WK7_HW_7

Anon Skywalker

10/5/2020

Question 10.1 a

For this question, we are tasked with creating a regression tree model. For this problem, I utilized the tree package in R. First, I created a test and train data set, using about 90% of the data for train and the remaining points for test. I chose this breakdown because of the small amount of data within the USCrime data set.

Next, I created a default model with all the predictor variables using tree(). By plotting this model, we can see its branches and leaves to get an idea of the unpruned breakdown. Surprisingly, not many predictors are used by the tree function by default. This poses a challenge in terms of whether or not to prune the tree, as too few of branches could create a oversimplified model that would not classify points well at all.

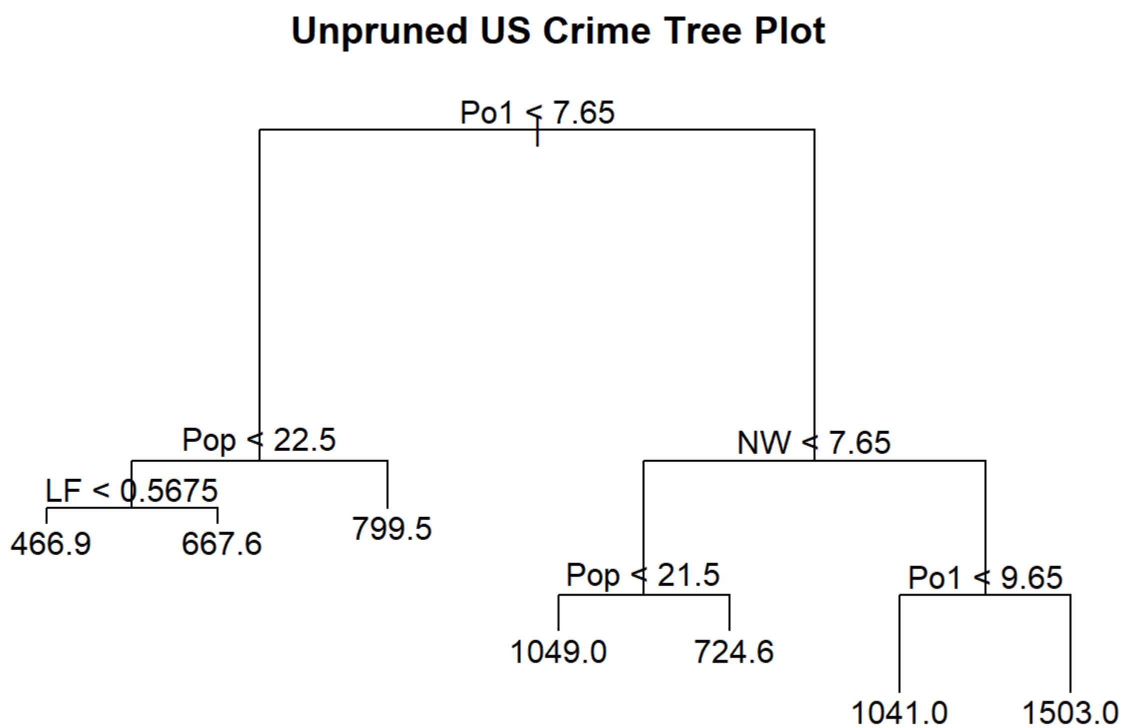
```
#data prep (create test and train)
set.seed(12345)

#selects 1/12 of the data points as a sample
sample.crime.data <- sample(1:nrow(crime.data), size = round(nrow(crime.data) / 12), replace = F
ALSE)
# 11/12th of the data
train.crime.data <- crime.data[-sample.crime.data, ]
# 1/12 of the data
test.crime.data <- crime.data[sample.crime.data, ]



#fit unpruned classification tree with all predictors
tree.crime <- tree(Crime~., data = crime.data)
summary(tree.crime)
```

