Homework 7-Regression with Tree-based Methods & Logistic Regression

Mark Pearl

26/02/2020

10.1 Regression Using Tree and Random Forest Methods with UsCrime Dataset

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) Regression Tree Model

```
uscrime_data <- read.table('C:/Users/mjpearl/Desktop/omsa/ISYE-6501-OAN/hw7/data/uscrime.txt',header = T
head(uscrime_data)
```

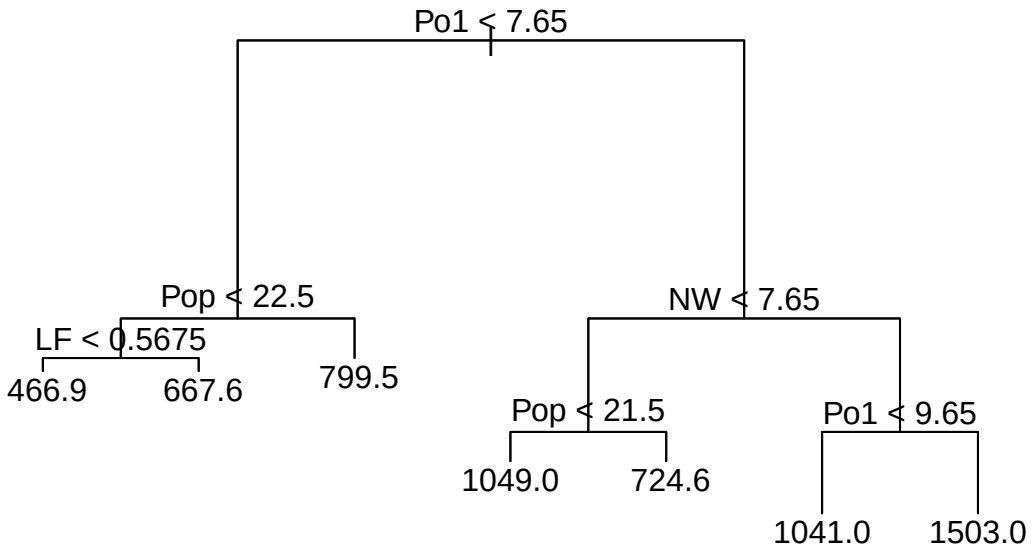
```
##      M So   Ed  Po1  Po2    LF   M.F Pop   NW    U1  U2 Wealth Ineq
## 1 15.1  1  9.1  5.8  5.6 0.510  95.0  33 30.1 0.108 4.1   3940 26.1
## 2 14.3  0 11.3 10.3  9.5 0.583 101.2  13 10.2 0.096 3.6   5570 19.4
## 3 14.2  1  8.9  4.5  4.4 0.533  96.9  18 21.9 0.094 3.3   3180 25.0
## 4 13.6  0 12.1 14.9 14.1 0.577  99.4 157  8.0 0.102 3.9   6730 16.7
## 5 14.1  0 12.1 10.9 10.1 0.591  98.5  18  3.0 0.091 2.0   5780 17.4
## 6 12.1  0 11.0 11.8 11.5 0.547  96.4  25  4.4 0.084 2.9   6890 12.6
##       Prob    Time Crime
## 1 0.084602 26.2011   791
## 2 0.029599 25.2999  1635
## 3 0.083401 24.3006   578
## 4 0.015801 29.9012  1969
## 5 0.041399 21.2998  1234
## 6 0.034201 20.9995   682
```

```
tree_model <- tree(Crime ~., uscrime_data)
summary(tree_model)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = uscrime_data)
## 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
```

