Week 7 Assignment - Advanced Regression

Omer Farooq (EDx ID: mfarooq4)

02/25/2020

Table of Contents

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).

Regression Tree Model

Before I jumped into building the model, I summarized the strategy I took below:

1. Build a regression tree based on full data using tree package.
2. Check if Pruning is required and whether pruning any branches improves the full data tree model.
3. Build a regression tree using cross validation.
4. Check if pruning is required on the CV tree model.
5. Use regression on branches of simpler tree

First, I loaded the Crime data.

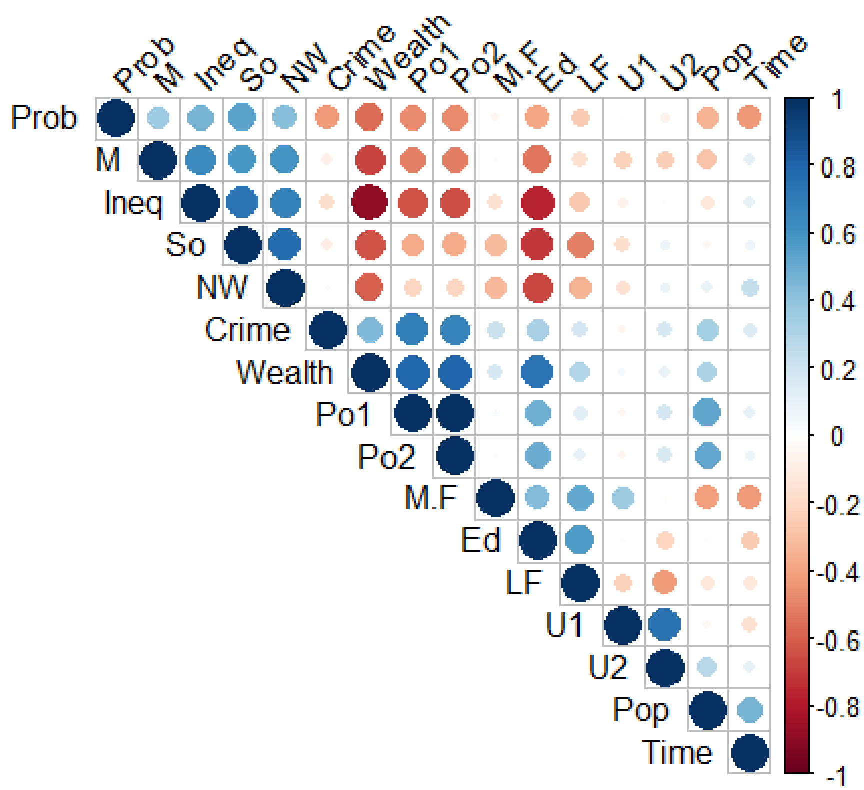
```
#setting the seed so that results are the same at every run
set.seed(101)

#loading data
crimedata <- read.delim("data_10.1/uscrime.txt")

#quick glance at the data
head(crimedata)

##      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
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4 0.029599
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0 0.083401
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7 0.015801
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4 0.041399
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6 0.034201
##      Time Crime
## 1 26.2011   791
## 2 25.2999  1635
## 3 24.3006   578
## 4 29.9012  1969
## 5 21.2998  1234
## 6 20.9995   682
```

Before I jumped into building the regression tree, I ran a correlation matrix on the data to get a sense on predictors that are highly correlated.

```
library(corrplot) #for correlation plot

#pearson correlation matrix
corrmat <- cor(crimedata)
round(corrmat, 2)

##            M    So    Ed   Po1   Po2    LF   M.F   Pop    NW    U1    U2 Wealth
## M       1.00  0.58 -0.53 -0.51 -0.51 -0.16 -0.03 -0.28  0.59 -0.22 -0.24  -0.67
## So      0.58  1.00 -0.70 -0.37 -0.38 -0.51 -0.31 -0.05  0.77 -0.17  0.07  -0.64
## Ed     -0.53 -0.70  1.00  0.48  0.50  0.56  0.44 -0.02 -0.66  0.02 -0.22   0.74
## Po1    -0.51 -0.37  0.48  1.00  0.99  0.12  0.03  0.53 -0.21 -0.04  0.19   0.79
## Po2    -0.51 -0.38  0.50  0.99  1.00  0.11  0.02  0.51 -0.22 -0.05  0.17   0.79
## LF     -0.16 -0.51  0.56  0.12  0.11  1.00  0.51 -0.12 -0.34 -0.23 -0.42   0.29
## M.F    -0.03 -0.31  0.44  0.03  0.02  0.51  1.00 -0.41 -0.33  0.35 -0.02   0.18
## Pop    -0.28 -0.05 -0.02  0.53  0.51 -0.12 -0.41  1.00  0.10 -0.04  0.27   0.31
## NW      0.59  0.77 -0.66 -0.21 -0.22 -0.34 -0.33  0.10  1.00 -0.16  0.08  -0.59
## U1     -0.22 -0.17  0.02 -0.04 -0.05 -0.23  0.35 -0.04 -0.16  1.00  0.75   0.04
## U2     -0.24  0.07 -0.22  0.19  0.17 -0.42 -0.02  0.27  0.08  0.75  1.00   0.09
## Wealth -0.67 -0.64  0.74  0.79  0.79  0.29  0.18  0.31 -0.59  0.04  0.09   1.00
## Ineq    0.64  0.74 -0.77 -0.63 -0.65 -0.27 -0.17 -0.13  0.68 -0.06  0.02  -0.88
## Prob    0.36  0.53 -0.39 -0.47 -0.47 -0.25 -0.05 -0.35  0.43 -0.01 -0.06  -0.56
## Time    0.11  0.07 -0.25  0.10  0.08 -0.12 -0.43  0.46  0.23 -0.17  0.10   0.00
## Crime  -0.09 -0.09  0.32  0.69  0.67  0.19  0.21  0.34  0.03 -0.05  0.18   0.44
##         Ineq  Prob  Time Crime
## M       0.64  0.36  0.11 -0.09
## So      0.74  0.53  0.07 -0.09
## Ed     -0.77 -0.39 -0.25  0.32
```

```
## Po1     -0.63 -0.47  0.10  0.69
## Po2     -0.65 -0.47  0.08  0.67
## LF      -0.27 -0.25 -0.12  0.19
## M.F     -0.17 -0.05 -0.43  0.21
## Pop     -0.13 -0.35  0.46  0.34
## NW       0.68  0.43  0.23  0.03
## U1      -0.06 -0.01 -0.17 -0.05
## U2       0.02 -0.06  0.10  0.18
## Wealth -0.88 -0.56  0.00  0.44
## Ineq     1.00  0.47  0.10 -0.18
## Prob     0.47  1.00 -0.44 -0.43
## Time     0.10 -0.44  1.00  0.15
## Crime   -0.18 -0.43  0.15  1.00
```

```
#plotting the correlation matrix
corrplot(cormat, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```

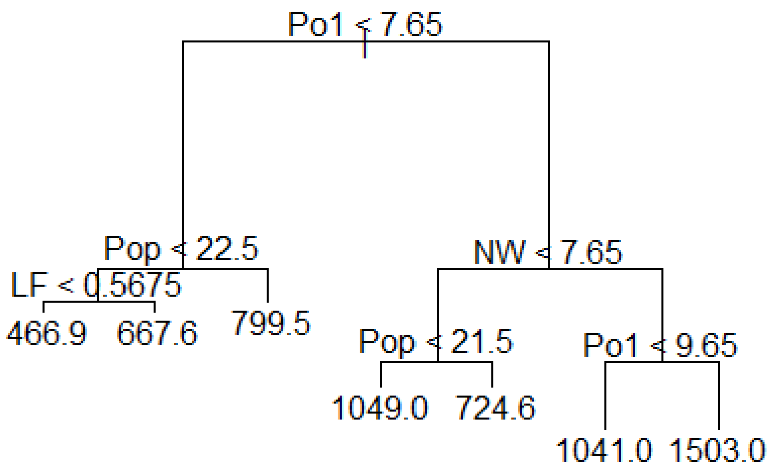


I wrote a function to calculate R-Sq for the tree models first, so that we can use it repeatedly in the code later on.

```
rsquare_func <- function(y_hat, y){
  Sum_Sq_Error <- sum((y_hat - y)^2)
  Sum_sq_total <- sum((y-mean(y))^2)
  r_sq <- 1 - Sum_Sq_Error/Sum_sq_total
  return (r_sq)
}
```

I then built the regression tree with all data. The output showed that this tree has 7 branches and the first split was done at Po1 which was not surprising given Po1 had the most correlation to Crime (from the pearson matrix above). The R-Sq of this model is 72% which was pretty high but there coud be overfitting going on here given we had on average 6 to 7 data points in each leaf. It was also not enough data points to perform regression on each leaf.

