Building a CART model to predict supreme court decisions

Nutan Sahoo

14 October 2017

Start by importing the data file. Look at the data using str

```
stevens<- read.csv("stevens.csv")
str(stevens)
```

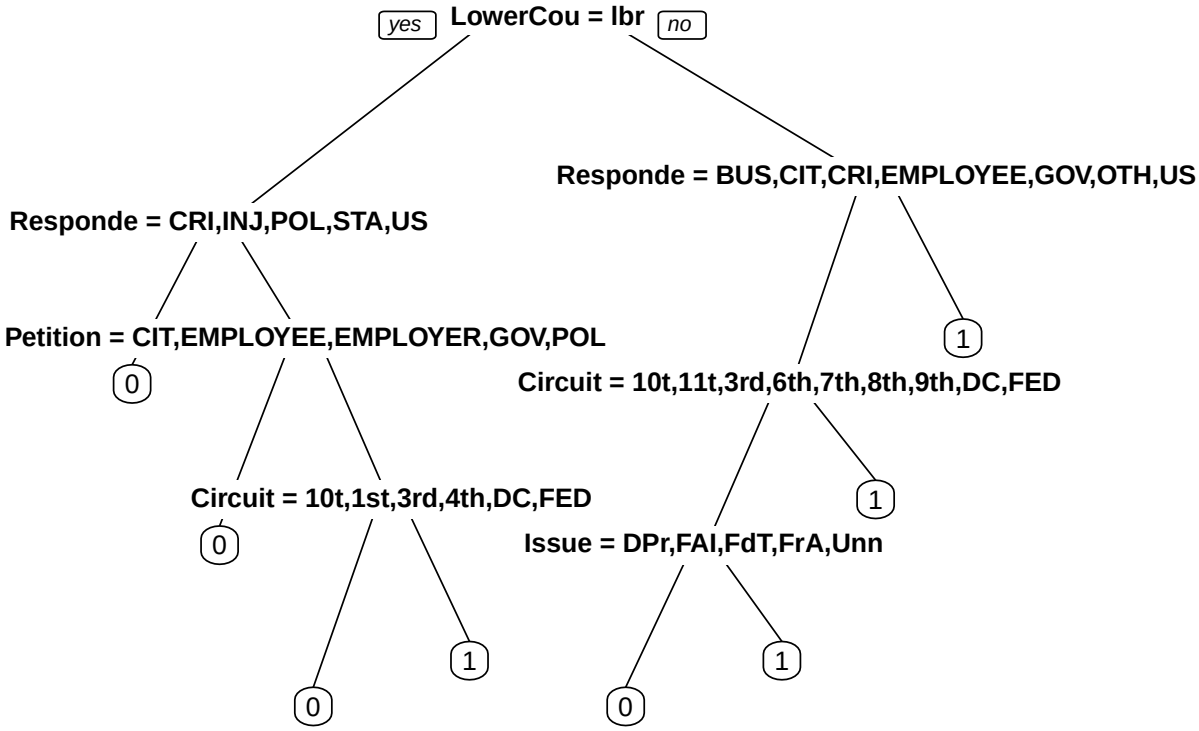
```
## 'data.frame':    566 obs. of  9 variables:
##  $ Docket    : Factor w/ 566 levels "00-1011","00-1045",..: 63 69 70 145 97 181 242 289 334 436 ...
##  $ Term      : int  1994 1994 1994 1994 1995 1995 1996 1997 1997 1999 ...
##  $ Circuit   : Factor w/ 13 levels "10th","11th",..: 4 11 7 3 9 11 13 11 12 2 ...
##  $ Issue     : Factor w/ 11 levels "Attorneys","CivilRights",..: 5 5 5 5 9 5 5 5 5 3 ...
##  $ Petitioner: Factor w/ 12 levels "AMERICAN.INDIAN",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Respondent: Factor w/ 12 levels "AMERICAN.INDIAN",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ LowerCourt: Factor w/ 2 levels "conser","liberal": 2 2 2 1 1 1 1 1 1 1 ...
##  $ Unconst   : int  0 0 0 0 0 1 0 1 0 0 ...
##  $ Reverse   : int  1 1 1 1 1 0 1 1 1 1 ...
```

We have 9 different var. The last one is the dependent variable. 0 means justice stevens decided to overturn the lower court's decision. 1 means he affirmed it. Then we need split the dataset into training and testing set using `CaTools` package

```
set.seed(3000)
#install.packages("CaTools")
library(caTools)
spl<- sample.split(stevens$Reverse, SplitRatio = 0.7)
Train1 <- subset(stevens, spl== TRUE )
Test1<- subset(stevens, spl==FALSE)
```

We would have to install and load a few packages for building a CART model. rpart works the same as lm, we give our dependent variable and then the independent ones. `minbucket` argument limits the tree so that it doesn't overfit to our training set. We selected a value of 25 but we can pick any value smaller or greater. `prp` function enables us to plot our tree.

```
#install.packages("rpart")
library(rpart)
#install.packages("rpart.plot")
library(rpart.plot)
stevens_tree<- rpart(Reverse~ Circuit+Issue+Petitioner+Respondent+LowerCourt+
                       Unconst, data= Train1, method = "class", minbucket= 25)
#class- because we want rpart to build a classification tree and not a regression tree
prp(stevens_tree)
```



Smaller the minbucket greater the number of splits in the model. As the number of observations in a particular cell decreases.
Now we will see how well our model performs. We will check the accuracy of the model by building a confusion matrix.

