= 6, importance = TRUE)
randomForest_train
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = train, ntree = 1000,      mtry =
6, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 81987.69
##                     % Var explained: 24.77
```

```r
pr <- predict(randomForest_train, test)

table1 <- table(pr, test$Crime)
table1
```

```
##
## pr                 373 742 754 946 968 1225 1234 1555 1674 1969
##   725.798883333333   1   0   0   0   0    0    0    0    0    0
##   872.156116666667   0   1   0   0   0    0    0    0    0    0
##   918.338416666667   0   0   0   0   0    0    1    0    0    0
##   926.298683333334   0   0   0   0   1    0    0    0    0    0
##   958.561450000001   0   0   1   0   0    0    0    0    0    0
##   977.297516666668   0   0   0   0   0    1    0    0    0    0
##   977.881300000001   0   0   0   1   0    0    0    0    0    0
##   991.398266666667   0   0   0   0   0    0    0    1    0    0
##   1035.88586666667   0   0   0   0   0    0    0    0    1    0
##   1210.71471666667   0   0   0   0   0    0    0    0    0    1
```

```r
Acc1 <- sum(diag(table1))/sum(table1)
Acc1
```

```
## [1] 0.6
```

Logistic Regression

```r
set.seed(123)
creditdata <- read.table("germancredit.txt", header = FALSE)

creditdata$V21 <- creditdata$V21 - 1

cd = sample(2, nrow(creditdata), replace = TRUE, prob = c(0.7, 0.3))
train <- creditdata[cd ==1, ]
test <- creditdata[cd ==2, ]

model.glm <- glm(V21 ~., data = train, family = 'binomial'(link = 'logit'))
summary(model.glm)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.346e+00  1.321e+00   1.019 0.308304
## V1A12       -6.757e-01  2.682e-01  -2.519 0.011775 *
## V1A13       -1.093e+00  4.317e-01  -2.532 0.011346 *
## V1A14       -2.029e+00  2.918e-01  -6.954 3.54e-12 ***
## V2           1.967e-02  1.138e-02   1.728 0.084024 .
## V3A31       -5.376e-01  6.611e-01  -0.813 0.416068
## V3A32       -1.067e+00  4.900e-01  -2.177 0.029514 *
## V3A33       -9.093e-01  5.406e-01  -1.682 0.092527 .
## V3A34       -1.859e+00  5.073e-01  -3.665 0.000247 ***
## V4A41       -1.802e+00  4.620e-01  -3.901 9.58e-05 ***
## V4A410      -1.745e+00  9.924e-01  -1.758 0.078731 .
## V4A42       -7.686e-01  3.208e-01  -2.396 0.016589 *
## V4A43       -8.801e-01  2.999e-01  -2.934 0.003343 **
## V4A44       -1.303e+00  9.777e-01  -1.333 0.182670
## V4A45        1.054e-01  6.687e-01   0.158 0.874787
## V4A46       -4.204e-02  5.183e-01  -0.081 0.935342
## V4A48       -2.149e+00  1.273e+00  -1.688 0.091347 .
## V4A49       -1.015e+00  4.036e-01  -2.515 0.011900 *
## V5           1.163e-04  5.284e-05   2.200 0.027786 *
## V6A62       -3.356e-01  3.640e-01  -0.922 0.356483
## V6A63       -5.177e-01  4.775e-01  -1.084 0.278264
## V6A64       -8.912e-01  5.809e-01  -1.534 0.124969
## V6A65       -9.149e-01  3.138e-01  -2.915 0.003555 **
## V7A72        5.262e-02  5.452e-01   0.097 0.923118
## V7A73        4.242e-02  5.376e-01   0.079 0.937108
## V7A74       -5.485e-01  5.671e-01  -0.967 0.333466
## V7A75        7.413e-02  5.220e-01   0.142 0.887084
## V8           3.148e-01  1.081e-01   2.911 0.003607 **
## V9A92       -3.023e-01  4.849e-01  -0.624 0.532920
## V9A93       -7.749e-01  4.762e-01  -1.627 0.103726
```