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

```
#check out branch splitting
tree.crime$frame
```

```
##        var  n        dev      yval splits.cutleft splits.cutright
## 1      Po1 47 6880927.66  905.0851          <7.65           >7.65
## 2      Pop 23  779243.48  669.6087          <22.5           >22.5
## 4       LF 12  243811.00  550.5000        <0.5675         >0.5675
## 8   <leaf>  7   48518.86  466.8571                               
## 9   <leaf>  5   77757.20  667.6000                               
## 5   <leaf> 11  179470.73  799.5455                               
## 3       NW 24 3604162.50 1130.7500          <7.65           >7.65
## 6      Pop 10  557574.90  886.9000          <21.5           >21.5
## 12 <leaf>  5  146390.80 1049.2000                               
## 13 <leaf>  5  147771.20  724.6000                               
## 7      Po1 14 2027224.93 1304.9286          <9.65           >9.65
## 14 <leaf>  6  170828.00 1041.0000                               
## 15 <leaf>  8 1124984.88 1502.8750                               
```

```
plot(tree.crime)
  title("Unpruned US Crime Tree Plot")
  text(tree.crime)
```




```
#view splits
tree.crime
```

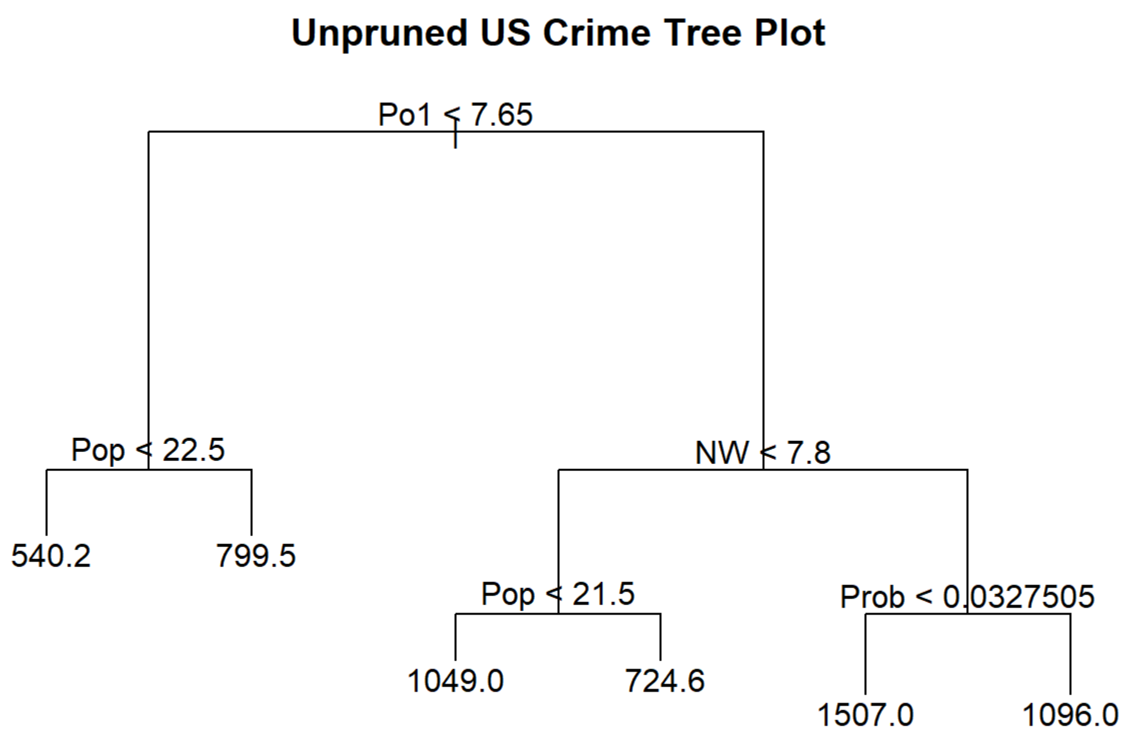
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 47 6881000  905.1
##    2) Po1 < 7.65 23  779200  669.6
##      4) Pop < 22.5 12  243800  550.5
##        8) LF < 0.5675 7   48520  466.9 *
##        9) LF > 0.5675 5   77760  667.6 *
##      5) Pop > 22.5 11  179500  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 *
```

The next step was to improve this model by using our train and test data sets. Using the train data and then using the CV.TREE() function in R, I was able to get a better idea of how to handle the pruning question. By plotting our cv model, we can more accurately assess the best number of nodes to use for our improved model. This came out to be 4 according to the plot.

