Homework7

Pablo

08/02/2019

Contents

Question 10.1

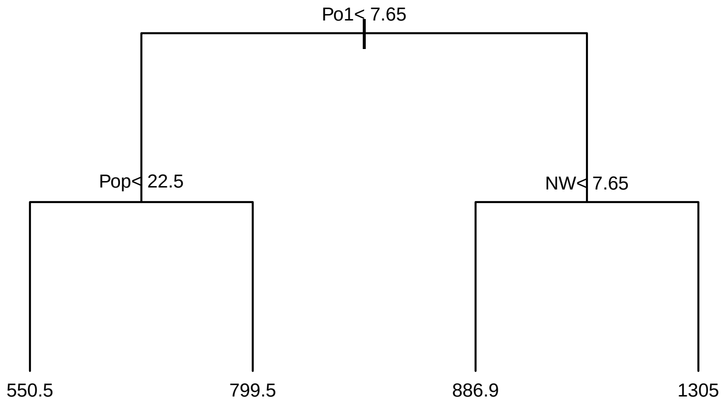
Using the same crime data set uscrime.txt as in Questions 8.2 and 9.1, find the best model you can using (a) a regression tree model, and (b) a random forest model

```
set.seed(42) #set the seed
# import data
uscrime <- read.delim("~/Documents/R/GeorgiaTech/AdvancedRegression/uscrime.txt")
library(rpart)
model_tree <- rpart(Crime~., uscrime)
summary(model_tree)
```

```
## Call:
## rpart(formula = Crime ~ ., data = uscrime)
##   n= 47
##
##           CP nsplit rel error    xerror      xstd
## 1 0.36296293      0 1.0000000 1.0261114 0.2572731
## 2 0.14814320      1 0.6370371 0.9439777 0.2107771
## 3 0.05173165      2 0.4888939 1.1083238 0.2534765
## 4 0.01000000      3 0.4371622 1.0962595 0.2546578
##
## Variable importance
##    Po1    Po2 Wealth   Ineq   Prob      M     NW    Pop   Time     Ed
##     17     17     11     11     10     10      9      5      4      4
##     LF     So
##      1      1
##
## Node number 1: 47 observations,    complexity param=0.3629629
##   mean=905.0851, MSE=146402.7
##   left son=2 (23 obs) right son=3 (24 obs)
##   Primary splits:
##       Po1    < 7.65      to the left,  improve=0.3629629, (0 missing)
```

```
##       Po2    < 7.2       to the left,  improve=0.3629629, (0 missing)
##       Prob   < 0.0418485 to the right, improve=0.3217700, (0 missing)
##       NW     < 7.65      to the left,  improve=0.2356621, (0 missing)
##       Wealth < 6240      to the left,  improve=0.2002403, (0 missing)
##   Surrogate splits:
##       Po2    < 7.2       to the left,  agree=1.000, adj=1.000, (0 split)
##       Wealth < 5330      to the left,  agree=0.830, adj=0.652, (0 split)
##       Prob   < 0.043598  to the right, agree=0.809, adj=0.609, (0 split)
##       M      < 13.25     to the right, agree=0.745, adj=0.478, (0 split)
##       Ineq   < 17.15     to the right, agree=0.745, adj=0.478, (0 split)
##
## Node number 2: 23 observations,    complexity param=0.05173165
##   mean=669.6087, MSE=33880.15
##   left son=4 (12 obs) right son=5 (11 obs)
##   Primary splits:
##       Pop < 22.5      to the left,  improve=0.4568043, (0 missing)
##       M   < 14.5      to the left,  improve=0.3931567, (0 missing)
##       NW  < 5.4       to the left,  improve=0.3184074, (0 missing)
##       Po1 < 5.75      to the left,  improve=0.2310098, (0 missing)
##       U1  < 0.093     to the right, improve=0.2119062, (0 missing)
##   Surrogate splits:
##       NW   < 5.4       to the left,  agree=0.826, adj=0.636, (0 split)
##       M    < 14.5      to the left,  agree=0.783, adj=0.545, (0 split)
##       Time < 22.30055  to the left,  agree=0.783, adj=0.545, (0 split)
##       So   < 0.5       to the left,  agree=0.739, adj=0.455, (0 split)
##       Ed   < 10.85     to the right, agree=0.739, adj=0.455, (0 split)
##
## Node number 3: 24 observations,    complexity param=0.1481432
##   mean=1130.75, MSE=150173.4
##   left son=6 (10 obs) right son=7 (14 obs)
##   Primary splits:
##       NW   < 7.65      to the left,  improve=0.2828293, (0 missing)
##       M    < 13.05     to the left,  improve=0.2714159, (0 missing)
##       Time < 21.9001   to the left,  improve=0.2060170, (0 missing)
##       M.F  < 99.2      to the left,  improve=0.1703438, (0 missing)
##       Po1  < 10.75     to the left,  improve=0.1659433, (0 missing)
##   Surrogate splits:
##       Ed   < 11.45     to the right, agree=0.750, adj=0.4, (0 split)
##       Ineq < 16.25     to the left,  agree=0.750, adj=0.4, (0 split)
##       Time < 21.9001   to the left,  agree=0.750, adj=0.4, (0 split)
##       Pop  < 30        to the left,  agree=0.708, adj=0.3, (0 split)
##       LF   < 0.5885    to the right, agree=0.667, adj=0.2, (0 split)
##
## Node number 4: 12 observations
##   mean=550.5, MSE=20317.58
##
## Node number 5: 11 observations
##   mean=799.5455, MSE=16315.52
##
## Node number 6: 10 observations
##   mean=886.9, MSE=55757.49
##
## Node number 7: 14 observations
##   mean=1304.929, MSE=144801.8
```

```r
# plot
plot(model_tree, margin=0.2, uniform = TRUE)
text(model_tree,cex=0.55)
```