Predictions

We will now make predictions from our CART model using `predict` function. Our third argument here is `type` which we set equal to class. We need to give this argument when making predictions for our CART model if we want majority class predictions.

```
PredictCART<- predict(stevens_tree, newdata = Test1, type="class")
#we will give class in type when making predictions(majority class prediction) on a cart model
table(Test1$Reverse, PredictCART)
```

```
##    PredictCART
##      0  1
##   0 41 36
##   1 22 71
```

```
#to get the accuracy we will add up the total no. of obsv we got correct divided by the total obsv.
accuracy<- (41+71)/(41+36+22+71)
```

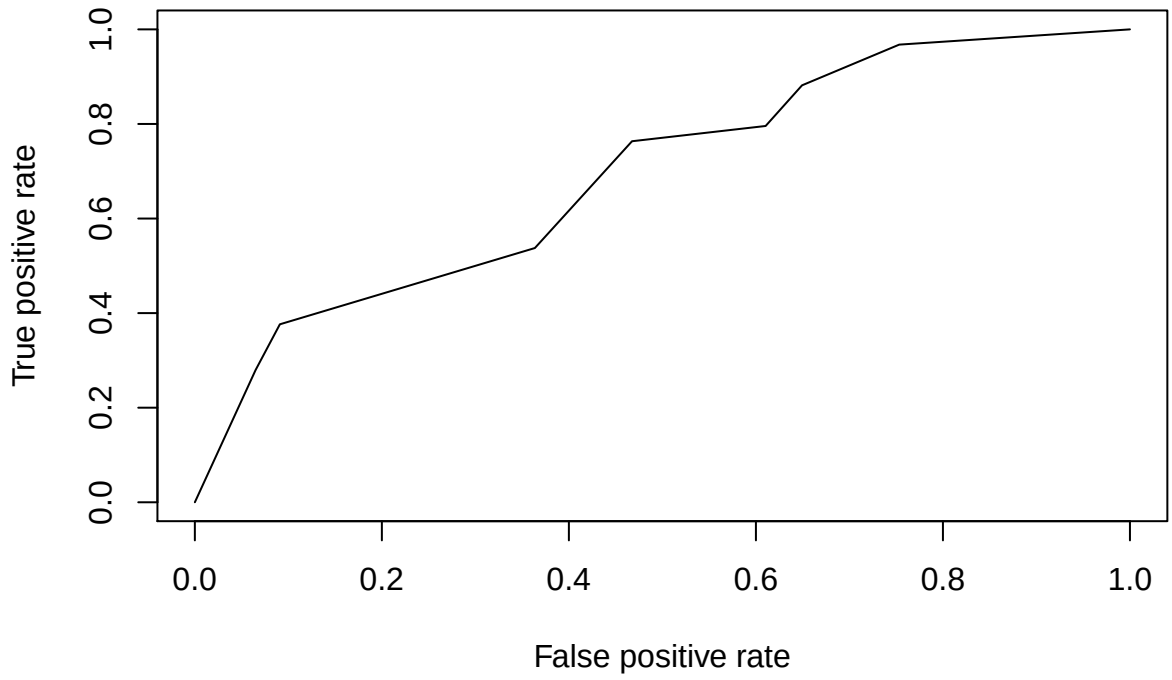
The baseline model always predicts Reverse and has an accuracy of 0.547. So our CART model sig. beats the baseline model. We will make the ROC curve using `ROCR` package.

```
PredictROC<- predict(stevens_tree, newdata = Test1)
head(PredictROC)
```

```
##            0         1
## 1  0.3035714 0.6964286
## 3  0.3035714 0.6964286
## 4  0.4000000 0.6000000
## 6  0.4000000 0.6000000
## 8  0.4000000 0.6000000
## 21 0.3035714 0.6964286
```

These numbers can be thought of as probabilities. The number written below the 0s are prob of getting the outcome 0 for that observation. These numbers actually give the percentage of the training set data in their respective subset with outcome 0 and the other coulumn gives percentage of the training set data in their respective subset with outcome 1. We will use the second column as the probabilities to generate an ROC curve.

```
library(ROCR)
pred<- prediction(PredictROC[ ,2], Test1$Reverse)
perf<- performance(pred, "tpr","fpr") #fpr and tpr are performance measures. There are other performanc
plot(perf)
```




```
as.numeric(performance(pred, "auc")@y.values)
```

```
## [1] 0.6927105
```

Random Forest Model

It builds many different trees on bootstrapped data. we need to select the value of a few parameters like minimum number of obsv. in a subset. In R this is controlled by the nodesize parameter. Smaller nodesize may take longer in R. Number of trees is controlled by ntree parameter. Should not be too small as bagging procedure may miss observations. More trees take more time. It is more computationally intense.

```
#install.packages("randomForest")
library(randomForest)
stevens_forest<- randomForest(Reverse~ Circuit+Issue+Petitioner+Respondent+LowerCourt+Unconst, data=Tra
Train1$Reverse<- as.factor(Train1$Reverse)
Test1$Reverse<- as.factor(Test1$Reverse)
```

When we are doing a classification problem we need to make sure that the outcome variable is factor. So we convert it to a factor.