```
set.seed(101)

#building regression tree with all data
library(tree)
tree1 <- tree(Crime~., data = crimedata)
tree1

## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 47 6881000  905.1
##    2) Po1 < 7.65 23  779200  669.6
##      4) Pop < 22.5 12  243800  550.5
##        8) LF < 0.5675 7   48520  466.9 *
##        9) LF > 0.5675 5   77760  667.6 *
##      5) Pop > 22.5 11  179500  799.5 *
##    3) Po1 > 7.65 24 3604000 1131.0
##      6) NW < 7.65 10  557600  886.9
##       12) Pop < 21.5 5  146400 1049.0 *
##       13) Pop > 21.5 5  147800  724.6 *
##      7) NW > 7.65 14 2027000 1305.0
##       14) Po1 < 9.65 6  170800 1041.0 *
##       15) Po1 > 9.65 8 1125000 1503.0 *

summary(tree1)

##
## 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

#plotting the tree
plot(tree1)
text(tree1)
```

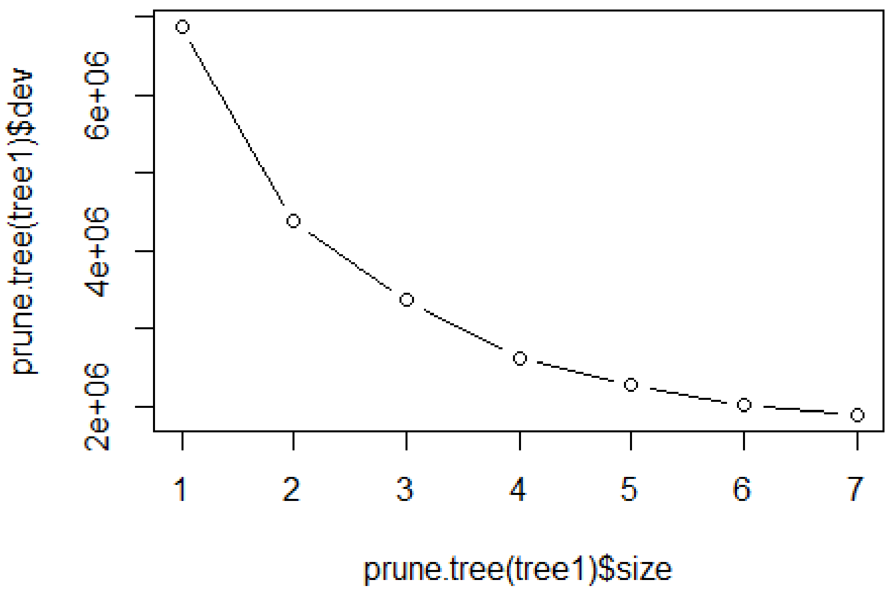



```
#R-Sq of the tree model
rsquare_func(predict(tree1),crimedata$Crime)

## [1] 0.7244962
```

I checked if pruning would improve this model. Plot of deviance of tree (it's a quality of fit measure for trees) vs number of terminal nodes (aka leaves) helped see if deviance stopped decreasing after certain number of leaves. Looking at the plot, It could be said that 4 leaves was a good point after which deviance decrease was not significant.

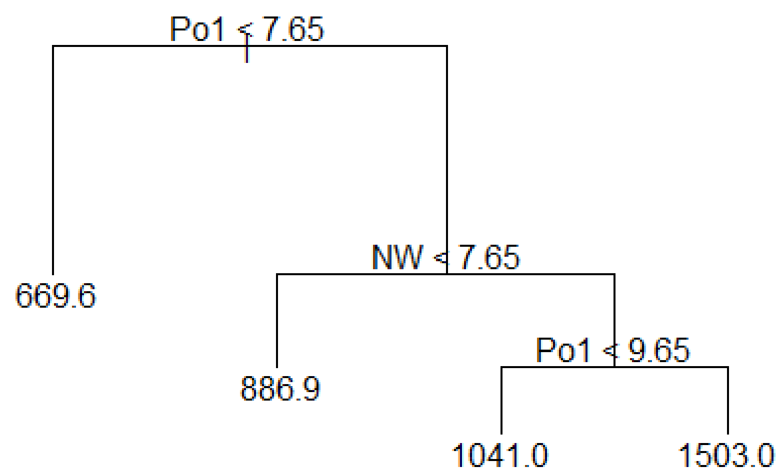
The pruned tree with 4 leaves had lower R-sq and slightly higher mean deviance compared to full 7 leaf tree.

```
#plotting deviation vs nodes
plot(prune.tree(tree1)$size, prune.tree(tree1)$dev, type = "b")
```




```
tree1_prune <- prune.tree(tree1, best = 4)
tree1_prune

## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 47 6881000  905.1
##    2) Po1 < 7.65 23  779200  669.6 *
##    3) Po1 > 7.65 24 3604000 1131.0
##      6) NW < 7.65 10  557600  886.9 *
##      7) NW > 7.65 14 2027000 1305.0
##       14) Po1 < 9.65 6  170800 1041.0 *
##       15) Po1 > 9.65 8 1125000 1503.0 *

summary(tree1_prune)

##
## Regression tree:
## snip.tree(tree = tree1, nodes = c(6L, 2L))
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  4
## Residual mean deviance:  61220 = 2633000 / 43
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -573.90 -152.60   35.39    0.00  158.90  490.10
```

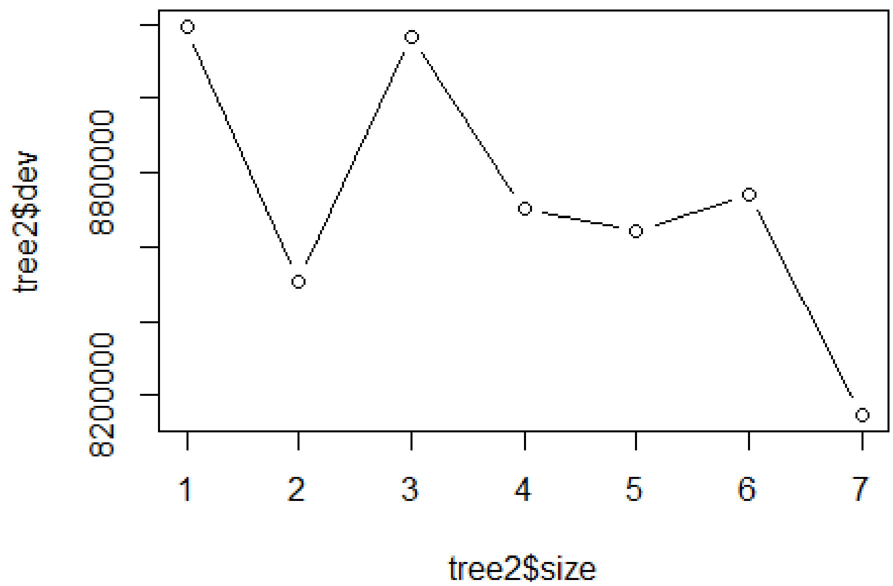
```
#plotting the tree
plot(tree1_prune)
text(tree1_prune)
```




```
#R-Sq of the tree model
rsquare_func(predict(tree1_prune),crimedata$Crime)

## [1] 0.6174017
```

I then built a tree with 10-fold cross-validation to see if a different pruning would help improve results. The deviance plot was all over the place. Error dropped and then got larger with more nodes. It's inconclusive. The deviance was actually larger than the one we had without CV.

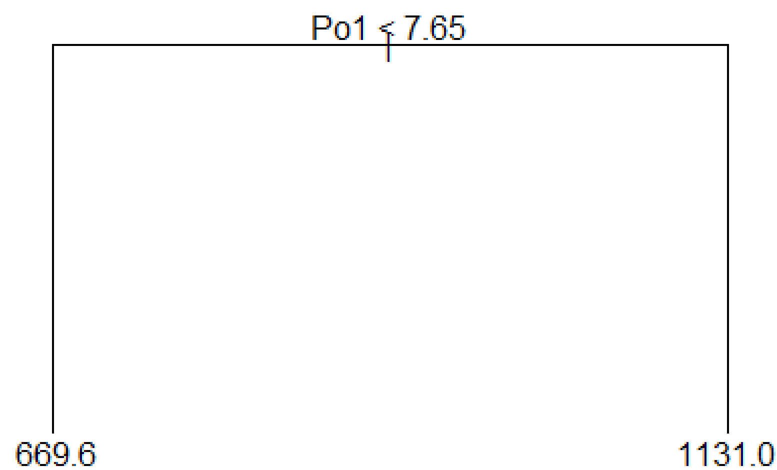
```
tree2 <- cv.tree(tree1)
plot(tree2$size, tree2$dev, type="b")
```



Then, I built a tree with 2 leaves only and performed regression on the data in those branches. The regression models had pretty high R-sq values for each branch. The second branch had only 1 significant factor which was not a indication of good model. Model on 1st branch is reasonable with 4 significant factors.

```
#pruned tree with 2 nodes only
set.seed(101)
tree3 <- prune.tree(tree1, best = 2)
summary(tree3)

##
## Regression tree:
## snip.tree(tree = tree1, nodes = 2:3)
## Variables actually used in tree construction:
## [1] "Po1"
## Number of terminal nodes:  2
## Residual mean deviance:  97410 = 4383000 / 45
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -622.800 -193.200   -5.609    0.000  147.300  862.200

tree3$where

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26
##  2  3  2  3  3  3  3  3  2  2  3  2  2  2  2  3  2  3  3  3  2  2  3  3  2
3
```

```
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
##  2  3  3  2  2  3  2  3  3  3  2  2  2  3  2  2  2  3  2  3  3

plot(tree3)
text(tree3)
```




```
#r-sq
rsquare_func(predict(tree3),crimedata$Crime)

## [1] 0.3629629

#separating data for lin regression
leaf1 <- crimedata[which(tree3$where == 2),]
leaf2 <- crimedata[which(tree3$where == 3),]

leaf1_reg <- lm(Crime~., data=leaf1)
summary(leaf1_reg)

