Tree Methods

LAEB

12-11-2019

Contents

1 Decision Tree Implementation in R

We will implement a regression tree and a classification tree. The same car dataset used to implement PCA will be used. Detailed documentation about the dataset to be used can be found here: http://jse.amstat.org/datasets/04cars.txt

```
## Load packages
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(caret) # splitting into training/test sets
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(rpart) # preforming decision trees
library(rpart.plot) # visualization decision trees
library(rattle) # Visualization decision trees
```

```
## 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(purrr) # grids
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:caret':
##
##     lift
```

```r
library(ipred) # bagging
```

```r
## Import dataset
cars04  <- readRDS(file =
                     '~/birwe_data/Data/playground/prepared-zone/methods-and-libraries/ml-reading-cours
## Look at dataset
str(cars04)
```

```
## 'data.frame':    387 obs. of  18 variables:
##  $ Sports    : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ SUV       : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ Wagon     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Minivan   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Pickup    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ AWD       : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ RWD       : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ Retail    : int  43755 46100 36945 89765 23820 33195 26990 25940 35940 42490 ...
##  $ Dealer    : int  39014 41100 33337 79978 21761 30299 24647 23508 32506 38325 ...
##  $ Engine    : num  3.5 3.5 3.5 3.2 2 3.2 2.4 1.8 1.8 3 ...
##  $ Cylinders : int  6 6 6 6 4 6 4 4 4 6 ...
##  $ Horsepower: int  225 225 265 290 200 270 200 170 170 220 ...
##  $ CityMPG   : int  18 18 17 17 24 20 22 22 23 20 ...
##  $ HighwayMPG: int  24 24 23 24 31 28 29 31 30 27 ...
##  $ Weight    : int  3880 3893 4451 3153 2778 3575 3230 3252 3638 3814 ...
##  $ Wheelbase : int  115 115 106 100 101 108 105 104 105 105 ...
##  $ Length    : int  197 197 189 174 172 186 183 179 180 180 ...
##  $ Width     : int  72 72 77 71 68 72 69 70 70 70 ...
```

```r
summary(cars04)
```

```
##      Sports            SUV              Wagon            Minivan
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.0000   Median :0.0000   Median :0.00000   Median :0.00000
##  Mean   :0.1163   Mean   :0.1525   Mean   :0.07235   Mean   :0.05426
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
##      Pickup       AWD               RWD              Retail
##  Min.   :0   Min.   :0.0000   Min.   :0.0000   Min.   : 10280
##  1st Qu.:0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 20997
##  Median :0   Median :0.0000   Median :0.0000   Median : 28495
##  Mean   :0   Mean   :0.2016   Mean   :0.2429   Mean   : 33231
##  3rd Qu.:0   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 39552
##  Max.   :0   Max.   :1.0000   Max.   :1.0000   Max.   :192465
##      Dealer           Engine        Cylinders        Horsepower
##  Min.   :  9875   Min.   :1.400   Min.   : 3.000   Min.   : 73.0
##  1st Qu.: 19575   1st Qu.:2.300   1st Qu.: 4.000   1st Qu.:165.0
##  Median : 26155   Median :3.000   Median : 6.000   Median :210.0
##  Mean   : 30441   Mean   :3.127   Mean   : 5.757   Mean   :214.4
##  3rd Qu.: 36124   3rd Qu.:3.800   3rd Qu.: 6.000   3rd Qu.:250.0
##  Max.   :173560   Max.   :6.000   Max.   :12.000   Max.   :493.0
##     CityMPG        HighwayMPG        Weight       Wheelbase
```

```
##  Min.   :10.00   Min.   :12.00   Min.   :1850   Min.   : 89.0
##  1st Qu.:18.00   1st Qu.:24.00   1st Qu.:3107   1st Qu.:103.0
##  Median :19.00   Median :27.00   Median :3469   Median :107.0
##  Mean   :20.31   Mean   :27.26   Mean   :3532   Mean   :107.2
##  3rd Qu.:21.50   3rd Qu.:30.00   3rd Qu.:3922   3rd Qu.:112.0
##  Max.   :60.00   Max.   :66.00   Max.   :6400   Max.   :130.0
##      Length         Width
##  Min.   :143   Min.   :64.00
##  1st Qu.:177   1st Qu.:69.00
##  Median :186   Median :71.00
##  Mean   :185   Mean   :71.28
##  3rd Qu.:193   3rd Qu.:73.00
##  Max.   :221   Max.   :81.00
```

```
head(cars04)
```

```
##                            Sports SUV Wagon Minivan Pickup AWD RWD Retail
## Acura 3.5 RL                    0   0     0       0      0   0   0  43755
## Acura 3.5 RL Navigation         0   0     0       0      0   0   0  46100
## Acura MDX                       0   1     0       0      0   1   0  36945
## Acura NSX S                     1   0     0       0      0   0   1  89765
## Acura RSX                       0   0     0       0      0   0   0  23820
## Acura TL                        0   0     0       0      0   0   0  33195
##                            Dealer Engine Cylinders Horsepower CityMPG
## Acura 3.5 RL                39014    3.5         6        225      18
## Acura 3.5 RL Navigation     41100    3.5         6        225      18
## Acura MDX                   33337    3.5         6        265      17
## Acura NSX S                 79978    3.2         6        290      17
## Acura RSX                   21761    2.0         4        200      24
## Acura TL                    30299    3.2         6        270      20
##                            HighwayMPG Weight Wheelbase Length Width
## Acura 3.5 RL                       24   3880       115    197    72
## Acura 3.5 RL Navigation            24   3893       115    197    72
## Acura MDX                          23   4451       106    189    77
## Acura NSX S                        24   3153       100    174    71
## Acura RSX                          31   2778       101    172    68
## Acura TL                           28   3575       108    186    72
```