Calculating the accuracy of the random forest model

```
PredictForest<- predict(stevens_forest, newdata= Test1)
table(Test1$Reverse, PredictForest)
(42+76)/(42+76+35+17)
```

So the accuracy of this model is 69%. This is an improvement over the accuracy of the CART model.

Cross-Validation

This method works by going through the following steps. First, we split the training set into k equally sized subsets, or folds. Then we select k-1 folds to estimate the model and compute predictions on the remaining one (validation set). Then we repeat the process by taking some other fold as the validation set this time.

```
#install.packages("caret")
library(caret)
#install.packages(e1071)
library(e1071)
#first we need to define how many folds we want
numfolds<- trainControl(method="cv", number=10)
#then we need to pick possible values for our cp
cpgrid<- expand.grid(.cp=seq(0.01,0.5, 0.01))
train(Reverse ~ Circuit+Issue+Petitioner+Respondent+LowerCourt+Unconst,
      data=Train1 , method = "rpart", trControl=numfolds, tuneGrid=cpgrid)
```

```
## CART
##
## 396 samples
##   6 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 357, 357, 357, 356, 356, 356, ...
## Resampling results across tuning parameters:
##
##   cp    Accuracy   Kappa
##   0.01  0.6285897  0.23480272
```

```
##   0.02  0.6183974  0.21533600
##   0.03  0.6236538  0.23278693
##   0.04  0.6338462  0.25943365
##   0.05  0.6338462  0.25943365
##   0.06  0.6438462  0.28283993
##   0.07  0.6438462  0.28283993
##   0.08  0.6438462  0.28283993
##   0.09  0.6438462  0.28283993
##   0.10  0.6438462  0.28283993
##   0.11  0.6438462  0.28283993
##   0.12  0.6438462  0.28283993
##   0.13  0.6438462  0.28283993
##   0.14  0.6438462  0.28283993
##   0.15  0.6438462  0.28283993
##   0.16  0.6438462  0.28283993
##   0.17  0.6438462  0.28283993
##   0.18  0.6438462  0.28283993
##   0.19  0.6438462  0.28283993
##   0.20  0.6438462  0.28283993
##   0.21  0.5756410  0.10184838
##   0.22  0.5656410  0.07476504
##   0.23  0.5479487  0.01264822
##   0.24  0.5453846  0.00000000
##   0.25  0.5453846  0.00000000
##   0.26  0.5453846  0.00000000
##   0.27  0.5453846  0.00000000
##   0.28  0.5453846  0.00000000
##   0.29  0.5453846  0.00000000
##   0.30  0.5453846  0.00000000
##   0.31  0.5453846  0.00000000
##   0.32  0.5453846  0.00000000
##   0.33  0.5453846  0.00000000
##   0.34  0.5453846  0.00000000
##   0.35  0.5453846  0.00000000
##   0.36  0.5453846  0.00000000
##   0.37  0.5453846  0.00000000
##   0.38  0.5453846  0.00000000
##   0.39  0.5453846  0.00000000
##   0.40  0.5453846  0.00000000
##   0.41  0.5453846  0.00000000
##   0.42  0.5453846  0.00000000
##   0.43  0.5453846  0.00000000
##   0.44  0.5453846  0.00000000
##   0.45  0.5453846  0.00000000
##   0.46  0.5453846  0.00000000
##   0.47  0.5453846  0.00000000
##   0.48  0.5453846  0.00000000
##   0.49  0.5453846  0.00000000
##   0.50  0.5453846  0.00000000
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.2.
```

We can see that the cp corresponding to the highest accuracy is the 0.19. You can also use 0.18 as there isn't

much difference in the accuracy. We will use this value of cp to create our CART model.

```
stevenscv<- rpart(Reverse~Circuit+ Issue+Petitioner+Respondent+LowerCourt+Unconst, data= Train1, method=
predictcv<- predict(stevenscv, newdata= Test1, type="class")
table(Test1$Reverse, predictcv)
```

```
##    predictcv
##      0  1
##   0 59 18
##   1 29 64
```

```
(59+64)/(59+64+18+29)
```

```
## [1] 0.7235294
```

The accuracy of this model is 72%, CART model we fitted above was 65%. There was a significant increase in accuracy in by selecting the right parameter value using this method.