##
## Call:
## lm(formula = Crime ~ ., data = leaf1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -109.147  -52.803   -6.495   53.784  127.196
##
## Coefficients:
##                  Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)   -48.5477  2044.9766  -0.024   0.9817  
## M              45.8622    58.6256   0.782   0.4597  
## So            380.4815   223.1072   1.705   0.1319  
## Ed            187.9074    89.5799   2.098   0.0741 .
## Po1            -3.5138   157.7513  -0.022   0.9829  
## Po2            44.6382   148.5528   0.300   0.7725  
## LF           1059.3652  1187.9722   0.892   0.4021  
## M.F           -22.5521    21.4677  -1.051   0.3284  
## Pop            10.6413     5.0929   2.089   0.0750 .
## NW              0.1010     7.9019   0.013   0.9902  
## U1           4878.2802  4874.8165   1.001   0.3503  
## U2             -5.5126   133.5094  -0.041   0.9682  
## Wealth         -0.1022     0.1752  -0.583   0.5779  
## Ineq            4.7779    35.5290   0.134   0.8968  
## Prob        -7317.4407  3280.7511  -2.230   0.0609 .
## Time          -20.0603     7.7287  -2.596   0.0357 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115.9 on 7 degrees of freedom
## Multiple R-squared:  0.8794, Adjusted R-squared:  0.6209 
## F-statistic: 3.403 on 15 and 7 DF,  p-value: 0.0541
```

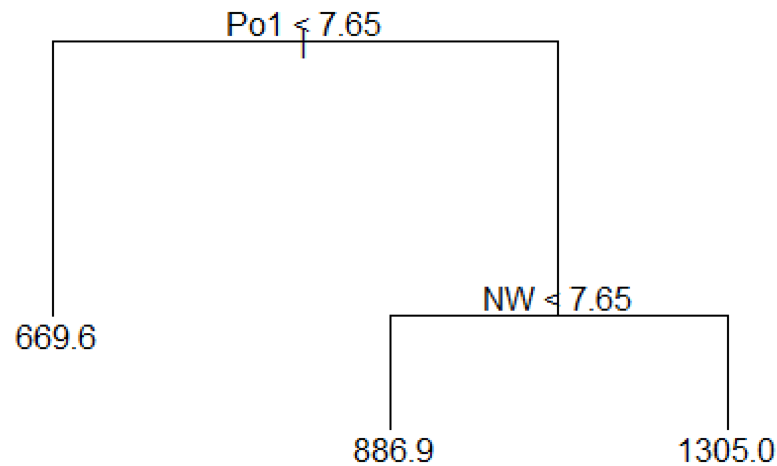
```
leaf2_reg <- lm(Crime~., data=leaf2)
summary(leaf2_reg)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = leaf2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -206.805 -120.407   -9.489  103.985  248.226 
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -8634.1701  2366.4043  -3.649  0.00651 **
## M               5.6032    96.1623   0.058  0.95496   
## So            179.6267   409.5210   0.439  0.67254   
## Ed            263.0845   146.4229   1.797  0.11010   
## Po1           235.2349   166.1289   1.416  0.19452   
## Po2          -140.7023   193.8759  -0.726  0.48869   
## LF           1442.4214  4832.4463   0.298  0.77294   
## M.F            -1.2379    54.8160  -0.023  0.98254   
## Pop            -3.7686     2.8833  -1.307  0.22751   
## NW             -0.5396    24.5039  -0.022  0.98297   
## U1          -3779.9843 10923.3434  -0.346  0.73823   
## U2            163.7106   150.5361   1.088  0.30848   
## Wealth          0.3017     0.2051   1.471  0.17946   
## Ineq          155.3754    65.5077   2.372  0.04511 * 
```

```
## Prob        -3624.0711  4381.4724  -0.827  0.43214
## Time           21.9335    14.6754   1.495  0.17338
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 229.9 on 8 degrees of freedom
## Multiple R-squared:  0.8827, Adjusted R-squared:  0.6626
## F-statistic: 4.012 on 15 and 8 DF,  p-value: 0.02669
```

Lastly, I repeated the process with 3 branches. This did not work since models for branch 2 and 3 had no siginificant factors. In a nutshell, regression tree had some success with regression on 2 branches and on pruned tree with 5 branches using all data. Due to the amount of data, there was overfitting in most of these models.

```
#pruned tree with 2 nodes only
set.seed(101)
tree4 <- prune.tree(tree1, best = 3)
summary(tree4)

##
## Regression tree:
## snip.tree(tree = tree1, nodes = c(6L, 2L, 7L))
## Variables actually used in tree construction:
## [1] "Po1" "NW"
## Number of terminal nodes:  3
## Residual mean deviance:  76460 = 3364000 / 44
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -550.9  -181.8   -37.9     0.0   158.9   688.1

tree4$where

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26
##  2  5  2  5  4  4  5  5  2  2  5  2  2  2  2  5  2  5  4  5  2  2  5  4  2
5
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
##  2  5  5  2  2  5  2  4  4  4  2  2  2  5  2  2  2  4  2  4  4

plot(tree4)
text(tree4)
```




```
#r-sq
rsquare_func(predict(tree4),crimedata$Crime)

## [1] 0.5111061

#separating data for lin regression
t4_leaf1 <- crimedata[which(tree4$where == 2),]
t4_leaf2 <- crimedata[which(tree4$where == 5),]
t4_leaf3 <- crimedata[which(tree4$where == 4),]

t4_leaf1_reg <- lm(Crime~., data=t4_leaf1)
summary(t4_leaf1_reg)

##
## Call:
## lm(formula = Crime ~ ., data = t4_leaf1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -109.147  -52.803   -6.495   53.784  127.196
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -48.5477  2044.9766  -0.024   0.9817
## M               45.8622    58.6256   0.782   0.4597
## So             380.4815   223.1072   1.705   0.1319
## Ed             187.9074    89.5799   2.098   0.0741 .
```

```
## Po1            -3.5138   157.7513  -0.022   0.9829
## Po2            44.6382   148.5528   0.300   0.7725
## LF           1059.3652  1187.9722   0.892   0.4021
## M.F           -22.5521    21.4677  -1.051   0.3284
## Pop            10.6413     5.0929   2.089   0.0750 .
## NW              0.1010     7.9019   0.013   0.9902
## U1           4878.2802  4874.8165   1.001   0.3503
## U2             -5.5126   133.5094  -0.041   0.9682
## Wealth         -0.1022     0.1752  -0.583   0.5779
## Ineq            4.7779    35.5290   0.134   0.8968
## Prob        -7317.4407  3280.7511  -2.230   0.0609 .
## Time          -20.0603     7.7287  -2.596   0.0357 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115.9 on 7 degrees of freedom
## Multiple R-squared:  0.8794, Adjusted R-squared:  0.6209
## F-statistic: 3.403 on 15 and 7 DF,  p-value: 0.0541
```

```
t4_leaf2_reg <- lm(Crime~., data=t4_leaf2)
summary(t4_leaf2_reg)
```

```
##
## Call:
## lm(formula = Crime ~ ., data = t4_leaf2)
##
## Residuals:
## ALL 14 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (2 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.381e+04         NA      NA       NA
## M            8.012e+01         NA      NA       NA
## So          -2.827e+02         NA      NA       NA
## Ed           2.663e+02         NA      NA       NA
## Po1         -2.943e+02         NA      NA       NA
## Po2          3.571e+02         NA      NA       NA
## LF          -1.648e+03         NA      NA       NA
## M.F          8.738e+01         NA      NA       NA
## Pop          1.155e+00         NA      NA       NA
## NW           8.841e+00         NA      NA       NA
## U1          -3.265e+04         NA      NA       NA
## U2           5.783e+02         NA      NA       NA
## Wealth       2.416e-01         NA      NA       NA
## Ineq         1.367e+02         NA      NA       NA
## Prob                NA         NA      NA       NA
## Time                NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
```

```
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 13 and 0 DF,  p-value: NA
```

```
t4_leaf3_reg <- lm(Crime~., data=t4_leaf3)
summary(t4_leaf3_reg)
```

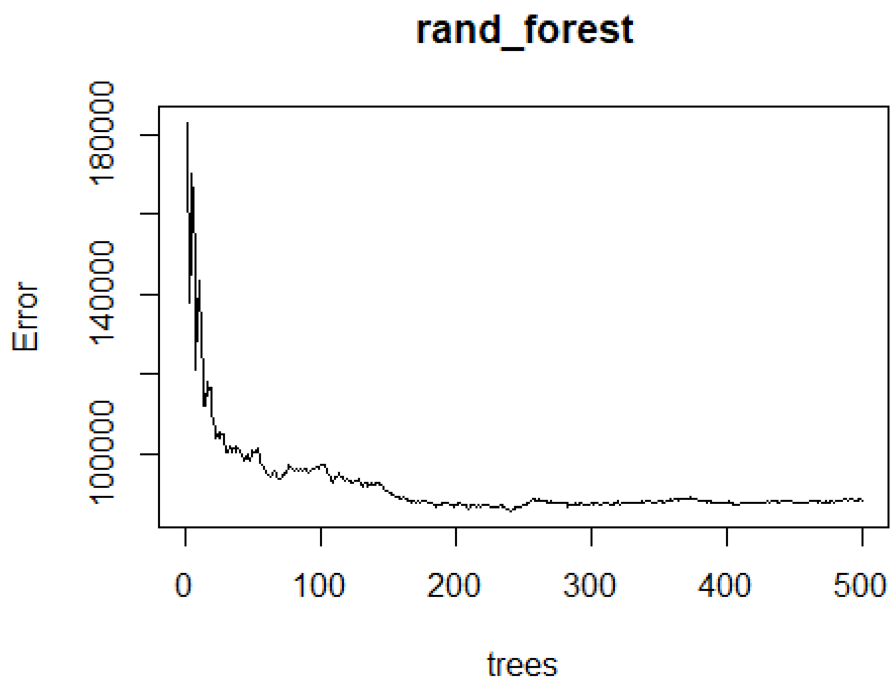
```
##
## Call:
## lm(formula = Crime ~ ., data = t4_leaf3)
##
## Residuals:
## ALL 10 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (6 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 32527.85         NA      NA       NA
## M             258.27         NA      NA       NA
## So                NA         NA      NA       NA
## Ed            -46.38         NA      NA       NA
## Po1         -1168.92         NA      NA       NA
## Po2           612.42         NA      NA       NA
## LF          16612.42         NA      NA       NA
## M.F          -384.45         NA      NA       NA
## Pop           -18.22         NA      NA       NA
## NW            124.13         NA      NA       NA
## U1           2064.68         NA      NA       NA
## U2                NA         NA      NA       NA
## Wealth            NA         NA      NA       NA
## Ineq              NA         NA      NA       NA
## Prob              NA         NA      NA       NA
## Time              NA         NA      NA       NA
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:    NaN
## F-statistic:   NaN on 9 and 0 DF,  p-value: NA
```

Random Forest

For random forest tree, I used the randomforest library. The function randomForest (ref: https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest) had a parameter for number of predictors to use in each tree. Give it was mentioned in the lectures that 1 + log (n) where n is the number of predictors is the ideal number to use, I stuck with that. Also, default number of trees in this function is 500, i stuck with that as well but tried a 1000 tree model too.

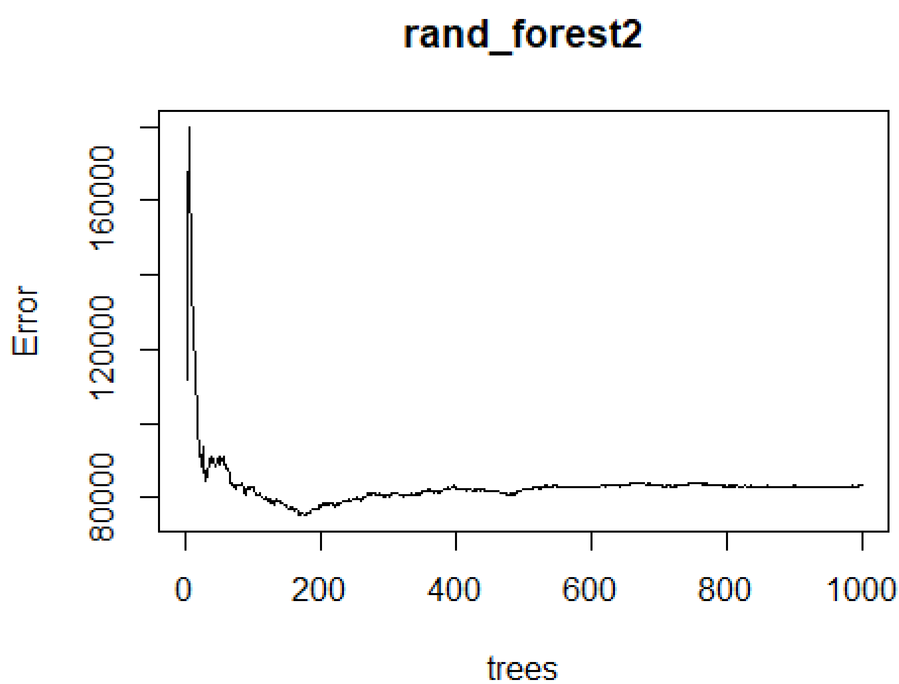
```
set.seed(101)

