HOMEWORK 7. ISYE 6501

Guillermo de la Hera Casado

October 5th, 2019

Fit a regression tree model to Crime data:

Let's first input the *Crime* data:

```
set.seed(101) # Set Seed so that same sample can be reproduced in future also
usCrime <- read.table("uscrime.txt", header = TRUE, sep = "\t")
```

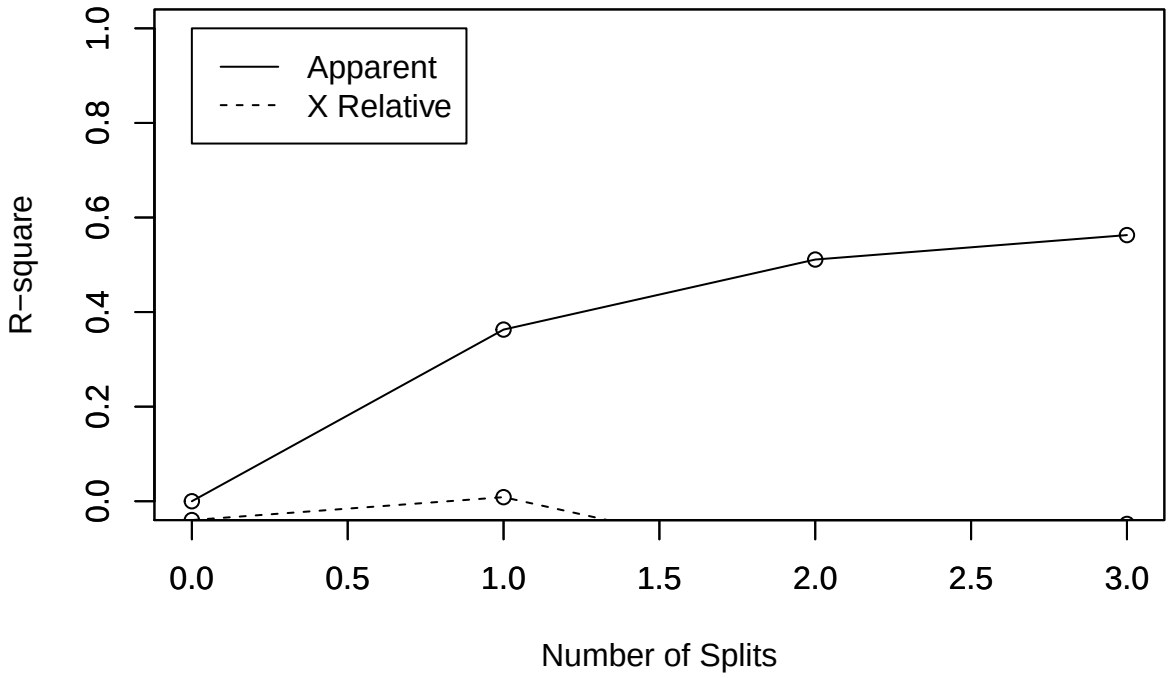
Now let's fit a regression tree model by using the *rpart* library. As this is a regression problem, we will set *method = anova*:

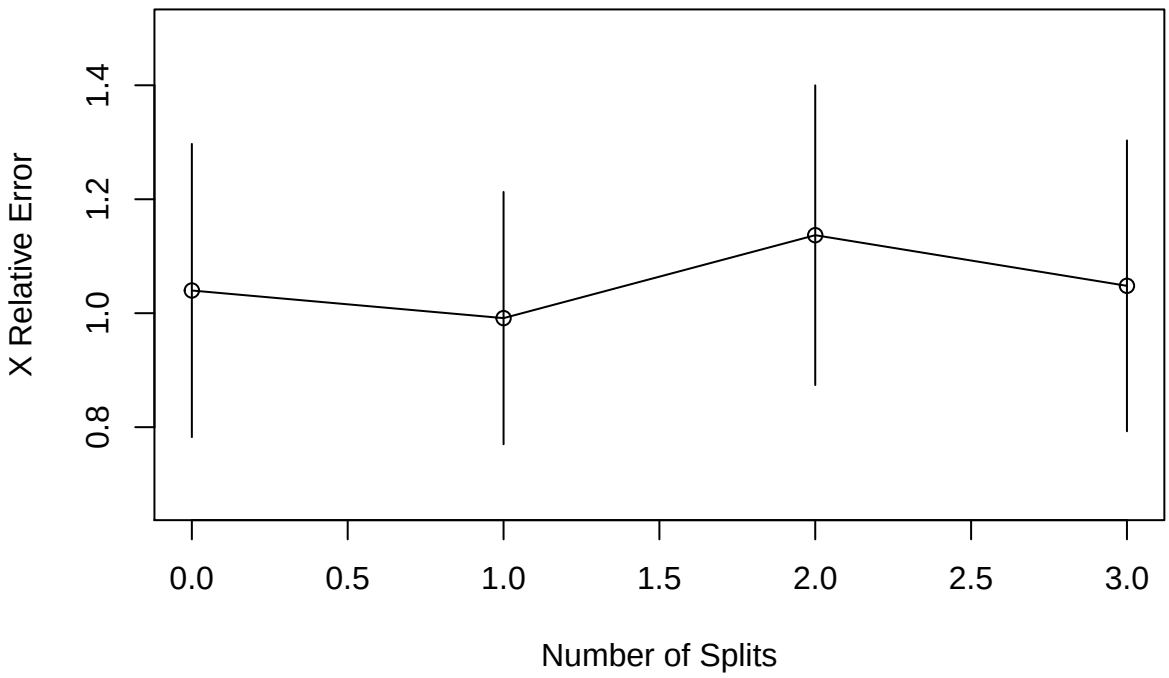
```
treeModel <- rpart(Crime~., data = usCrime, method = 'anova')
printcp(treeModel)
```

```
##
## Regression tree:
## rpart(formula = Crime ~ ., data = usCrime, method = "anova")
##
## 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 1.03982 0.25728
## 2 0.148143      1   0.63704 0.99136 0.22138
## 3 0.051732      2   0.48889 1.13689 0.26302
## 4 0.010000      3   0.43716 1.04803 0.25525
```