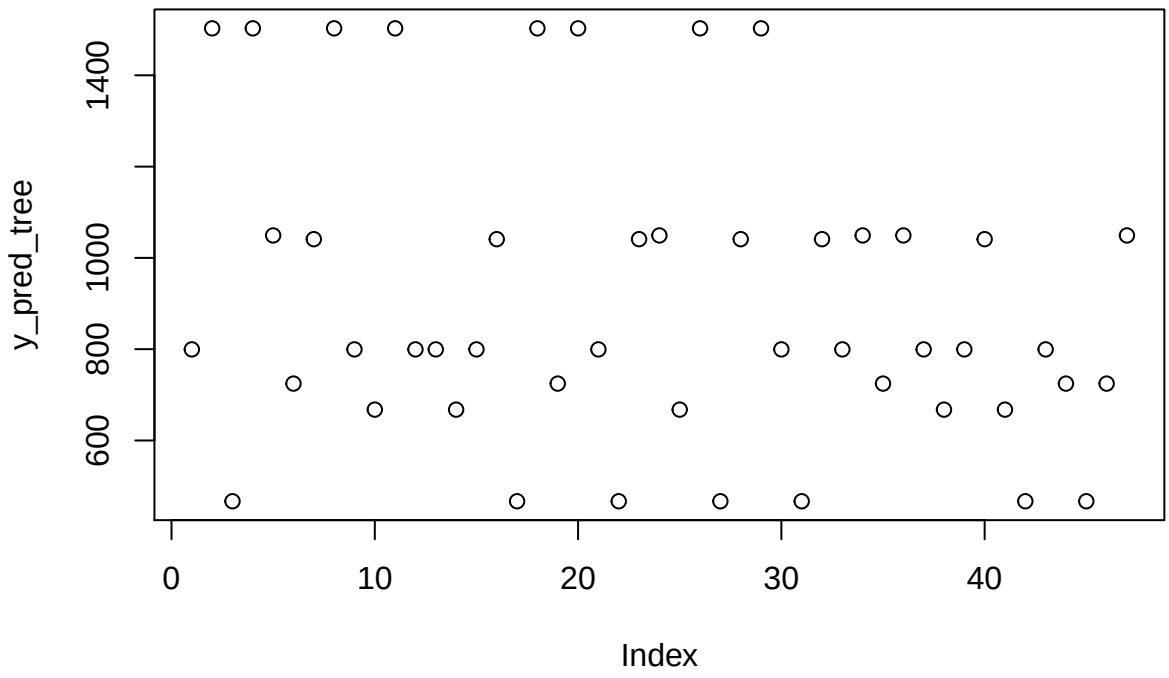
We can see with our tree output that it's selected the Po1, Pop, LF and NW variables. For the initial summary, this seems to be the combination of factors that are the best candidates for splitting based on the default settings for the model.

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



Now we can use the predict function for this tree to compare the observed values against the predicted observations, and determine the R^2 value to determine it's relative performance.

```
y_pred_tree <- predict(tree_model)
plot(y_pred_tree)
```



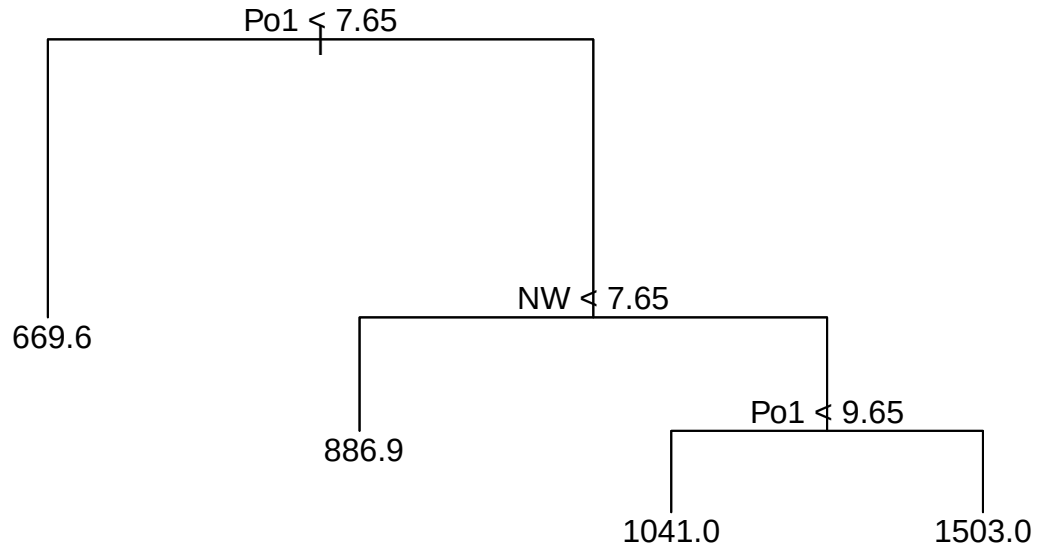

```
#Based on source to create helper function for R2, which represents 1 - the residual sum of squares div
rsq <- function (y_pred, y_actual) {
  residual_ss <- sum((y_pred - y_actual) ^ 2)
  total_ss <- sum((y_actual - mean(y_actual)) ^ 2)
  rsq <- 1 - residual_ss / total_ss
  return (rsq)
}


rsq_tree <- rsq(y_pred_tree, uscrime_data$Crime)
rsq_tree
```

```
## [1] 0.7244962
```

Based on our results for R^2 we can see the initial results provide a good estimator for the crime variable.

