# Classification Trees

### Miguel Fernandez

### BAN-502

# Import libraries  
library(tidyverse)  
library(caret)  
library(rpart)  
library(rattle)  
library(RColorBrewer)

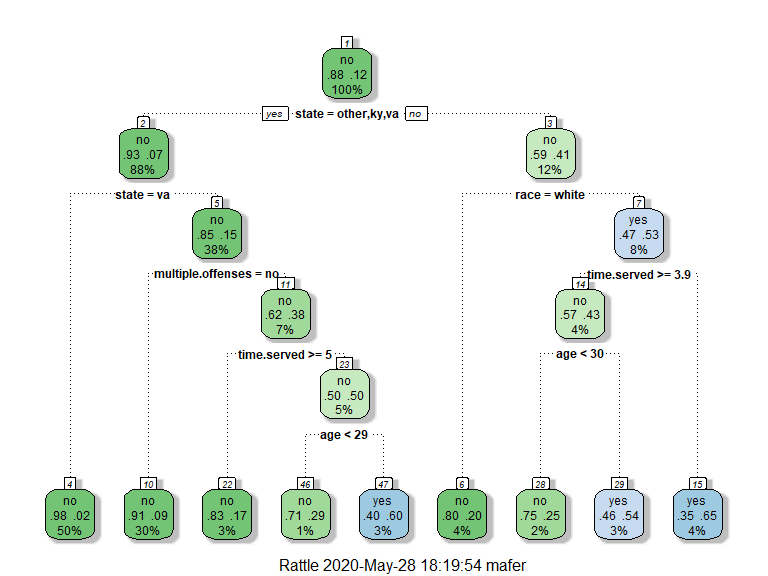
# Read in data  
parole <- read\_csv("parole.csv")  
  
# Recode variables and levels  
parole <- parole %>%  
 mutate(male = as\_factor(as.character(male))) %>%  
 mutate(male = fct\_recode(male,"male" = "1", "female" = "0")) %>%  
 mutate(race = as\_factor(as.character(race))) %>%  
 mutate(race = fct\_recode(race,"white" = "1", "not.white" = "2")) %>%  
 mutate(state = as\_factor(as.character(state))) %>%  
 mutate(state = fct\_recode(state,"ky" = "2",  
 "la" = "3",  
 "va" = "4",  
 "other" = "1")) %>%  
 mutate(multiple.offenses = as\_factor(as.character(multiple.offenses))) %>%  
 mutate(multiple.offenses = fct\_recode(multiple.offenses,"yes" = "1",  
 "no" = "0")) %>%  
 mutate(crime = as\_factor(as.character(crime))) %>%  
 mutate(crime = fct\_recode(crime,"larceny" = "2",  
 "drug" = "3",  
 "driving" = "4",  
 "other" = "1")) %>%  
 mutate(violator = as\_factor(as.character(violator))) %>%  
 mutate(violator = fct\_recode(violator,"yes" = "1", "no" = "0"))

#### Task 1

# Set # random seed  
set.seed(12345)  
# Split data for training and testing  
train.rows <- createDataPartition(y = parole$violator,  
 p = 0.7,  
 list = FALSE)  
train <- slice(parole, train.rows)  
test <- slice(parole, -train.rows)

