classtree\_m4\_cohen

#necessary libraries  
library(tidyverse, quiet=TRUE)

## Warning: package 'tidyverse' was built under R version 3.5.3

## -- Attaching packages ---------------------------------------------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 3.1.1 v purrr 0.3.2  
## v tibble 2.1.1 v dplyr 0.8.1  
## v tidyr 0.8.3 v stringr 1.4.0  
## v readr 1.3.1 v forcats 0.4.0

## Warning: package 'ggplot2' was built under R version 3.5.3

## Warning: package 'tibble' was built under R version 3.5.3

## Warning: package 'tidyr' was built under R version 3.5.3

## Warning: package 'readr' was built under R version 3.5.3

## Warning: package 'purrr' was built under R version 3.5.3

## Warning: package 'dplyr' was built under R version 3.5.3

## Warning: package 'stringr' was built under R version 3.5.3

## Warning: package 'forcats' was built under R version 3.5.3

## -- Conflicts ------------------------------------------------------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

library(caret)

## Warning: package 'caret' was built under R version 3.5.3

## Loading required package: lattice

##   
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':  
##   
## lift

library(rpart)  
library(rattle)

## Warning: package 'rattle' was built under R version 3.5.3

## Rattle: A free graphical interface for data science with R.  
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(RColorBrewer)

#read in parole.csv, rename variables per specifications and convert select variables to factors  
paroledata = read\_csv("parole.csv")

## Parsed with column specification:  
## cols(  
## male = col\_double(),  
## race = col\_double(),  
## age = col\_double(),  
## state = col\_double(),  
## time.served = col\_double(),  
## max.sentence = col\_double(),  
## multiple.offenses = col\_double(),  
## crime = col\_double(),  
## violator = col\_double()  
## )

#data comes in as doubles, below convert select to factors-interpreting numbers as characters  
paroledata = paroledata %>% mutate(male = as\_factor(as.character(male)),  
 race = as\_factor(as.character(race)),  
 state = as\_factor(as.character(state)),  
 crime = as\_factor(as.character(crime)),  
 multiple.offenses = as\_factor(as.character(multiple.offenses)),  
 violator = as\_factor(as.character(violator))) %>%  
#recode variables to match specifications  
 mutate(male = fct\_recode(male, "Male" = "1", "Female" = "0"),  
 race = fct\_recode(race, "White" = "1", "Not white" = "2"),  
 state = fct\_recode(state, "KY" = "2", "LA" = "3", "VA" = "4", "Other" = "1"),  
 crime = fct\_recode(crime,"Larceny" = "2", "Drug-related" = "3", "Driving-related" = "4", "Other" = "1"),  
 multiple.offenses = fct\_recode(multiple.offenses, "Multiple" = "1", "Not multiple" = "0"),  
 violator = fct\_recode(violator, "Violated" = "1", "Did not violate" = "0"))  
#str(paroledata)

# Task 1

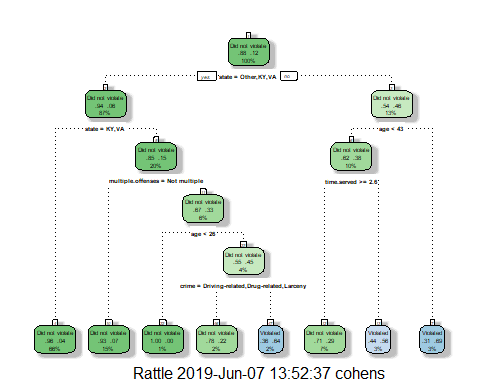
Split data into training and testing datasets (70%, 30%).

set.seed(12345)  
train.rows = createDataPartition(y = paroledata$violator, p=0.7, list = FALSE) #70% in training  
train = paroledata[train.rows,]   
test = paroledata[-train.rows,]

# Task 2 and 3

Create classification tree to predict violator and plot tree. Relate tree to case of 40 yr old LA parolee who served 5 yrs.

tree1 = rpart(violator ~., train, method="class")  
fancyRpartPlot(tree1)



I would classify this parolee as a non-violator. Using the classification tree created from the rpart function, parolees from LA would be on the right side of the first node because it is “false” that their state = Other, KY or VA. The next node for this group branches parollees based on age. Here a 40 year old LA parolee would be on the left side of that node because it is “true” that their age is < 43. The next node for this group branches parolees based on time served. This 40 yr old LA parolee would be on the on the left side of this node because it is “true” that he/she served >= 2.6 yrs. 71% of parollees such as this are predicted not to violate parole.