```
#Now we will use train/test data to evaluate how good our tree model is:
set.seed(12345)
trained.tree.crime <- tree(Crime~., data = train.crime.data)
summary(trained.tree.crime)
```

```
##
## Regression tree:
## tree(formula = Crime ~ ., data = train.crime.data)
## Variables actually used in tree construction:
## [1] "Po1"  "Pop"  "NW"   "Prob"
## Number of terminal nodes:  6
## Residual mean deviance:  44700 = 1654000 / 37
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -464.400 -114.900   -1.182    0.000   92.030  461.600
```

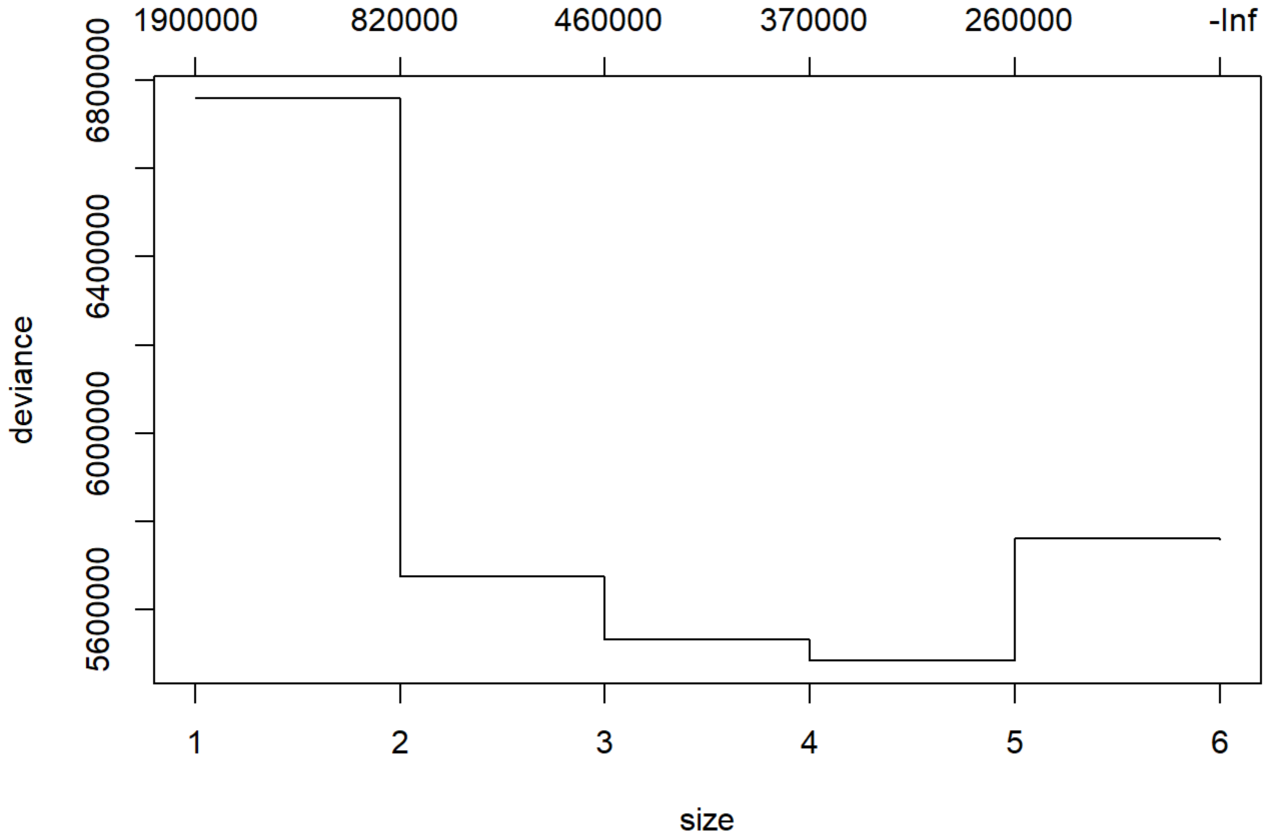
```
#used Po1, Pop, LF, NW