#### Task 2

# Create classification tree  
tree1 = rpart(violator ~ ., train, method = "class")  
fancyRpartPlot(tree1)



#### Task 3

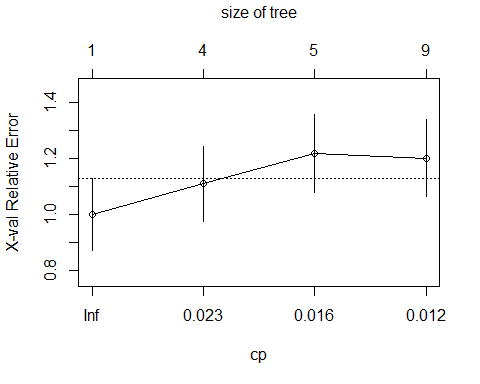
A 40 year old parolee from Louisiana who served a 5 year prison sentence could violate their parole or they could not. It depends on their race according to the classification tree above. Starting with the root node, we move to the right because the parolee is from Louisiana which is not listed in the expression. This leads to a node about race. If the parolee is white then they are not likely to violate their parole. However, if they are any other race, the next node asks was their prison sentence greater than or equal to 3.9 years. In our example, the person did serve more than 3.9 years. They served 5 years so this leads to the left and a node regarding age. The parolee is 40 years old which returns a no from the expression and the model concludes that this person will violate their parole.

#### Task 4

# Assessment of model  
printcp(tree1)

##   
## Classification tree:  
## rpart(formula = violator ~ ., data = train, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] age multiple.offenses race state   
## [5] time.served   
##   
## Root node error: 55/473 = 0.11628  
##   
## n= 473   
##   
## CP nsplit rel error xerror xstd  
## 1 0.030303 0 1.00000 1.0000 0.12676  
## 2 0.018182 3 0.90909 1.1091 0.13253  
## 3 0.013636 4 0.89091 1.2182 0.13788  
## 4 0.010000 8 0.83636 1.2000 0.13702

plotcp(tree1)



Looking at the CP table, the complexity parameter that generates the lowest cross-validated error is 0.03. What is interesting is that there will be zero splits to achieve that result.

#### Task 5

# Prune tree back  
tree2 = prune(tree1,cp = tree1$cptable[which.min(tree1$cptable[,"xerror"]),"CP"])  
printcp(tree2)

##   
## Classification tree:  
## rpart(formula = violator ~ ., data = train, method = "class")  
##   
## Variables actually used in tree construction:  
## character(0)  
##   
## Root node error: 55/473 = 0.11628  
##   
## n= 473   
##   
## CP nsplit rel error xerror xstd  
## 1 0.030303 0 1 1 0.12676

# Check majority group  
train %>%   
 group\_by(violator) %>%  
 summarise(no\_rows = length(violator))

## # A tibble: 2 x 2  
## violator no\_rows  
## <fct> <int>  
## 1 no 418  
## 2 yes 55

From the table above, we can see that no is the majority in the data set. In fact, yes responses only represent 11 percent of the data as seen in the Root node error in the complex Parameter table.

#### Task 6

# Predict using training set  
treepred = predict(tree1, train, type = "class")  
head(treepred)

## 1 2 3 4 5 6   
## no no no no no no   
## Levels: no yes

# Create confusion matrix  
confusionMatrix(treepred, train$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 400 28  
## yes 18 27  
##   
## Accuracy : 0.9027   
## 95% CI : (0.8724, 0.9279)  
## No Information Rate : 0.8837   
## P-Value [Acc > NIR] : 0.1095   
##   
## Kappa : 0.4862   
##   
## Mcnemar's Test P-Value : 0.1845   
##   
## Sensitivity : 0.49091   
## Specificity : 0.95694   
## Pos Pred Value : 0.60000   
## Neg Pred Value : 0.93458   
## Prevalence : 0.11628   
## Detection Rate : 0.05708   
## Detection Prevalence : 0.09514   
## Balanced Accuracy : 0.72392   
##   
## 'Positive' Class : yes   
##

The model has an accuracy of 0.90 which looks pretty good. The no information rate is 0.88 which is below the accuracy of the model. This means that if the model predicted no for all responses, it would have an accuracy of 0.88. The model does a great job predicting all the true negatives with a specificity of 0.95. However, it correctly identified only 50 percent of all the positives. Given that there are so few parole violators in the data set, this isn’t horrible. If the purpose of the model is to identify those that are likely to violate parole, then some adjustments must be made to improve sensitivity.