library(randomForest)

#number of predictors to use in each tree
```

```r
pred <- 1 + log(15)

#random forest model using 1+log(n) predictors and 500 trees
rand_forest <- randomForest(Crime~., data = crimedata, mtry = round(pred),
importance=TRUE)
rand_forest

##
## Call:
##  randomForest(formula = Crime ~ ., data = crimedata, mtry = round(pred),
importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 88447.44
##                     % Var explained: 39.59

rand_forest$importance

##           %IncMSE IncNodePurity
## M       2084.3065     211058.28
## So       501.9829      24155.77
## Ed      7133.9581     339679.06
## Po1    30492.8426    1180906.69
## Po2    23678.9590    1083612.33
## LF      3972.3000     318500.45
## M.F      769.2636     274437.07
## Pop      528.4682     463665.38
## NW     15252.1425     546203.49
## U1      -855.9384     158824.17
## U2       116.0106     175565.42
## Wealth  4228.0048     637614.03
## Ineq     959.8675     208316.32
## Prob   17278.4296     649865.56
## Time    1606.0544     211732.67

plot(rand_forest)
```

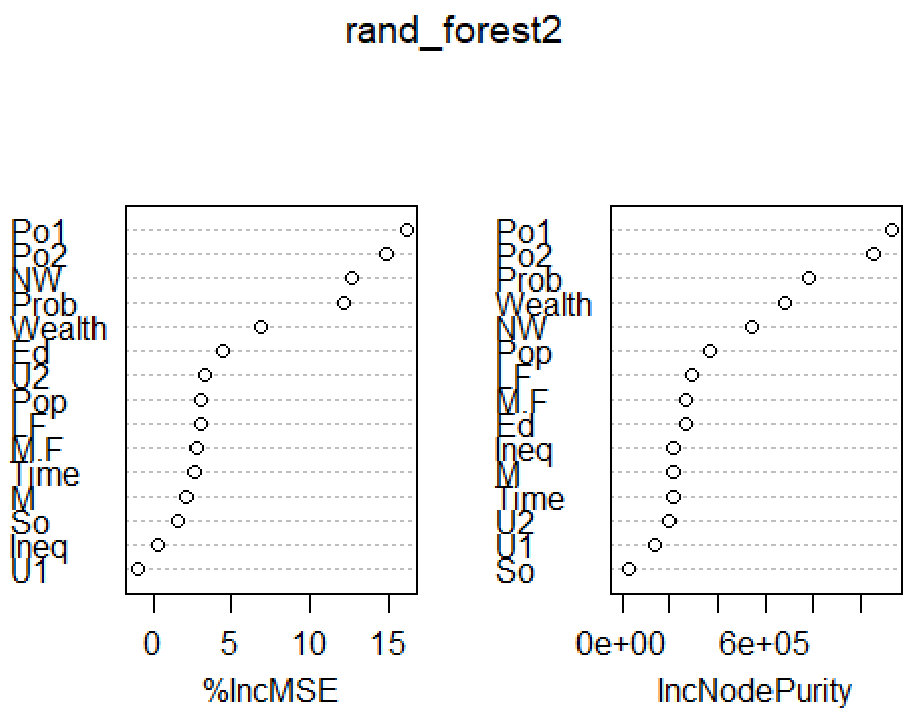



```
#r-sq of the model
rsquare_func(predict(rand_forest), crimedata$Crime)

## [1] 0.395862

#random forest model using 1+log(n) predictors and 1000 trees
rand_forest2 <- randomForest(Crime~., data = crimedata, mtry = round(pred),
ntree=1000, importance=TRUE)
rand_forest2

##
## Call:
##  randomForest(formula = Crime ~ ., data = crimedata, mtry = round(pred),
ntree = 1000, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 83207.26
##                     % Var explained: 43.17

rand_forest2$importance

##           %IncMSE IncNodePurity
## M       1480.6003      216756.8
## So       373.9914       27486.7
## Ed      3421.6233      263441.2
## Po1    31303.7823     1128142.7
```

```
## Po2    27574.1054     1049115.5
## LF      2566.9546      290776.5
## M.F     1698.3690      269441.2
## Pop     2586.4288      368187.2
## NW     17259.9042      545191.0
## U1      -632.5481      137844.7
## U2      2038.0173      195958.6
## Wealth  8710.3991      683853.8
## Ineq     219.5889      216791.9
## Prob   17314.9868      780722.5
## Time    1338.5286      214518.4

plot(rand_forest2)
```




```
#r-sq of the model
rsquare_func(predict(rand_forest2), crimedata$Crime)

## [1] 0.4316549

varImpPlot(rand_forest2)
```



The results of random forest tree seemed better than regression tree. The r-sq for both 500 and 1000 tree models was 40% and 43% which is low but also showed less over-fitting (professor had mentioned in the model quality lectures that 30-40% r-sq is more close to reality). The importance of predictors was shown in the Importance output of the model and higher values in both measures (%incMSE and incNodePurity) indicate how important that predictor was. Unsurprisingly, both graphs show that Po1 and po2 were most important with po1 being highest in the stack, which resemebles with the regression tree where 1st split was on po1.

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.

At my job at the T-Mobile HQ in the Seattle area, my team helps get analytics products built for our network supply chain team. This team manages the planning, procurement and logistics of getting the right equipment to the right locations so that T-Mobile's network could get built or improved. The equipment used to build or enhance the cellular network includes items like radios, antenna, cables, etc. This equipment is packaged together on pallets at the distribution center for each project according to the Bill of Material (BOM) of a project. This packaged material is called a kit. Kits are shipped from the distribution center to the market staging locations where the crew picks up the material to go complete the buid or enhancement on the celluar tower.

Often times, this kits is packaged incorrectly and is comprised. Which means either the quantities of equipment needed are incorrect or centain equipment is missing altogeter. This data is available in our data warehouse i.e. how many and which kits were compromised in the past. A model could be built using this data to predict whether any kits for unpcoming projets could be comprimised (higher or lower probability). Based on this prediction, a closer attention could be paid to those kits in the Distribution Center to minimize the overall quantity of compromised kits. Some useful predicotrs for this model wold be:

- Number of SKUs (equipments types) in a kit
- Total quantity of SKUs in a kit
- Number of pallets used for a kit
- Equipment size disparity. If a kit has some equipment that is too large in size (like an antenna) and too small (like small cables), chances are higher that smaller size equipment would get lost or would be forgotten.
- Average distance that has to be covered in the distribution center to pick the materials for a kit. High distance would increase chances of errors.

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.

Part 1 - Building Logistics Regression Model

This part required demonstrating,

- Building a model and showing what features were selected and how
- Showing the model output

- Quality of fit

First of all, I loaded the data.

```
#setting the seed so that results are the same at every run
set.seed(101)

#loading data
germandata <- read.table("data_10.3/germancredit.txt", header = F)

#quick glance at the data
head(germandata)

##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   2
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   2
## 6 A192 A201   1
```

The response column V21 has 1 and 2 indicating bad and good.I converted that to 0 and 1 for use in the model. I splitted the data into training (80%) and testing (20%).