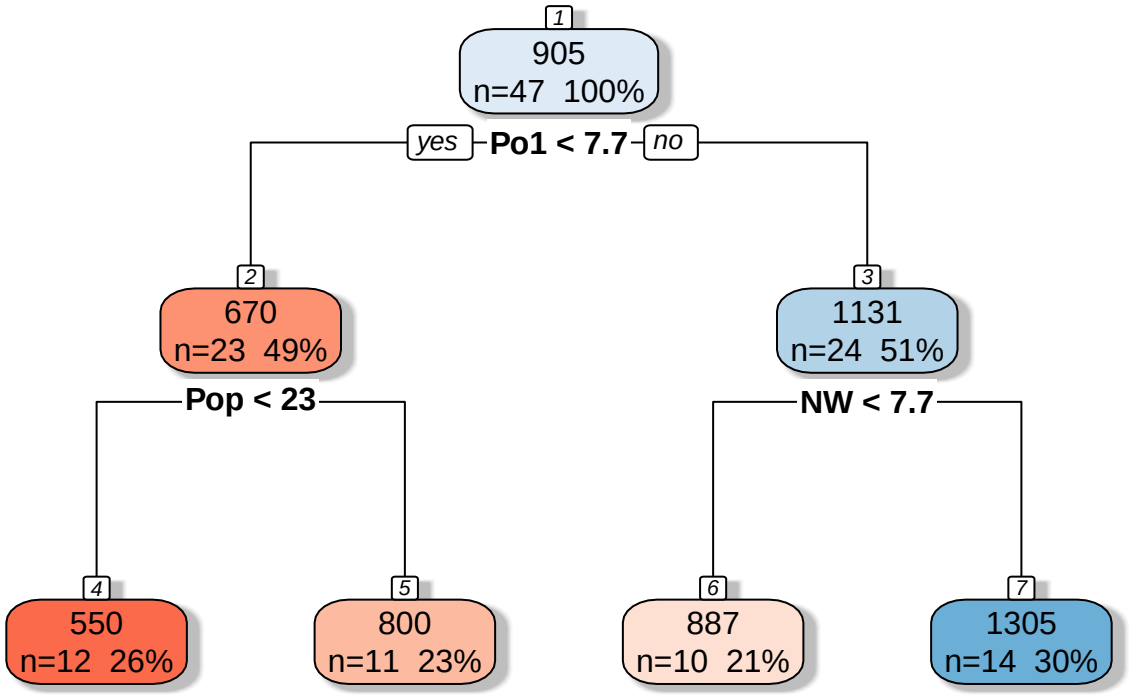
```
rsq.rpart(treeModel)
```

```
##
## Regression tree:
## rpart(formula = Crime ~ ., data = usCrime, method = "anova")
##
## 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 1.03982 0.25728
## 2 0.148143      1   0.63704 0.99136 0.22138
## 3 0.051732      2   0.48889 1.13689 0.26302
## 4 0.010000      3   0.43716 1.04803 0.25525
```


Apparent
X Relative
R−square
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Number of Splits




```
rpart.plot(treeModel, box.palette="RdBu", shadow.col = "gray", nn=TRUE, extra = 101)
```



As we can see in the summary, *xerror*, that is the cross validation error, gets to the lowest level for **CP = 0.148143**. **That's for 1 split.**

We also see from *rsq.rpart* method that *Apparent* R-squared keeps growing incrementally with number of splits proposed. However, *X Relative* parameter, that is the R-squared value from cross-validation gives the best value **with 1 split**.

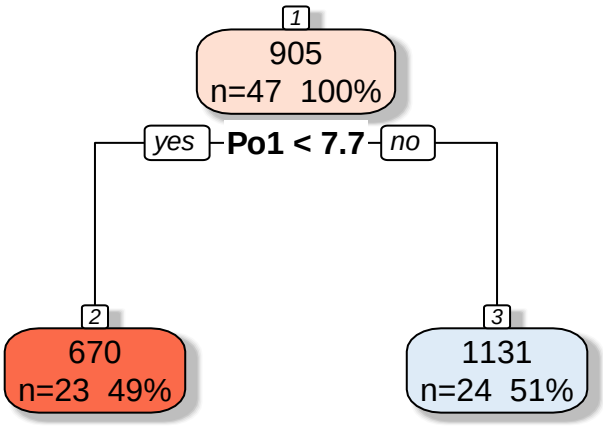
We can also see the the tree visualization, with details like data points per node, % of instances on each leaf and the business rule that is applied to the predictors for each split. The plot also shows the predicted regression value that apply per node e.g. those observations with Po1 < 7.7 and Pop<23 will have a predicted crime rate of: *550*

Let's now prune the tree, in order to reduce overfitting in the sample. For that, we will use the *prune* function. The complexity parameter will be the one that minimizes the xerror, pretty much the same value as I said above: **CP = 0.148143**

```
pruned <- prune(treeModel, cp = treeModel$cptable[which.min(treeModel$cptable[,"xerror"]),
                                                  "CP"])
```

The next step is to plot again the tree and understand what has happened during the pruning process:

```
rpart.plot(pruned, box.palette="RdBu", shadow.col = "gray", nn=TRUE, extra = 101)
```



