Decision Trees

James D. Wilson

April 2nd, 2018

In this presentation, we will investigate how to use the following classification methods:

1) Classification Trees
2) Regression Trees
3) Random Forests

Much of this lab is a replication of Section 8.3 in the "Introduction to Statistical Learning" textbook by James, Witten, Hastie, and Tibshirani (with my own commentary throughout).

Classification and Regression Trees rely on the package *tree*. Furthermore, we'll be using decision trees on several data sets in the *ISLR* package. We first install and load these packages.

```r
install.packages("tree", repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpfGU1gX/downloaded_packages
```

```r
install.packages("ISLR", repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpfGU1gX/downloaded_packages
```

```r
library(tree, quietly = TRUE)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```r
library(ISLR, quietly = TRUE)
```

```
## Warning: package 'ISLR' was built under R version 3.4.2
```

Classification Trees

We are first going to use classification trees to analyze the *Carseats* data in the *ISLR* package. We will focus on classifying the *Sales* of different car seats. Here, the *Sales* variable is continuous. As a result, we first binarize *Sales* and create a categorical variable which specifies which car seats had a *High* number of sales.

```r
#look at the data
?Carseats

attach(Carseats)

#Create a binary variable specifying whether or not the Sales of the carseat was High (> 8)

High <- ifelse(Sales <= 8, "No", "Yes")
```

```r
#Merge the binary variable with the remaining data set
Carseats <- data.frame(Carseats, High)
```

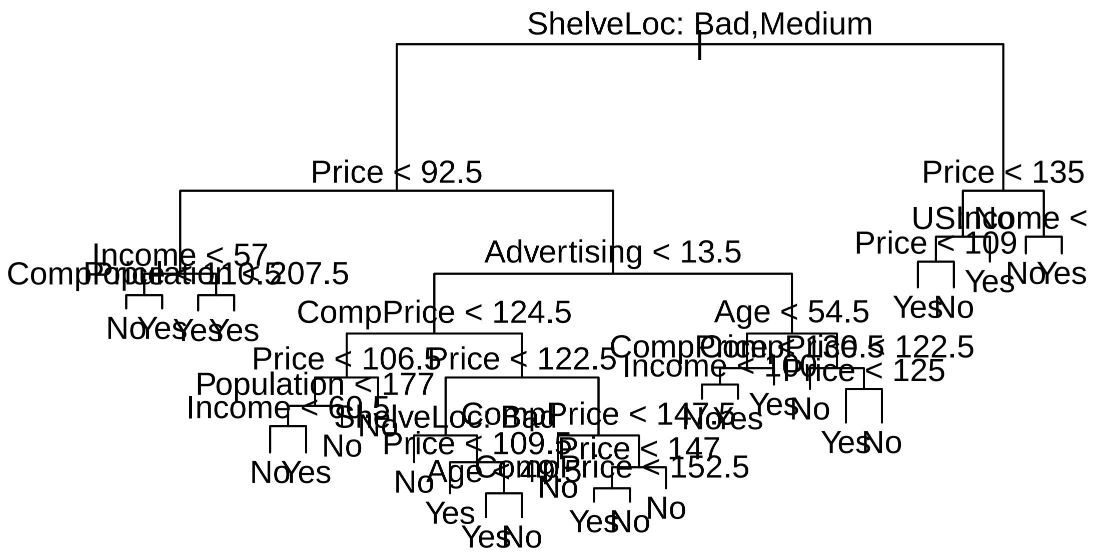
Now we use the *tree()* function to fit a classification tree in order to predict *High* using all variables except *Sales*. We then look at a summary of the fitted tree.

```r
tree.carseats <- tree(High ~.-Sales, Carseats)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

We can now also plot the decision tree for visualization. Note that we have to add text to the tree using the *text()* function in R.

```r
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```



```r
#Look at the regions and breakdown of the High variable in the tree
tree.carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##            44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##        12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
##        14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

Now we run this on a training set and leave out a test set for later validation.