```
#replacing response values to 0 and 1 because 2 is bad and 1 is good.
germandata$V21[germandata$V21 == 2] <- 0
head(germandata)

##    V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 1 A11  6 A34 A43 1169 A65 A75  4 A93 A101   4 A121  67 A143 A152   2 A173   1
## 2 A12 48 A32 A43 5951 A61 A73  2 A92 A101   2 A121  22 A143 A152   1 A173   1
## 3 A14 12 A34 A46 2096 A61 A74  2 A93 A101   3 A121  49 A143 A152   1 A172   2
## 4 A11 42 A32 A42 7882 A61 A74  2 A93 A103   4 A122  45 A143 A153   1 A173   2
## 5 A11 24 A33 A40 4870 A61 A73  3 A93 A101   4 A124  53 A143 A153   2 A173   2
## 6 A14 36 A32 A46 9055 A65 A73  2 A93 A101   4 A124  35 A143 A153   1 A172   2
##    V19  V20 V21
## 1 A192 A201   1
## 2 A191 A201   0
## 3 A191 A201   1
## 4 A191 A201   1
## 5 A191 A201   0
## 6 A192 A201   1

#splitting data to train & test

nrow(germandata)
```

```
## [1] 1000

set.seed(101) #this ensures that same datasets are reproduced in the future.
sample <- sample.int(n = nrow(germandata), size =
floor(.80*nrow(germandata)), replace = F)
germandata_train <- germandata[sample,]
germandata_test <- germandata[-sample,]

nrow(germandata_train)

## [1] 800

nrow(germandata_test)

## [1] 200

head(germandata_train)

##      V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 841 A11 36 A32 A42 5179 A61 A74  4 A93 A101   2 A122  29 A143 A152   1 A173   1
## 825 A14 18 A34 A42 3780 A61 A72  3 A91 A101   2 A123  35 A143 A152   2 A174   1
## 430 A11 18 A34 A45 1190 A61 A71  2 A92 A101   4 A124  55 A143 A153   3 A171   2
## 95  A12 12 A32 A40 1318 A64 A75  4 A93 A101   4 A121  54 A143 A152   1 A173   1
## 209 A11 24 A32 A49 6568 A61 A73  2 A94 A101   2 A123  21 A142 A152   1 A172   1
## 442 A11 12 A32 A42 1620 A61 A73  2 A92 A102   3 A122  30 A143 A152   1 A173   1
##      V19  V20 V21
## 841 A191 A201   0
## 825 A192 A201   1
## 430 A191 A201   0
## 95  A192 A201   1
## 209 A191 A201   1
## 442 A191 A201   1

head(germandata_test)

##     V1 V2  V3  V4   V5  V6  V7 V8  V9  V10 V11  V12 V13  V14  V15 V16  V17 V18
## 8  A12 36 A32 A41 6948 A61 A73  2 A93 A101   2 A123  35 A143 A151   1 A174   1
## 10 A12 30 A34 A40 5234 A61 A71  4 A94 A101   2 A123  28 A143 A152   2 A174   1
## 12 A11 48 A32 A49 4308 A61 A72  3 A92 A101   4 A122  24 A143 A151   1 A173   1
## 16 A11 24 A32 A43 1282 A62 A73  4 A92 A101   2 A123  32 A143 A152   1 A172   1
## 29 A12  7 A32 A43 2415 A61 A73  3 A93 A103   2 A121  34 A143 A152   1 A173   1
## 32 A11 24 A32 A42 4020 A61 A73  2 A93 A101   2 A123  27 A142 A152   1 A173   1
##     V19  V20 V21
## 8  A192 A201   1
## 10 A191 A201   0
## 12 A191 A201   0
## 16 A191 A201   0
## 29 A191 A201   1
## 32 A191 A201   1
```

I was now ready to build the first Logistic Regression model. I used all features to see what the outcome was.

The output of the model shows all the featues the model considered along with respective P-values indicating whether a feature was significant or not (significant if P-value was less than 0.05). Also worth noting, that each categorical value of a feature is considered separately (as a 0, 1 response for that value). 14 features were found to be significant.

```
logit_full <- glm(V21~., family = binomial(link = "logit"), data =
germandata_train)
summary(logit_full)

##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data =
germandata_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5356  -0.5668   0.3479   0.6514   2.4077
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.392e-01  1.245e+00  -0.192 0.847601
## V1A12        3.832e-01  2.575e-01   1.488 0.136687
## V1A13        8.974e-01  4.651e-01   1.929 0.053686 .
## V1A14        1.623e+00  2.622e-01   6.191 5.98e-10 ***
## V2          -4.245e-02  1.110e-02  -3.825 0.000131 ***
## V3A31        1.654e-01  6.490e-01   0.255 0.798794
## V3A32        7.187e-01  5.083e-01   1.414 0.157431
## V3A33        1.262e+00  5.548e-01   2.274 0.022941 *
## V3A34        1.874e+00  5.227e-01   3.585 0.000337 ***
## V4A41        1.658e+00  4.358e-01   3.805 0.000142 ***
## V4A410       1.715e+00  8.353e-01   2.053 0.040046 *
## V4A42        9.437e-01  3.063e-01   3.081 0.002060 **
## V4A43        1.017e+00  2.910e-01   3.493 0.000478 ***
## V4A44        5.671e-01  7.782e-01   0.729 0.466197
## V4A45        3.368e-01  6.131e-01   0.549 0.582749
## V4A46       -2.504e-01  4.372e-01  -0.573 0.566761
## V4A48        1.701e+00  1.233e+00   1.380 0.167502
## V4A49        1.117e+00  4.010e-01   2.787 0.005323 **
## V5          -1.222e-04  5.152e-05  -2.372 0.017679 *
## V6A62        2.744e-01  3.226e-01   0.851 0.394966
## V6A63       -1.843e-01  4.340e-01  -0.425 0.671154
## V6A64        1.173e+00  6.248e-01   1.878 0.060411 .
## V6A65        9.169e-01  3.016e-01   3.040 0.002366 **
## V7A72        2.263e-02  5.146e-01   0.044 0.964921
## V7A73        3.145e-01  4.872e-01   0.646 0.518596
## V7A74        9.861e-01  5.276e-01   1.869 0.061631 .
## V7A75        4.695e-01  4.922e-01   0.954 0.340107
## V8          -3.423e-01  1.030e-01  -3.322 0.000892 ***
## V9A92        2.575e-01  4.695e-01   0.548 0.583394
## V9A93        8.931e-01  4.644e-01   1.923 0.054478 .
```

```
## V9A94        6.506e-01  5.598e-01   1.162 0.245181
## V10A102     -4.590e-01  4.683e-01  -0.980 0.326949
## V10A103      6.301e-01  4.527e-01   1.392 0.163960
## V11          4.850e-02  1.010e-01   0.480 0.630997
## V12A122     -3.154e-01  2.913e-01  -1.082 0.279071
## V12A123     -2.972e-01  2.729e-01  -1.089 0.276145
## V12A124     -6.402e-01  4.970e-01  -1.288 0.197645
## V13          1.491e-02  1.053e-02   1.416 0.156805
## V14A142      1.740e-01  4.774e-01   0.364 0.715596
## V14A143      7.658e-01  2.684e-01   2.853 0.004331 **
## V15A152      4.095e-01  2.676e-01   1.530 0.126004
## V15A153      6.199e-01  5.575e-01   1.112 0.266151
## V16         -4.804e-01  2.277e-01  -2.110 0.034863 *
## V17A172     -7.000e-01  7.901e-01  -0.886 0.375678
## V17A173     -6.351e-01  7.627e-01  -0.833 0.405016
## V17A174     -5.370e-01  7.751e-01  -0.693 0.488398
## V18         -2.198e-01  2.891e-01  -0.760 0.446998
## V19A192      1.528e-01  2.326e-01   0.657 0.511312
## V20A202      1.077e+00  7.690e-01   1.401 0.161353
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 963.44  on 799  degrees of freedom
## Residual deviance: 683.78  on 751  degrees of freedom
## AIC: 781.78
##
## Number of Fisher Scoring iterations: 5
```

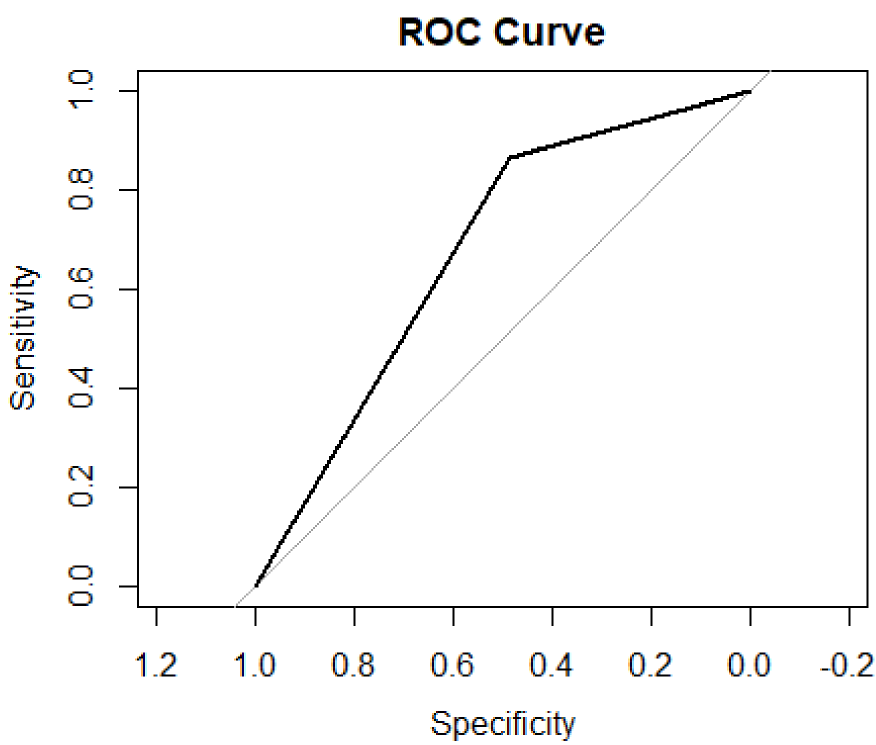
I calculated the accuracy along with other quality measures of the model like ROC curve (area was 0.6745). I used a threshold of 0.5 (right in the middle) for this exercise. I will find a better threshold based on the cost next.

```r
library(caret)
library(pROC)
predictions <- predict(logit_full,newdata = germandata_test, type =
"response")
predictions_01 <- as.integer(predictions > 0.5)
confusionMatrix(factor(germandata_test$V21, levels =
c(1,0)),factor(predictions_01,levels = c(1,0)))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   1   0
##          1 114  18
##          0  35  33
##
##                Accuracy : 0.735
```

```
##                  95% CI : (0.6681, 0.7948)
##     No Information Rate : 0.745
##     P-Value [Acc > NIR] : 0.66148
##
##                   Kappa : 0.3714
##
##  Mcnemar's Test P-Value : 0.02797
##
##             Sensitivity : 0.7651
##             Specificity : 0.6471
##          Pos Pred Value : 0.8636
##          Neg Pred Value : 0.4853
##              Prevalence : 0.7450
##          Detection Rate : 0.5700
##    Detection Prevalence : 0.6600
##       Balanced Accuracy : 0.7061
##
##        'Positive' Class : 1
##
```

```
AUC <- roc(germandata_test$V21,predictions_01)
plot(AUC, main = "ROC Curve")
```




```
AUC
```

```
##
## Call:
```

```
## roc.default(response = germandata_test$V21, predictor = predictions_01)
##
## Data: predictions_01 in 68 controls (germandata_test$V21 0) < 132 cases
(germandata_test$V21 1).
## Area under the curve: 0.6745
```

Before I looked at the cost and better threshold, I wanted to make sure the model is good. I used stepwise method to select the features.