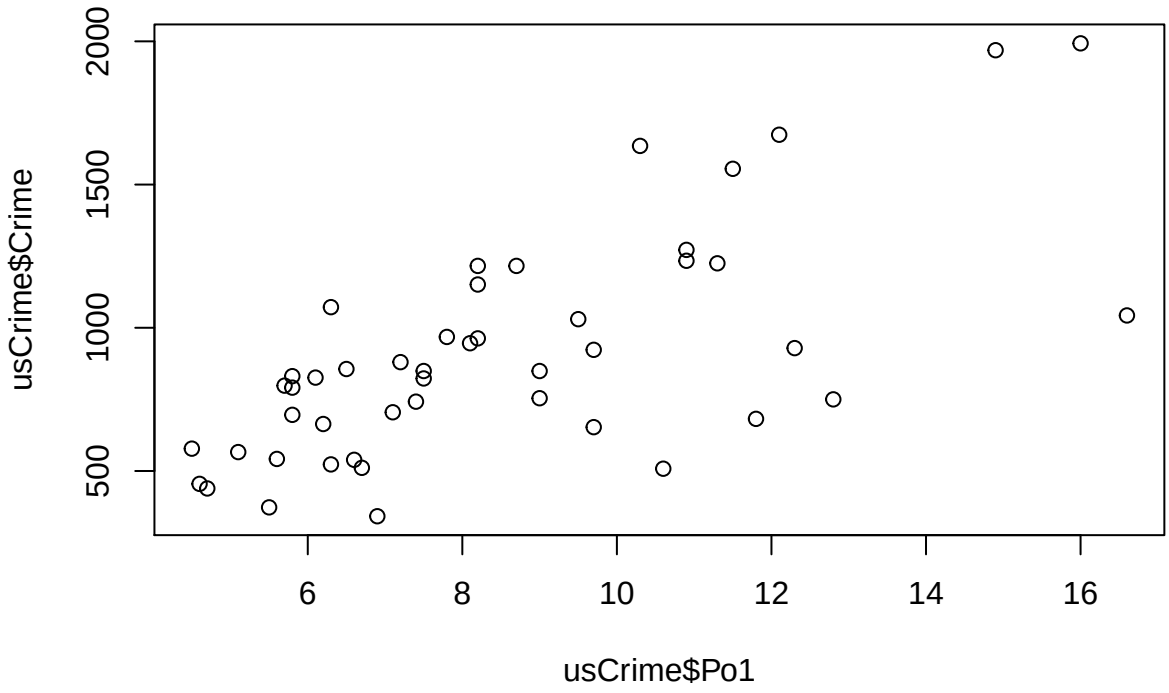
As we could expect, the pruned tree has only 1 split by the predictor *Po1*, getting more instances on each leaf and reducing the possibility of overfitting, that is quite nice having in mind the reduced amount of data points available in this set.

Let's interpret the model. We can say that:

- For instances where *Po1* (per capita expenditure on police protection in 1960) is < 7.7, the predicted crime rate will be: **670** (number of offenses per 100,000 population)
- For instances where *Po1* is > 7.7, the predicted crime rate will be: **1131**

So this takes us to a very interesting reading... there is a positive correlation between more expenditure on police protection and crime rate!. Let's verify that by plotting both variables:

```
plot(usCrime$Po1, usCrime$Crime)
```



As we can see, it's quite true... for higher values of Po1 we can see on average higher values of Crime rate. Let's take the first row of the dataset to verify that this works, by predicting it's crime rate with the pruned model:

```
obs <- usCrime[1,]
print(obs)
```

```
##       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 0.51  95  33 30.1 0.108 4.1   3940 26.1 0.084602
##      Time Crime
## 1 26.2011   791
```

```
predict <- predict(pruned, obs[,1:15])
print(round(predict,0))
```

```
##   1
## 670
```

In this observation, Po1 = 5.8. Therefore as per the pruned model, it's predicted value should be 670, because $5.8 < 7.7$. When we look at the predicted value, we can see that our understanding is correct.

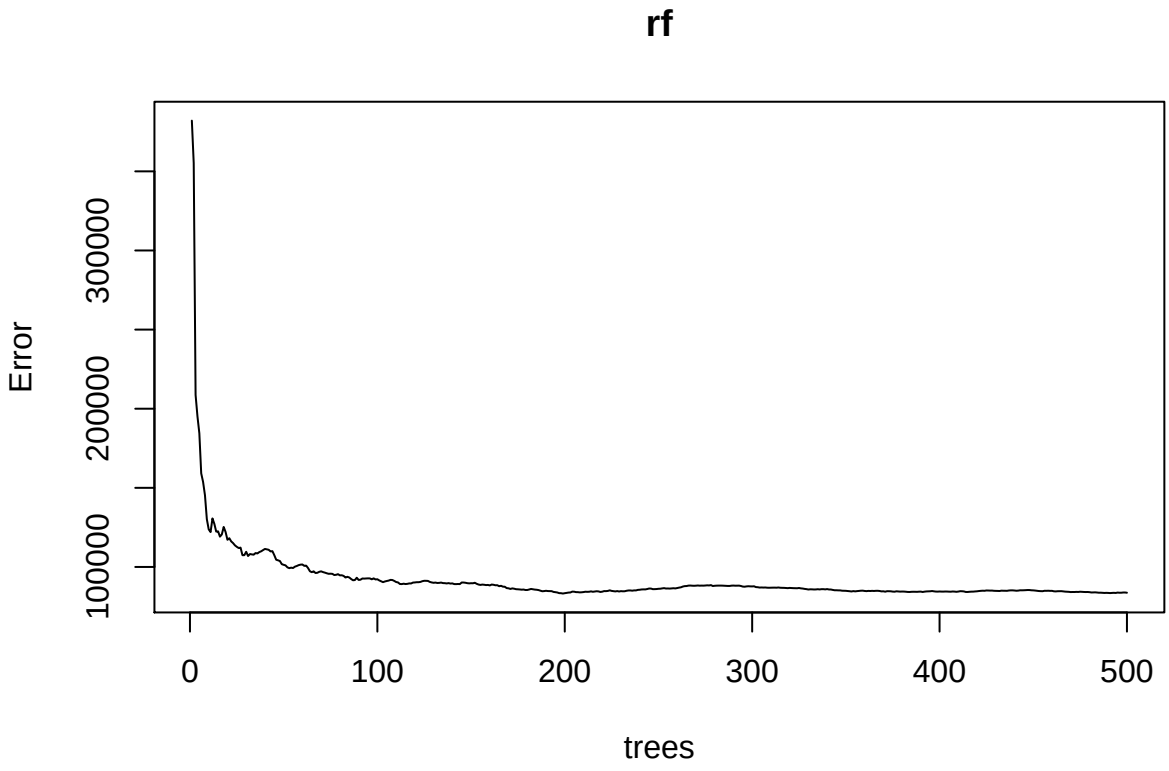
Fit a Random Forest model to Crime data:

Let's create a Random Forest model with default parameters:

```
rf <- randomForest(Crime ~ ., data = usCrime, importance = TRUE)
rf
```

```
##
## 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: 83704.32
##                     % Var explained: 42.83
```

```
plot(rf)
```



As we can see by default a forest of 500 trees will be created and 5 variables will be used per split. It also recognizes automatically that we want to have a regression model, likely because it identifies the response variable as continuous.