Before proceeding to the tree implementation, we split the data into a training and test sets uset the Caret package.

```
## Splitting the data into a train and test set
# Set seed
set.seed(123)
# Do the split
cars04.sub <- cars04  %>%
  select(-c("SUV","Wagon","Minivan","Pickup","AWD","RWD"))
training.sample <- cars04.sub$Sports %>% createDataPartition(p = 0.8, list = FALSE)
cars04.train.data  <- cars04.sub[training.sample, ]
cars04.test.data <- cars04.sub[-training.sample, ]
# Check rows of the train and test datasets
nrow(cars04.train.data)
```

```
## [1] 310
```

```
nrow(cars04.test.data)
```

```
## [1] 77
```

```r
# First rows of the train and test datasets
head(cars04.train.data)
```

```
##                           Sports Retail Dealer Engine Cylinders Horsepower
## Acura 3.5 RL Navigation        0  46100  41100    3.5         6        225
## Acura NSX S                    1  89765  79978    3.2         6        290
## Acura RSX                      0  23820  21761    2.0         4        200
## Acura TSX                      0  26990  24647    2.4         4        200
## Audi A4 1.8T                   0  25940  23508    1.8         4        170
## Audi A4 1.8T convertible       0  35940  32506    1.8         4        170
##                           CityMPG HighwayMPG Weight Wheelbase Length Width
## Acura 3.5 RL Navigation        18         24   3893       115    197    72
## Acura NSX S                    17         24   3153       100    174    71
## Acura RSX                      24         31   2778       101    172    68
## Acura TSX                      22         29   3230       105    183    69
## Audi A4 1.8T                   22         31   3252       104    179    70
## Audi A4 1.8T convertible       23         30   3638       105    180    70
```

```r
head(cars04.test.data)
```

```
##                                      Sports Retail Dealer Engine Cylinders
## Acura 3.5 RL                              0  43755  39014    3.5         6
## Acura MDX                                 0  36945  33337    3.5         6
## Acura TL                                  0  33195  30299    3.2         6
## Audi A6 2.7 Turbo Quattro four-door       0  42840  38840    2.7         6
## Audi A6 3.0 Avant Quattro                 0  40840  37060    3.0         6
## Audi A6 3.0 Quattro                       0  39640  35992    3.0         6
##                                      Horsepower CityMPG HighwayMPG Weight
## Acura 3.5 RL                                225      18         24   3880
## Acura MDX                                   265      17         23   4451
## Acura TL                                    270      20         28   3575
## Audi A6 2.7 Turbo Quattro four-door         250      18         25   3836
## Audi A6 3.0 Avant Quattro                   220      18         25   4035
## Audi A6 3.0 Quattro                         220      18         25   3880
##                                      Wheelbase Length Width
## Acura 3.5 RL                               115    197    72
## Acura MDX                                  106    189    77
## Acura TL                                   108    186    72
## Audi A6 2.7 Turbo Quattro four-door        109    192    71
## Audi A6 3.0 Avant Quattro                  109    192    71
## Audi A6 3.0 Quattro                        109    192    71
```

There are many methodologies for constructing regression trees but one of the oldest is known as the classification and regression tree (CART) approach developed by Breiman et al. (1984).
Basic regression trees partition a data set into smaller subgroups and then fit a simple constant for each observation in the subgroup. The partitioning is achieved by successive binary partitions (aka recursive partitioning) based on the different predictors. The constant to predict is based on the average response values for all observations that fall in that subgroup (if regression tree) or majority vote (if clasification tree).

There exist different packages in R to implement decision trees. In the following we will use the rpart package to implement the tree and the rpart.plot and rattle plackages for visualizations.

1.1 Regression Tree

Let's consider that we want to predict the Highway Mileage of a car. We fit a regression tree to the training dataset only and look at the output.

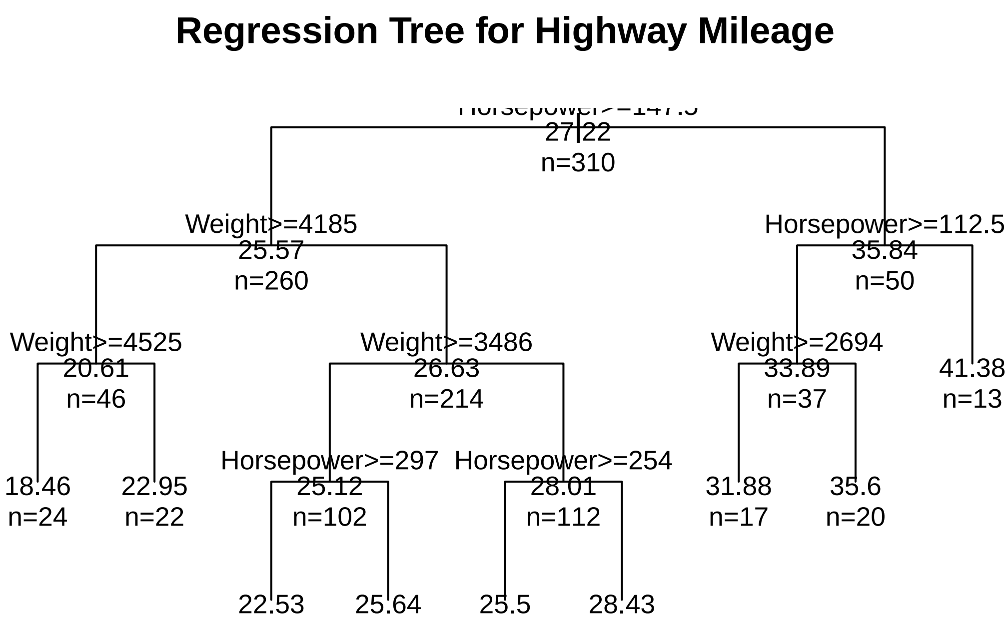
```
## Regression tree
reg.tree.fit <- rpart(
                  formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
                    Horsepower + Weight + Wheelbase + Length + Width,
                  method ="anova",
                  data = cars04.train.data)
# Look at regression tree
reg.tree.fit
```

```
## n= 310
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 310 10023.64000 27.22258
##    2) Horsepower>=147.5 260  3489.88800 25.56538
##      4) Weight>=4185 46   474.95650 20.60870
##        8) Weight>=4525 24   127.95830 18.45833 *
##        9) Weight< 4525 22   114.95450 22.95455 *
##      5) Weight< 4185 214  1641.83600 26.63084
##       10) Weight>=3486 102   566.58820 25.11765
##         20) Horsepower>=297 17    30.23529 22.52941 *
##         21) Horsepower< 297 85   399.69410 25.63529 *
##       11) Weight< 3486 112   628.99110 28.00893
##         22) Horsepower>=254 16    34.00000 25.50000 *
##         23) Horsepower< 254 96   477.48960 28.42708 *
##    3) Horsepower< 147.5 50  2106.72000 35.84000
##      6) Horsepower>=112.5 37   399.56760 33.89189
##       12) Weight>=2694.5 17    99.76471 31.88235 *
##       13) Weight< 2694.5 20   172.80000 35.60000 *
##      7) Horsepower< 112.5 13  1167.07700 41.38462 *
```