#### Task 7

# Predict using testing set  
treepred1 = predict(tree1, test, type = "class")  
head(treepred1)

## 1 2 3 4 5 6   
## no no no no no no   
## Levels: no yes

# Create confusion matrix  
confusionMatrix(treepred1, test$violator, positive = "yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction no yes  
## no 171 13  
## yes 8 10  
##   
## Accuracy : 0.896   
## 95% CI : (0.8455, 0.9345)  
## No Information Rate : 0.8861   
## P-Value [Acc > NIR] : 0.3797   
##   
## Kappa : 0.4309   
##   
## Mcnemar's Test P-Value : 0.3827   
##   
## Sensitivity : 0.43478   
## Specificity : 0.95531   
## Pos Pred Value : 0.55556   
## Neg Pred Value : 0.92935   
## Prevalence : 0.11386   
## Detection Rate : 0.04950   
## Detection Prevalence : 0.08911   
## Balanced Accuracy : 0.69504   
##   
## 'Positive' Class : yes   
##

The model performed nearly identical on the testing data as it did on the training data. Measures of accuracy, no information rate, sensitivity and specificity are very close between the two models. Therefore, we can conclude that the model fits the data well and is not subject to over fitting.

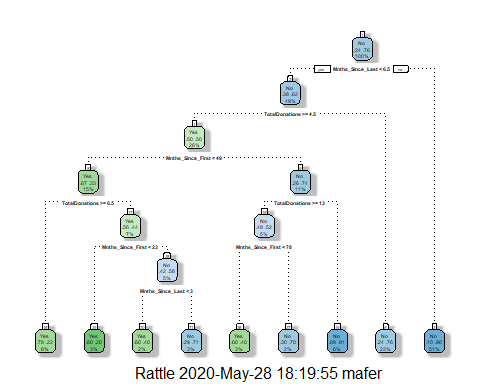
#### Task 8

# Read in data  
blood <- read\_csv("Blood.csv")  
# Create factor  
blood <- blood %>%  
 mutate(DonatedMarch = as\_factor(as.character(DonatedMarch))) %>%  
 mutate(DonatedMarch = fct\_recode(DonatedMarch,"Yes" = "1", "No" = "0"))

#### Task 9

# Set # random seed  
set.seed(1234)  
# Split data for training and testing  
train.rows.b <- createDataPartition(y = blood$DonatedMarch,  
 p = 0.7,  
 list = FALSE)  
train.b <- slice(blood, train.rows.b)  
test.b <- slice(blood, -train.rows.b)

tree.b = rpart(DonatedMarch ~ ., train.b, method = "class")  
fancyRpartPlot(tree.b)

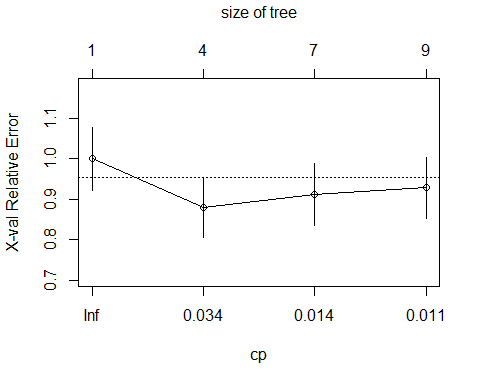


Looking at the decision tree, we see that the only variable not used is TotalDonated. This variable looks like a measure of the amount of blood each observation has given. The variable TotalDonations which is included in the model is likely a near-perfect predictor of TotalDonated which would cause the variable to be left out of the model.

# Assessment of model  
printcp(tree.b)

##   
## Classification tree:  
## rpart(formula = DonatedMarch ~ ., data = train.b, 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 0.880 0.074580  
## 3 0.012 6 0.736 0.912 0.075556  
## 4 0.010 8 0.712 0.928 0.076030

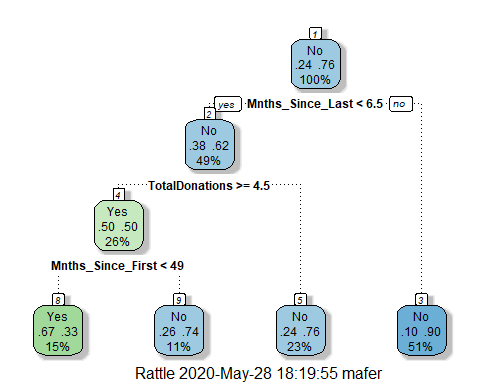
plotcp(tree.b)



Looking at the plot above, we can see that a complexity parameter of 0.01 does not yield the lowest cross-validated error. When we examine the table, a complexity parameter of 0.016 yields the smallest error of 0.88. To achieve these results there would need to be three splits in the model. Pruning the classification tree back will reveal those three splits needed.