The outcome also states that the model explains 42.83% of the variance. Quite conveniently, random forest sets automatically some data points aside of the training set in order to validate the quality of the predictions. Those data points are called **out-of-bag** predictions. Unexplained variance would be to due true random behaviour or lack of fit.

We can also see a lot of error reduction by using at least 100 trees in the model. The incremental benefit of going above and beyond that number is not great for this specific dataset, therefore we assume we can use 100 trees as reference.

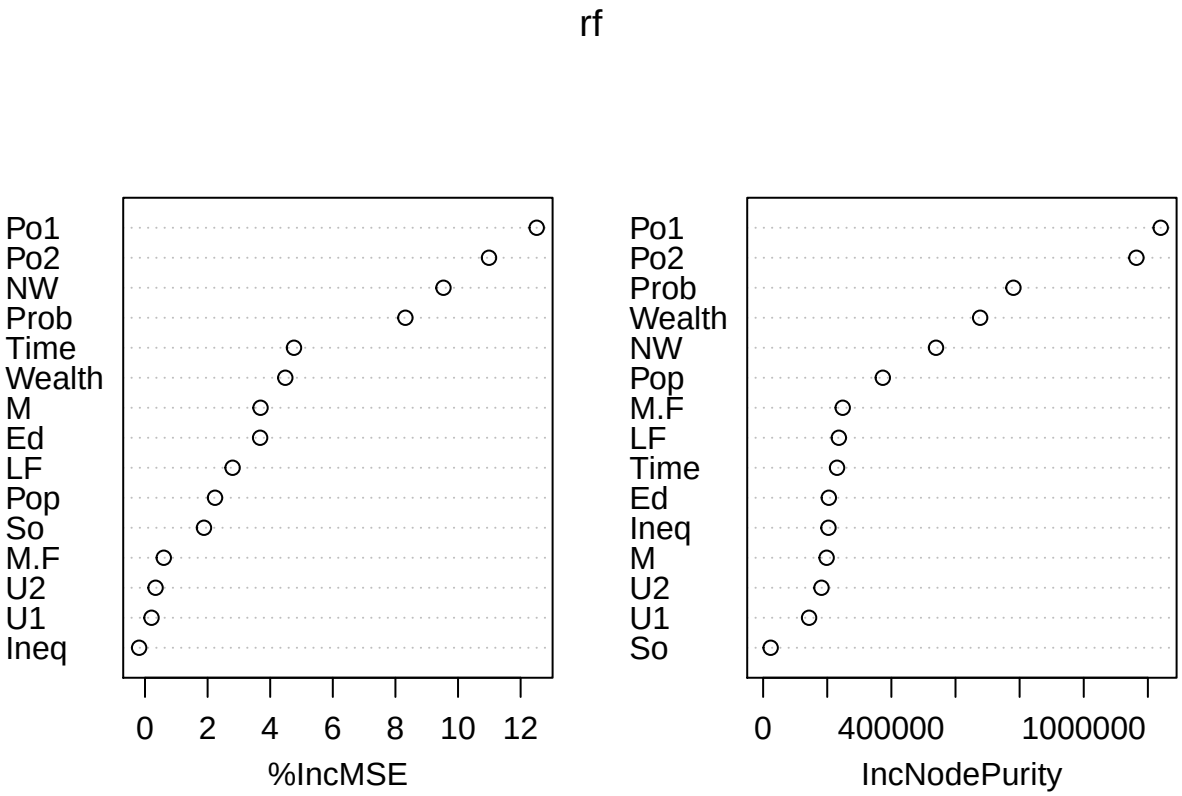
What about the importance of the predictors? Let's plot it. As we can see, Po1 would be the most relevant predictor for the model - meaning that if we drop it - we would see a huge % increase in MSE / reduction of accuracy. This speaks to the result that we had from the CART model, where the same variable was also

used to build the first split.

```
importance(rf)
```

```
##           %IncMSE IncNodePurity
## M       3.6903211     198056.94
## So      1.8836320      23511.33
## Ed      3.6777234     204887.87
## Po1    12.5130053    1239997.40
## Po2    10.9889560    1164234.52
## LF      2.7983524     236133.94
## M.F     0.5995331     248238.04
## Pop     2.2370864     373259.59
## NW      9.5343810     539227.44
## U1      0.2080082     143355.71
## U2      0.3378314     182013.18
## Wealth  4.4821268     676896.82
## Ineq   -0.1875176     203754.74
## Prob    8.3176820     780656.44
## Time    4.7593440     230556.37
```

```
varImpPlot(rf, sort=TRUE)
```



Would we get same or better outcomes with different values of *mtry*? Probably this should be answered through cross-validation. For that, we will use the very handy function **rfcv** using 5 folds and testing the following vaues of mtry: 15, 8, 4, and 1:

```
rfcv <- rfcv(trainx = usCrime[,1:15], trainy =  usCrime[,16], cv.fold = 5,
             mtry = function(p) max(1, floor(sqrt(p))))
rfcv$error.cv
```

```
##        15         8         4         1
##  94539.82  91253.36 114918.76 164418.78
```

As we can see from the results, the cross-validation error is minimized when we use 8 variables on each split. Let's build a final randomForest model with 100 trees and *mtry=8*. We will compare the % of variance explained vs the default model:

```
rf2 <- randomForest(Crime ~ ., data = usCrime, importance = TRUE, mtry=8, ntree=100)
rf2
```

```
##
## Call:
##  randomForest(formula = Crime ~ ., data = usCrime, importance = TRUE,      mtry = 8, ntree = 100)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 85188.75
##                     % Var explained: 41.81
```

From the outcome we can see that the performance of the new model doesn't seem to improve in terms of % variance explained vs the default model (41.81% in new model vs 42.83% in default). We will then continue with the default model.

The last question is, could we get the same performance with less predictors? In theory, if we remove the less important predictors from the model, the % of variance explained wouldn't be much worse... Let's try it. We will take out of the model: *Ineq*, *U1*,*U2* and *M.F* and see:

```
rfSimple <- randomForest(Crime ~ Po1 + Po2 + NW + Prob + Time + Wealth + M + Ed + LF + Pop + So,
                         data = usCrime, importance = TRUE)
rfSimple
```

```
##
## Call:
##  randomForest(formula = Crime ~ Po1 + Po2 + NW + Prob + Time +      Wealth + M + Ed + LF + Pop + So,
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 85298.43
##                     % Var explained: 41.74
```

Very interesting! Even taking out these 4 predictors, the % of variance explained keeps at the same level!. In other words, the following predictors don't seem to be relevant for the regression model:

- Ineq: % of families earning below half the median income
- U1: unemployment rate of urban males 14-24
- U2: unemployment rate of urban males: 35-39
- M.F: number of males per 100 females.