```
stepwise <- step(logit_full)

## Start:  AIC=781.78
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14 + V15 + V16 + V17 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V17   3   684.60 776.60
## - V12   3   686.10 778.10
## - V11   1   684.01 780.01
## - V19   1   684.21 780.21
## - V18   1   684.35 780.35
## - V15   2   686.50 780.50
## - V10   2   687.01 781.01
## <none>      683.78 781.78
## - V13   1   685.81 781.81
## - V20   1   686.12 782.12
## - V7    4   692.72 782.72
## - V16   1   688.29 784.29
## - V9    3   692.66 784.66
## - V5    1   689.47 785.47
## - V14   2   692.51 786.51
## - V6    4   697.62 787.62
## - V8    1   695.23 791.23
## - V2    1   698.92 794.92
## - V4    9   715.91 795.91
## - V3    4   708.54 798.54
## - V1    3   728.99 820.99
##
## Step:  AIC=776.6
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V12 + V13 + V14 + V15 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V12   3   686.79 772.79
## - V11   1   684.75 774.75
## - V15   2   687.18 775.18
## - V18   1   685.20 775.20
## - V19   1   685.22 775.22
## - V10   2   687.70 775.70
## - V13   1   686.60 776.60
```

```
## <none>       684.60 776.60
## - V20   1   686.91 776.91
## - V7    4   693.12 777.12
## - V16   1   688.76 778.76
## - V9    3   693.30 779.30
## - V5    1   690.41 780.41
## - V14   2   693.46 781.46
## - V6    4   699.22 783.22
## - V8    1   696.21 786.21
## - V2    1   699.87 789.87
## - V4    9   716.60 790.60
## - V3    4   709.19 793.19
## - V1    3   729.34 815.34
##
## Step:  AIC=772.79
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V11 +
##     V13 + V14 + V15 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V11   1   686.94 770.94
## - V15   2   689.03 771.03
## - V19   1   687.16 771.16
## - V18   1   687.29 771.29
## - V10   2   690.60 772.60
## <none>       686.79 772.79
## - V13   1   688.83 772.83
## - V20   1   689.05 773.05
## - V7    4   695.68 773.68
## - V16   1   691.03 775.03
## - V9    3   695.16 775.16
## - V5    1   693.15 777.15
## - V14   2   696.13 778.13
## - V6    4   701.57 779.57
## - V8    1   699.11 783.11
## - V2    1   703.61 787.61
## - V4    9   720.30 788.30
## - V3    4   711.90 789.90
## - V1    3   732.53 812.53
##
## Step:  AIC=770.94
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 +
##     V14 + V15 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V15   2   689.03 769.03
## - V19   1   687.35 769.35
## - V18   1   687.42 769.42
## - V10   2   690.74 770.74
## <none>       686.94 770.94
## - V20   1   689.14 771.14
```

```
## - V13   1   689.18 771.18
## - V7    4   696.24 772.24
## - V16   1   691.06 773.06
## - V9    3   695.24 773.24
## - V5    1   693.46 775.46
## - V14   2   696.57 776.57
## - V6    4   701.82 777.82
## - V8    1   699.22 781.22
## - V2    1   703.65 785.65
## - V4    9   720.54 786.54
## - V3    4   712.00 788.00
## - V1    3   732.53 810.53
##
## Step:  AIC=769.03
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 +
##     V14 + V16 + V18 + V19 + V20
##
##        Df Deviance    AIC
## - V19   1   689.48 767.48
## - V18   1   689.51 767.51
## - V10   2   692.67 768.67
## <none>      689.03 769.03
## - V20   1   691.10 769.10
## - V13   1   691.80 769.80
## - V7    4   698.42 770.42
## - V16   1   693.28 771.28
## - V9    3   698.91 772.91
## - V5    1   695.72 773.72
## - V14   2   698.31 774.31
## - V6    4   703.82 775.82
## - V8    1   701.33 779.33
## - V2    1   706.05 784.05
## - V4    9   723.20 785.20
## - V3    4   715.06 787.06
## - V1    3   736.15 810.15
##
## Step:  AIC=767.48
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 +
##     V14 + V16 + V18 + V20
##
##        Df Deviance    AIC
## - V18   1   689.99 765.99
## - V10   2   693.11 767.11
## - V20   1   691.43 767.43
## <none>      689.48 767.48
## - V13   1   692.47 768.47
## - V7    4   699.22 769.22
## - V16   1   693.58 769.58
## - V9    3   699.55 771.55
## - V5    1   695.75 771.75
```

```
## - V14   2   698.75 772.75
## - V6    4   704.40 774.40
## - V8    1   701.65 777.65
## - V2    1   706.84 782.84
## - V4    9   723.87 783.87
## - V3    4   715.57 785.57
## - V1    3   737.33 809.33
##
## Step:  AIC=765.99
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V10 + V13 +
##     V14 + V16 + V20
##
##        Df Deviance    AIC
## - V10   2   693.51 765.51
## - V20   1   691.90 765.90
## <none>      689.99 765.99
## - V13   1   692.79 766.79
## - V7    4   699.75 767.75
## - V16   1   694.34 768.34
## - V9    3   699.57 769.57
## - V5    1   696.11 770.11
## - V14   2   699.17 771.17
## - V6    4   704.87 772.87
## - V8    1   701.78 775.78
## - V2    1   707.24 781.24
## - V4    9   724.80 782.80
## - V3    4   716.67 784.67
## - V1    3   738.27 808.27
##
## Step:  AIC=765.51
## V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 + V13 + V14 +
##     V16 + V20
##
##        Df Deviance    AIC
## <none>      693.51 765.51
## - V20   1   695.82 765.82
## - V13   1   696.35 766.35
## - V16   1   697.88 767.88
## - V7    4   703.90 767.90
## - V9    3   703.23 769.23
## - V14   2   702.61 770.61
## - V5    1   700.70 770.70
## - V6    4   708.04 772.04
## - V8    1   705.99 775.99
## - V2    1   710.28 780.28
## - V4    9   729.73 783.73
## - V3    4   720.04 784.04
## - V1    3   740.66 806.66

stepwise
```

```
##
## Call:  glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
##     V13 + V14 + V16 + V20, family = binomial(link = "logit"),
##     data = germandata_train)
##
## Coefficients:
## (Intercept)        V1A12        V1A13        V1A14           V2        V3A31
##  -0.6704949    0.4358373    0.8930260    1.6427744   -0.0428229    0.1789102
##       V3A32        V3A33        V3A34        V4A41       V4A410        V4A42
##   0.7435816    1.2083921    1.9161344    1.6410814    1.7640438    0.8733331
##       V4A43        V4A44        V4A45        V4A46        V4A48        V4A49
##   1.0923438    0.6256206    0.4100021   -0.3670424    1.5906901    1.1542921
##          V5        V6A62        V6A63        V6A64        V6A65        V7A72
##  -0.0001277    0.1287677   -0.2012722    1.0670816    0.9364022   -0.2243264
##       V7A73        V7A74        V7A75           V8        V9A92        V9A93
##   0.1287617    0.8273522    0.2638559   -0.3460853    0.2114054    0.8320429
##       V9A94          V13      V14A142      V14A143          V16       V20A202
##   0.6445083    0.0163685    0.2554991    0.7784872   -0.4584550    1.0475803
##
## Degrees of Freedom: 799 Total (i.e. Null);  764 Residual
## Null Deviance:       963.4
## Residual Deviance: 693.5     AIC: 765.5
```

The final model wasbuilt below with accuracy and AUC calculations. It showed that the accuracy had droppped a bit for this model compared to the one where we used all features. But the drop was very insignificant and the true positive and true negative were close. False positives (high cost) were actually the same. Thus, we could go ahead with this model.

```
#final model
logit_selected <- glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 +
V9 + V13 + V14 + V16 + V20,
                      family = binomial(link = "logit"),
                      data = germandata_train)

summary(logit_selected)