plot(trained.tree.crime)
  title("Unpruned US Crime Tree Plot")
  text(trained.tree.crime)
```



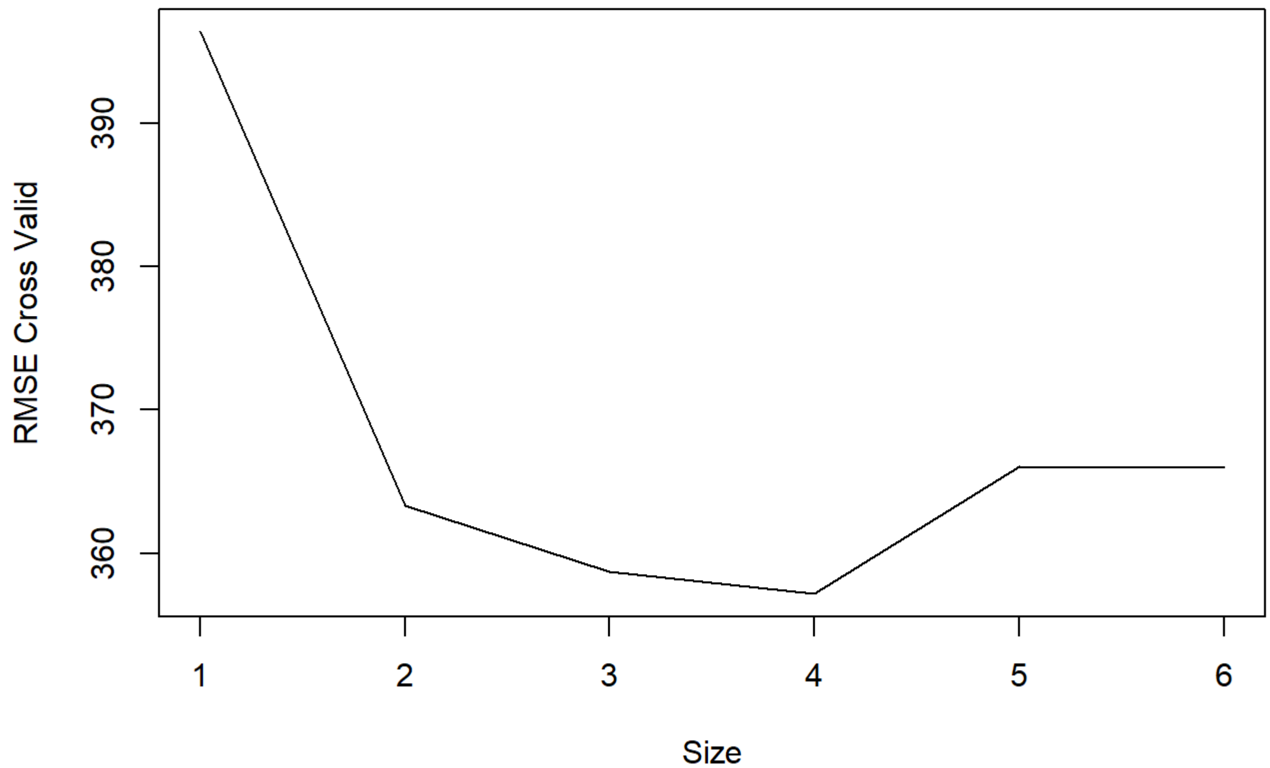

```
#perform cross validation - find best number of nodes
# plot indicates 4 has the least amount of error! 4 is a good prune point of nodes
# set seed for random number generator

cv.crime.tree <- cv.tree(trained.tree.crime)
summary(cv.crime.tree)
```

```
##        Length Class  Mode
## size   6      -none- numeric
## dev    6      -none- numeric
## k      6      -none- numeric
## method 1      -none- character
```