```r
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train ,]
High.test <- High[-train]
tree.carseats <- tree(High ~.-Sales, Carseats, subset=train)
```

```r
#Look at predictons on the test set
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")

#Evaluate misclassifications
table(tree.pred, High.test)
```

```
##          High.test
## tree.pred No  Yes
##       No  86  27
##       Yes 30  57
```

```r
#Calculate the accuracy
mean(tree.pred == High.test)
```

```
## [1] 0.715
```

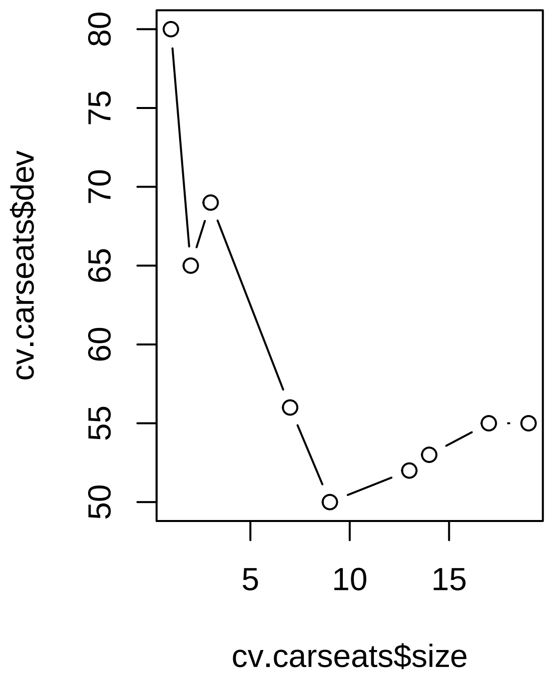
Next, we use cross validation to determine how the tree should be pruned using **weakest-link pruning, *or **cost complexity pruning**. Here, the parameter* k* corresponds to the α we referred to in class.

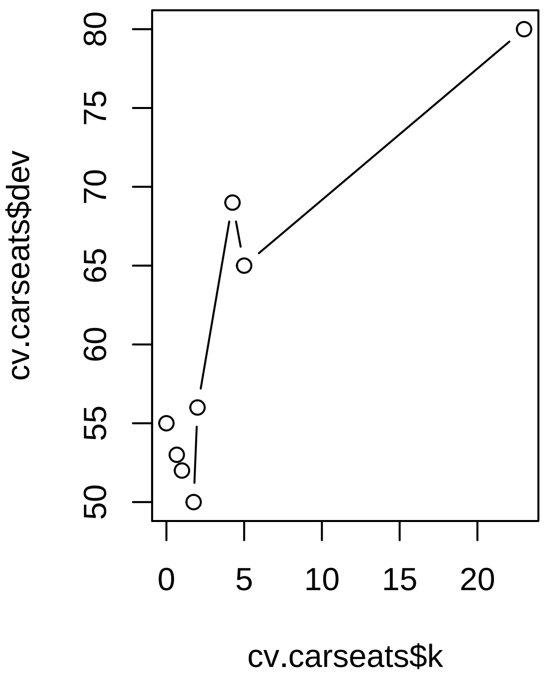
```r
set.seed(3)

cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)

#look at the results
cv.carseats
```

```
## $size
## [1] 19 17 14 13  9  7  3  2  1
##
## $dev
## [1] 55 55 53 52 50 56 69 65 80
##
## $k
## [1]       -Inf  0.0000000  0.6666667  1.0000000  1.7500000  2.0000000
## [7]  4.2500000  5.0000000 23.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

Above, *dev* corresponds to the cross-validation error in this instance. We see that the tree with 9 terminal nodes results in the lowest cross validation rate. We plot the error now as a function of *size* and *k*

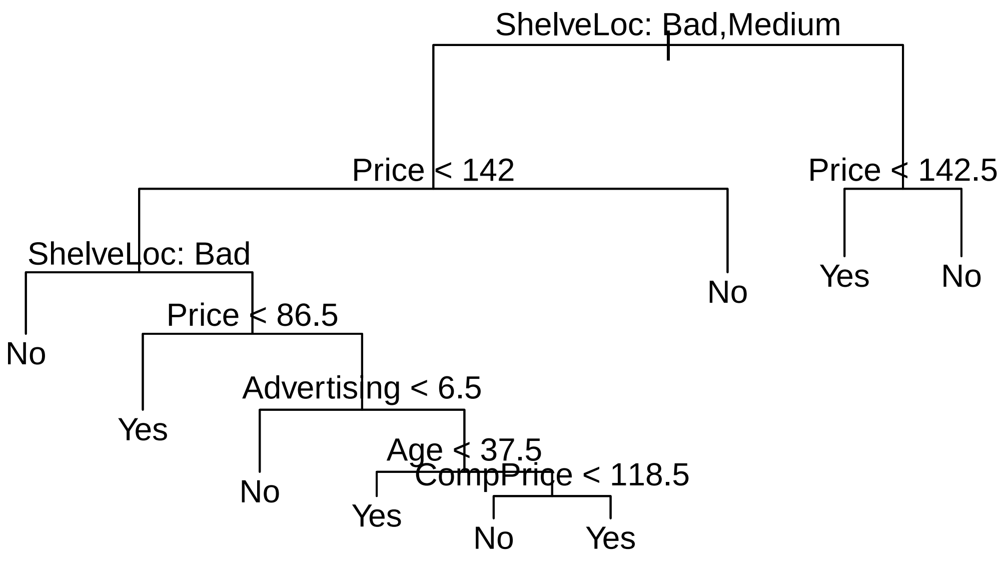
```r
par(mfrow = c(1, 2))
plot(cv.carseats$size, cv.carseats$dev, type = "b")
plot(cv.carseats$k, cv.carseats$dev, type = "b")
```