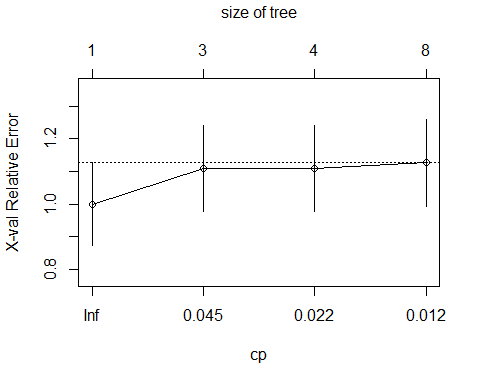
# Task 4

Evaluate tree performance as a function of complexity parameter.

printcp(tree1)

##   
## Classification tree:  
## rpart(formula = violator ~ ., data = train, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] age crime multiple.offenses state   
## [5] time.served   
##   
## Root node error: 55/473 = 0.11628  
##   
## n= 473   
##   
## CP nsplit rel error xerror xstd  
## 1 0.054545 0 1.00000 1.0000 0.12676  
## 2 0.036364 2 0.89091 1.1091 0.13253  
## 3 0.013636 3 0.85455 1.1091 0.13253  
## 4 0.010000 7 0.80000 1.1273 0.13345

plotcp(tree1)



#plot(tree1)  
#text(tree1)

The complexity parameter that minimizes cross-validated error is .05. The cross-validated error associated with this complexity parameter is .12676 and the data does not need to be split. I think this means that just knowing whether one was in LA or not is the best predictor of violating parole, without risking having a model that overfits the test data.

# Task 5

Prune the tree back to the chosen complexity parameter.

tree2 = prune(tree1,cp= tree1$cptable[which.min(tree1$cptable[,"xerror"]),"CP"])  
summary(tree2)

## Call:  
## rpart(formula = violator ~ ., data = train, method = "class")  
## n= 473   
##   
## CP nsplit rel error xerror xstd  
## 1 0.05454545 0 1 1 0.1267582  
##   
## Node number 1: 473 observations  
## predicted class=Did not violate expected loss=0.1162791 P(node) =1  
## class counts: 418 55  
## probabilities: 0.884 0.116

#fancyRpartPlot(tree2)

Pruning the tree back to the complexity parameter, would predict that 88.4% of parolees will not violate and 11.6% would violate.

# Task 6

Use original tree to make predictions on the training data. Create a confusion matrix and calculate results.

tree1pred = predict(tree1, train, type = "class")  
head(tree1pred)

## 1 2 3 4   
## Did not violate Did not violate Did not violate Did not violate   
## 5 6   
## Did not violate Did not violate   
## Levels: Did not violate Violated

confusionMatrix(tree1pred,train$violator,positive="Violated")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Did not violate Violated  
## Did not violate 402 28  
## Violated 16 27  
##   
## Accuracy : 0.907   
## 95% CI : (0.8771, 0.9316)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 0.06272   
##   
## Kappa : 0.5   
##   
## Mcnemar's Test P-Value : 0.09725   
##   
## Sensitivity : 0.49091   
## Specificity : 0.96172   
## Pos Pred Value : 0.62791   
## Neg Pred Value : 0.93488   
## Prevalence : 0.11628   
## Detection Rate : 0.05708   
## Detection Prevalence : 0.09091   
## Balanced Accuracy : 0.72632   
##   
## 'Positive' Class : Violated   
##

The predictions made from the unpruned classification tree have 90.7 accuracy on training dataset. The “naive” model which places all parollees into the “did not violate” category had an 88.4% accuracy. There is a small gain by using the model, but the p value is not significant. The specificity, which is the how well the model classified those who did not violate parole is 96%. The sensitivity, which is how well the model classified those who violated parole is 49%.

# Task 7

Use the unpruned tree to make predictions on the testing data. Create confusion matrix, calculate results and discuss model utility.

tree1pred = predict(tree1, test, type = "class")  
head(tree1pred)

## 1 2 3 4   
## Did not violate Violated Did not violate Did not violate   
## 5 6   
## Did not violate Did not violate   
## Levels: Did not violate Violated

confusionMatrix(tree1pred,test$violator,positive="Violated")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Did not violate Violated  
## Did not violate 170 19  
## Violated 9 4  
##   
## Accuracy : 0.8614   
## 95% CI : (0.8059, 0.9059)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.88631   
##   
## Kappa : 0.1525   
##   
## Mcnemar's Test P-Value : 0.08897   
##   
## Sensitivity : 0.17391   
## Specificity : 0.94972   
## Pos Pred Value : 0.30769   
## Neg Pred Value : 0.89947   
## Prevalence : 0.11386   
## Detection Rate : 0.01980   
## Detection Prevalence : 0.06436   
## Balanced Accuracy : 0.56182   
##   
## 'Positive' Class : Violated   
##