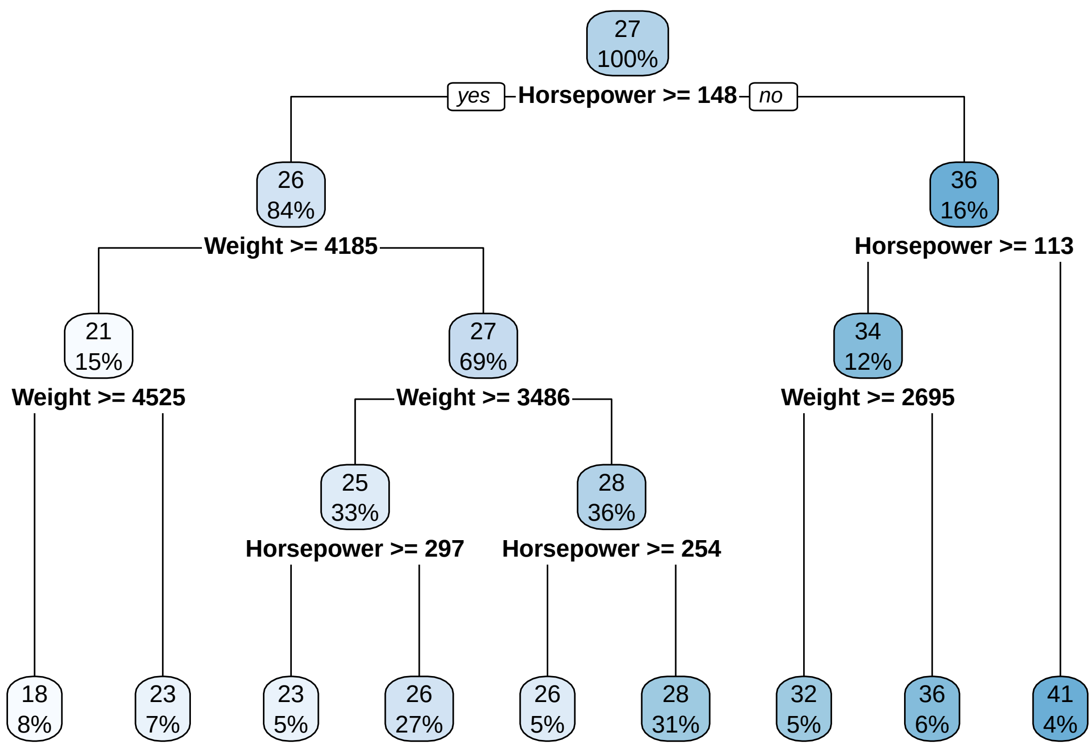
The output tells us that there are 310 observations in the Root branch 310 observations with SSE = 10023 and a HighwayMPG prediction of 27.22. The First split happens on the Horsepower variable. The number of observations i.e. cars with a Horsepower >= 147.5 is 260 and the SSE = 3489 and HighwayMPG prediction = 25.56 for this branch. This means that Horsepower is the variable having the most important reduction in SSE, then it's Engine. The second most important variable is Weight and so on.

The tree can be plotted using base R or the rpart.plot and rattle packages as follow

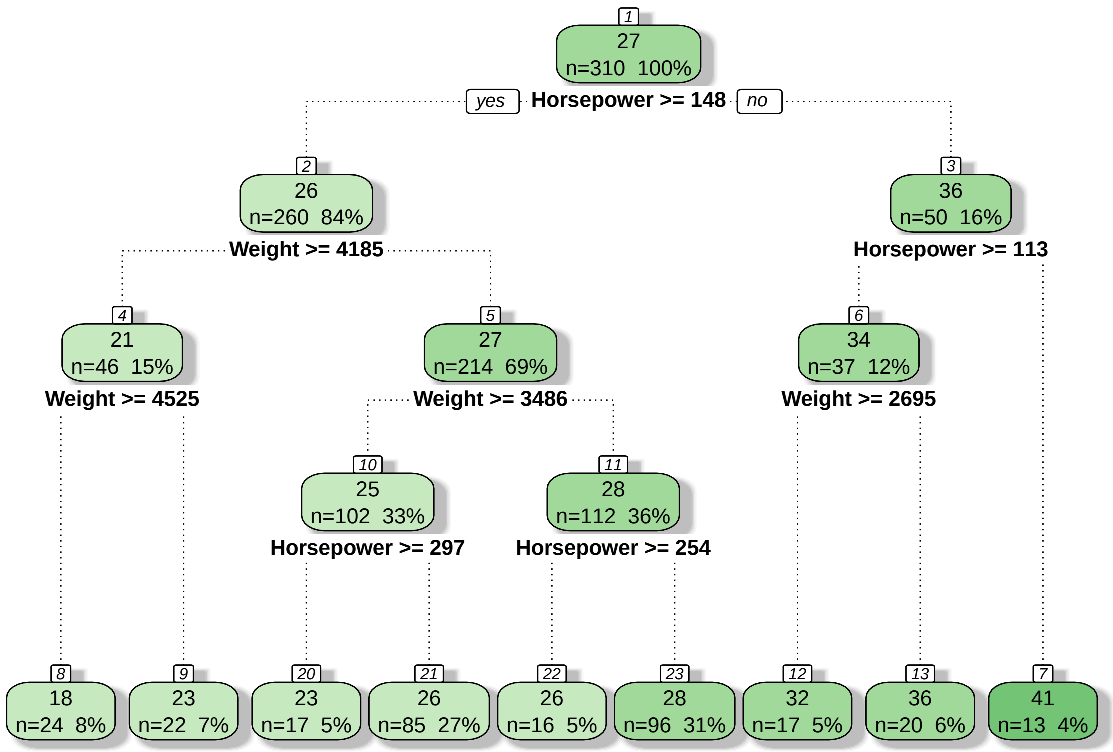
```
# Directly using base R
plot(reg.tree.fit, uniform=TRUE,
     main="Regression Tree for Highway Mileage")
text(reg.tree.fit, use.n=TRUE, all=TRUE, cex=.8)
```




```r
# Using rpart package
rpart.plot(reg.tree.fit)
```




```r
# Using the rattle packages
fancyRpartPlot(reg.tree.fit)
```



Rattle 2019−Nov−12 14:58:31 laeb

We can use this model to predict the HIghwayMPG on the test dataset and compared with the observed value. This is done as follows

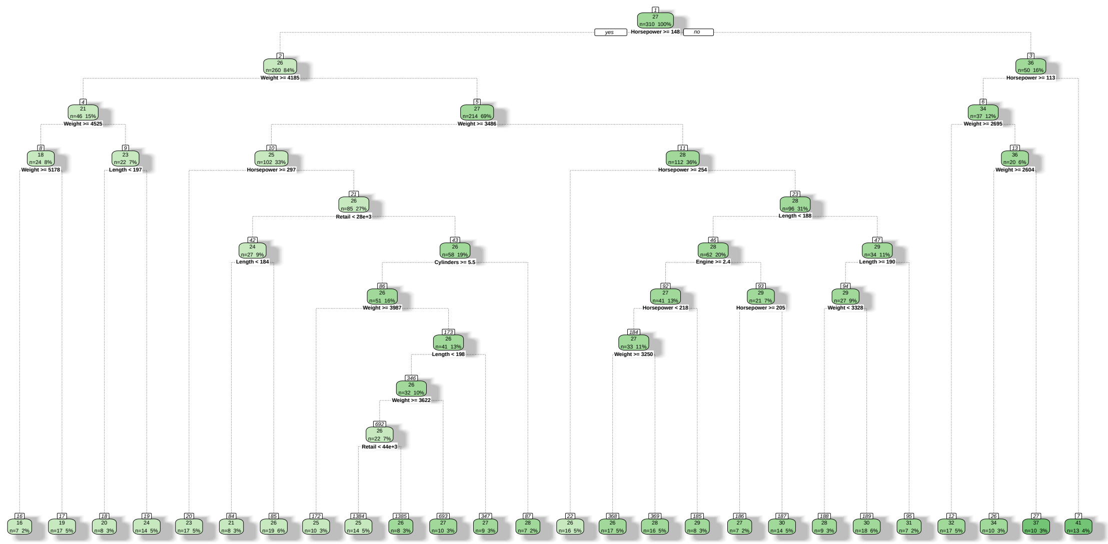
```
# Predict on test dataset
pred <- predict(reg.tree.fit, newdata = cars04.test.data)
RMSE(pred = pred, obs = cars04.test.data$HighwayMPG)
```

```
## [1] 3.296994
```