Describe a situation when Logistic Regression woud be appropiate and mention few predictors;

I work for a large Pay-TV company in Africa. It's a prepaid business, meaning that on a monthly basis each customer goes through the decision of paying for the package renewal or not.

The ability of predicting the risk of customers to disconnect on due date is priceless in this business, as dormancy is a key issue and drops the ARPU received from the subscribers.

Logistic Regression does help on this task, as we are interested in measuring risk as a "percentage of disconnection likelihood". Therefore we want to use probabilities, not binary solutions.

Regarding predictors, we currently use some as follows:

- Disconnection day of week
- Disconnection day of month
- Activity rate over the last 6 months
- Package type

Logistic Regression exercise

Find good prediction model for whether credit applicants are good credit risks or not

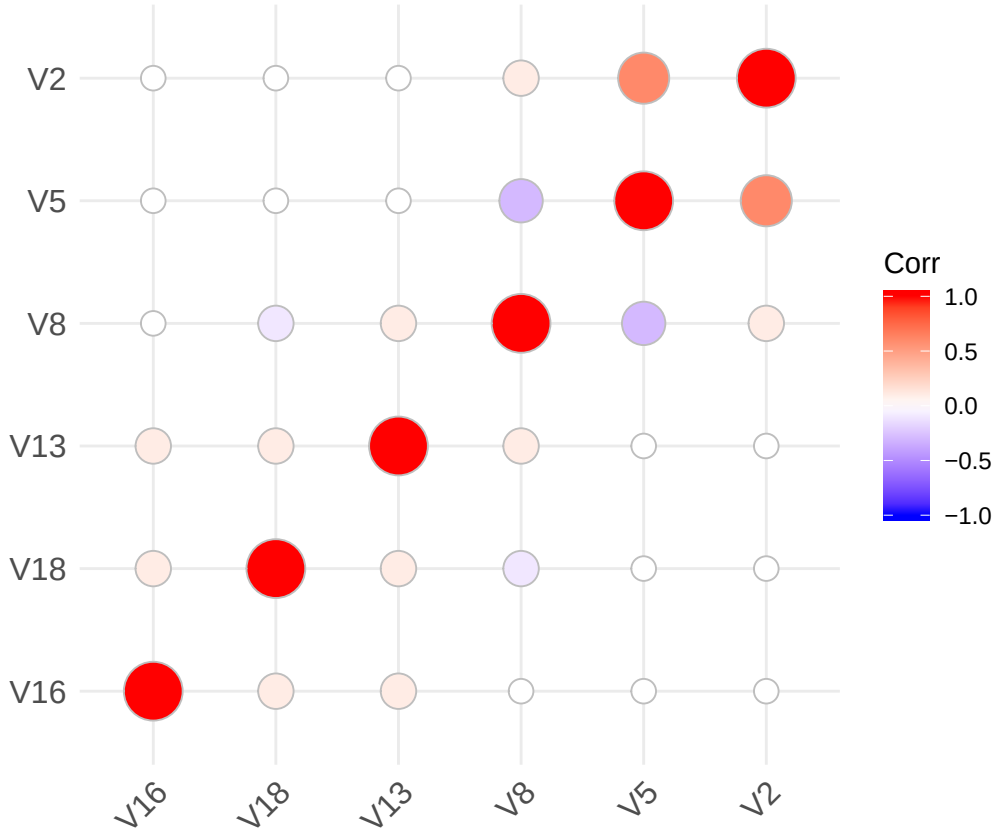
Let's load the data. We will find 1000 observations, 20 predictors and one response variable.

```
germanCredit <- read.table("germancredit.txt", header = FALSE)
str(germanCredit)
```

```
## 'data.frame':    1000 obs. of  21 variables:
##  $ V1 : Factor w/ 4 levels "A11","A12","A13",..: 1 2 4 1 1 4 4 2 4 2 ...
##  $ V2 : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ V3 : Factor w/ 5 levels "A30","A31","A32",..: 5 3 5 3 4 3 3 3 3 5 ...
##  $ V4 : Factor w/ 10 levels "A40","A41","A410",..: 5 5 8 4 1 8 4 2 5 1 ...
##  $ V5 : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...
##  $ V6 : Factor w/ 5 levels "A61","A62","A63",..: 5 1 1 1 1 5 3 1 4 1 ...
##  $ V7 : Factor w/ 5 levels "A71","A72","A73",..: 5 3 4 4 3 3 5 3 4 1 ...
##  $ V8 : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ V9 : Factor w/ 4 levels "A91","A92","A93",..: 3 2 3 3 3 3 3 3 1 4 ...
##  $ V10: Factor w/ 3 levels "A101","A102",..: 1 1 1 3 1 1 1 1 1 1 ...
##  $ V11: int  4 2 3 4 4 4 4 2 4 2 ...
##  $ V12: Factor w/ 4 levels "A121","A122",..: 1 1 1 2 4 4 2 3 1 3 ...
##  $ V13: int  67 22 49 45 53 35 53 35 61 28 ...
##  $ V14: Factor w/ 3 levels "A141","A142",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ V15: Factor w/ 3 levels "A151","A152",..: 2 2 2 3 3 3 2 1 2 2 ...
##  $ V16: int  2 1 1 1 2 1 1 1 1 2 ...
##  $ V17: Factor w/ 4 levels "A171","A172",..: 3 3 2 3 3 2 3 4 2 4 ...
##  $ V18: int  1 1 2 2 2 2 1 1 1 1 ...
##  $ V19: Factor w/ 2 levels "A191","A192": 2 1 1 1 1 2 1 2 1 1 ...
##  $ V20: Factor w/ 2 levels "A201","A202": 1 1 1 1 1 1 1 1 1 1 ...
##  $ V21: int  1 2 1 1 2 1 1 1 1 2 ...
```