We can now use the function *prune.misclass()* to prune the tree to the nine-node tree.

```
prune.carseats <- prune.misclass(tree.carseats, best = 9)
plot(prune.carseats)
text(prune.carseats, pretty = 0)
```



Finally, we evaluate how well our pruned tree performed on the test set.

```
#Make predictions on the test set
prune.pred <- predict(prune.carseats, Carseats.test, type = "class")

#Look at confusion matrix
table(prune.pred, High.test)
```

```
##             High.test
## prune.pred No Yes
##        No  94  24
##        Yes 22  60
```

```
#Calculate the accuracy
mean(prune.pred == High.test)
```

```
## [1] 0.77
```

Regression Trees

Regression trees work the same way as classification trees and rely on the function *tree*. Below, we fit a regression tree using the *Boston* data from the *MASS* package.

```
install.packages("MASS", repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpfGU1gX/downloaded_packages
```

```
library(MASS, quietly = TRUE)
```

```
## Warning: package 'MASS' was built under R version 3.4.3
```

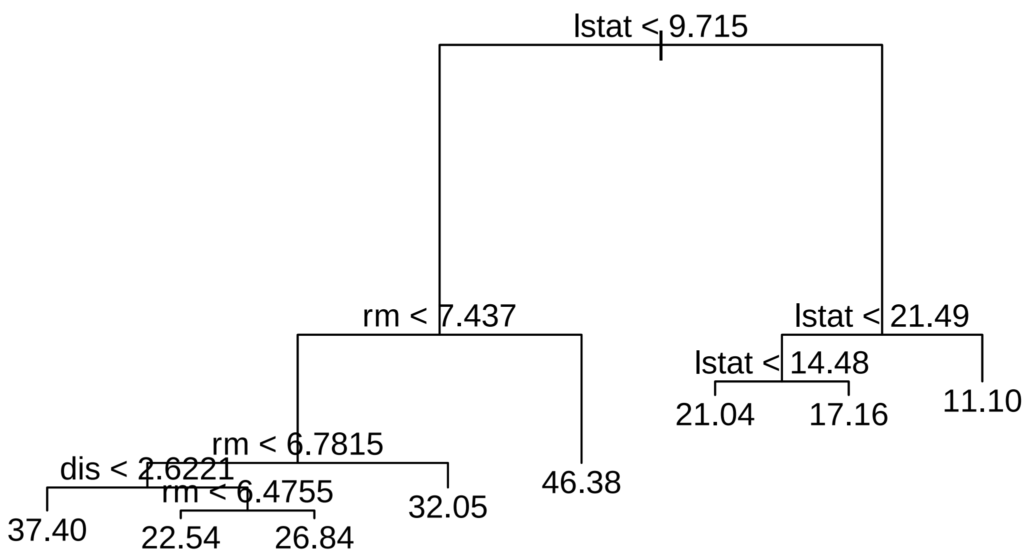
```
#Investigate the Boston data
?Boston
```

Below we will now build, prune, and assess the fit of a regression tree following the same template we did above for classification trees. Here, we regress the median value of owner-occupied homes *medv* against the remaining predictors.

```
#Randomly split the data into a training and test set
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)

#Fit a tree on the training data
tree.boston <- tree(medv ~ ., data = Boston, subset = train)
summary(tree.boston)
```