```
plot(cv.crime.tree)
```




```
plot(cv.crime.tree$size, sqrt(cv.crime.tree$dev / nrow(train.crime.data)), xlab = "Size", type =
"l" , ylab = "RMSE Cross Valid")
```

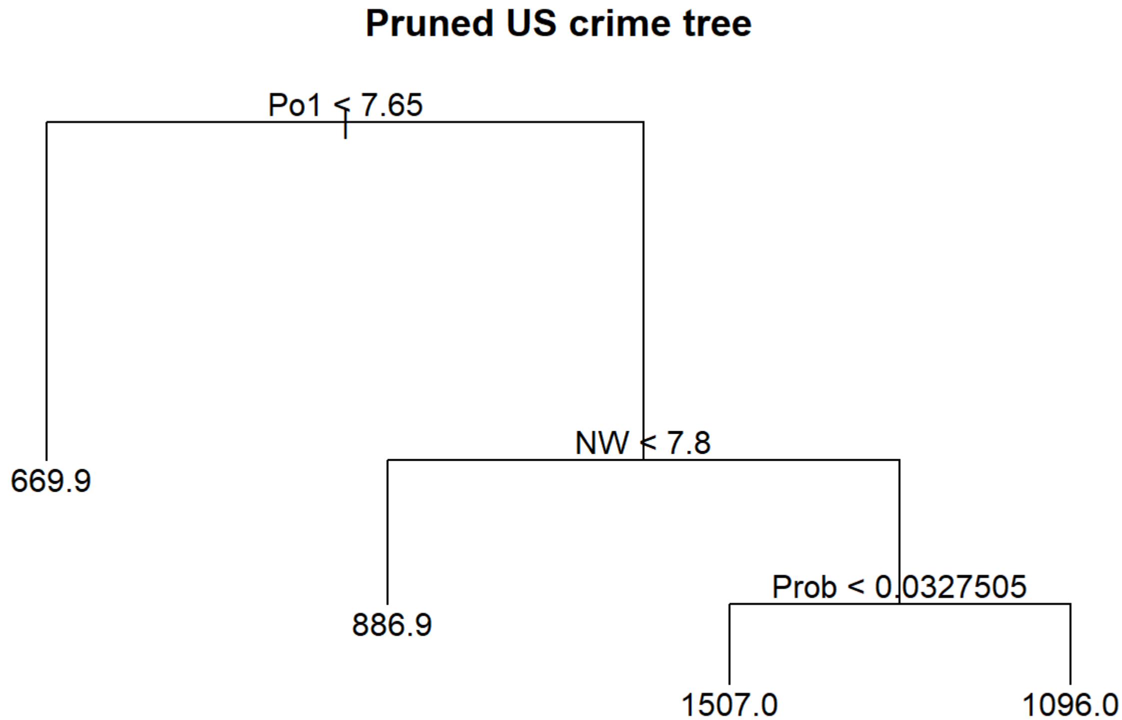


By

using the prune.tree() function and setting best = 4, we can set the pruned model to utilize only 4 nodes. To compare the two models, I did a manual calculation for RMSE (square root of the residuals variance). By using this metric, we can see that the original model did better than the pruned one! So, it looks like that will be our winner. Note: before setting the seed, I noticed several different outcomes. This makes sense as the cross validation uses a random number for the number of folds, and this often resulted in a different outcome when plotting where sometimes 5 or even 7 nodes looked like the lowest point of error.

The rmse for this model came out to be 485.54

```
#test against new data using the previously discovered 4 nodes fom CV
pruned.tree.crime <- prune.tree(trained.tree.crime,  best = 4)

#plot the pruned tree with 7 nodes
plot(pruned.tree.crime)
text(pruned.tree.crime)
title("Pruned US crime tree")
```




```
summary(pruned.tree.crime)
```

```
##
## Regression tree:
## snip.tree(tree = trained.tree.crime, nodes = c(6L, 2L))
## Variables actually used in tree construction:
## [1] "Po1"  "NW"   "Prob"
## Number of terminal nodes:  4
## Residual mean deviance:  58650 = 2287000 / 39
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -464.40 -163.00   35.14    0.00  154.60  461.60
```

```
#worse than not pruning at all!!!
prun.rmse <- sqrt(summary(pruned.tree.crime)$dev / nrow(train.crime.data))
#unpruned tree model wins the rmse test
rmse.tree <- sqrt(summary(trained.tree.crime)$dev / nrow(train.crime.data))
#simple function to use for predicting


#408.6 vs 368.7 for pruned means the pruned of 2 is a better fit!
unpruned.tree.pred <- predict(trained.tree.crime, newdata = test.crime.data)
sqrt(mean((unpruned.tree.pred - test.crime.data$Crime) ^ 2))
```

```
## [1] 485.5408
```

Question 10.1 b

While searching through https://topepo.github.io/caret/model-training-and-tuning.html#model-training-and-parameter-tuning (https://topepo.github.io/caret/model-training-and-tuning.html#model-training-and-parameter-tuning), I found an extremely helpful function called train() in R that allows for bootsrapping (random sampling with replacement) over 1000 samples. The train function then found the best RMSE provided the training data. This well fitted model was then used to predict against the test data and I calculated for its RMSE which came out to be 428.95! This beats our previous model by a wide margin, as lower RMSE indicates a better model fit!