We can now look at pruning the tree to work backward from the leaves to determine if the estimation error has increased from the branch, in which case we'll remove it. We will cap our terminal node setting for "best" to 4 as we want to ensure we don't want a tree with too high of a max depth as this will likely produce leaves with splits that result in less< 5% of total observations and ultimately, overfitting.

```
#Calculate the new prediction on the pruned tree with number of terminal nodes set to 4
tree_pruned <- prune.tree(tree_model,best= 4)
plot(tree_pruned)
text(tree_pruned)
```



We can see from the result that we have fewer leaves, and our tree is split by a reduced number of features compared to our original tree.

```
y_pred_pruned <- predict(tree_pruned)
rsq_pruned <- rsq(y_pred_pruned, uscrime_data$Crime)
rsq_pruned
```

```
## [1] 0.6174017
```

We can see that with a lower R^2 for our pruned tree that it accounts for less of the variance compared to our original tree. However, this is likely due to overfitting, in that the original tree will likely perform better on our training dataset as there's more features that are split on. But our pruned tree will do better on our test dataset, as it will react to newer observations with less bias.

Another exercise we could do is reduce our tree a given leaf, as it splits our data appropriately in addition to accounting for a significant amount of variance with minimal feature engineering (i.e. in comparison to our previous homework exercises.). From here we can calculate the r^2 value to determine the impact on performance.

Now we will run the random forest model on our dataset. From the lecture we determined that the appropriate number of features to use for each tree is 1 + log(n), and a number of trees ideally specified between 500-1000. This means the number of features that will be split on to form a given sub-tree, each-tree will have different permutations of both feature and dataset combinations (using boosting). The average results across all tree will be used, compared to a classifcation result which uses the mode.

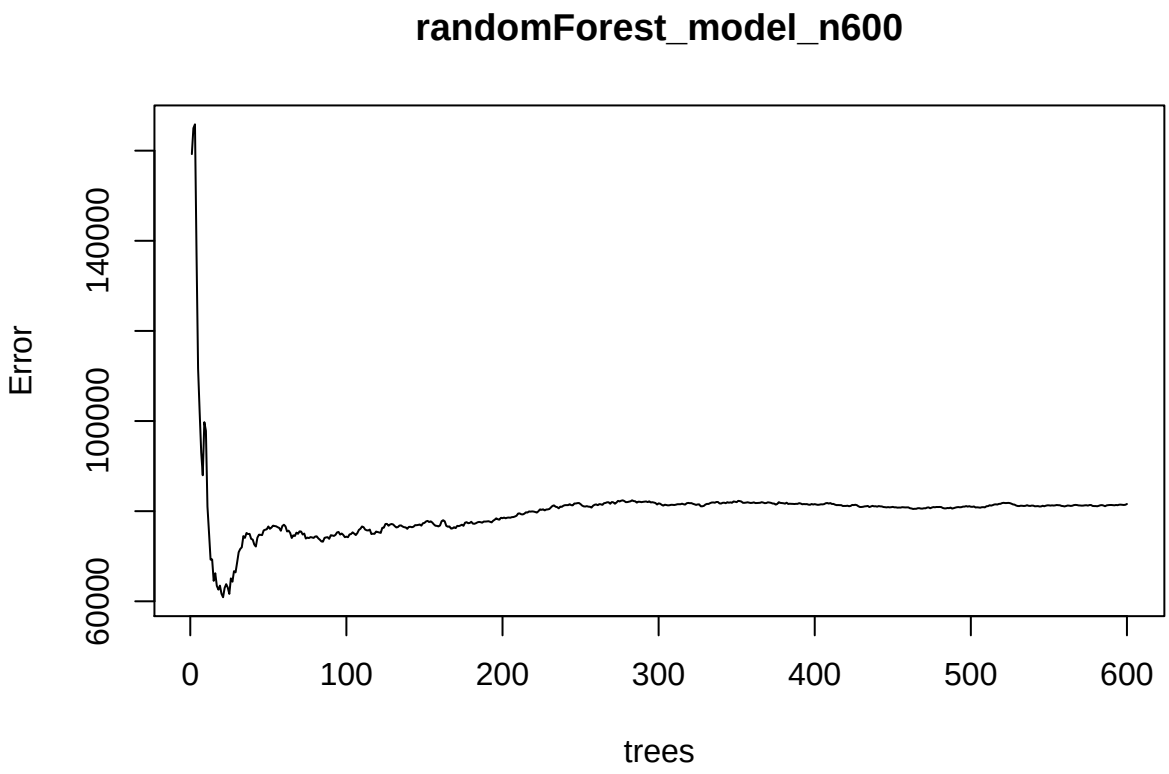
```r
num_features_rf <- 1 + log(ncol(uscrime_data))
randomForest_model_n600 <- randomForest(Crime~., data = uscrime_data, mtry = num_features_rf, importanc
randomForest_model_n900 <- randomForest(Crime~., data = uscrime_data, mtry = num_features_rf, importanc
randomForest_model_n600
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime_data, mtry = num_features_rf,      importance = T,
##                Type of random forest: regression
##                      Number of trees: 600
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 81592.27
##                     % Var explained: 44.27
```