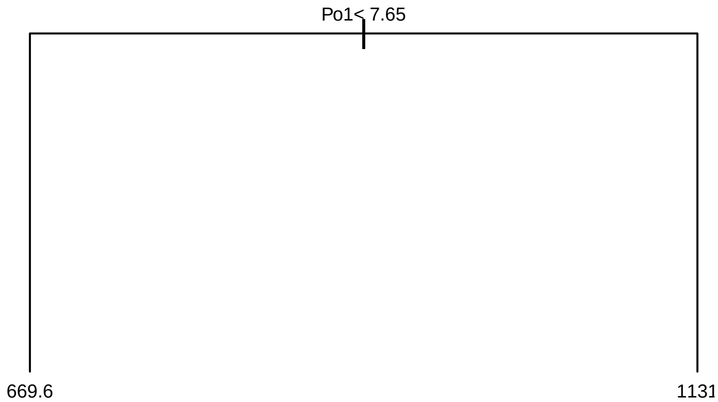

```r
# the plot shows that only 3 predictors were used to construct the regression tree

# test predicting the same data with the model
y_predicted <- predict(model_tree)
# calculate square error
RSS <- sum((y_predicted-uscrime$Crime)^2)

# measure quality by calculating R^2 and RSE
TSS <- sum((uscrime$Crime - mean(uscrime$Crime))^2)
R <- 1 - RSS/TSS
R
```

```
## [1] 0.5628378
```

```r
# Residual Standard Error (RSE)
RSE <- sqrt((1/(nrow(uscrime)-2))*RSS)
error_rate <- RSE/mean(uscrime$Crime)
error_rate
```

```
## [1] 0.2856598
```

```r
# try to prune the tree

# use the rpart.control to modify the rpart fit parameter
# The complexity parameter (cp) in rpart is the minimum improvement in the model needed at each node.
bestcp <- model_tree$cptable[which.min(model_tree$cptable[,"xerror"]),"CP"]
model_tree.pruned <- prune(model_tree, cp = bestcp)

# plot
plot(model_tree.pruned,margin=0.2, uniform = TRUE)
text(model_tree.pruned, cex=0.55)
```



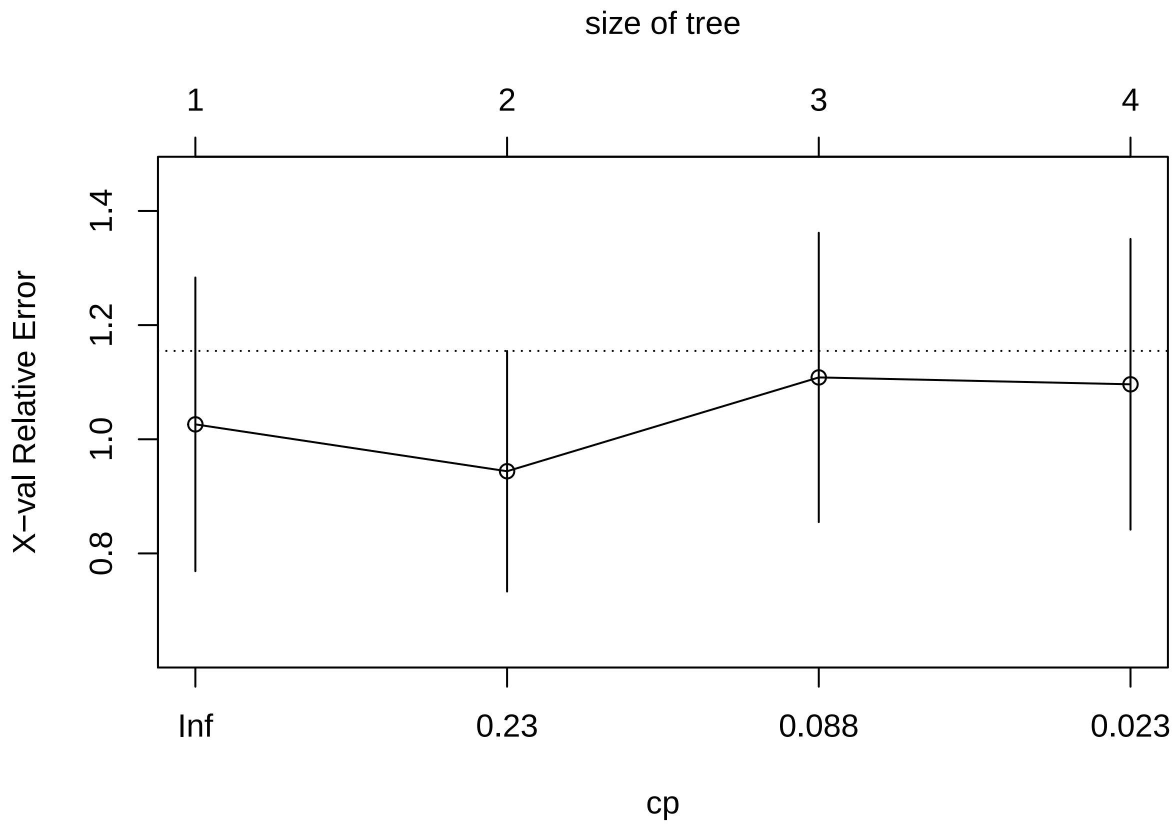

```r
# the plot shows that only 3 predictors were used to construct the regression tree

# test predicting the same data with the model
y_predicted <- predict(model_tree.pruned)
# calculate square error
RSS <- sum((y_predicted-uscrime$Crime)^2)

# measure quality by calculating R^2 and RSE
TSS <- sum((uscrime$Crime - mean(uscrime$Crime))^2)
R <- 1 - RSS/TSS
R
```

```
## [1] 0.3629629
```

```r
# Residual Standard Error (RSE)
RSE <- sqrt((1/(nrow(uscrime)-2))*RSS)
error_rate <- RSE/mean(uscrime$Crime)
error_rate
```

```
## [1] 0.3448341
```

```r
# I found out that this package can show the Cross Validated results
plotcp(model_tree)
```




```r
printcp(model_tree)
```

```
##
## Regression tree:
## rpart(formula = Crime ~ ., data = uscrime)
##
## Variables actually used in tree construction:
## [1] NW  Po1 Pop
##
## Root node error: 6880928/47 = 146403
##
## n= 47
##
##         CP nsplit rel error  xerror    xstd
## 1 0.362963      0   1.00000 1.02611 0.25727
## 2 0.148143      1   0.63704 0.94398 0.21078
## 3 0.051732      2   0.48889 1.10832 0.25348
## 4 0.010000      3   0.43716 1.09626 0.25466
```