```
set.seed(12345)
  fitted.crime.forest <- train(
    Crime~., data = train.crime.data, method = 'rf',
    trControl = trainControl(method = 'boot_all', number = 1000), metric = 'RMSE', importance =
TRUE)

summary(fitted.crime.forest)
```

```
##                 Length Class      Mode
## call              5    -none-     call
## type              1    -none-     character
## predicted        43    -none-     numeric
## mse             500    -none-     numeric
## rsq             500    -none-     numeric
## oob.times        43    -none-     numeric
## importance       30    -none-     numeric
## importanceSD     15    -none-     numeric
## localImportance   0    -none-     NULL
## proximity         0    -none-     NULL
## ntree             1    -none-     numeric
## mtry              1    -none-     numeric
## forest           11    -none-     list
## coefs             0    -none-     NULL
## y                43    -none-     numeric
## test              0    -none-     NULL
## inbag             0    -none-     NULL
## xNames           15    -none-     character
## problemType       1    -none-     character
## tuneValue         1    data.frame list
## obsLevels         1    -none-     logical
## param             1    -none-     list
```

```
best.forest <- fitted.crime.forest$finalModel
best.forest
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 82184.72
##                     % Var explained: 35.83
```

```
rf.predict <- predict(fitted.crime.forest, newdata = test.crime.data)

#RMSE:  428.9532
sqrt(mean((rf.predict - test.crime.data$Crime) ^ 2))
```

```
## [1] 428.9532
```

Question 10.2

As a data analyst in the medical field, one area of concern is HAIs or hospital acquired infections. To combat this, there are many procedures and resources in place to ensure that staff can wash their hands between patient encounters. This data is collected and scored per hospital and being an outlier in this score is very bad for business! To get the probability that the hospital I work at will meet the state mandated criteria for HAIs in one month from now, we could use predictors such as newly onboarded staff, contact positions used, number of hand washing stations, number of doctors, and number of re-admits for HAI related symptoms in order to predict our performance come next month!

Question 10.3

For question 10.3, We are asked to apply logistic regression to a new dataset “germancredit”. The details behind the dataset can be seen here: http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29 (http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29). After studying the dataset, I loaded in the data without headers and realized that the response column was 1 and 2s instead of 1 and 0s. I corrected this with the index/assign functionality in R and then went on to create a sample of the data to use for portioning out a test and train data set. I created my default model with an AIC of 889.1 and then improved on this model using only the significant factors to an AIC of 879.1.

```r
set.seed(12345)

gcredit.data <- read.table("germancredit.txt", stringsAsFactors = FALSE, header = FALSE)
#replace 1 and 2s in response column with usable 0/1 values

gcredit.data$V21[gcredit.data$V21 == 1] <- 0
gcredit.data$V21[gcredit.data$V21 == 2] <- 1
#selects 1/10 of the data points as a sample
sample <- sample(1:nrow(gcredit.data ), size = round(nrow(gcredit.data ) / 10), replace = FALSE)
# 9/10th of the data
train.gcredit.data <- gcredit.data[-sample, ]
# 1/10 of the data
test.gcredit.data <- gcredit.data[sample,  ]

gcredit.glm <- glm(V21~., family = binomial(link = "logit"), data = train.gcredit.data)
#AIC 889.1
summary(gcredit.glm)
```