Often a balance needs to be found in the depth and complexity of the tree to optimize predictive performance on unseen data. This is done by pruning the tree. The package rpart is actually behind the scene already aplying a cost complexity alpha values to prune the tree. It is performing a 10 fold CV on the training dataset to decide the optimal value of alpha i.e. the value of alpha for which the CV error is minimal. In our example, the tree was pruned to have 9 leaves i.e. 9 leaves was found to be the size of the tree for which the CV error is minimal. The dashed line indicates the 1 standard deviation of the minimum CV error. In practice it is common to use a tree with the minimum size within 1 standard deviation of the minimum CV error.

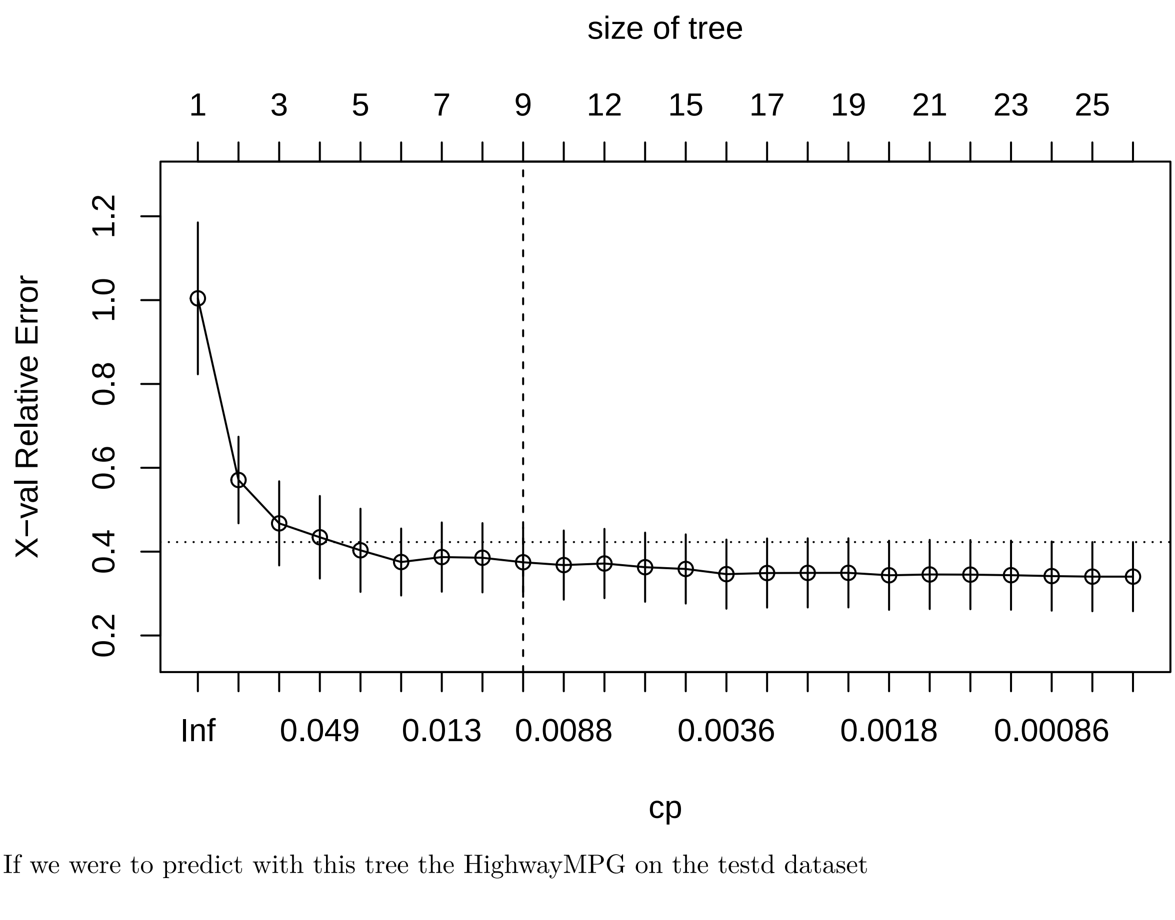
Let's force rpart to generate the full tree (no penalty done i.e. no pruning)

```
# Force rpart to generate a full tree with no penalty
reg.tree.fit2 <- rpart(
  formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
    Horsepower + Weight + Wheelbase + Length + Width,
  data    = cars04.train.data,
  method  = "anova",
  control = list(cp = 0)
)
fancyRpartPlot(reg.tree.fit2)
```



Rattle 2019−Nov−12 14:58:32 laeb

```r
plotcp(reg.tree.fit2)
abline(v = 9, lty = "dashed")
```



If we were to predict with this tree the HighwayMPG on the testd dataset

```r
# Predict on test dataset
pred <- predict(reg.tree.fit2, newdata = cars04.test.data)
RMSE(pred = pred, obs = cars04.test.data$HighwayMPG)
```

```
## [1] 3.085369
```

In addition to the cost complexity parameter, one can tune other parameters e.g. minsplit and maxdepth using the control arguments of rpart.

- minsplit: Set the minimum number of observations in the node before the algorithm perform a split
- maxdepth: Set the maximum depth of any node of the final tree. The root node is treated a depth 0

One can manually tune these parameters and assess the performance of your model. But this might be cumbersome, instead one can make a grid search.

```r
# Make a grid to search
hypergrid <- expand.grid(
  minsplit = seq(5, 20, 1),
  maxdepth = seq(3, 9, 1)
)
head(hypergrid) # head of the grid
```

```
##   minsplit maxdepth
## 1        5        3
## 2        6        3
## 3        7        3
## 4        8        3
## 5        9        3
## 6       10        3
```

```r
nrow(hypergrid) # total number of combinations
```

```
## [1] 112
```

```r
# One model for each hyperparameter combination
models <- list()
for (i in 1:nrow(hypergrid)) {
  # get minsplit, maxdepth values at row i
  minsplit <- hypergrid$minsplit[i]
  maxdepth <- hypergrid$maxdepth[i]
  # train a model and store in the list
  models[[i]] <- rpart(
    formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
      Horsepower + Weight + Wheelbase + Length + Width,
    data    = cars04.train.data,
    method  = "anova",
    control = list(minsplit = minsplit, maxdepth = maxdepth)
  )
}
# function to get optimal cp
getcp <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  cp <- x$cptable[min, "CP"]
}# function to get minimum error
getminerror <- function(x) {
  min    <- which.min(x$cptable[, "xerror"])
  xerror <- x$cptable[min, "xerror"]
```

```r
}

hypergrid_min <- hypergrid %>%
  mutate(
    cp    = purrr::map_dbl(models, getcp),
    error = purrr::map_dbl(models, getminerror)
  ) %>%
  arrange(error) %>%
  top_n(-5, wt = error)
hypergrid_min
```

```
##   minsplit maxdepth         cp     error
## 1       10        9 0.01000000 0.3463026
## 2       14        3 0.01267033 0.3481227
## 3        7        8 0.01000000 0.3493651
## 4       12        7 0.01000000 0.3515852
## 5       15        3 0.01267033 0.3517852
```

```r
# Optimal Model
reg.tree.fit.opt <- rpart(
  formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
    Horsepower + Weight + Wheelbase + Length + Width,
  data    = cars04.train.data,
  method  = "anova",
  control = list(minsplit = 10, maxdepth = 9, cp = 0.01)
)
# Predict on test dataset
pred <- predict(reg.tree.fit.opt, newdata = cars04.test.data)
RMSE(pred = pred, obs = cars04.test.data$HighwayMPG)
```