```
## V9A94       -2.404e-01  5.606e-01  -0.429 0.668082
## V10A102     -3.240e-02  5.227e-01  -0.062 0.950574
## V10A103     -1.285e+00  5.818e-01  -2.209 0.027191 *
## V11         -7.169e-02  1.037e-01  -0.691 0.489495
## V12A122      1.237e-01  3.029e-01   0.408 0.682998
## V12A123      7.141e-02  2.875e-01   0.248 0.803875
## V12A124      7.512e-01  5.107e-01   1.471 0.141267
## V13         -2.546e-02  1.145e-02  -2.224 0.026164 *
## V14A142      5.335e-02  5.161e-01   0.103 0.917658
## V14A143     -7.322e-01  3.045e-01  -2.404 0.016212 *
## V15A152     -8.709e-02  2.938e-01  -0.296 0.766869
## V15A153     -2.876e-01  5.831e-01  -0.493 0.621813
## V16          2.735e-01  2.179e-01   1.255 0.209426
## V17A172      3.759e-01  7.719e-01   0.487 0.626269
## V17A173      4.243e-01  7.417e-01   0.572 0.567288
## V17A174      5.765e-01  7.484e-01   0.770 0.441097
## V18          4.816e-01  3.189e-01   1.510 0.130938
## V19A192     -2.451e-01  2.432e-01  -1.008 0.313603
## V20A202     -1.694e+00  8.253e-01  -2.052 0.040164 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 853.56  on 704  degrees of freedom
## Residual deviance: 618.21  on 656  degrees of freedom
## AIC: 716.21
##
## Number of Fisher Scoring iterations: 5
```

```
model.glm2 <- glm(V21 ~ V1+V2+V3+V4+V5+V6+V8+V10+V13+V14+V20, data = train, f
amily = 'binomial'(link = 'logit'))
summary(model.glm2)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V10 +
##     V13 + V14 + V20, family = binomial(link = "logit"), data = train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.186e+00  7.476e-01   2.925 0.003449 **
## V1A12       -6.865e-01  2.571e-01  -2.670 0.007580 **
## V1A13       -1.114e+00  4.212e-01  -2.645 0.008175 **
## V1A14       -2.066e+00  2.781e-01  -7.432 1.07e-13 ***
## V2           2.063e-02  1.063e-02   1.942 0.052154 .
## V3A31       -7.429e-01  6.265e-01  -1.186 0.235730
## V3A32       -1.295e+00  4.604e-01  -2.813 0.004906 **
## V3A33       -1.034e+00  5.244e-01  -1.971 0.048725 *
## V3A34       -1.943e+00  4.896e-01  -3.969 7.23e-05 ***
```

```
## V4A41       -1.675e+00  4.313e-01  -3.884 0.000103 ***
## V4A410      -1.640e+00  9.042e-01  -1.813 0.069761 .
## V4A42       -6.818e-01  3.053e-01  -2.233 0.025536 *
## V4A43       -9.197e-01  2.901e-01  -3.170 0.001525 **
## V4A44       -1.155e+00  9.086e-01  -1.271 0.203575
## V4A45        1.065e-01  6.772e-01   0.157 0.874974
## V4A46        1.079e-01  5.099e-01   0.212 0.832463
## V4A48       -2.198e+00  1.221e+00  -1.800 0.071866 .
## V4A49       -1.095e+00  3.886e-01  -2.819 0.004821 **
## V5           9.669e-05  4.671e-05   2.070 0.038473 *
## V6A62       -3.061e-01  3.440e-01  -0.890 0.373498
## V6A63       -5.642e-01  4.670e-01  -1.208 0.226992
## V6A64       -7.806e-01  5.607e-01  -1.392 0.163820
## V6A65       -9.759e-01  3.031e-01  -3.220 0.001283 **
## V8           2.805e-01  1.013e-01   2.770 0.005608 **
## V10A102      4.232e-02  5.087e-01   0.083 0.933698
## V10A103     -1.192e+00  5.497e-01  -2.169 0.030109 *
## V13         -2.302e-02  9.399e-03  -2.450 0.014301 *
## V14A142      1.250e-02  4.923e-01   0.025 0.979748
## V14A143     -7.712e-01  2.930e-01  -2.632 0.008492 **
## V20A202     -1.606e+00  8.071e-01  -1.990 0.046627 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 853.56  on 704  degrees of freedom
## Residual deviance: 635.83  on 675  degrees of freedom
## AIC: 695.83
##
## Number of Fisher Scoring iterations: 5
```