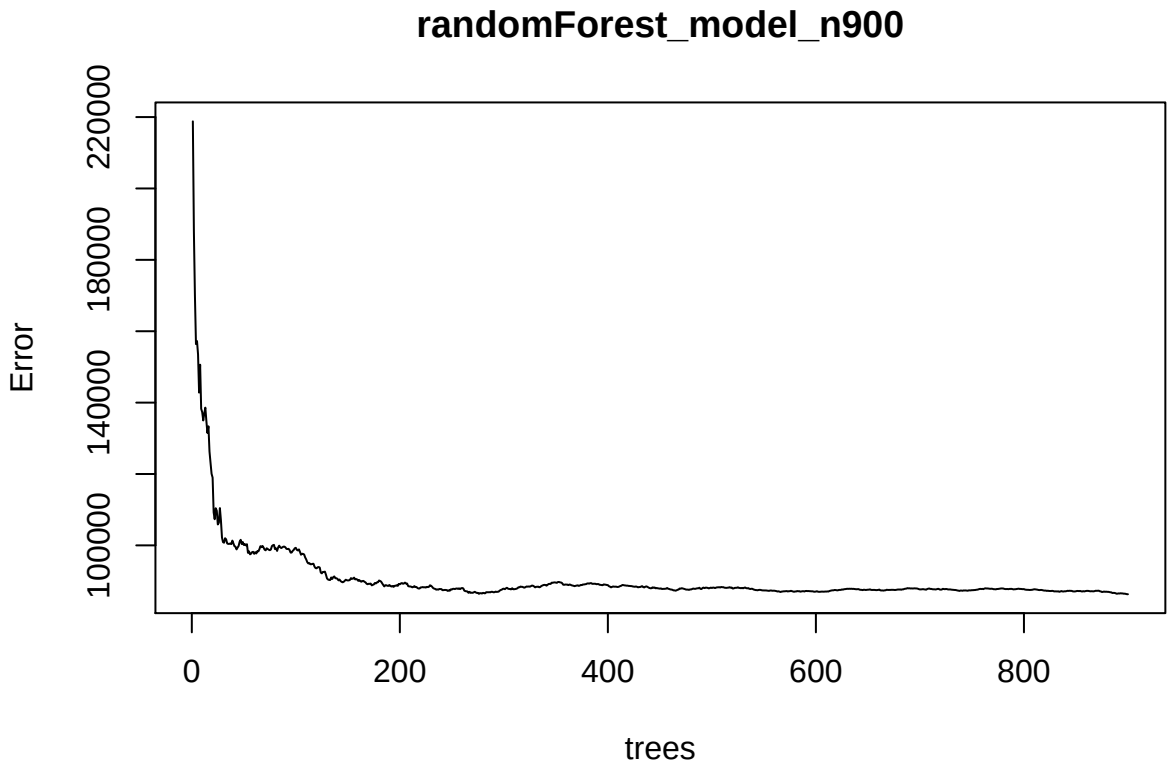
```r
randomForest_model_n900
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime_data, mtry = num_features_rf,      importance = T,
##                Type of random forest: regression
##                      Number of trees: 900
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 86310.99
##                     % Var explained: 41.05
```

```r
plot(randomForest_model_n600)
```