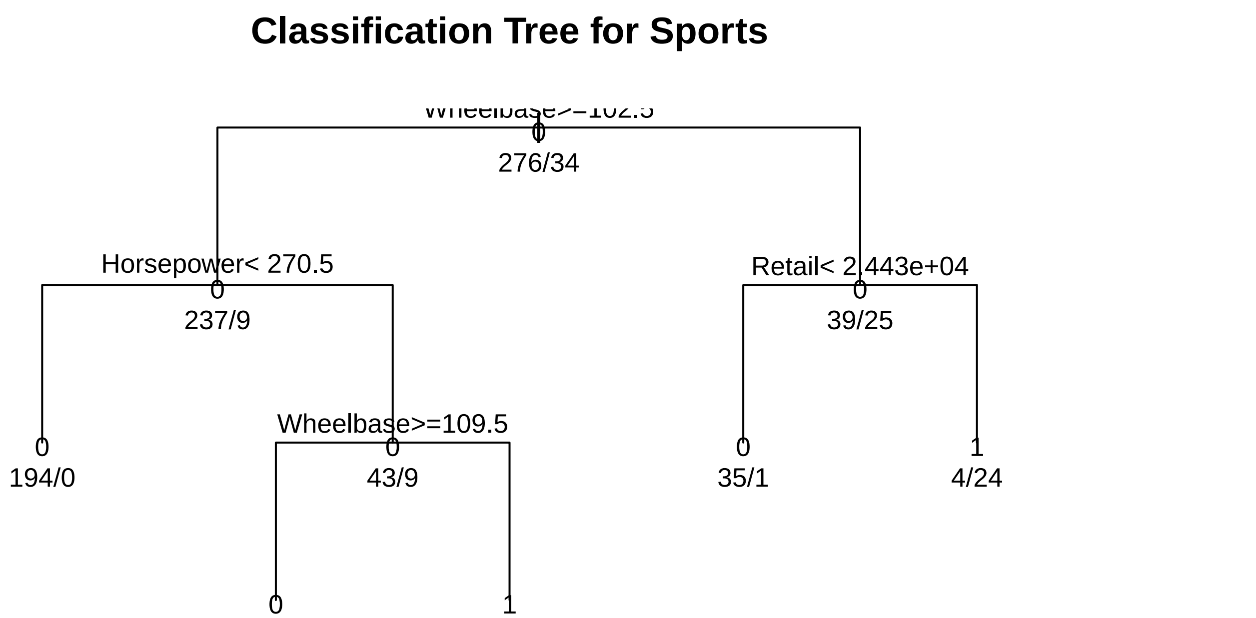
```
## [1] 3.296994
```

1.2 Classification Tree

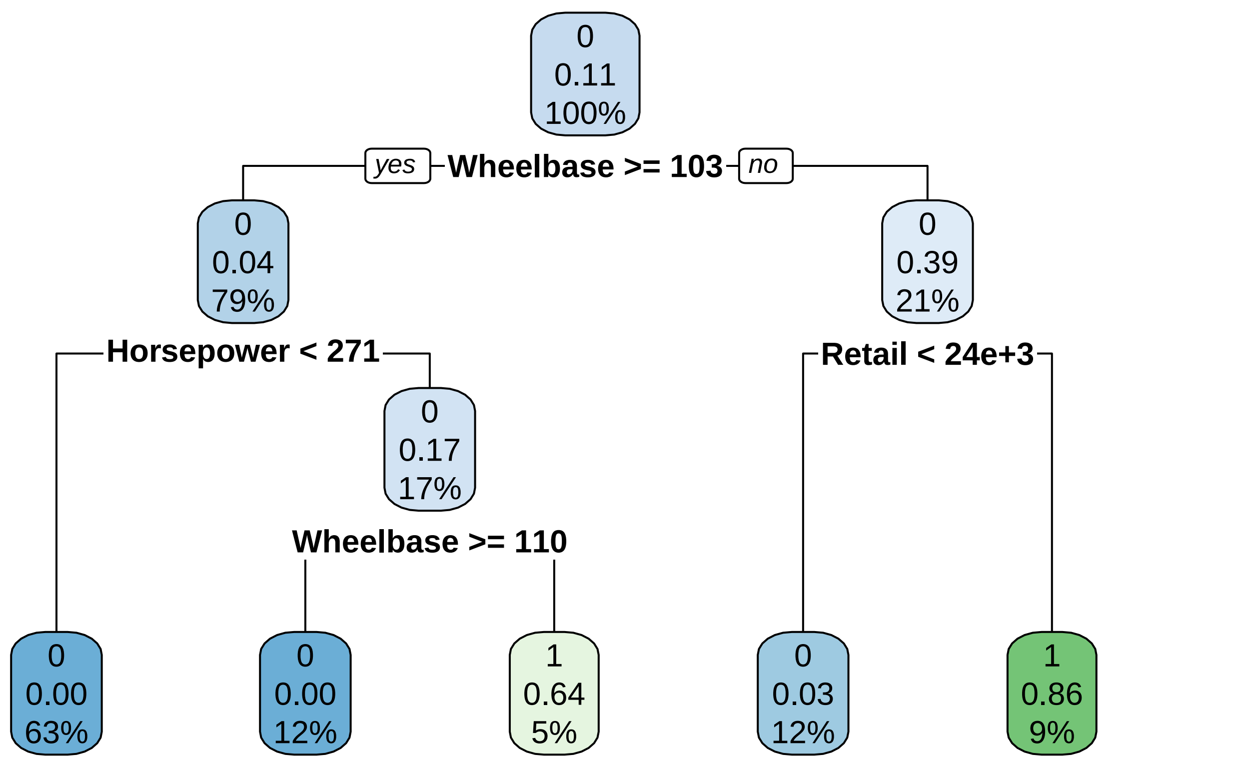
```r
# Grow tree
tree.fit <- rpart(Sports ~ Retail + Dealer + Engine + Cylinders + Horsepower + CityMPG + HighwayMPG + We
             method="class", data = cars04.train.data)
# Look at output
tree.fit
```

```
## n= 310
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 310 34 0 (0.89032258 0.10967742)
##    2) Wheelbase>=102.5 246  9 0 (0.96341463 0.03658537)
##      4) Horsepower< 270.5 194  0 0 (1.00000000 0.00000000) *
##      5) Horsepower>=270.5 52  9 0 (0.82692308 0.17307692)
##       10) Wheelbase>=109.5 38  0 0 (1.00000000 0.00000000) *
##       11) Wheelbase< 109.5 14  5 1 (0.35714286 0.64285714) *
##    3) Wheelbase< 102.5 64 25 0 (0.60937500 0.39062500)
##      6) Retail< 24432.5 36  1 0 (0.97222222 0.02777778) *
##      7) Retail>=24432.5 28  4 1 (0.14285714 0.85714286) *
```