10.1 a) Analysis

First I chose to use the rpart package to test the regression tree. This model shows the usage of three variables: `Pop`, `NW` and `Po1`.

The quality of a linear regression fit is typically assessed using two quantities: the residual standard error (RSE) and the R^2.

The resulting R^2 is `0.562` and the RSE/mean can be interpreted as `28%` errror rate of the model (The less is better). This is not the best model and now it is time to analyze why.

I tried to prune the tree but I actually got a worse model and it only used one variable, so for this model it is not a good idea to use the pruned version. This is mainly because the data set has too few data points.

Question 10.1 b)

Now construct a random forest

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
random_forest <- randomForest(Crime~., data = uscrime,importance = TRUE)
random_forest
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 85865.34
##                     % Var explained: 41.35
```

```
# mtry = Number of variables randomly sampled as candidates at each split
# after some test, the mtry had higher variance explanied
random_forest <- randomForest(Crime~., data = uscrime,importance = TRUE, mtry = 3)
random_forest
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = uscrime, importance = TRUE,      mtry = 3)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 82243.21
##                     % Var explained: 43.82
```

```
y_predicted <- predict(random_forest)
RSS <- sum((y_predicted-uscrime$Crime)^2)
TSS <- sum((uscrime$Crime - mean(uscrime$Crime))^2)
R <- 1 - RSS/TSS
R
```

```
## [1] 0.4382399
```

```
# Residual Standard Error (RSE)
RSE <- sqrt((1/(nrow(uscrime)-2))*RSS)
error_rate <- RSE/mean(uscrime$Crime)
error_rate
```

```
## [1] 0.3238197
```

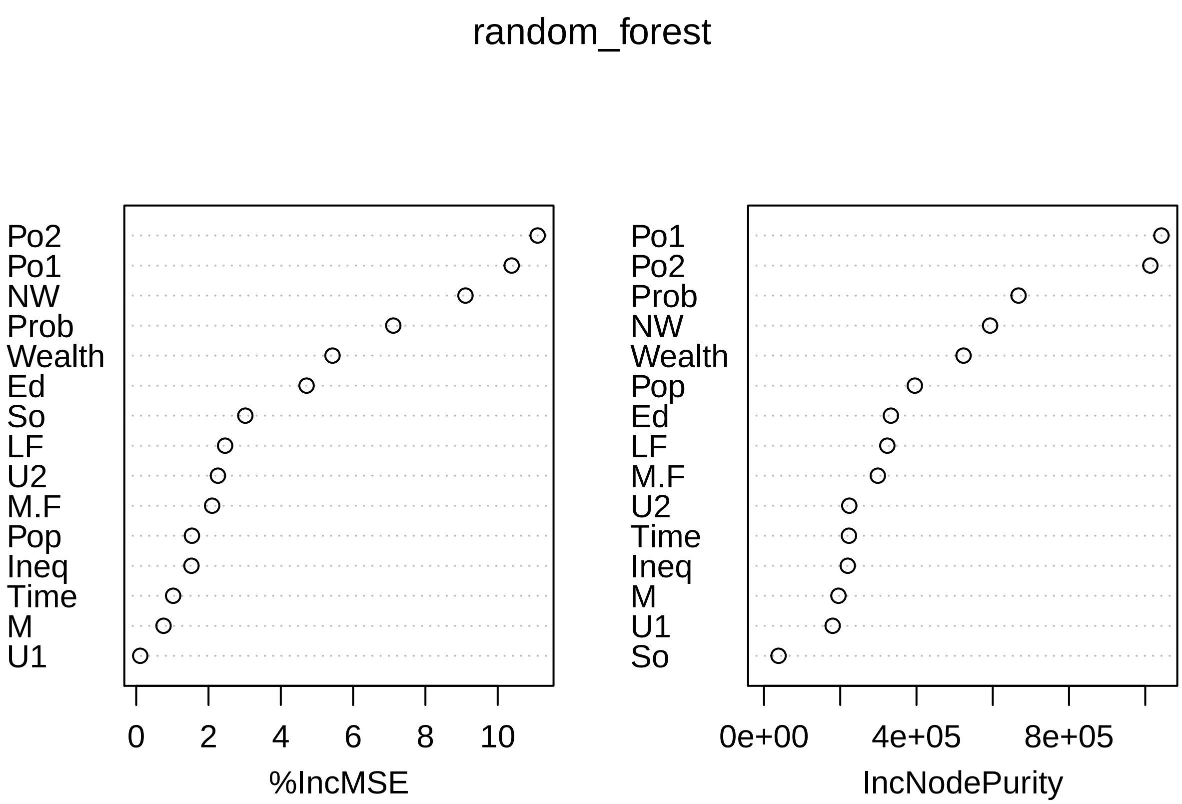
Analysis 10.1 b)

This a pretty straight forward model, the model chooses n random trees to fit the model and for each one it chooses random predictors to build them and to predict it uses the average response from all of them. It is harded to analyze because we can't graph all of the trees (500 for default) Although it is possible to see which are the most important variables like this