As we are new to this dataset, it would be interesting to run *corrplot* and start undertanding the relationship between variables:

```
corr <- round(cor(germanCredit[, c("V16", "V18", "V13", "V8", "V5", "V2")]), 1)
ggcorrplot(corr, method = "circle")
```



There is a significant positive correlation between V2 (credit duration in months) and V5 (credit amount). There is also an interesting negative correlation between V5 (credit amount) and V8 (installment rate in percentage of disposable income)

As part of our data preparation efforts, the *binomial family* requires the dependent variable to be mapped to [0,1] values:

- As it is now, 1 represents good credit rate, 2 represents bad credit rate.
- Let's make it become 0 for good credit rate and 1 for bad credit rate

We will keep 25% of the data aside to test our predictions:

```
germanCredit$V21 <- apply(germanCredit, 1, function(x) {ifelse(x[21] == 1, 0, 1)})
sample <- sample.split(germanCredit$V21, SplitRatio = 0.75)
trainSet <- subset(germanCredit, sample == TRUE)
testSet <- subset(germanCredit, sample == FALSE)
```

Let's run the logistic regression model on the dataset in order to gain more insights on the relationship between predictors and response:

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

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = trainSet)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.3894  -0.7057  -0.3644   0.6959   2.6162
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  6.250e-01  1.251e+00   0.500  0.61731
## V1A12       -5.171e-01  2.548e-01  -2.029  0.04242 *
## V1A13       -6.951e-01  4.246e-01  -1.637  0.10159
## V1A14       -1.677e+00  2.670e-01  -6.280 3.38e-10 ***
## V2           2.555e-02  1.110e-02   2.301  0.02137 *
## V3A31        2.509e-01  6.378e-01   0.393  0.69402
## V3A32       -6.157e-01  5.109e-01  -1.205  0.22822
## V3A33       -6.695e-01  5.510e-01  -1.215  0.22438
## V3A34       -1.331e+00  5.327e-01  -2.498  0.01248 *
## V4A41       -1.838e+00  4.409e-01  -4.167 3.08e-05 ***
## V4A410      -1.319e+00  9.954e-01  -1.325  0.18507
## V4A42       -7.856e-01  2.963e-01  -2.651  0.00802 **
## V4A43       -1.212e+00  2.905e-01  -4.171 3.03e-05 ***
## V4A44       -1.764e-01  9.124e-01  -0.193  0.84672
## V4A45       -4.990e-01  6.452e-01  -0.773  0.43924
## V4A46       -7.633e-02  4.934e-01  -0.155  0.87705
## V4A48       -2.335e+00  1.334e+00  -1.751  0.07994 .
## V4A49       -1.098e+00  3.911e-01  -2.807  0.00500 **
## V5           1.572e-04  5.048e-05   3.115  0.00184 **
## V6A62       -6.819e-01  3.455e-01  -1.974  0.04840 *
## V6A63       -8.349e-02  4.396e-01  -0.190  0.84936
## V6A64       -1.606e+00  6.589e-01  -2.437  0.01480 *
## V6A65       -7.451e-01  3.033e-01  -2.456  0.01403 *
## V7A72        2.647e-01  4.766e-01   0.555  0.57865
## V7A73        7.365e-02  4.625e-01   0.159  0.87346
## V7A74       -4.408e-01  5.013e-01  -0.879  0.37921
## V7A75       -4.210e-02  4.690e-01  -0.090  0.92848
## V8           4.258e-01  1.044e-01   4.079 4.53e-05 ***
## V9A92       -2.508e-01  4.511e-01  -0.556  0.57820
## V9A93       -9.640e-01  4.473e-01  -2.155  0.03116 *
## V9A94       -1.928e-01  5.155e-01  -0.374  0.70848
## V10A102      4.600e-01  4.568e-01   1.007  0.31392
## V10A103     -7.392e-01  4.790e-01  -1.543  0.12279
## V11         -1.891e-02  1.015e-01  -0.186  0.85215
## V12A122      4.430e-01  2.907e-01   1.524  0.12747
## V12A123      1.967e-01  2.800e-01   0.703  0.48229
## V12A124      6.860e-01  5.094e-01   1.347  0.17805
## V13         -1.320e-02  1.095e-02  -1.205  0.22806
## V14A142     -2.690e-01  4.754e-01  -0.566  0.57146
## V14A143     -9.166e-01  2.813e-01  -3.259  0.00112 **
## V15A152     -4.172e-01  2.749e-01  -1.518  0.12910
## V15A153     -6.217e-01  5.634e-01  -1.103  0.26985
## V16          7.879e-02  2.227e-01   0.354  0.72343
## V17A172      6.401e-02  8.269e-01   0.077  0.93830
## V17A173      2.008e-01  7.986e-01   0.251  0.80152
## V17A174      8.391e-02  8.042e-01   0.104  0.91690
## V18          3.965e-01  2.976e-01   1.332  0.18277
## V19A192     -3.137e-01  2.349e-01  -1.335  0.18185
## V20A202     -1.817e+00  7.674e-01  -2.368  0.01789 *
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 916.30  on 749  degrees of freedom
## Residual deviance: 667.43  on 701  degrees of freedom
## AIC: 765.43
##
## Number of Fisher Scoring iterations: 5
```

The AIC of the model is: **765*. The below is the list of predictors that are statistically significant at 95% confidence level:

- V1A12: status of existing checking account - 0 <= ... < 200 DM
- V1A14: no checking account
- V2: Duration in month
- V3A34: critical account/ other credits existing (not at this bank)
- V4A41: purpose: car (used)
- V4A42: purpose: furniture/equipment
- V4A43: purpose: radio/television
- V4A49: purpose: business
- V5: Credit amount
- V6A62: savings account - 100 <= ... < 500 DM
- V6A64: savings account - >= 1000 DM
- V6A65: savings account - unknown/ no savings account
- V8: Installment rate in percentage of disposable income
- V9a93: male single
- V14a143: none installment plans
- V20a202: no foreign worker

While no exact equivalent to the R2 of linear regression exists, the McFadden R2 index can be used to assess the model fit.

Logistic regression models are fitted using the method of maximum likelihood - i.e. the parameter estimates are those values which maximize the likelihood of the data which have been observed. McFadden's R squared measure is defined as: *1 - (log(Lc) / log(Lnull))*, where Lc denotes the (maximized) likelihood value from the current fitted model, and Lnull denotes the corresponding value but for the null model - the model with only an intercept and no covariates.