#### Task 10

# Prune the tree  
tree.b2 = prune(tree.b,cp = tree.b$cptable[which.min(tree.b$cptable[,"xerror"]),"CP"])  
  
# Plot the tree  
fancyRpartPlot(tree.b2)



Pruning the tree back results in a much simpler tree that is unlikely to over fit the data. This model uses the same three variables as the first model, but with many less splits.

# Print cp table  
printcp(tree.b2)

##   
## Classification tree:  
## rpart(formula = DonatedMarch ~ ., data = train.b, 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 0.88 0.074580

The table above verifies that the tree was pruned back to the lowest error rate. The new model will be applied to the training and testing data sets and the performance will be measured using a confusion matrix.

# Predictions on the training set  
treepred.b = predict(tree.b2, train.b, type = "class")  
head(treepred.b)

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

# Create confusion matrix  
confusionMatrix(treepred.b, train.b$DonatedMarch, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 53 26  
## No 72 373  
##   
## Accuracy : 0.813   
## 95% CI : (0.7769, 0.8455)  
## No Information Rate : 0.7615   
## P-Value [Acc > NIR] : 0.002713   
##   
## Kappa : 0.4107   
##   
## Mcnemar's Test P-Value : 5.476e-06   
##   
## Sensitivity : 0.4240   
## Specificity : 0.9348   
## Pos Pred Value : 0.6709   
## Neg Pred Value : 0.8382   
## Prevalence : 0.2385   
## Detection Rate : 0.1011   
## Detection Prevalence : 0.1508   
## Balanced Accuracy : 0.6794   
##   
## 'Positive' Class : Yes   
##

The confusion matrix and statistics table above reveals an accuracy of 81 percent on the training data. The no information rate is 76 percent meaning the model performed better than just predicting “no” for all observations. This is confirmed by the p-value of less than 0.05 for accuracy being greater than the no information rate. However, the model performed less than ideal classifying true positives. It correctly identified less than half of the positive values while 93 percent of the true negatives were correctly identified. One possible scenario where this model would perform well is if we were trying to market or advertise to a population to donate blood. We would want to target those who have not donated yet. However, these results are from the training data, so lets see if our possible scenario plays out well on the testing data.

# Predictions on the testing data  
treepred.b2 = predict(tree.b2, test.b, type = "class")  
head(treepred.b2)

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

# Create confusion matrix  
confusionMatrix(treepred.b2, test.b$DonatedMarch, positive = "Yes")

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction Yes No  
## Yes 18 20  
## No 35 151  
##   
## Accuracy : 0.7545   
## 95% CI : (0.6927, 0.8094)  
## No Information Rate : 0.7634   
## P-Value [Acc > NIR] : 0.65710   
##   
## Kappa : 0.2468   
##   
## Mcnemar's Test P-Value : 0.05906   
##   
## Sensitivity : 0.33962   
## Specificity : 0.88304   
## Pos Pred Value : 0.47368   
## Neg Pred Value : 0.81183   
## Prevalence : 0.23661   
## Detection Rate : 0.08036   
## Detection Prevalence : 0.16964   
## Balanced Accuracy : 0.61133   
##   
## 'Positive' Class : Yes   
##

The model did not perform as well on the testing data set. The no information rate is slightly higher than the model accuracy meaning it would perform better predicting “no” for every observation. However, model specificity is strong at 88 percent which would still make it a decent model for the hypothetical marketing example above.