```
importance(random_forest)
```

```
##            %IncMSE IncNodePurity
## M        0.7556645     195283.19
## So       3.0181423      38139.18
## Ed       4.7128505     332868.45
## Po1     10.3864254    1042365.74
## Po2     11.1051940    1013387.20
## LF       2.4591587     323412.40
## M.F      2.0999588     298567.76
## Pop      1.5414657     395669.57
## NW       9.1083199     593057.74
## U1       0.1111776     180111.33
## U2       2.2600855     223692.25
## Wealth   5.4302213     523419.73
## Ineq     1.5278646     219673.19
## Prob     7.1108972     667630.66
## Time     1.0213302     222736.37
```

```
varImpPlot(random_forest)
```



The top 3 predictors are Po1, Po2 and NW. Almost the same ones that the regression tree used. One benefit of this model is that the over fitting is removed, so there is no need to do cv.

The resulting R^2 is 0.4382399 and the RSE/mean can be interpreted as 32% errror rate of the model (The less is better). The quality of this model is worse than the regression tree because it is not over-fitted.

Question 10.2

A logistic regression could be used on Tesla cars, so they can measure the probability of crashing to an object. The predictors that could be used are

1. The speed of the car (mph).
2. The distance of the object (mi).
3. The speed of the object (mph).
4. The weather condition (temp).
5. The condition of the tires (miles used).

Question 10.3

```
library(caTools)

germancredit <- read.table("~/Documents/R/GeorgiaTech/AdvancedRegression/germancredit.txt", quote="\"",

# convert response variable to 0 and 1
germancredit$V21[germancredit$V21==1]<-0
germancredit$V21[germancredit$V21==2]<-1

# split data 70% training and 30% test
sample = sample.split(germancredit, SplitRatio = 0.70)
train = subset(germancredit, sample == TRUE)
test  = subset(germancredit, sample == FALSE)

model <- glm(V21 ~.,family=binomial(link = "logit"),data=train)
# AIC = 680
summary(model)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8718  -0.6513  -0.3207   0.6264   2.8332
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.366e+00  1.316e+00   1.038 0.299327
## V1A12       -3.993e-01  2.903e-01  -1.375 0.168979
## V1A13       -5.077e-01  4.442e-01  -1.143 0.253114
## V1A14       -1.665e+00  2.986e-01  -5.575 2.47e-08 ***
## V2           3.564e-02  1.194e-02   2.984 0.002843 **
## V3A31       -4.141e-01  7.034e-01  -0.589 0.556082
## V3A32       -1.209e+00  5.698e-01  -2.122 0.033838 *
## V3A33       -1.493e+00  6.174e-01  -2.418 0.015608 *
```

```
## V3A34       -2.310e+00  5.837e-01  -3.958 7.57e-05 ***
## V4A41       -1.804e+00  4.680e-01  -3.854 0.000116 ***
## V4A410      -2.146e+00  1.411e+00  -1.521 0.128235
## V4A42       -9.069e-01  3.436e-01  -2.639 0.008308 **
## V4A43       -1.198e+00  3.256e-01  -3.680 0.000233 ***
## V4A44        8.008e-01  1.147e+00   0.698 0.485192
## V4A45       -9.730e-02  6.098e-01  -0.160 0.873225
## V4A46       -5.913e-02  4.940e-01  -0.120 0.904721
## V4A48       -1.519e+01  4.853e+02  -0.031 0.975026
## V4A49       -9.472e-01  4.293e-01  -2.206 0.027361 *
## V5           1.738e-04  5.866e-05   2.963 0.003045 **
## V6A62       -4.230e-01  3.564e-01  -1.187 0.235249
## V6A63       -1.265e-03  5.001e-01  -0.003 0.997982
## V6A64       -8.810e-01  6.501e-01  -1.355 0.175348
## V6A65       -9.353e-01  3.342e-01  -2.799 0.005128 **
## V7A72       -1.582e-01  5.173e-01  -0.306 0.759690
## V7A73       -6.408e-01  4.986e-01  -1.285 0.198685
## V7A74       -1.289e+00  5.504e-01  -2.342 0.019176 *
## V7A75       -4.312e-01  4.973e-01  -0.867 0.385877
## V8           4.145e-01  1.120e-01   3.700 0.000216 ***
## V9A92        4.003e-02  4.902e-01   0.082 0.934921
## V9A93       -7.883e-01  4.955e-01  -1.591 0.111596
## V9A94       -5.538e-01  5.889e-01  -0.940 0.347028
## V10A102      8.147e-01  5.183e-01   1.572 0.115995
## V10A103     -8.400e-01  5.308e-01  -1.582 0.113550
## V11          6.816e-02  1.116e-01   0.611 0.541473
## V12A122      7.844e-04  3.353e-01   0.002 0.998133
## V12A123     -1.063e-01  2.985e-01  -0.356 0.721680
## V12A124      3.754e-01  6.031e-01   0.622 0.533667
## V13         -1.844e-02  1.174e-02  -1.571 0.116234
## V14A142     -8.537e-02  4.882e-01  -0.175 0.861188
## V14A143     -8.251e-01  2.978e-01  -2.771 0.005590 **
## V15A152     -6.844e-01  3.082e-01  -2.220 0.026403 *
## V15A153     -8.163e-01  6.567e-01  -1.243 0.213848
## V16          1.131e-01  2.311e-01   0.489 0.624513
## V17A172      4.825e-01  7.945e-01   0.607 0.543657
## V17A173      3.038e-01  7.587e-01   0.400 0.688871
## V17A174      2.338e-01  7.607e-01   0.307 0.758566
## V18          4.103e-01  3.139e-01   1.307 0.191180
## V19A192     -2.076e-01  2.595e-01  -0.800 0.423694
## V20A202     -1.076e+00  7.167e-01  -1.501 0.133462
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 803.18  on 667  degrees of freedom
## Residual deviance: 559.54  on 619  degrees of freedom
## AIC: 657.54
##
## Number of Fisher Scoring iterations: 14
```