The predictions made from the unpruned classification tree have 86% accuracy on testing dataset. The “naive” model which places all parollees into the “did not violate” category had an 87% accuracy - which is higher than the accuracy of the model itself! The model is not performing well on the testing data set, perhaps it is “overfitted” to the training dataset. The specificity, which is the how well the model classified those who did not violate parole is 95%. The sensitivity, which is how well the model classified those who violated parole is 17%. I don’t think this is a great model. The sensitivity, to me, is not at an acceptable level to detect violators of parole which seems very important to be able to do.

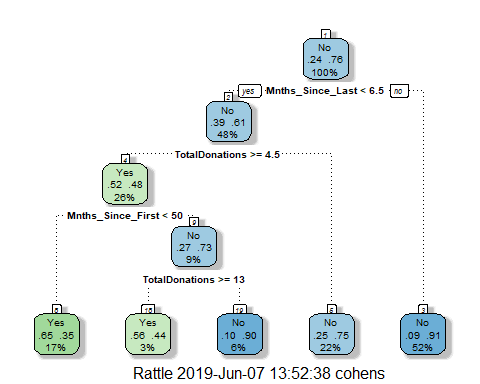
# Task 8 & 9

Read in new dataset and convert and recode select variables to factors. Split into train and test datasets (70%,30%). Create classification tree on training dataset and evaluate complexity parameter of model to predict DonatedMarch.

#read in blood.csv, rename variables per specifications and convert select variables to factors  
blooddata = read\_csv("blood.csv")

## Parsed with column specification:  
## cols(  
## Mnths\_Since\_Last = col\_double(),  
## TotalDonations = col\_double(),  
## Total\_Donated = col\_double(),  
## Mnths\_Since\_First = col\_double(),  
## DonatedMarch = col\_double()  
## )

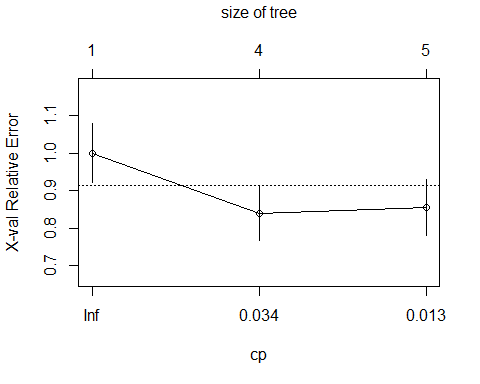
#DonatedMarch comes in as double, below convert to factor-interpreting numbers as characters  
blooddata = blooddata %>% mutate(DonatedMarch = as\_factor(as.character(DonatedMarch)))%>%  
 mutate(DonatedMarch = fct\_recode(DonatedMarch, "Yes" = "1", "No" = "0"))  
#str(blooddata)  
#split into train, test datasets  
set.seed(1234)  
train.rows = createDataPartition(y = blooddata$DonatedMarch, p=0.7, list = FALSE) #70% in training  
train2 = blooddata[train.rows,]   
test2 = blooddata[-train.rows,]  
#create classification tree & plot  
tree3 = rpart(DonatedMarch ~., train2, method="class")  
fancyRpartPlot(tree3)



#plot complexity parameter  
printcp(tree3)

##   
## Classification tree:  
## rpart(formula = DonatedMarch ~ ., data = train2, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] Mnths\_Since\_First Mnths\_Since\_Last TotalDonations   
##   
## Root node error: 125/524 = 0.23855  
##   
## n= 524   
##   
## CP nsplit rel error xerror xstd  
## 1 0.072 0 1.000 1.000 0.078049  
## 2 0.016 3 0.784 0.840 0.073304  
## 3 0.010 4 0.768 0.856 0.073822

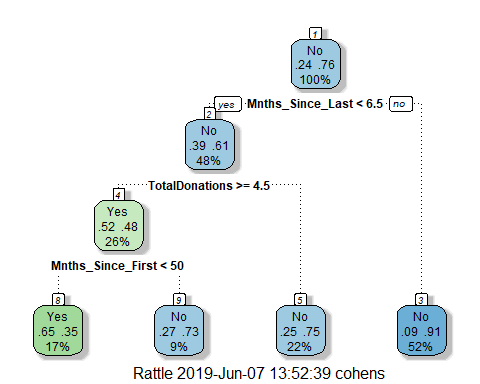
plotcp(tree3)

 The complexity parameter that minimizes cross-validated error is .016. This is associated with .856 cross-validated error.