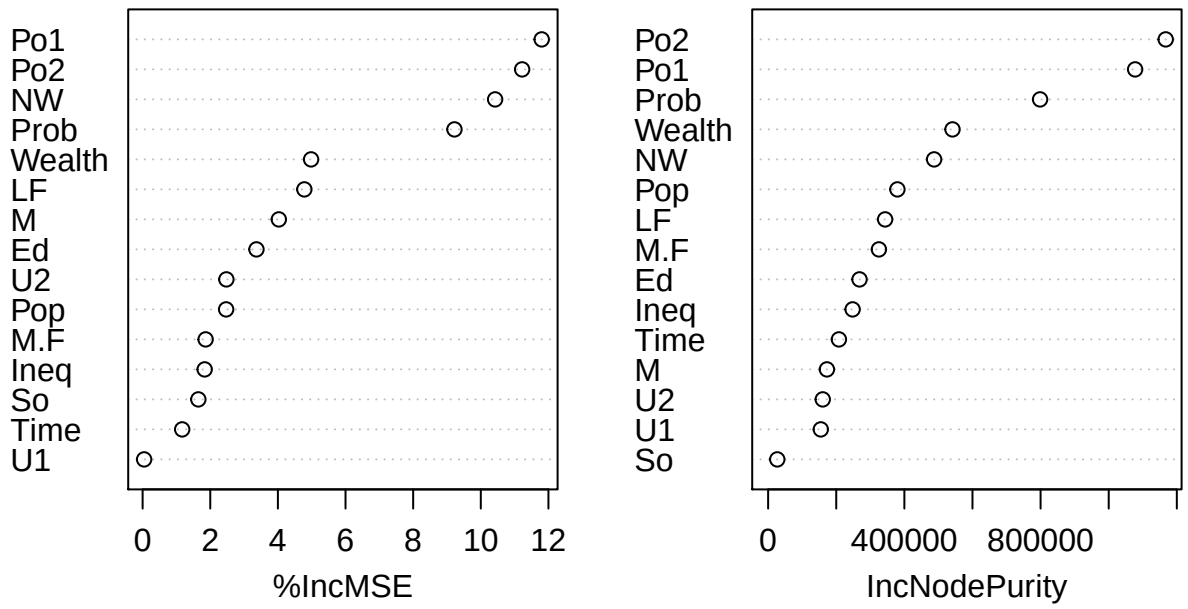



```
plot(randomForest_model_n900)
```



From our results we can see that our error tends to flatten around 100 trees. We can see when tweaking the ntrees number that the tree flattens out earlier. This makes sense as increasing the number of trees will help our model generalize for marginally more variation.

```
# Importance Plot
varImpPlot(randomForest_model_n600)
```



We can see based on the feature importance plot, that the features used for splitting in the original tree model align well with this output, as a majority of the features used to split ([1] "Po1" "Pop" "LF" "NW") can be seen at the top of graph. This is further clarified with the pruned result as the Pop feature is removed, and can be represented with moderate importance.

However, when looking at the results of our summary we can see that the R2 is only 42% (i.e. shown by % Var explained in the summary output) compared to our tree-based model used previously.

10.2 Real-Life Application

An example I used at work for Logistic regresison was for predicting a fraudulent customer based on various features. In this case we're looking for a binary response of 0,1. Many factors can attribute to mortgage fraud including involved parties (i.e. brokers, lawyers), FICO score, credit scores, demographic information, etc. This can be used in production to help determine which customers are more likely for fraud, and can help minimize risk for the client.

10.3 Logistic Regression with German Credit Dataset

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.

```r
german_data <- read.table('C:/Users/mjpearl/Desktop/omsa/ISYE-6501-OAN/hw7/data/germancredit.txt',header
#Replace target variable value of 2 with 0 as we're training on a binary response variable
german_data$V21[german_data$V21 == 2] <- 0
head(german_data)
```