```
##
## Call:
## glm(formula = V21 ~ ., family = binomial(link = "logit"), data = train.gcredit.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3698  -0.6667  -0.3470   0.6751   2.6406
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.913e-01  1.169e+00   0.249 0.803145
## V1A12       -4.347e-01  2.344e-01  -1.854 0.063684 .
## V1A13       -1.069e+00  4.010e-01  -2.666 0.007679 **
## V1A14       -1.927e+00  2.533e-01  -7.608 2.77e-14 ***
## V2           3.422e-02  1.011e-02   3.384 0.000714 ***
## V3A31        1.186e-01  5.909e-01   0.201 0.840954
## V3A32       -6.733e-01  4.720e-01  -1.427 0.153680
## V3A33       -1.057e+00  5.174e-01  -2.043 0.041068 *
## V3A34       -1.591e+00  4.860e-01  -3.273 0.001063 **
## V4A41       -1.637e+00  3.948e-01  -4.145 3.39e-05 ***
## V4A410      -2.025e+00  8.659e-01  -2.338 0.019367 *
## V4A42       -7.682e-01  2.832e-01  -2.713 0.006674 **
## V4A43       -8.879e-01  2.679e-01  -3.315 0.000917 ***
## V4A44       -6.168e-01  7.928e-01  -0.778 0.436580
## V4A45       -2.103e-01  5.962e-01  -0.353 0.724338
## V4A46        2.392e-02  4.276e-01   0.056 0.955394
## V4A48       -1.939e+00  1.289e+00  -1.505 0.132390
## V4A49       -9.090e-01  3.665e-01  -2.480 0.013123 *
## V5           1.199e-04  4.702e-05   2.549 0.010796 *
## V6A62       -2.725e-01  3.080e-01  -0.885 0.376247
## V6A63       -1.613e-01  4.162e-01  -0.388 0.698266
## V6A64       -1.354e+00  5.481e-01  -2.471 0.013468 *
## V6A65       -8.977e-01  2.805e-01  -3.201 0.001371 **
## V7A72       -1.002e-01  4.639e-01  -0.216 0.829073
## V7A73       -2.701e-01  4.430e-01  -0.610 0.542024
## V7A74       -1.021e+00  4.844e-01  -2.107 0.035092 *
## V7A75       -3.792e-01  4.469e-01  -0.849 0.396152
## V8           3.154e-01  9.463e-02   3.333 0.000860 ***
## V9A92       -1.717e-02  4.151e-01  -0.041 0.967002
## V9A93       -6.852e-01  4.075e-01  -1.681 0.092670 .
## V9A94       -1.284e-01  4.861e-01  -0.264 0.791758
## V10A102      3.973e-01  4.622e-01   0.859 0.390089
## V10A103     -1.141e+00  4.747e-01  -2.404 0.016212 *
## V11         -4.652e-02  9.428e-02  -0.493 0.621716
## V12A122      2.903e-01  2.728e-01   1.064 0.287193
## V12A123      2.485e-01  2.563e-01   0.969 0.332350
## V12A124      6.872e-01  4.765e-01   1.442 0.149297
## V13         -9.946e-03  9.914e-03  -1.003 0.315730
## V14A142      1.152e-01  4.558e-01   0.253 0.800514
## V14A143     -6.827e-01  2.585e-01  -2.641 0.008275 **
## V15A152     -4.270e-01  2.548e-01  -1.676 0.093813 .
## V15A153     -7.138e-01  5.241e-01  -1.362 0.173224
## V16          4.297e-01  2.041e-01   2.105 0.035258 *
```

```
## V17A172      4.183e-01  7.300e-01   0.573 0.566590
## V17A173      4.610e-01  7.040e-01   0.655 0.512604
## V17A174      4.120e-01  7.078e-01   0.582 0.560459
## V18          2.502e-01  2.709e-01   0.924 0.355677
## V19A192     -2.450e-01  2.166e-01  -1.131 0.257988
## V20A202     -1.565e+00  7.157e-01  -2.186 0.028808 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1096.15  on 899  degrees of freedom
## Residual deviance:  781.87  on 851  degrees of freedom
## AIC: 879.87
##
## Number of Fisher Scoring iterations: 5
```

```
#Chose significant values V1+V2+V3+V4+V5+V6+V8+V9+V10+V14+V15+V20
gcredit.glm.improv <- glm(V21~ V1+V2+V3+V4+V5+V6+V8+V9+V10+V14+V15+V20,
                   family = binomial(link = "logit"), data = train.gcredit.data)
#AIC 879.1 so better!
summary(gcredit.glm.improv)
```