```
pR2(creditlm)
```

```
##          llh      llhNull           G2     McFadden         r2ML
## -333.7132055 -458.1482265  248.8700420    0.2716043    0.2823883
##         r2CU
##    0.4003920
```

Let's now include only those predictors who were detected as significant, and make a new model based on them only. We will compare the improvements in AIC and McFadden R2.

```
creditlmSimple <- glm(V21~V1+V2+V3+V4+V5+V6+V8+V9+V14+V20,
                      data = trainSet, family=binomial(link="logit"))
summary(creditlmSimple)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V14 +
##     V20, family = binomial(link = "logit"), data = trainSet)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1893  -0.7038  -0.4017   0.7628   2.6852
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  8.243e-01  7.437e-01   1.108  0.26771
## V1A12       -5.638e-01  2.433e-01  -2.317  0.02049 *
## V1A13       -7.531e-01  4.061e-01  -1.854  0.06368 .
## V1A14       -1.708e+00  2.593e-01  -6.588 4.47e-11 ***
## V2           2.613e-02  1.055e-02   2.477  0.01325 *
## V3A31        6.984e-02  5.979e-01   0.117  0.90702
## V3A32       -8.132e-01  4.783e-01  -1.700  0.08907 .
## V3A33       -7.760e-01  5.317e-01  -1.459  0.14449
## V3A34       -1.563e+00  5.083e-01  -3.074  0.00211 **
## V4A41       -1.739e+00  4.227e-01  -4.114 3.89e-05 ***
## V4A410      -1.739e+00  9.780e-01  -1.778  0.07544 .
## V4A42       -6.289e-01  2.794e-01  -2.251  0.02438 *
## V4A43       -1.297e+00  2.781e-01  -4.662 3.13e-06 ***
## V4A44       -3.248e-01  9.095e-01  -0.357  0.72096
## V4A45       -5.757e-01  6.158e-01  -0.935  0.34983
## V4A46        9.101e-02  4.683e-01   0.194  0.84591
## V4A48       -2.289e+00  1.370e+00  -1.672  0.09457 .
## V4A49       -1.176e+00  3.775e-01  -3.116  0.00184 **
## V5           1.385e-04  4.592e-05   3.017  0.00256 **
## V6A62       -5.576e-01  3.303e-01  -1.688  0.09139 .
## V6A63       -1.001e-01  4.270e-01  -0.234  0.81471
## V6A64       -1.508e+00  6.088e-01  -2.477  0.01326 *
## V6A65       -7.737e-01  2.906e-01  -2.663  0.00775 **
## V8           3.883e-01  9.900e-02   3.922 8.79e-05 ***
## V9A92       -1.118e-01  4.275e-01  -0.261  0.79372
## V9A93       -8.896e-01  4.223e-01  -2.106  0.03517 *
## V9A94       -1.393e-01  4.945e-01  -0.282  0.77823
## V14A142     -1.987e-01  4.637e-01  -0.428  0.66834
## V14A143     -8.591e-01  2.727e-01  -3.151  0.00163 **
## V20A202     -1.626e+00  7.483e-01  -2.173  0.02978 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 916.30  on 749  degrees of freedom
## Residual deviance: 689.27  on 720  degrees of freedom
## AIC: 749.27
##
## Number of Fisher Scoring iterations: 5
```

```
pR2(creditlmSimple)
```

```
##          llh      llhNull           G2     McFadden         r2ML
## -344.6334083 -458.1482265  227.0296365    0.2477688    0.2611839
##         r2CU
##    0.3703266
```

Mc Fadden R squared metric decreases from 0.266 to 0.246. However, this can be biased as the formula

doesn't account for the benefits on complexity reduction like the adjusted R-Squared metric.

AIC decreases from 765 to 749. What's the likelihood of the default model to be better than the simplified model? As per the formula explained by Dr. Sokol, just only: e^ (749-765/2) = 0.03%. **Therefore we will stick to our simplified model**

Let's make the predictions for our test set based on the simplified model and compute the Area Under the Curve. We will set up *type = "response"* in order to get probabilities:

```
predicted <- predict(creditlmSimple, testSet, type = 'response')
print(head(predicted))
```

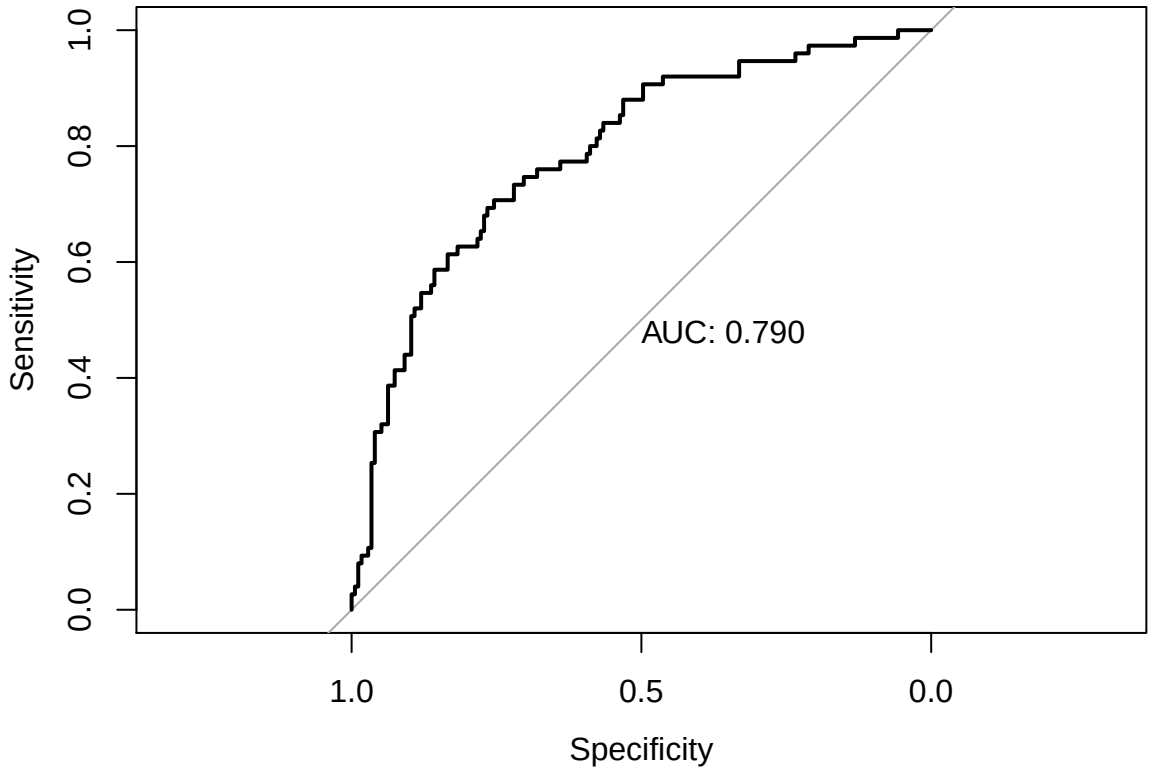
```
##          1          2         11         15         19         23
## 0.06382613 0.50873664 0.53348313 0.59939938 0.65933507 0.04096345
```