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

```r
sample <- sample(1:nrow(german_data), size = round(0.75*nrow(german_data)))
train <- german_data[sample,]
test <- german_data[-sample,]
```

```r
logistic_full <- glm(V21 ~., family = binomial(link="logit"), train)
summary(logistic_full)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6784  -0.6542   0.3762   0.6945   2.2438
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.225e-01  1.234e+00   0.180 0.856938
## V1A12        3.993e-01  2.595e-01   1.539 0.123907
## V1A13        1.185e+00  4.732e-01   2.505 0.012257 *
## V1A14        1.730e+00  2.670e-01   6.479 9.22e-11 ***
## V2          -2.733e-02  1.112e-02  -2.457 0.014002 *
## V3A31       -9.697e-01  6.539e-01  -1.483 0.138125
## V3A32       -5.362e-02  4.988e-01  -0.107 0.914401
## V3A33        7.557e-02  5.639e-01   0.134 0.893388
## V3A34        7.116e-01  5.214e-01   1.365 0.172278
## V4A41        1.593e+00  4.267e-01   3.733 0.000189 ***
## V4A410       1.129e+00  9.016e-01   1.252 0.210612
## V4A42        7.992e-01  3.018e-01   2.648 0.008094 **
## V4A43        9.695e-01  2.929e-01   3.310 0.000934 ***
## V4A44        7.413e-01  1.258e+00   0.589 0.555779
## V4A45        1.851e-01  6.459e-01   0.287 0.774391
```

```
## V4A46       -4.489e-02  4.742e-01  -0.095 0.924584
## V4A48        1.817e+00  1.239e+00   1.467 0.142441
## V4A49        6.783e-01  3.912e-01   1.734 0.082981 .
## V5          -1.556e-04  5.231e-05  -2.976 0.002922 **
## V6A62        1.328e-01  3.445e-01   0.386 0.699847
## V6A63        6.311e-01  4.712e-01   1.339 0.180523
## V6A64        1.480e+00  5.825e-01   2.540 0.011069 *
## V6A65        4.633e-01  3.032e-01   1.528 0.126432
## V7A72       -2.870e-01  5.261e-01  -0.545 0.585410
## V7A73       -1.568e-01  5.101e-01  -0.307 0.758582
## V7A74        4.186e-01  5.408e-01   0.774 0.438896
## V7A75       -8.453e-02  5.084e-01  -0.166 0.867939
## V8          -2.215e-01  1.037e-01  -2.136 0.032661 *
## V9A92       -4.440e-03  4.434e-01  -0.010 0.992010
## V9A93        9.385e-01  4.388e-01   2.139 0.032458 *
## V9A94        4.580e-01  5.285e-01   0.867 0.386192
## V10A102     -1.273e-01  4.608e-01  -0.276 0.782390
## V10A103      9.881e-01  4.894e-01   2.019 0.043467 *
## V11          4.222e-02  1.022e-01   0.413 0.679432
## V12A122     -1.686e-01  2.952e-01  -0.571 0.567820
## V12A123     -2.007e-01  2.756e-01  -0.728 0.466318
## V12A124     -9.255e-01  5.131e-01  -1.804 0.071288 .
## V13          1.146e-02  1.054e-02   1.087 0.277036
## V14A142      5.252e-02  5.105e-01   0.103 0.918058
## V14A143      5.393e-01  2.904e-01   1.857 0.063315 .
## V15A152      2.513e-01  2.787e-01   0.902 0.367213
## V15A153      7.148e-01  5.612e-01   1.274 0.202749
## V16         -3.279e-01  2.210e-01  -1.484 0.137913
## V17A172     -2.190e-01  8.018e-01  -0.273 0.784775
## V17A173     -4.109e-01  7.716e-01  -0.532 0.594382
## V17A174     -1.606e-01  7.754e-01  -0.207 0.835926
## V18         -6.355e-02  3.007e-01  -0.211 0.832631
## V19A192      3.741e-01  2.416e-01   1.549 0.121483
## V20A202      1.026e+00  6.578e-01   1.559 0.118927
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 905.90  on 749  degrees of freedom
## Residual deviance: 666.04  on 701  degrees of freedom
## AIC: 764.04
##
## Number of Fisher Scoring iterations: 5
```

```
logistics_impFeatures <- glm(V21~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V9 + +V10 + V13,
                family=binomial(link = 'logit'),
                data=train)
summary(logistics_impFeatures)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V7 + V9 + +V10 +
##     V13, family = binomial(link = "logit"), data = train)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5939  -0.7369   0.4079   0.6912   2.2468
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.290e-01  9.071e-01  -0.363 0.716807
## V1A12        4.587e-01  2.487e-01   1.844 0.065112 .
## V1A13        1.259e+00  4.634e-01   2.718 0.006571 **
## V1A14        1.703e+00  2.568e-01   6.629 3.38e-11 ***
## V2          -3.680e-02  1.018e-02  -3.616 0.000299 ***
## V3A31       -1.034e+00  6.084e-01  -1.699 0.089364 .
## V3A32        1.848e-01  4.698e-01   0.393 0.694089
## V3A33        1.436e-01  5.437e-01   0.264 0.791643
## V3A34        8.083e-01  5.001e-01   1.616 0.106012
## V4A41        1.509e+00  4.011e-01   3.761 0.000169 ***
## V4A410       1.224e+00  9.103e-01   1.345 0.178621
## V4A42        6.892e-01  2.886e-01   2.388 0.016962 *
## V4A43        8.861e-01  2.820e-01   3.142 0.001679 **
## V4A44        7.450e-01  1.196e+00   0.623 0.533350
## V4A45       -4.649e-02  6.214e-01  -0.075 0.940363
## V4A46       -2.684e-01  4.542e-01  -0.591 0.554534
## V4A48        1.614e+00  1.186e+00   1.362 0.173307
## V4A49        5.797e-01  3.753e-01   1.544 0.122498
## V5          -1.049e-04  4.396e-05  -2.386 0.017026 *
## V6A62        3.076e-02  3.278e-01   0.094 0.925239
## V6A63        7.616e-01  4.656e-01   1.636 0.101855
## V6A64        1.255e+00  5.449e-01   2.304 0.021245 *
## V6A65        4.999e-01  2.936e-01   1.702 0.088672 .
## V7A72       -4.844e-01  4.453e-01  -1.088 0.276704
## V7A73       -3.093e-01  4.252e-01  -0.727 0.466965
## V7A74        2.774e-01  4.608e-01   0.602 0.547153
## V7A75       -3.712e-01  4.304e-01  -0.862 0.388417
## V9A92       -1.280e-01  4.275e-01  -0.300 0.764550
## V9A93        7.316e-01  4.164e-01   1.757 0.078909 .
## V9A94        4.316e-01  5.124e-01   0.842 0.399653
## V10A102     -1.564e-01  4.418e-01  -0.354 0.723418
## V10A103      1.027e+00  4.621e-01   2.223 0.026188 *
## V13          1.198e-02  9.779e-03   1.226 0.220378
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 905.90  on 749  degrees of freedom
## Residual deviance: 688.27  on 717  degrees of freedom
## AIC: 754.27
##
## Number of Fisher Scoring iterations: 5
```