```
##
## Call:
## glm(formula = V21 ~ V1 + V2 + V3 + V4 + V5 + V6 + V8 + V9 + V10 +
##     V14 + V15 + V20, family = binomial(link = "logit"), data = train.gcredit.data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2554  -0.6912  -0.3739   0.6983   2.8013
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.331e+00  7.291e-01   1.825 0.067984 .
## V1A12       -4.762e-01  2.262e-01  -2.105 0.035252 *
## V1A13       -1.118e+00  3.866e-01  -2.892 0.003823 **
## V1A14       -1.942e+00  2.464e-01  -7.881 3.25e-15 ***
## V2           3.236e-02  9.555e-03   3.387 0.000707 ***
## V3A31       -2.473e-01  5.630e-01  -0.439 0.660451
## V3A32       -1.034e+00  4.467e-01  -2.314 0.020691 *
## V3A33       -1.196e+00  5.085e-01  -2.352 0.018666 *
## V3A34       -1.716e+00  4.714e-01  -3.641 0.000271 ***
## V4A41       -1.595e+00  3.817e-01  -4.178 2.94e-05 ***
## V4A410      -2.041e+00  8.338e-01  -2.448 0.014357 *
## V4A42       -7.080e-01  2.726e-01  -2.597 0.009411 **
## V4A43       -9.190e-01  2.612e-01  -3.518 0.000435 ***
## V4A44       -6.092e-01  7.575e-01  -0.804 0.421234
## V4A45       -4.461e-03  5.712e-01  -0.008 0.993769
## V4A46        7.604e-02  4.210e-01   0.181 0.856662
## V4A48       -2.172e+00  1.269e+00  -1.711 0.087050 .
## V4A49       -9.578e-01  3.566e-01  -2.686 0.007237 **
## V5           1.101e-04  4.331e-05   2.542 0.011017 *
## V6A62       -2.383e-01  2.945e-01  -0.809 0.418363
## V6A63       -2.448e-01  4.042e-01  -0.606 0.544712
## V6A64       -1.350e+00  5.237e-01  -2.577 0.009955 **
## V6A65       -9.439e-01  2.725e-01  -3.464 0.000532 ***
## V8           2.945e-01  9.054e-02   3.253 0.001143 **
## V9A92        1.040e-02  3.961e-01   0.026 0.979058
## V9A93       -6.906e-01  3.870e-01  -1.784 0.074351 .
## V9A94       -1.282e-01  4.661e-01  -0.275 0.783196
## V10A102      4.930e-01  4.494e-01   1.097 0.272608
## V10A103     -1.199e+00  4.681e-01  -2.561 0.010450 *
## V14A142      1.844e-01  4.433e-01   0.416 0.677380
## V14A143     -7.276e-01  2.516e-01  -2.891 0.003835 **
## V15A152     -4.043e-01  2.341e-01  -1.727 0.084131 .
## V15A153     -3.478e-01  3.427e-01  -1.015 0.310143
## V20A202     -1.570e+00  7.130e-01  -2.202 0.027674 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1096.15  on 899  degrees of freedom
## Residual deviance:  802.33  on 866  degrees of freedom
## AIC: 870.33
```

```
##
## Number of Fisher Scoring iterations: 5
```

```
train.gcredit.data$V1A13[train.gcredit.data$V1 == "A13"] <- 1
train.gcredit.data$V1A13[train.gcredit.data$V1 != "A13"] <- 0

#last section before mental breakdown D:
predict(gcredit.glm.improv, test.gcredit.data[, -21], type = "response")
```

```
##        142        51       720       730       220       664       826
## 0.65731797 0.29383208 0.66742825 0.03114865 0.15399918 0.38405467 0.59584020
##        605       587       352       216       770        86        75
## 0.26724864 0.35082041 0.05279273 0.01090397 0.01965379 0.02015842 0.54177770
##         38       615       862       778       465       928        40
## 0.45859868 0.15745194 0.11649511 0.31105317 0.17683073 0.84730051 0.19766470
##        935       806       286       257       972       724       840
## 0.67158978 0.77719971 0.89767587 0.05630762 0.17048915 0.28736054 0.14533817
##        506        12       771       393        14       653       704
## 0.01863170 0.82249308 0.21698449 0.79745481 0.46966641 0.76420345 0.49404109
##        148       618       887       528       592        62       480
## 0.11445593 0.28690447 0.08945582 0.01498994 0.54199125 0.01958462 0.25161986
##        354       500       635       572       873       800       537
## 0.77596824 0.12506400 0.62452216 0.11242096 0.21459453 0.21321856 0.43910018
##         36       744       166       649       901       723       145
## 0.39018197 0.80970194 0.02650363 0.54968742 0.29336221 0.70622896 0.11429704
##        895       945       677       570       106       451       946
## 0.01592501 0.38696424 0.08815587 0.80807157 0.17472995 0.04194610 0.90519057
##         13       735       579       433       586        56       363
## 0.20354493 0.06580032 0.65920867 0.21732327 0.66841418 0.03433085 0.50786784
##        889       504        91       628       546       675       621
## 0.25372362 0.26394289 0.03122533 0.39766390 0.61538841 0.12310652 0.12222919
##        726       567       234       255       535       863       154
## 0.03694626 0.57287299 0.15433120 0.07245001 0.04592715 0.61769616 0.09082598
##        439       903       399       741        46       471       764
## 0.76696849 0.02355627 0.47772008 0.79766273 0.14656651 0.45224347 0.14384742
##        908        90       124       472       867       517       931
## 0.46396105 0.76710327 0.15679159 0.66220109 0.76711232 0.05493520 0.23808069
##        580       377
## 0.04658243 0.07177960
```