```
# before removing the predictors with higher p-value than 5%, first
# we need to categorize the columns with categorical values.
```

```r
# now remove predictors with p-values above 5%


model <- glm(V21~V1+V2+V3+V4+V5+V6+V8+V9+V10+V14+V20,family=binomial(link = "logit"),data=train)
# AIC 673 ç8lower better)
summary(model)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10 +
##     V14 + V20, family = binomial(link = "logit"), data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7707  -0.6804  -0.3612   0.6533   2.8317
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  9.126e-01  8.181e-01   1.116 0.264619
## V1A12       -4.586e-01  2.727e-01  -1.682 0.092648 .
## V1A13       -6.478e-01  4.347e-01  -1.490 0.136144
## V1A14       -1.739e+00  2.845e-01  -6.113 9.77e-10 ***
## V2           3.153e-02  1.121e-02   2.813 0.004907 **
## V3A31       -5.469e-01  6.655e-01  -0.822 0.411162
## V3A32       -1.319e+00  5.368e-01  -2.457 0.014015 *
## V3A33       -1.519e+00  6.006e-01  -2.529 0.011426 *
## V3A34       -2.371e+00  5.644e-01  -4.200 2.67e-05 ***
## V4A41       -1.561e+00  4.338e-01  -3.598 0.000321 ***
## V4A410      -2.101e+00  1.386e+00  -1.516 0.129595
## V4A42       -7.479e-01  3.212e-01  -2.328 0.019908 *
## V4A43       -1.213e+00  3.149e-01  -3.853 0.000117 ***
## V4A44        5.151e-01  1.014e+00   0.508 0.611261
## V4A45       -2.030e-02  5.890e-01  -0.034 0.972508
## V4A46        8.415e-02  4.679e-01   0.180 0.857264
## V4A48       -1.558e+01  4.794e+02  -0.032 0.974076
## V4A49       -9.568e-01  4.098e-01  -2.335 0.019560 *
## V5           1.493e-04  5.241e-05   2.849 0.004389 **
## V6A62       -3.940e-01  3.424e-01  -1.150 0.249956
## V6A63       -1.229e-01  4.684e-01  -0.262 0.792959
## V6A64       -1.029e+00  6.112e-01  -1.683 0.092382 .
## V6A65       -9.678e-01  3.202e-01  -3.023 0.002506 **
## V8           3.709e-01  1.068e-01   3.474 0.000512 ***
## V9A92        2.802e-01  4.630e-01   0.605 0.545062
## V9A93       -6.863e-01  4.664e-01  -1.471 0.141212
## V9A94       -2.777e-01  5.583e-01  -0.497 0.618902
## V10A102      8.418e-01  5.071e-01   1.660 0.096943 .
## V10A103     -8.070e-01  5.171e-01  -1.561 0.118640
## V14A142     -8.593e-02  4.715e-01  -0.182 0.855370
## V14A143     -7.930e-01  2.864e-01  -2.769 0.005626 **
## V20A202     -1.018e+00  7.006e-01  -1.453 0.146175
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 803.18  on 667  degrees of freedom
## Residual deviance: 585.25  on 636  degrees of freedom
## AIC: 649.25
##
## Number of Fisher Scoring iterations: 14
```

```r
predicted_y<-predict(model,test,type = "response")
predicted_y
```

```
##            1            7           13           14           15
## 3.657325e-02 9.950023e-02 1.668725e-01 3.733570e-01 6.236716e-01
##           18           21           22           28           34
## 7.675886e-01 6.975560e-02 1.305916e-01 1.847217e-01 1.045266e-02
##           35           36           39           42           43
## 6.365341e-01 2.693102e-01 3.147051e-01 2.091144e-01 4.184936e-01
##           49           55           56           57           60
## 5.316833e-02 7.684752e-01 3.755981e-02 8.020662e-02 8.418285e-01
##           63           64           70           76           77
## 7.167604e-01 9.188258e-01 1.343208e-01 8.254752e-02 5.747028e-01
##           78           81           84           85           91
## 1.238151e-01 5.600772e-02 2.100944e-01 2.099405e-01 1.915893e-02
##           97           98           99          102          105
## 8.081786e-02 2.223203e-01 1.622476e-01 3.561965e-01 1.393067e-02
##          106          112          118          119          120
## 2.019771e-01 6.300527e-01 7.674179e-02 3.119334e-01 2.712586e-01
##          123          126          127          133          139
## 6.801770e-02 3.189560e-01 3.265770e-01 1.234068e-01 3.815417e-02
##          140          141          144          147          148
## 2.011171e-01 4.170189e-02 2.775877e-01 2.171367e-01 1.052548e-01
##          154          160          161          162          165
## 8.611920e-02 2.437371e-03 1.824187e-02 1.822315e-01 2.702931e-01
##          168          169          175          181          182
## 2.519674e-01 2.117225e-01 4.780501e-01 2.386108e-01 6.068980e-01
##          183          186          189          190          196
## 5.769908e-01 2.342477e-02 2.456951e-01 8.036326e-01 2.505567e-01
##          202          203          204          207          210
## 6.363274e-01 1.136861e-01 2.885208e-07 3.649359e-02 3.949857e-03
##          211          217          223          224          225
## 3.191039e-03 8.042975e-01 1.112946e-01 4.459657e-02 5.192968e-02
##          228          231          232          238          244
## 5.993943e-01 3.776889e-01 8.723913e-02 5.849922e-01 4.747791e-02
##          245          246          249          252          253
## 2.477778e-07 1.040244e-01 1.831338e-01 2.262745e-01 7.955931e-01
##          259          265          266          267          270
## 3.916171e-02 1.658120e-02 1.852473e-01 4.560319e-02 3.178640e-02
##          273          274          280          286          287
## 9.575462e-01 4.741530e-01 6.251014e-02 9.265353e-01 5.660531e-01
##          288          291          294          295          301
## 1.608806e-01 1.964135e-02 1.649524e-01 2.717763e-01 1.194192e-01
##          307          308          309          312          315
## 1.116677e-01 2.471646e-01 3.598761e-01 2.696770e-01 2.189759e-02
##          316          322          328          329          330
## 8.765817e-01 5.292395e-01 2.281640e-01 3.869617e-01 1.511934e-01
##          333          336          337          343          349
```