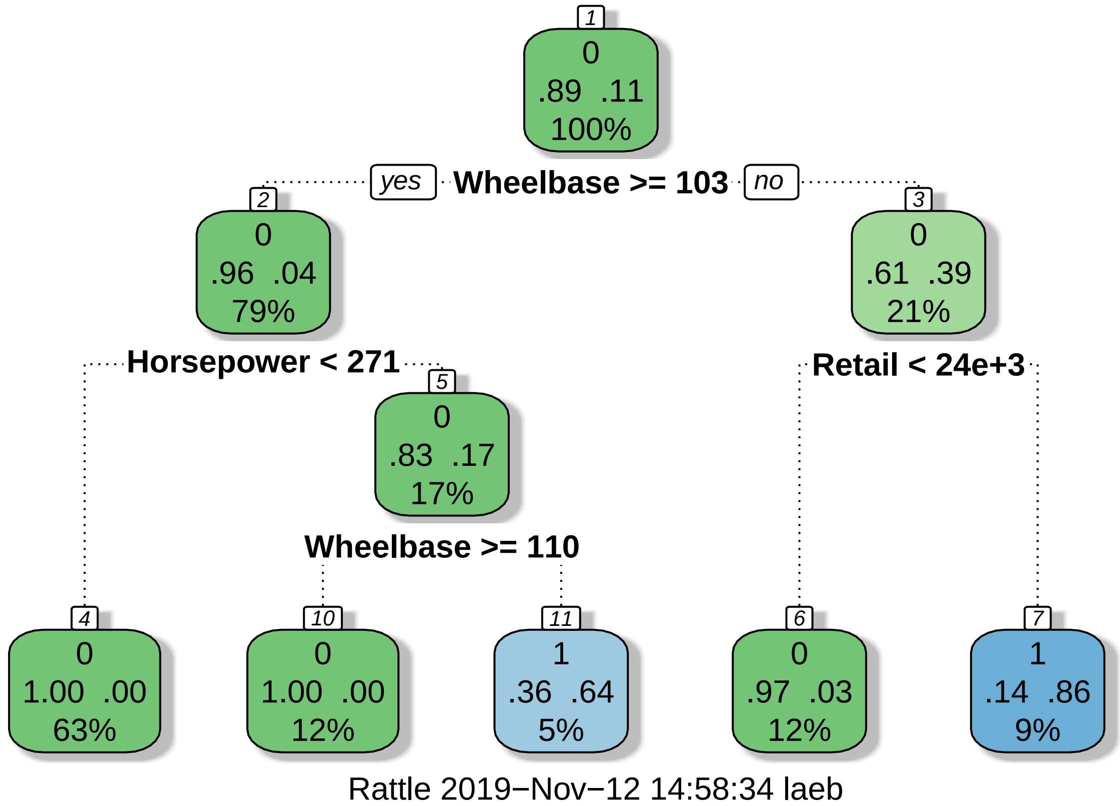
```r
# plot tree
# Directly using base R
plot(tree.fit, uniform=TRUE,
     main="Classification Tree for Sports")
text(tree.fit, use.n=TRUE, all=TRUE, cex=.8)
```




```r
# Using rpart package
rpart.plot(tree.fit)
```