```
roc <- roc(testSet$V21, predicted)
```

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

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

```
plot(roc, print.auc=TRUE)
```



The model does a pretty good job with AUC = 0.79. **This means that 79% of the times it will provide higher risk of bad credit to a random customer with bad credit, vs a random customer with good credit**

Determine a good threshold probability

We know that incorrectly classifying a bad customer as good is 5 times worse than classifying a good customer as bad. This is very important as we know that False Negatives (predicting that a customer is good credit when is bad credit) is the parameter to reduce.

Therefore, let's use **sensitivity: TP/ (TP + FN) as our key metric**. It will guarantee that we maximize true positives (those who are bad credit being identified as bad credit) and also drop FN to the minimum (those who are bad credit being incorrectly identified as good credit).

As a side effect, probably this will cause some increase in False Positives (those who are good credit being incorrectly identified as bad) but they don'e penalize that much / the company can always treat FPs manually on a 1 to 1 basis to then discern if that'FP is correct or not. Let's explore the impact of modifying the threshold in the confusion matrix (caret package). Let's remember to define *positive=1*, so that the caret knows how to identify the positive instance:

Threshold 50%

```
predicted <- predict(creditlmSimple, testSet, type = "response")
class_prediction_50 <- ifelse(predicted> 0.50,1,0)
caret::confusionMatrix(table(class_prediction_50, testSet$V21), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## class_prediction_50   0   1
##                   0 156  37
##                   1  19  38
##
##                Accuracy : 0.776
##                  95% CI : (0.7192, 0.8261)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.004485
##
##                   Kappa : 0.4274
##
##  Mcnemar's Test P-Value : 0.023103
##
##             Sensitivity : 0.5067
##             Specificity : 0.8914
##          Pos Pred Value : 0.6667
##          Neg Pred Value : 0.8083
##              Prevalence : 0.3000
##          Detection Rate : 0.1520
##    Detection Prevalence : 0.2280
##       Balanced Accuracy : 0.6990
##
##        'Positive' Class : 1
##
```

As we can see, sensitivity is quite low, at: 0.507. If we assume the cost function, then the cost for our mistakes would be: FN [37] X 5 + FP [19] x 1 = 204.

Threshold 30%

```
predicted <- predict(creditlmSimple, testSet, type = "response")
class_prediction_30 <- ifelse(predicted> 0.3,1,0)
caret::confusionMatrix(table(class_prediction_30, testSet$V21), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## class_prediction_30   0   1
##                   0 129  22
##                   1  46  53
##
##                Accuracy : 0.728
##                  95% CI : (0.6683, 0.7822)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.185316
##
##                   Kappa : 0.4066
##
##  Mcnemar's Test P-Value : 0.005285
##
##             Sensitivity : 0.7067
##             Specificity : 0.7371
##          Pos Pred Value : 0.5354
##          Neg Pred Value : 0.8543
##              Prevalence : 0.3000
##          Detection Rate : 0.2120
##    Detection Prevalence : 0.3960
##       Balanced Accuracy : 0.7219
##
##        'Positive' Class : 1
##
```

Reducing the threshold results on higher sensitivity, at 0.706. The cost function for our mistakes would be: FN [22] X 5 + FP [46] x 1 = 156.

Threshold 10%

```
predicted <- predict(creditlmSimple, testSet, type = "response")
class_prediction_10 <- ifelse(predicted> 0.1,1,0)
caret::confusionMatrix(table(class_prediction_10, testSet$V21), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## class_prediction_10   0   1
##                   0  73   6
##                   1 102  69
##
##                Accuracy : 0.568
##                  95% CI : (0.5041, 0.6303)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2469
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9200
##             Specificity : 0.4171
##          Pos Pred Value : 0.4035
```

```
##          Neg Pred Value : 0.9241
##              Prevalence : 0.3000
##          Detection Rate : 0.2760
##    Detection Prevalence : 0.6840
##       Balanced Accuracy : 0.6686
##
##        'Positive' Class : 1
##
```

Reducing the threshold results on higher sensitivity, at 0.92. The cost function for our mistakes would be: FN [6] X 5 + FP [102] x 1 = 132.

Threshold 5%

```
predicted <- predict(creditlmSimple, testSet, type = "response")
class_prediction_5 <- ifelse(predicted> 0.05,1,0)
caret::confusionMatrix(table(class_prediction_5, testSet$V21), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##
## class_prediction_5   0   1
##                  0  40   3
##                  1 135  72
##
##                Accuracy : 0.448
##                  95% CI : (0.3853, 0.5119)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1255
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9600
##             Specificity : 0.2286
##          Pos Pred Value : 0.3478
##          Neg Pred Value : 0.9302
##              Prevalence : 0.3000
##          Detection Rate : 0.2880
##    Detection Prevalence : 0.8280
##       Balanced Accuracy : 0.5943
##
##        'Positive' Class : 1
##
```

Reducing the threshold results on higher sensitivity, at 0.96. The cost function for our mistakes would be: FN [3] X 5 + FP [135] x 1 = 150. Therefore we can see that making threshold $< 10\%$ doesn't bring better cost function results, as FP are just too high.

We will then agree that threshold for this exercise should be marked at 10%