```
## 9.376158e-01 1.263464e-01 1.264496e-01 1.354203e-01 1.267413e-02
##          350          351          354          357          358
## 8.292460e-02 2.134933e-01 7.325523e-01 7.254937e-03 1.476487e-01
##          364          370          371          372          375
## 4.147159e-02 4.089891e-01 1.803433e-01 3.545564e-02 9.480587e-01
##          378          379          385          391          392
## 1.049373e-02 9.177726e-01 5.467565e-02 1.631315e-01 9.464931e-02
##          393          396          399          400          406
## 7.806576e-01 7.054234e-01 3.260569e-01 2.973425e-02 1.522575e-01
##          412          413          414          417          420
## 4.737493e-02 6.168651e-02 4.313665e-02 5.894245e-01 4.653823e-01
##          421          427          433          434          435
## 1.195816e-01 4.253988e-02 1.502265e-01 1.880323e-01 2.853696e-01
##          438          441          442          448          454
## 9.062557e-02 1.852160e-01 6.310775e-01 4.664360e-02 7.266406e-02
##          455          456          459          462          463
## 7.638655e-01 1.709840e-01 7.892042e-01 3.979978e-01 4.541279e-01
##          469          475          476          477          480
## 1.651712e-01 1.230687e-01 7.418054e-01 9.872209e-02 1.812564e-01
##          483          484          490          496          497
## 5.683014e-01 5.186498e-02 4.854513e-02 2.232515e-01 7.897532e-01
##          498          501          504          505          511
## 4.986392e-02 8.566398e-01 1.432764e-01 8.178109e-01 5.224471e-01
##          517          518          519          522          525
## 5.618849e-02 1.799189e-01 3.104603e-01 4.141415e-01 2.349301e-01
##          526          532          538          539          540
## 2.397646e-01 4.590731e-01 2.773288e-01 9.553728e-01 4.466861e-01
##          543          546          547          553          559
## 4.443570e-01 5.875329e-01 2.372663e-01 3.616950e-01 7.910808e-01
##          560          561          564          567          568
## 7.265622e-02 2.812657e-01 5.109430e-01 5.313933e-01 6.463775e-03
##          574          580          581          582          585
## 5.501077e-01 7.437845e-08 1.719402e-01 1.998409e-01 1.906803e-01
##          588          589          595          601          602
## 2.000676e-01 7.809327e-01 2.627254e-01 7.879571e-02 4.449611e-01
##          603          606          609          610          616
## 9.419260e-01 7.998603e-01 7.722151e-02 3.880414e-02 8.722108e-01
##          622          623          624          627          630
## 1.218750e-01 3.802412e-01 6.159785e-01 1.156454e-01 3.620554e-02
##          631          637          643          644          645
## 4.781388e-01 1.752615e-01 1.599485e-01 2.249776e-02 1.961935e-01
##          648          651          652          658          664
## 1.893653e-01 9.098718e-01 2.802937e-01 3.599641e-01 3.196225e-01
##          665          666          669          672          673
## 1.904286e-01 1.235322e-01 5.296275e-01 9.610552e-02 6.356867e-01
##          679          685          686          687          690
## 5.720973e-01 2.854913e-01 4.396875e-01 2.261450e-02 1.652390e-01
##          693          694          700          706          707
## 1.170558e-01 6.654655e-02 4.481589e-01 2.418580e-01 7.042503e-01
##          708          711          714          715          721
## 8.173880e-01 4.416052e-02 9.855528e-02 9.579444e-01 3.868119e-01
##          727          728          729          732          735
## 1.487445e-02 7.296251e-01 9.162697e-01 2.135353e-01 6.247583e-02
##          736          742          748          749          750
```

```
## 9.294497e-01 4.692796e-01 6.818421e-01 1.290916e-02 2.878884e-02
##          753          756          757          763          769
## 1.850131e-01 6.330685e-01 2.085193e-02 2.877470e-01 5.444660e-02
##          770          771          774          777          778
## 1.074295e-02 1.780939e-01 5.124230e-02 1.780099e-01 4.099137e-01
##          784          790          791          792          795
## 7.326691e-01 8.292536e-01 3.201297e-01 3.418052e-02 1.421008e-01
##          798          799          805          811          812
## 4.109280e-02 1.154031e-01 3.820893e-01 1.990854e-01 1.432133e-01
##          813          816          819          820          826
## 3.913570e-01 8.035657e-01 8.564604e-01 5.877403e-01 4.641304e-01
##          832          833          834          837          840
## 7.881373e-01 9.169475e-01 2.139404e-01 5.492881e-02 1.182216e-01
##          841          847          853          854          855
## 6.810690e-01 9.331964e-02 2.380690e-02 7.600685e-01 4.196372e-01
##          858          861          862          868          874
## 4.886378e-02 1.538474e-02 1.194037e-01 4.311350e-02 2.642808e-01
##          875          876          879          882          883
## 4.661817e-01 1.930801e-01 4.312615e-01 7.021022e-02 1.668006e-01
##          889          895          896          897          900
## 2.553700e-01 9.641349e-03 1.127687e-01 5.992706e-01 5.343895e-01
##          903          904          910          916          917
## 2.266646e-02 3.791038e-02 2.290168e-01 9.047041e-01 2.592199e-02
##          918          921          924          925          931
## 8.444276e-01 1.057026e-01 3.649070e-01 8.949623e-01 4.234948e-01
##          937          938          939          942          945
## 2.188142e-01 2.371864e-01 8.863983e-01 4.815597e-02 4.398419e-01
##          946          952          958          959          960
## 9.502933e-01 3.004190e-01 3.193783e-02 6.700083e-01 3.127434e-01
##          963          966          967          973          979
## 7.756235e-02 2.945767e-01 1.170140e-01 8.808439e-01 2.304931e-01
##          980          981          984          987          988
## 4.615624e-01 2.307694e-01 4.156255e-01 8.205684e-01 5.197422e-02
##          994         1000
## 6.402698e-01 1.056661e-01
```