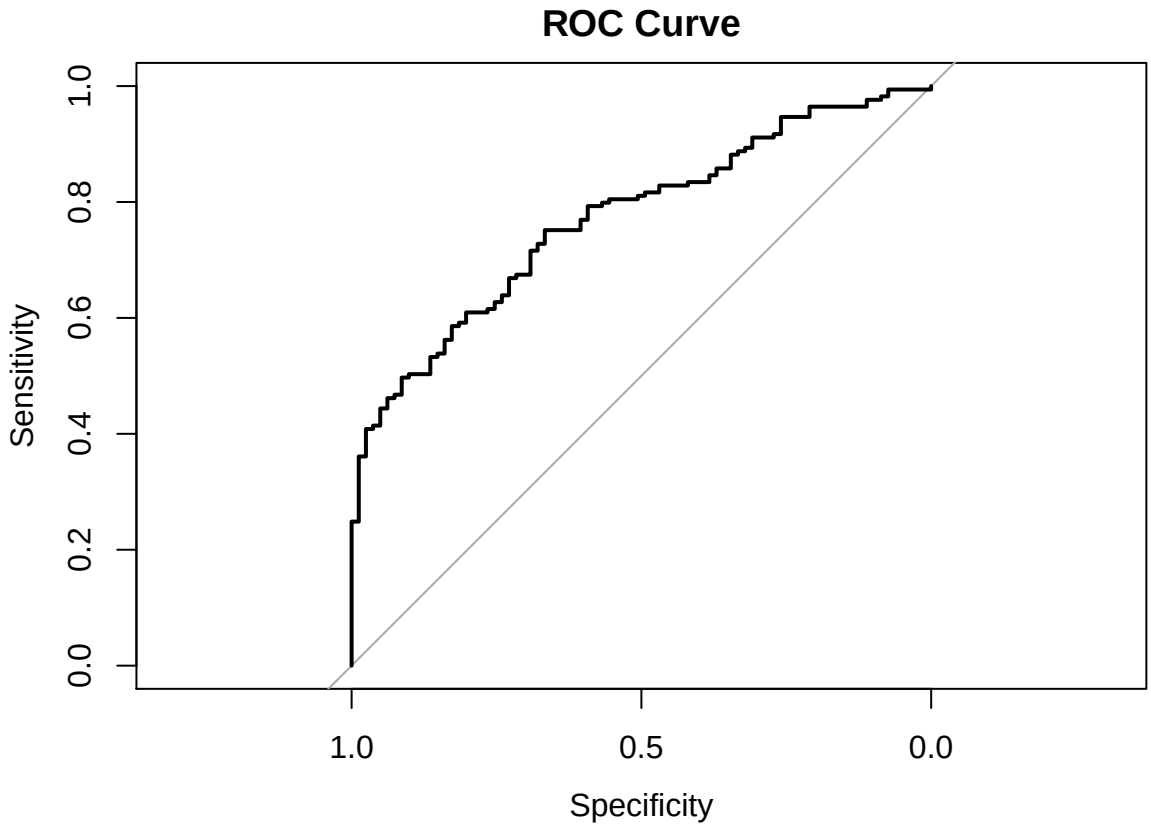
```
y_pred_feature_selection <- predict(logistics_impFeatures, test, type = "response")
```

```
AUC <- roc(test$V21, y_pred_feature_selection)
```