##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V8 + V9 +
##     V13 + V14 + V16 + V20, family = binomial(link = "logit"),
##     data = germandata_train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6475  -0.6321   0.3634   0.6766   2.2186
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.705e-01  1.029e+00  -0.652 0.514647
## V1A12        4.358e-01  2.508e-01   1.737 0.082304 .
```

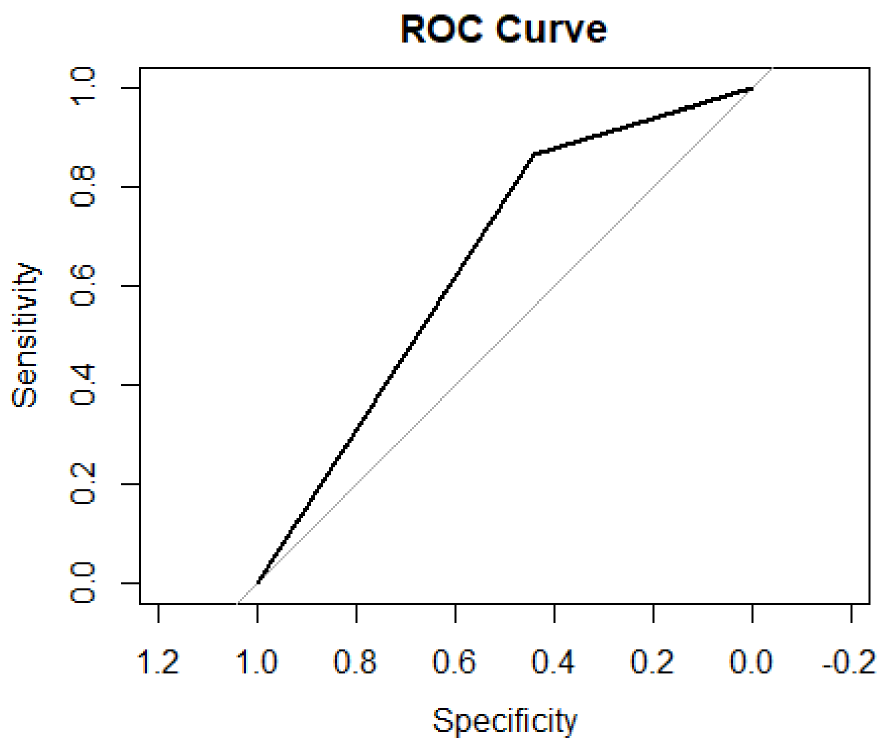
```
## V1A13        8.930e-01  4.597e-01   1.942 0.052085 .  
## V1A14        1.643e+00  2.582e-01   6.362 1.99e-10 ***
## V2          -4.282e-02  1.063e-02  -4.028 5.63e-05 ***
## V3A31        1.789e-01  6.360e-01   0.281 0.778492    
## V3A32        7.436e-01  5.039e-01   1.476 0.140061    
## V3A33        1.208e+00  5.492e-01   2.200 0.027798 *  
## V3A34        1.916e+00  5.190e-01   3.692 0.000222 ***
## V4A41        1.641e+00  4.224e-01   3.885 0.000102 ***
## V4A410       1.764e+00  7.915e-01   2.229 0.025833 *  
## V4A42        8.733e-01  2.945e-01   2.966 0.003018 ** 
## V4A43        1.092e+00  2.840e-01   3.847 0.000120 ***
## V4A44        6.256e-01  7.724e-01   0.810 0.417984    
## V4A45        4.100e-01  5.870e-01   0.698 0.484868    
## V4A46       -3.670e-01  4.271e-01  -0.859 0.390112    
## V4A48        1.591e+00  1.212e+00   1.312 0.189494    
## V4A49        1.154e+00  3.904e-01   2.957 0.003109 ** 
## V5          -1.277e-04  4.809e-05  -2.656 0.007916 ** 
## V6A62        1.288e-01  3.120e-01   0.413 0.679850    
## V6A63       -2.013e-01  4.338e-01  -0.464 0.642670    
## V6A64        1.067e+00  5.900e-01   1.809 0.070524 .  
## V6A65        9.364e-01  2.960e-01   3.164 0.001557 ** 
## V7A72       -2.243e-01  4.488e-01  -0.500 0.617213    
## V7A73        1.288e-01  4.124e-01   0.312 0.754882    
## V7A74        8.274e-01  4.595e-01   1.801 0.071745 .  
## V7A75        2.639e-01  4.251e-01   0.621 0.534848    
## V8          -3.461e-01  9.993e-02  -3.463 0.000534 ***
## V9A92        2.114e-01  4.552e-01   0.464 0.642352    
## V9A93        8.320e-01  4.456e-01   1.867 0.061889 .  
## V9A94        6.445e-01  5.440e-01   1.185 0.236078    
## V13          1.637e-02  9.803e-03   1.670 0.094962 .  
## V14A142      2.555e-01  4.755e-01   0.537 0.591029    
## V14A143      7.785e-01  2.632e-01   2.957 0.003104 ** 
## V16         -4.585e-01  2.208e-01  -2.077 0.037835 *  
## V20A202      1.048e+00  7.580e-01   1.382 0.166983    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 963.44  on 799  degrees of freedom
## Residual deviance: 693.51  on 764  degrees of freedom
## AIC: 765.51
## 
## Number of Fisher Scoring iterations: 5
```

```
#model fit and accuracy
steppredict <- predict(logit_selected,newdata = germandata_test, type = "response")
steppredict_01 <- as.integer(steppredict > 0.5)
```

```r
confusionMatrix(factor(germandata_test$V21, levels =
c(1,0)),factor(steppredict_01,levels = c(1,0)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   0
##          1 114  18
##          0  38  30
##
##                Accuracy : 0.72
##                  95% CI : (0.6523, 0.781)
##     No Information Rate : 0.76
##     P-Value [Acc > NIR] : 0.91847
##
##                   Kappa : 0.3282
##
##  Mcnemar's Test P-Value : 0.01112
##
##             Sensitivity : 0.7500
##             Specificity : 0.6250
##          Pos Pred Value : 0.8636
##          Neg Pred Value : 0.4412
##              Prevalence : 0.7600
##          Detection Rate : 0.5700
##    Detection Prevalence : 0.6600
##       Balanced Accuracy : 0.6875
##
##        'Positive' Class : 1
##
```

```r
AUC <- roc(germandata_test$V21,steppredict_01)
plot(AUC, main = "ROC Curve")
```




```
AUC

##
## Call:
## roc.default(response = germandata_test$V21, predictor = steppredict_01)
##
## Data: steppredict_01 in 68 controls (germandata_test$V21 0) < 132 cases
(germandata_test$V21 1).
## Area under the curve: 0.6524
```

Part 2 - Finding the right threshold

I used a loop from 0.01 to 1 as thresholds to see which threshold had the lowest cost and lowest false positive ratio (false positive / false positive + true negative). Once I printed the table with the results, it was clear that I had to discard the first 7 reading false positive ratio was either NaN or 1. Looking for the lowest cost in the remaining data showed it to be 76 and lowest false positive ratio was 0.25. Unsurprisingly the cost for 0.25 ratio was also 76, thus the 0.28 threshold seemed the right threshold.

```
results <- matrix(NA, nrow=100, ncol=4)
colnames(results) <- c("threshold","Cost", "False Positive Ratio",
"Accuracy")

#looping on the model and documenting accuracy percentage
for(i in 1:100){
    threshold <- i/100
```

```
    steppredict <- predict(logit_selected,newdata = germandata_test, type =
"response")
    steppredict <- as.integer(steppredict > threshold)
    matrix <- confusionMatrix(factor(germandata_test$V21, levels =
c(1,0)),factor(steppredict,levels = c(1,0)))
    cost <- matrix$table[2,1] + 5*matrix$table[1,2]
    ratio <- matrix$table[1,2] / (matrix$table[1,2] + matrix$table[2,2])
    results[i,] <- c(threshold, cost, round(ratio, digits = 2),
round(matrix$overall[1],digits = 2))
}

results

##         threshold Cost False Positive Ratio Accuracy
##    [1,]      0.01   68                  NaN     0.66
##    [2,]      0.02   68                  NaN     0.66
##    [3,]      0.03   73                 1.00     0.66
##    [4,]      0.04   73                 1.00     0.66
##    [5,]      0.05   73                 1.00     0.66
##    [6,]      0.06   78                 1.00     0.65
##    [7,]      0.07   78                 1.00     0.65
##    [8,]      0.08   78                 1.00     0.65
##    [9,]      0.09   77                 0.67     0.66
##   [10,]      0.10   77                 0.67     0.66
##   [11,]      0.11   77                 0.67     0.66
##   [12,]      0.12   77                 0.67     0.66
##   [13,]      0.13   76                 0.50     0.66
##   [14,]      0.14   81                 0.60     0.66
##   [15,]      0.15   86                 0.67     0.65
##   [16,]      0.16   85                 0.57     0.66
##   [17,]      0.17   84                 0.50     0.66
##   [18,]      0.18   83                 0.44     0.66
##   [19,]      0.19   83                 0.44     0.66
##   [20,]      0.20   82                 0.40     0.67
##   [21,]      0.21   81                 0.36     0.68
##   [22,]      0.22   81                 0.36     0.68
##   [23,]      0.23   81                 0.36     0.68
##   [24,]      0.24   81                 0.36     0.68
##   [25,]      0.25   80                 0.33     0.68
##   [26,]      0.26   80                 0.33     0.68
##   [27,]      0.27   79                 0.31     0.68
##   [28,]      0.28   76                 0.25     0.70
##   [29,]      0.29   91                 0.37     0.68
##   [30,]      0.30   90                 0.35     0.69
##   [31,]      0.31   90                 0.35     0.69
##   [32,]      0.32  100                 0.41     0.68
##   [33,]      0.33  105                 0.43     0.68
##   [34,]      0.34  103                 0.40     0.68
##   [35,]      0.35  101                 0.37     0.70
##   [36,]      0.36  101                 0.37     0.70
```

```
## [37,]     0.37  100              0.36    0.70
## [38,]     0.38   99              0.34    0.70
## [39,]     0.39   98              0.33    0.71
## [40,]     0.40   98              0.33    0.71
## [41,]     0.41  106              0.35    0.71
## [42,]     0.42  109              0.35    0.72
## [43,]     0.43  114              0.37    0.71
## [44,]     0.44  113              0.36    0.72
## [45,]     0.45  112              0.35    0.72
## [46,]     0.46  120              0.36    0.72
## [47,]     0.47  120              0.36    0.72
## [48,]     0.48  128              0.38    0.72
## [49,]     0.49  128              0.38    0.72
## [50,]     0.50  128              0.38    0.72
## [51,]     0.51  133              0.39    0.72
## [52,]     0.52  143              0.41    0.70
## [53,]     0.53  142              0.40    0.71
## [54,]     0.54  140              0.39    0.72
## [55,]     0.55  139              0.38    0.72
## [56,]     0.56  148              0.40    0.72
## [57,]     0.57  147              0.39    0.72
## [58,]     0.58  155              0.40    0.72
## [59,]     0.59  157              0.39    0.74
## [60,]     0.60  156              0.38    0.74
## [61,]     0.61  156              0.38    0.74
## [62,]     0.62  171              0.41    0.72
## [63,]     0.63  176              0.42    0.72
## [64,]     0.64  191              0.44    0.70
## [65,]     0.65  206              0.46    0.69
## [66,]     0.66  204              0.45    0.70
## [67,]     0.67  219              0.47    0.68
## [68,]     0.68  222              0.47    0.69
## [69,]     0.69  231              0.47    0.68
## [70,]     0.70  239              0.47    0.68
## [71,]     0.71  254              0.49    0.67
## [72,]     0.72  263              0.49    0.66
## [73,]     0.73  268              0.50    0.66
## [74,]     0.74  287              0.51    0.64
## [75,]     0.75  292              0.52    0.64
## [76,]     0.76  307              0.53    0.62
## [77,]     0.77  311              0.53    0.62
## [78,]     0.78  321              0.54    0.62
## [79,]     0.79  330              0.54    0.61
## [80,]     0.80  344              0.55    0.60
## [81,]     0.81  348              0.55    0.60
## [82,]     0.82  358              0.56    0.59
## [83,]     0.83  378              0.57    0.57
## [84,]     0.84  392              0.58    0.56
## [85,]     0.85  401              0.58    0.56
## [86,]     0.86  416              0.59    0.54
```