```r
predicted_round <- round(predicted_y)

confusion_matrix <- as.matrix(table(predicted_round,test$V21))
confusion_matrix
```

```
##
## predicted_round   0   1
##               0 196  55
##               1  29  52
```

```r
# accuracy
accuracy <- (confusion_matrix[1,1] + confusion_matrix[2,2]) / sum(confusion_matrix)
accuracy
```

```
## [1] 0.746988
```

```r
# sensitivity
sensitivity <- (confusion_matrix[1,1]) / (confusion_matrix[1,1] + confusion_matrix[2,1])
sensitivity
```

```
## [1] 0.8711111
```

```r
# specificity
specificity <-  (confusion_matrix[2,2]) / (confusion_matrix[2,2] + confusion_matrix[2,1])
specificity
```

```
## [1] 0.6419753
```

Analysis 10.3

For this a logistic regression, there can be multiple aproaches. For example, the categorical predictors could be converted to 1 and 0 (although I didn't do it here). I first used all the predictors to construct the model and then I only used the predictors that had p-values under 5% threshold. Then I started to measure the quality of the model. Remember that here there is not a pure R^2 that will show the quality easily, but there are other methods to measure it. To have a general idea of the model, I confusion matrix can be constructed with the rounded values (this is because the response is 0 or 1) and the test set.

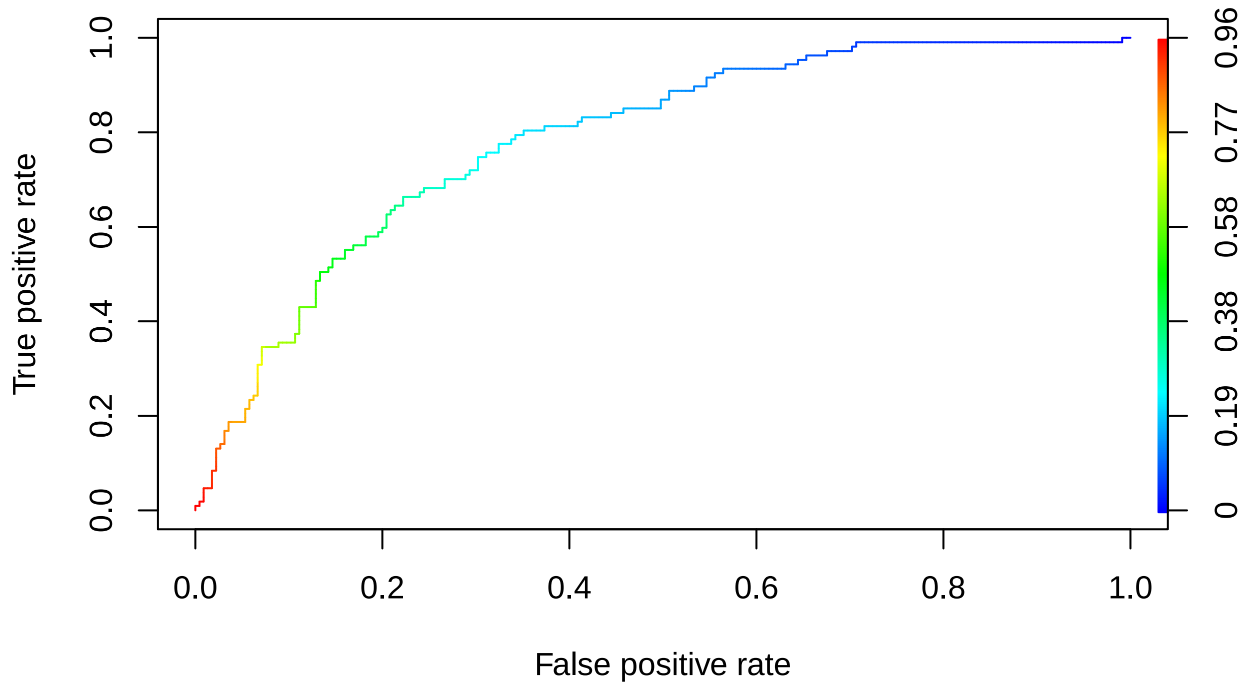
The accuracy is the measure by adding True Positives + True Negatives/sum(all_data). The reported accuracy is ´0.75´ The specificity can be measured by True Positive/TN + FP. The accuracy measures the fraction of category of members correctly classified. The reported specificity is `0.87` The sensitivity can be measure by TP/FN+TN. This measures the non category members correctly classified. The sensitivity is `0.64`;

Receiver Operating Characteristic(ROC) summarizes the model's performance by evaluating the trade offs between true positive rate (sensitivity) and false positive rate(1- specificity). For plotting ROC, it is advisable to assume $p > 0.5$ since we are more concerned about success rate. ROC summarizes the predictive power for all possible values of $p > 0.5$. The area under curve (AUC), referred to as index of accuracy(A) or concordance index, is a perfect performance metric for ROC curve. Higher the area under curve, better the prediction power of the model.