# Task 10a

Prune the tree back to the chosen complexity parameter. Make predictions on the training dataset. Create confusion matrices, calculate results and discuss model utility.

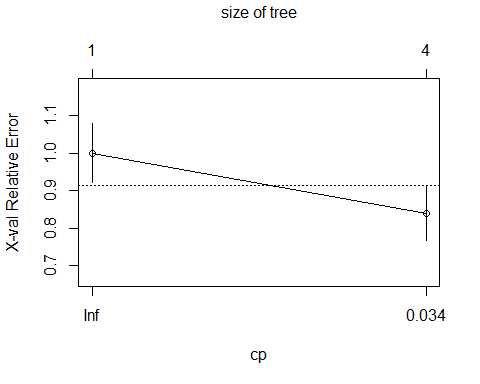
#Prune tree & plot  
tree4 = prune(tree3,cp= tree3$cptable[which.min(tree3$cptable[,"xerror"]),"CP"])  
#summary(tree2)  
fancyRpartPlot(tree4)



#plot complexity parameter  
printcp(tree4)

##   
## Classification tree:  
## rpart(formula = DonatedMarch ~ ., data = train2, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] Mnths\_Since\_First Mnths\_Since\_Last TotalDonations   
##   
## Root node error: 125/524 = 0.23855  
##   
## n= 524   
##   
## CP nsplit rel error xerror xstd  
## 1 0.072 0 1.000 1.00 0.078049  
## 2 0.016 3 0.784 0.84 0.073304

plotcp(tree4)



#make predictions, predict confusion matrix  
tree2pred = predict(tree4, train2, type = "class")  
head(tree2pred)

## 1 2 3 4 5 6   
## Yes Yes No No Yes Yes   
## Levels: Yes No

confusionMatrix(tree2pred,train2$DonatedMarch,positive="Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 58 31  
## No 67 368  
##   
## Accuracy : 0.813   
## 95% CI : (0.7769, 0.8455)  
## No Information Rate : 0.7615   
## P-Value [Acc > NIR] : 0.002713   
##   
## Kappa : 0.4287   
##   
## Mcnemar's Test P-Value : 0.000407   
##   
## Sensitivity : 0.4640   
## Specificity : 0.9223   
## Pos Pred Value : 0.6517   
## Neg Pred Value : 0.8460   
## Prevalence : 0.2385   
## Detection Rate : 0.1107   
## Detection Prevalence : 0.1698   
## Balanced Accuracy : 0.6932   
##   
## 'Positive' Class : Yes   
##

Using the pruned tree to predict DonatedMarch on the training dataset yields an accuracy rate of 81% which is higher than the naive model’s accuracy rate of 76%. The difference had a significant p value. The pruned tree only includes has 3 splits whereas the unpruned one has 4.

# Task 10b

Use pruned model on the testing dataset. Make predictions, create confusion matrices, calculate results and discuss model utility.

tree3pred = predict(tree4, test2, type = "class")  
head(tree3pred)

## 1 2 3 4 5 6   
## No Yes Yes No No Yes   
## Levels: Yes No

confusionMatrix(tree3pred,test2$DonatedMarch,positive="Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 14 16  
## No 39 155  
##   
## Accuracy : 0.7545   
## 95% CI : (0.6927, 0.8094)  
## No Information Rate : 0.7634   
## P-Value [Acc > NIR] : 0.657104   
##   
## Kappa : 0.2006   
##   
## Mcnemar's Test P-Value : 0.003012   
##   
## Sensitivity : 0.2642   
## Specificity : 0.9064   
## Pos Pred Value : 0.4667   
## Neg Pred Value : 0.7990   
## Prevalence : 0.2366   
## Detection Rate : 0.0625   
## Detection Prevalence : 0.1339   
## Balanced Accuracy : 0.5853   
##   
## 'Positive' Class : Yes   
##

It’s interesting that when the pruned model was run on the testing blood data set, the accuracy of the naive model was higher than the accuracy rate of the model by 1 percentage point. That suggests this might be an overfitted model. The sensitivity on the training set was 46% but here it is just 26%. The accuracy decreased as well from when it was run on the training dataset.