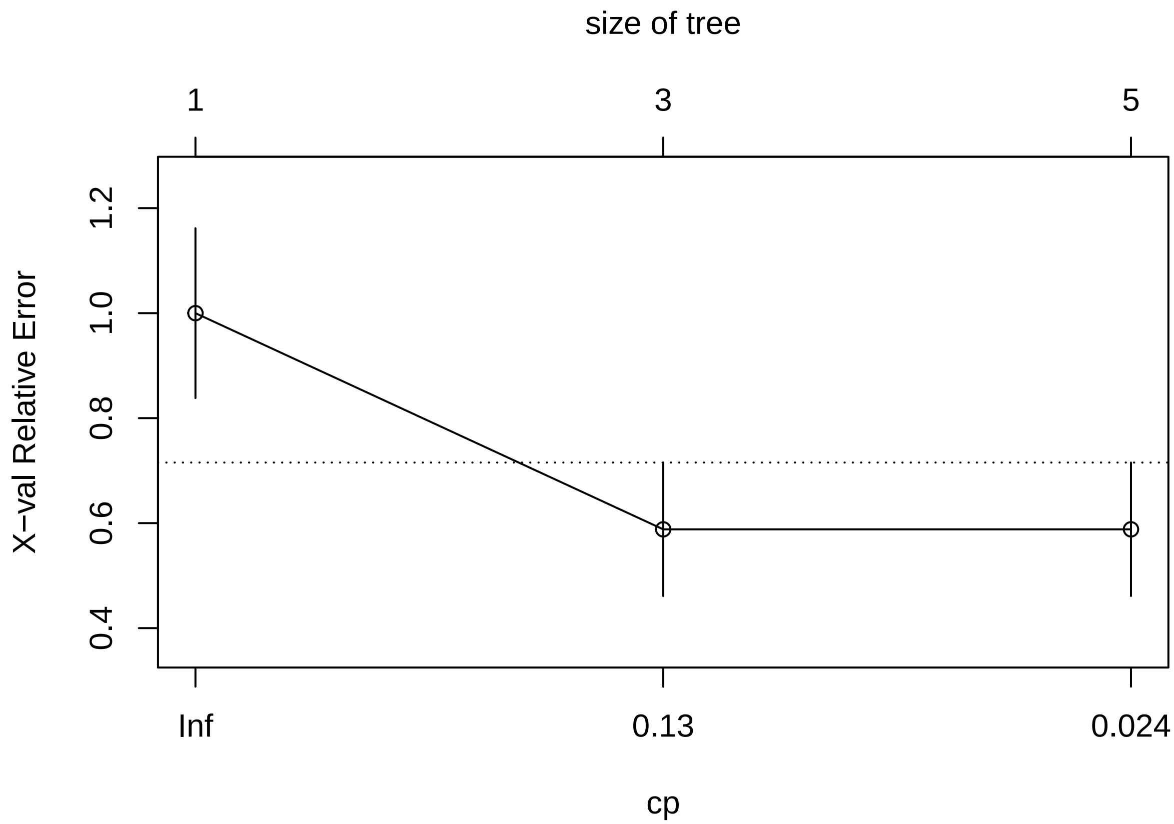

```r
# Using the rattle packages
fancyRpartPlot(tree.fit)
```




```
# display the results
printcp(tree.fit)
```

```
##
## Classification tree:
## rpart(formula = Sports ~ Retail + Dealer + Engine + Cylinders +
##     Horsepower + CityMPG + HighwayMPG + Weight + Wheelbase +
##     Length + Width, data = cars04.train.data, method = "class")
##
## Variables actually used in tree construction:
## [1] Horsepower Retail     Wheelbase
##
## Root node error: 34/310 = 0.10968
##
## n= 310
##
##          CP nsplit rel error  xerror    xstd
## 1 0.294118      0   1.00000 1.00000 0.16182
## 2 0.058824      2   0.41176 0.58824 0.12722
## 3 0.010000      4   0.29412 0.58824 0.12722
```

```
# visualize cross-validation results
plotcp(tree.fit)
```




```r
# detailed summary of splits
summary(tree.fit)
```

```
## Call:
## rpart(formula = Sports ~ Retail + Dealer + Engine + Cylinders +
##     Horsepower + CityMPG + HighwayMPG + Weight + Wheelbase +
##     Length + Width, data = cars04.train.data, method = "class")
##   n= 310
##
##           CP nsplit rel error    xerror      xstd
## 1 0.29411765      0 1.0000000 1.0000000 0.1618208
## 2 0.05882353      2 0.4117647 0.5882353 0.1272197
## 3 0.01000000      4 0.2941176 0.5882353 0.1272197
##
## Variable importance
##     Retail Horsepower     Dealer  Wheelbase      Width    CityMPG
##         14         13         13         12         11         10
## HighwayMPG     Length     Weight     Engine  Cylinders
##          9          7          4          4          2
##
## Node number 1: 310 observations,    complexity param=0.2941176
##   predicted class=0  expected loss=0.1096774  P(node) =1
##     class counts:   276    34
##    probabilities: 0.890 0.110
##   left son=2 (246 obs) right son=3 (64 obs)
##   Primary splits:
##       Wheelbase  < 102.5   to the right, improve=12.731720, (0 missing)
##       Retail     < 75880   to the left,  improve=11.449810, (0 missing)
```

```
##       Dealer     < 69198.5 to the left,  improve= 9.848602, (0 missing)
##       Length     < 177.5   to the right, improve= 7.497176, (0 missing)
##       Horsepower < 256.5   to the left,  improve= 6.378697, (0 missing)
##   Surrogate splits:
##       Length     < 167.5   to the right, agree=0.884, adj=0.438, (0 split)
##       Weight     < 3163    to the right, agree=0.852, adj=0.281, (0 split)
##       Horsepower < 126.5   to the right, agree=0.842, adj=0.234, (0 split)
##       Engine     < 1.95    to the right, agree=0.839, adj=0.219, (0 split)
##       Retail     < 14775   to the right, agree=0.829, adj=0.172, (0 split)
##
## Node number 2: 246 observations,    complexity param=0.05882353
##   predicted class=0  expected loss=0.03658537  P(node) =0.7935484
##     class counts:   237     9
##    probabilities: 0.963 0.037
##   left son=4 (194 obs) right son=5 (52 obs)
##   Primary splits:
##       Horsepower < 270.5   to the left,  improve=2.4568480, (0 missing)
##       Length     < 180.5   to the right, improve=1.0178080, (0 missing)
##       Retail     < 56172.5 to the left,  improve=0.8531517, (0 missing)
##       Dealer     < 50459.5 to the left,  improve=0.8531517, (0 missing)
##       HighwayMPG < 26.5    to the right, improve=0.5414634, (0 missing)
##   Surrogate splits:
##       Retail    < 46182.5 to the left,  agree=0.907, adj=0.558, (0 split)
##       Dealer    < 41433.5 to the left,  agree=0.898, adj=0.519, (0 split)
##       Cylinders < 7       to the left,  agree=0.898, adj=0.519, (0 split)
##       Engine    < 4.1     to the left,  agree=0.890, adj=0.481, (0 split)
##       CityMPG   < 14.5    to the right, agree=0.833, adj=0.212, (0 split)
##
## Node number 3: 64 observations,    complexity param=0.2941176
##   predicted class=0  expected loss=0.390625  P(node) =0.2064516
##     class counts:    39    25
##    probabilities: 0.609 0.391
##   left son=6 (36 obs) right son=7 (28 obs)
##   Primary splits:
##       Retail     < 24432.5 to the left,  improve=21.66716, (0 missing)
##       Dealer     < 22391.5 to the left,  improve=21.66716, (0 missing)
##       Width      < 68.5    to the left,  improve=21.37158, (0 missing)
##       Horsepower < 205     to the left,  improve=21.32156, (0 missing)
##       CityMPG    < 21.5    to the right, improve=18.92757, (0 missing)
##   Surrogate splits:
##       Dealer     < 22391.5 to the left,  agree=1.000, adj=1.000, (0 split)
##       Horsepower < 167.5   to the left,  agree=0.938, adj=0.857, (0 split)
##       CityMPG    < 21.5    to the right, agree=0.906, adj=0.786, (0 split)
##       HighwayMPG < 29.5    to the right, agree=0.906, adj=0.786, (0 split)
##       Width      < 68.5    to the left,  agree=0.906, adj=0.786, (0 split)
##
## Node number 4: 194 observations
##   predicted class=0  expected loss=0  P(node) =0.6258065
##     class counts:   194     0
##    probabilities: 1.000 0.000
##
## Node number 5: 52 observations,    complexity param=0.05882353
##   predicted class=0  expected loss=0.1730769  P(node) =0.1677419
##     class counts:    43     9
```

```
##    probabilities: 0.827 0.173
##   left son=10 (38 obs) right son=11 (14 obs)
##   Primary splits:
##       Wheelbase < 109.5   to the right, improve=8.456044, (0 missing)
##       Length    < 180.5   to the right, improve=6.875092, (0 missing)
##       Weight    < 3555.5  to the right, improve=4.738584, (0 missing)
##       Engine    < 4.05    to the right, improve=3.869094, (0 missing)
##       Cylinders < 7       to the right, improve=2.646520, (0 missing)
##   Surrogate splits:
##       Length    < 181.5   to the right, agree=0.942, adj=0.786, (0 split)
##       Weight    < 3650    to the right, agree=0.865, adj=0.500, (0 split)
##       Engine    < 3.7     to the right, agree=0.846, adj=0.429, (0 split)
##       Width     < 70.5    to the right, agree=0.846, adj=0.429, (0 split)
##       Cylinders < 7       to the right, agree=0.808, adj=0.286, (0 split)
##
## Node number 6: 36 observations
##   predicted class=0  expected loss=0.02777778  P(node) =0.116129
##     class counts:    35     1
##    probabilities: 0.972 0.028
##
## Node number 7: 28 observations
##   predicted class=1  expected loss=0.1428571  P(node) =0.09032258
##     class counts:     4    24
##    probabilities: 0.143 0.857
##
## Node number 10: 38 observations
##   predicted class=0  expected loss=0  P(node) =0.1225806
##     class counts:    38     0
##    probabilities: 1.000 0.000
##
## Node number 11: 14 observations
##   predicted class=1  expected loss=0.3571429  P(node) =0.04516129
##     class counts:     5     9
##    probabilities: 0.357 0.643
```

```r
# make predictions from the tree
tree.pred <- predict(tree.fit, cars04.test.data, type = "class")
cars04.test.data.pred <- cars04.test.data %>% mutate(Sports.pred = tree.pred)
# Performance
tab <- table(cars04.test.data.pred$Sports, cars04.test.data.pred$Sports.pred)
tab
```

```
##
##       0  1
##   0 64  2
##   1  3  8
```

```r
# Sensitivity TP/(TP + FN)
tab[2,2]/(tab[2,2] + tab[2,1])
```

```
## [1] 0.7272727
```

```r
# Specificity  TN/(TN + FP)
tab[1,1]/(tab[1,1] + tab[1,2])
```

```
## [1] 0.969697
```

```r
# Precision TP/predicted yes.
tab[2,2]/(tab[2,2] + tab[1,2])
```

```
## [1] 0.8
```

```r
# Accuracy (TP + TN)/(All)
(tab[2,2] + tab[1,1])/(tab[2,2] + tab[1,1] + tab[1,2] + tab[2,1])
```

```
## [1] 0.9350649
```