```r
library(ROCR)
```

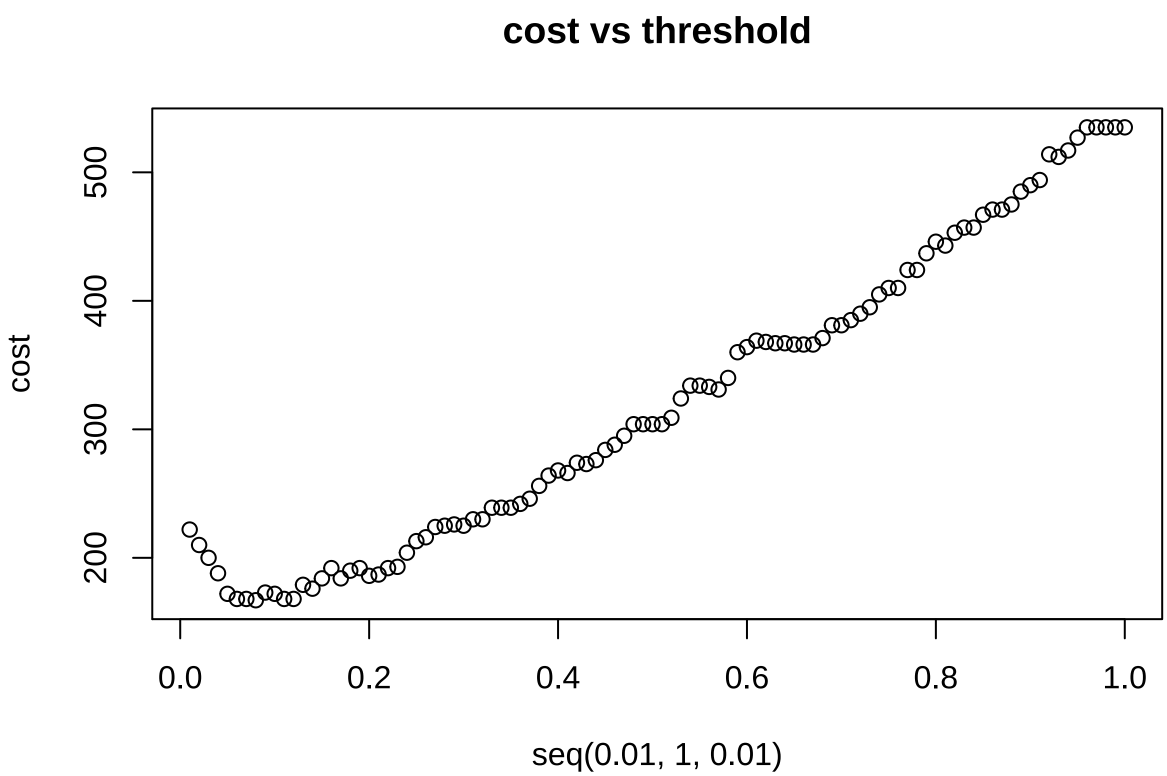
```
## Loading required package: gplots

##
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':
##
##     lowess
```

```r
ROCRpred <- prediction(predicted_y, test$V21)
ROCRperf <- performance(ROCRpred, 'tpr','fpr')
plot(ROCRperf, colorize = TRUE)
```



Now measure the cost

```
cost <- c()
# test threshold from 1% to 100%
for(i in seq(0.01,1,0.01))
{
  # threshold calculation
  y_round <- as.integer(predicted_y > i)
  # V21 is wheter the credit is given or not (0, 1)
  confusion <-as.matrix(table(y_round,test$V21))
  cost_fn <- 0
  cost_fp <- 0
  if (nrow(confusion) > 1) {
    cost_fn <- confusion[2,1]
  }
  if (ncol(confusion) > 1) {
    cost_fp <- confusion[1,2]
  }
  # save cost result
  # the cost of a false positive is 5 times worse than a false negative.
  cost <- c(cost, cost_fp*5 + cost_fn)
}

plot(seq(0.01,1,0.01),cost,main = "cost vs threshold")
```




```
# this will give the index
which.min(cost)
```

```
## [1] 8
```

```
# the threshold is the following one
which.min(cost)*0.01
```

```
## [1] 0.08
```

```
min(cost)
```

```
## [1] 167
```

Here it is necessary to iterate through all the possible thresholds from 1% to 100% to choose between 0 and 1. To calculate the cost of the misclassification it is only needed to use the false positive and false negative from the confusion matrix because the other ones measure the correct classification. So in each iteration a new confusion matrix is constructed using the threshold i in the loop. Then the FP and FN are used from the confusion matrix and the cost is calculated and ploted. The cost of a False Positive is 5 times higher than the cost of a False Negative because here we are dealing with credit scores, so it is better to not give credit/money to people with a "false" good score (FP).

The min threshold probability to have low costs/loss is expected to be 0.08 and the cost asociated with this threshold is 167. There seems to be a range of threshold that can be tolerable form 0.01 to 0.2. If for example a threshold of 0.6 is used the cost is the following one.

```
y_round <- as.integer(predicted_y > 0.6)
# V21 is wheter the credit is given or not (0, 1)
confusion <-as.matrix(table(y_round,test$V21))
cost_fn <- 0
cost_fp <- 0
if (nrow(confusion) > 1) {
  cost_fn <- confusion[2,1]
```

```
}
if (ncol(confusion) > 1) {
  cost_fp <- confusion[1,2]
}
# save cost result
# the cost of a false positive is 5 times worse than a false negative.
cost <-  cost_fp*5 + cost_fn
cost
```

```
## [1] 364
```

The cost would be 364, more than the double of the previous threshold. This could be millions of dollars of cost. So it can be concluded that it can be costly to choose a random or bad threshold.