```
pr1 <- predict(model.glm, test, type = 'response')
t = 0.50
for (i in 1:length(pr1)){
  if (pr1[[i]] >= t){
    pr1[[i]] <- 1
  }else{
    pr1[[i]] <- 0
  }
  }
table1 <- table(pr1, test$V21)
table1
```

```
##
## pr1   0   1
##   0 166  46
##   1  36  47
```

```r
Acc1 <- sum(diag(table1))/sum(table1)
Acc1
```

```
## [1] 0.7220339
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
testc = test%>%select('V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V8', 'V10', 'V13',
'V14', 'V20')
pr2 <- predict(model.glm2, testc, type = 'response')
t = 0.50
for (i in 1:length(pr2)){
  if (pr2[[i]] >= t){
    pr2[[i]] <- 1
  }else{
    pr2[[i]] <- 0
  }
}
table2 <- table(pr2, test$V21)
table2
```

```
##
## pr2   0   1
##   0 172  47
##   1  30  46
```

```r
Acc2 <- sum(diag(table2))/sum(table2)
Acc2
```

```
## [1] 0.7389831
```

```r
l <- c(seq(0,1, by = 0.05))
accuracy_model2 <- c()
false_positives <- c()
j = 0
for(k in seq(0,1, by = 0.05)){
  pr2 <- predict(model.glm2, testc, type = 'response')

  for(i in 1:length(pr2)){
    if(pr2[[i]] >= k){
      pr2[[i]] <- 1
```

```
    } else {
      pr2[[i]] <- 0
    }
  }

  table2 <- table(pr2, test$V21)
  Acc2 <- sum(diag(table2))/sum(table2)
  accuracy_model2[j] = Acc2
  if(j < length(l)-2 && j > 2){
    false_positives[j] = table2[1,2]/sum(table2)
  } else {
    false_positives[j] = NA
  }
  j = j + 1
}
cbind("Threshold" = l, accuracy_model2, false_positives)
```

```
## Warning in cbind(Threshold = l, accuracy_model2, false_positives): number
of
## rows of result is not a multiple of vector length (arg 2)
```

```
##       Threshold accuracy_model2 false_positives
##  [1,]      0.00       0.4372881              NA
##  [2,]      0.05       0.5627119              NA
##  [3,]      0.10       0.6474576      0.02711864
##  [4,]      0.15       0.6610169      0.05084746
##  [5,]      0.20       0.6813559      0.06779661
##  [6,]      0.25       0.6915254      0.08474576
##  [7,]      0.30       0.6949153      0.11186441
##  [8,]      0.35       0.7084746      0.13220339
##  [9,]      0.40       0.7288136      0.13559322
## [10,]      0.45       0.7389831      0.15932203
## [11,]      0.50       0.7322034      0.17966102
## [12,]      0.55       0.7288136      0.20338983
## [13,]      0.60       0.7322034      0.22033898
## [14,]      0.65       0.7288136      0.23050847
## [15,]      0.70       0.7254237      0.25762712
## [16,]      0.75       0.7084746      0.27457627
## [17,]      0.80       0.6949153      0.30169492
## [18,]      0.85       0.6915254      0.30847458
## [19,]      0.90       0.6881356              NA
## [20,]      0.95       0.6847458              NA
## [21,]      1.00       0.4372881              NA
```

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var

par(pty = 's')
roc_data <- roc(train$V21 , model.glm2$fitted.values, plot = TRUE, legacy.axe
s = TRUE, col = "blue", lwd = 4, print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases

plot.roc(train$V21 , model.glm$fitted.values,legacy.axes = TRUE, col = "green
", lwd = 4, print.auc = TRUE, add = TRUE, print.auc.y = 0.2)

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```