```
##  [87,]      0.87  417                     0.57     0.56
##  [88,]      0.88  426                     0.58     0.55
##  [89,]      0.89  436                     0.58     0.54
##  [90,]      0.90  444                     0.58     0.54
##  [91,]      0.91  452                     0.58     0.54
##  [92,]      0.92  462                     0.58     0.53
##  [93,]      0.93  471                     0.58     0.52
##  [94,]      0.94  501                     0.60     0.50
##  [95,]      0.95  541                     0.62     0.46
##  [96,]      0.96  561                     0.63     0.44
##  [97,]      0.97  581                     0.63     0.42
##  [98,]      0.98  610                     0.64     0.39
##  [99,]      0.99  640                     0.65     0.36
## [100,]      1.00  660                     0.66     0.34
```

```r
#right threshold by eliminating first rows
results_new <- results[8:100,]
results_new
```

```
##       threshold Cost False Positive Ratio Accuracy
##  [1,]      0.08   78                 1.00     0.65
##  [2,]      0.09   77                 0.67     0.66
##  [3,]      0.10   77                 0.67     0.66
##  [4,]      0.11   77                 0.67     0.66
##  [5,]      0.12   77                 0.67     0.66
##  [6,]      0.13   76                 0.50     0.66
##  [7,]      0.14   81                 0.60     0.66
##  [8,]      0.15   86                 0.67     0.65
##  [9,]      0.16   85                 0.57     0.66
## [10,]      0.17   84                 0.50     0.66
## [11,]      0.18   83                 0.44     0.66
## [12,]      0.19   83                 0.44     0.66
## [13,]      0.20   82                 0.40     0.67
## [14,]      0.21   81                 0.36     0.68
## [15,]      0.22   81                 0.36     0.68
## [16,]      0.23   81                 0.36     0.68
## [17,]      0.24   81                 0.36     0.68
## [18,]      0.25   80                 0.33     0.68
## [19,]      0.26   80                 0.33     0.68
## [20,]      0.27   79                 0.31     0.68
## [21,]      0.28   76                 0.25     0.70
## [22,]      0.29   91                 0.37     0.68
## [23,]      0.30   90                 0.35     0.69
## [24,]      0.31   90                 0.35     0.69
## [25,]      0.32  100                 0.41     0.68
## [26,]      0.33  105                 0.43     0.68
## [27,]      0.34  103                 0.40     0.68
## [28,]      0.35  101                 0.37     0.70
## [29,]      0.36  101                 0.37     0.70
## [30,]      0.37  100                 0.36     0.70
```

```
## [31,]       0.38   99                          0.34     0.70
## [32,]       0.39   98                          0.33     0.71
## [33,]       0.40   98                          0.33     0.71
## [34,]       0.41  106                          0.35     0.71
## [35,]       0.42  109                          0.35     0.72
## [36,]       0.43  114                          0.37     0.71
## [37,]       0.44  113                          0.36     0.72
## [38,]       0.45  112                          0.35     0.72
## [39,]       0.46  120                          0.36     0.72
## [40,]       0.47  120                          0.36     0.72
## [41,]       0.48  128                          0.38     0.72
## [42,]       0.49  128                          0.38     0.72
## [43,]       0.50  128                          0.38     0.72
## [44,]       0.51  133                          0.39     0.72
## [45,]       0.52  143                          0.41     0.70
## [46,]       0.53  142                          0.40     0.71
## [47,]       0.54  140                          0.39     0.72
## [48,]       0.55  139                          0.38     0.72
## [49,]       0.56  148                          0.40     0.72
## [50,]       0.57  147                          0.39     0.72
## [51,]       0.58  155                          0.40     0.72
## [52,]       0.59  157                          0.39     0.74
## [53,]       0.60  156                          0.38     0.74
## [54,]       0.61  156                          0.38     0.74
## [55,]       0.62  171                          0.41     0.72
## [56,]       0.63  176                          0.42     0.72
## [57,]       0.64  191                          0.44     0.70
## [58,]       0.65  206                          0.46     0.69
## [59,]       0.66  204                          0.45     0.70
## [60,]       0.67  219                          0.47     0.68
## [61,]       0.68  222                          0.47     0.69
## [62,]       0.69  231                          0.47     0.68
## [63,]       0.70  239                          0.47     0.68
## [64,]       0.71  254                          0.49     0.67
## [65,]       0.72  263                          0.49     0.66
## [66,]       0.73  268                          0.50     0.66
## [67,]       0.74  287                          0.51     0.64
## [68,]       0.75  292                          0.52     0.64
## [69,]       0.76  307                          0.53     0.62
## [70,]       0.77  311                          0.53     0.62
## [71,]       0.78  321                          0.54     0.62
## [72,]       0.79  330                          0.54     0.61
## [73,]       0.80  344                          0.55     0.60
## [74,]       0.81  348                          0.55     0.60
## [75,]       0.82  358                          0.56     0.59
## [76,]       0.83  378                          0.57     0.57
## [77,]       0.84  392                          0.58     0.56
## [78,]       0.85  401                          0.58     0.56
## [79,]       0.86  416                          0.59     0.54
## [80,]       0.87  417                          0.57     0.56
```

```
## [81,]      0.88  426                    0.58     0.55
## [82,]      0.89  436                    0.58     0.54
## [83,]      0.90  444                    0.58     0.54
## [84,]      0.91  452                    0.58     0.54
## [85,]      0.92  462                    0.58     0.53
## [86,]      0.93  471                    0.58     0.52
## [87,]      0.94  501                    0.60     0.50
## [88,]      0.95  541                    0.62     0.46
## [89,]      0.96  561                    0.63     0.44
## [90,]      0.97  581                    0.63     0.42
## [91,]      0.98  610                    0.64     0.39
## [92,]      0.99  640                    0.65     0.36
## [93,]      1.00  660                    0.66     0.34
```

```
results_new[which.min(results_new[,2]),2]
```

```
## Cost
##   76
```

```
results_new[which.min(results_new[,3]),3]
```

```
## False Positive Ratio
##                 0.25
```

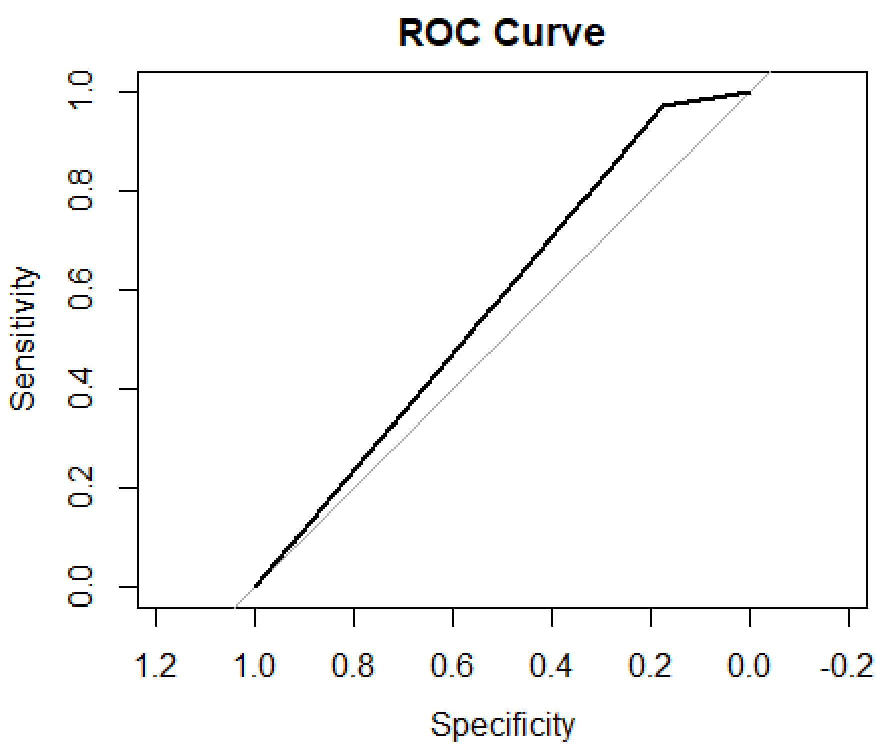
Final model fit measures for 0.28 threshold.

```
final_predict <- predict(logit_selected,newdata = germandata_test, type =
"response")
final_predict_01 <- as.integer(final_predict > 0.28)
confusionMatrix(factor(germandata_test$V21, levels =
c(1,0)),factor(final_predict_01,levels = c(1,0)))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   1   0
##          1 128   4
##          0  56  12
##
##                Accuracy : 0.7
##                  95% CI : (0.6314, 0.7626)
##     No Information Rate : 0.92
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1794
##
##  Mcnemar's Test P-Value : 4.577e-11
##
##             Sensitivity : 0.6957
##             Specificity : 0.7500
##          Pos Pred Value : 0.9697
```

```
##          Neg Pred Value : 0.1765
##              Prevalence : 0.9200
##          Detection Rate : 0.6400
##    Detection Prevalence : 0.6600
##       Balanced Accuracy : 0.7228
##
##        'Positive' Class : 1
##
```

```
AUC <- roc(germandata_test$V21,final_predict_01)
plot(AUC, main = "ROC Curve")
```




```
AUC

##
## Call:
## roc.default(response = germandata_test$V21, predictor = final_predict_01)
##
## Data: final_predict_01 in 68 controls (germandata_test$V21 0) < 132 cases
(germandata_test$V21 1).
## Area under the curve: 0.5731
```