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

```
## Setting direction: controls < cases
```

```
plot(AUC, main = "ROC Curve")
```




```
AUC
```

```
##
## Call:
## roc.default(response = test$V21, predictor = y_pred_feature_selection)
##
## Data: y_pred_feature_selection in 81 controls (test$V21 0) < 169 cases (test$V21 1).
## Area under the curve: 0.7711
```

```
results <- vector("list",92)

for (i in seq(8,95, by=1)){
    thresh <- i/100
    y_pred <- as.integer(y_pred_feature_selection > thresh)
    table <- as.matrix(table(y_pred, test$V21))
```

```
    cost <- table[2,1] + 5*table[1,2]
    results[i] <- cost
}

results
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
##
## [[7]]
## NULL
##
## [[8]]
## [1] 85
##
## [[9]]
## [1] 85
##
## [[10]]
## [1] 84
##
## [[11]]
## [1] 81
##
## [[12]]
## [1] 81
##
## [[13]]
## [1] 81
##
## [[14]]
## [1] 80
##
## [[15]]
## [1] 80
##
## [[16]]
## [1] 90
```

```
##
## [[17]]
## [1] 89
##
## [[18]]
## [1] 94
##
## [[19]]
## [1] 94
##
## [[20]]
## [1] 97
##
## [[21]]
## [1] 102
##
## [[22]]
## [1] 102
##
## [[23]]
## [1] 102
##
## [[24]]
## [1] 102
##
## [[25]]
## [1] 101
##
## [[26]]
## [1] 100
##
## [[27]]
## [1] 96
##
## [[28]]
## [1] 96
##
## [[29]]
## [1] 99
##
## [[30]]
## [1] 109
##
## [[31]]
## [1] 107
##
## [[32]]
## [1] 110
##
## [[33]]
## [1] 115
##
## [[34]]
## [1] 120
```

```
##
## [[35]]
## [1] 125
##
## [[36]]
## [1] 130
##
## [[37]]
## [1] 130
##
## [[38]]
## [1] 129
##
## [[39]]
## [1] 134
##
## [[40]]
## [1] 131
##
## [[41]]
## [1] 141
##
## [[42]]
## [1] 149
##
## [[43]]
## [1] 153
##
## [[44]]
## [1] 168
##
## [[45]]
## [1] 171
##
## [[46]]
## [1] 171
##
## [[47]]
## [1] 181
##
## [[48]]
## [1] 189
##
## [[49]]
## [1] 187
##
## [[50]]
## [1] 192
##
## [[51]]
## [1] 189
##
## [[52]]
## [1] 193
```

```
##
## [[53]]
## [1] 197
##
## [[54]]
## [1] 204
##
## [[55]]
## [1] 204
##
## [[56]]
## [1] 205
##
## [[57]]
## [1] 208
##
## [[58]]
## [1] 228
##
## [[59]]
## [1] 237
##
## [[60]]
## [1] 242
##
## [[61]]
## [1] 241
##
## [[62]]
## [1] 239
##
## [[63]]
## [1] 252
##
## [[64]]
## [1] 266
##
## [[65]]
## [1] 265
##
## [[66]]
## [1] 270
##
## [[67]]
## [1] 285
##
## [[68]]
## [1] 300
##
## [[69]]
## [1] 303
##
## [[70]]
## [1] 312
```

```
##
## [[71]]
## [1] 326
##
## [[72]]
## [1] 336
##
## [[73]]
## [1] 348
##
## [[74]]
## [1] 347
##
## [[75]]
## [1] 346
##
## [[76]]
## [1] 356
##
## [[77]]
## [1] 365
##
## [[78]]
## [1] 384
##
## [[79]]
## [1] 393
##
## [[80]]
## [1] 416
##
## [[81]]
## [1] 421
##
## [[82]]
## [1] 426
##
## [[83]]
## [1] 433
##
## [[84]]
## [1] 452
##
## [[85]]
## [1] 460
##
## [[86]]
## [1] 475
##
## [[87]]
## [1] 474
##
## [[88]]
## [1] 499
```

```
##
## [[89]]
## [1] 517
##
## [[90]]
## [1] 532
##
## [[91]]
## [1] 596
##
## [[92]]
## [1] 616
##
## [[93]]
## [1] 645
##
## [[94]]
## [1] 685
##
## [[95]]
## [1] 705
```

We use these values from 0 to 92 as with the values being closer to zero or one, we get only one column. Thus helping us find threshold.

We can see with the proceeding results the threshold cuts off well at 15.

```
y_pred_feature_selection_final <- predict(logistics_impFeatures, test, type = "response")
y_pred_final <- as.integer(y_pred_feature_selection_final > 0.15)
table(y_pred_final, test$V21)
```

```
##
## y_pred_final   0   1
##            0   6   1
##            1  75 168
```