2 Bagging

Bagging combines and averages multiple models. Averaging across multiple trees reduces the variability of any one tree and reduces overfitting, which improves predictive performance. Steps:

- Create m bootstrapped samples from the training data
- For each bootstrapped sample train, make an unpruned tree model
- Average the prediction for each tree to create an overall average

Bagging will be implemented with the ipred and caret package.

```r
# train bagged model
bagged.reg.tree.fit <- bagging(
  formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
    Horsepower + Weight + Wheelbase + Length + Width,
  data    = cars04.train.data,
  coob    = TRUE
)
bagged.reg.tree.fit
```

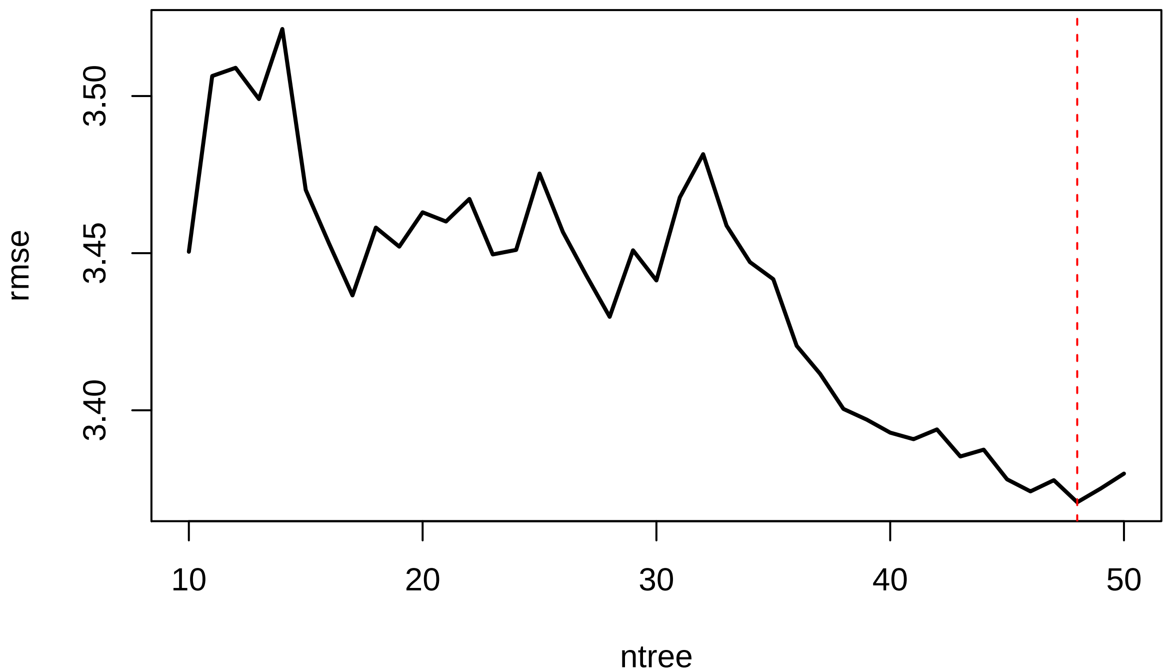
```
##
## Bagging regression trees with 25 bootstrap replications
##
## Call: bagging.data.frame(formula = HighwayMPG ~ Retail + Dealer + Engine +
##     Cylinders + Horsepower + Weight + Wheelbase + Length + Width,
##     data = cars04.train.data, coob = TRUE)
##
## Out-of-bag estimate of root mean squared error:  3.4163
```

```r
# By default the number of trees used is 25 but this might not be enough
# assess 10-50 bagged trees
ntree <- 10:50
# create empty vector to store OOB RMSE values
rmse <- vector(mode = "numeric", length = length(ntree))

for (i in seq_along(ntree)) {
  # reproducibility
  set.seed(123)

  # perform bagged model
  model <- bagging(
    formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
      Horsepower + Weight + Wheelbase + Length + Width,
    data    = cars04.train.data,
    coob    = TRUE,
```

```r
    nbagg   = ntree[i]
  )
  # get OOB error
  rmse[i] <- model$err
}
plot(ntree, rmse, type = 'l', lwd = 2)
abline(v = 48, col = "red", lty = "dashed")
```




```r
# Use 45 trees instead of 25
opt.bagged.reg.tree.fit <- bagging(
  formula = HighwayMPG ~ Retail + Dealer + Engine + Cylinders +
    Horsepower + Weight + Wheelbase + Length + Width,
  data    = cars04.train.data,
  coob    = TRUE,
  nbagg = 48
)
opt.bagged.reg.tree.fit
```

```
##
## Bagging regression trees with 48 bootstrap replications
##
## Call: bagging.data.frame(formula = HighwayMPG ~ Retail + Dealer + Engine +
##     Cylinders + Horsepower + Weight + Wheelbase + Length + Width,
##     data = cars04.train.data, coob = TRUE, nbagg = 48)
##
## Out-of-bag estimate of root mean squared error:  3.3648
```

3 Boosting

Several supervised machine learning models are founded on a single predictive model (i.e. linear regression, penalized models, naive Bayes, support vector machines). Alternatively, other approaches such as bagging and random forests are built on the idea of building an ensemble of models where each individual model predicts the outcome and then the ensemble simply averages the predicted values. The family of boosting methods is based on a different, constructive strategy of ensemble formation. The main idea of boosting is to add new models to the ensemble sequentially. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.