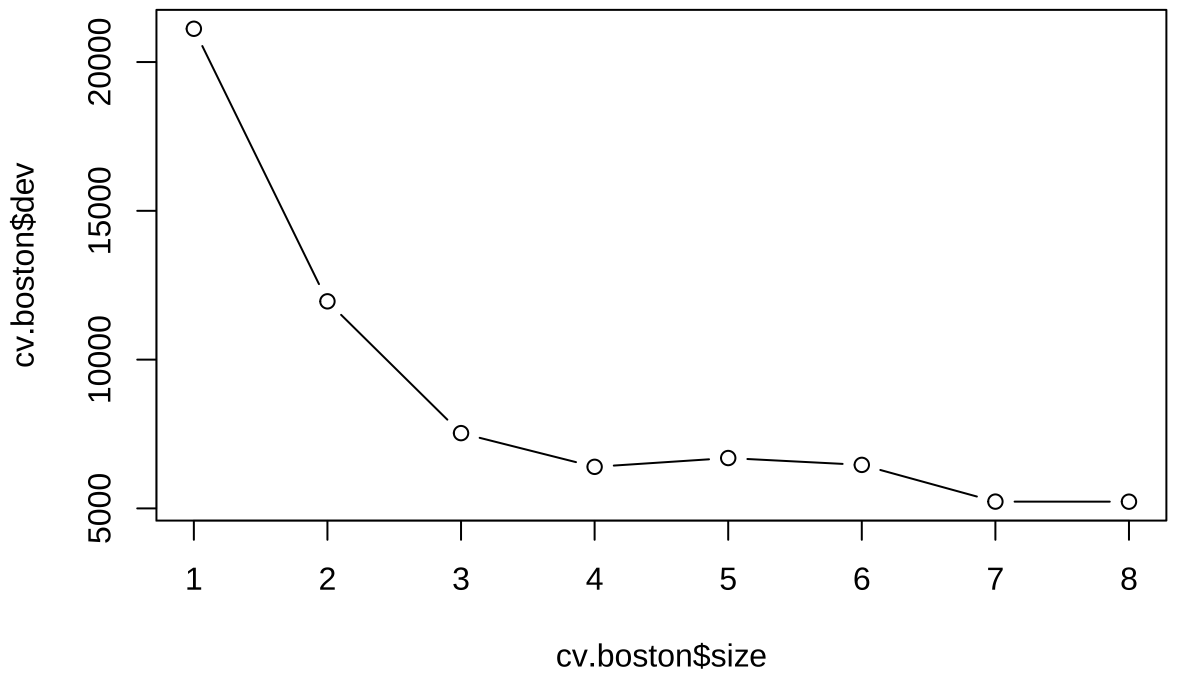
```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "lstat" "rm"    "dis"
## Number of terminal nodes:  8
## Residual mean deviance:  12.65 = 3099 / 245
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

```
#Visualize the tree
plot(tree.boston)
text(tree.boston, pretty = 0)
```



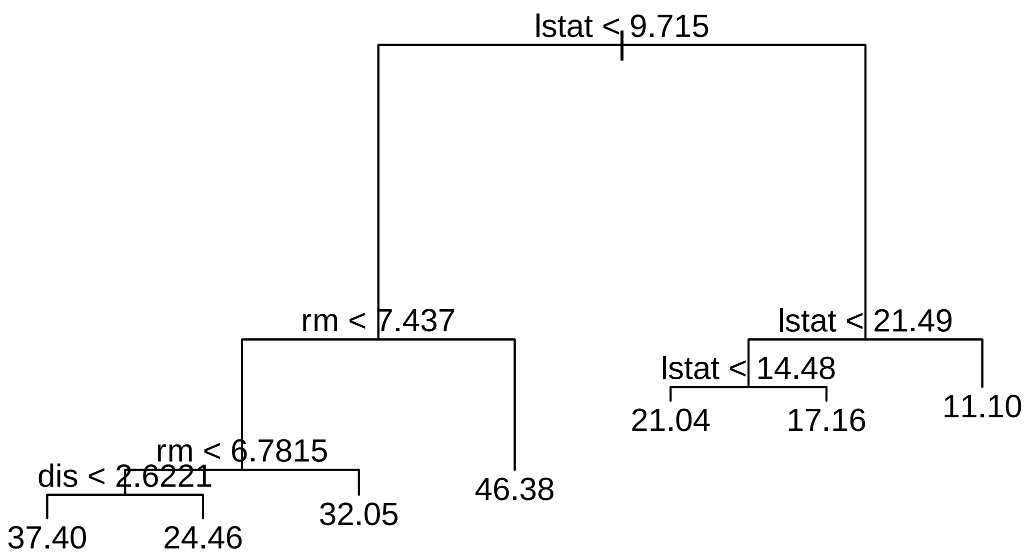
We perform pruning on this originally fit tree now below.

```r
cv.boston <- cv.tree(tree.boston)
#Plot the cross-validation error against the size of the pruned tree
plot(cv.boston$size, cv.boston$dev, type = "b")
```



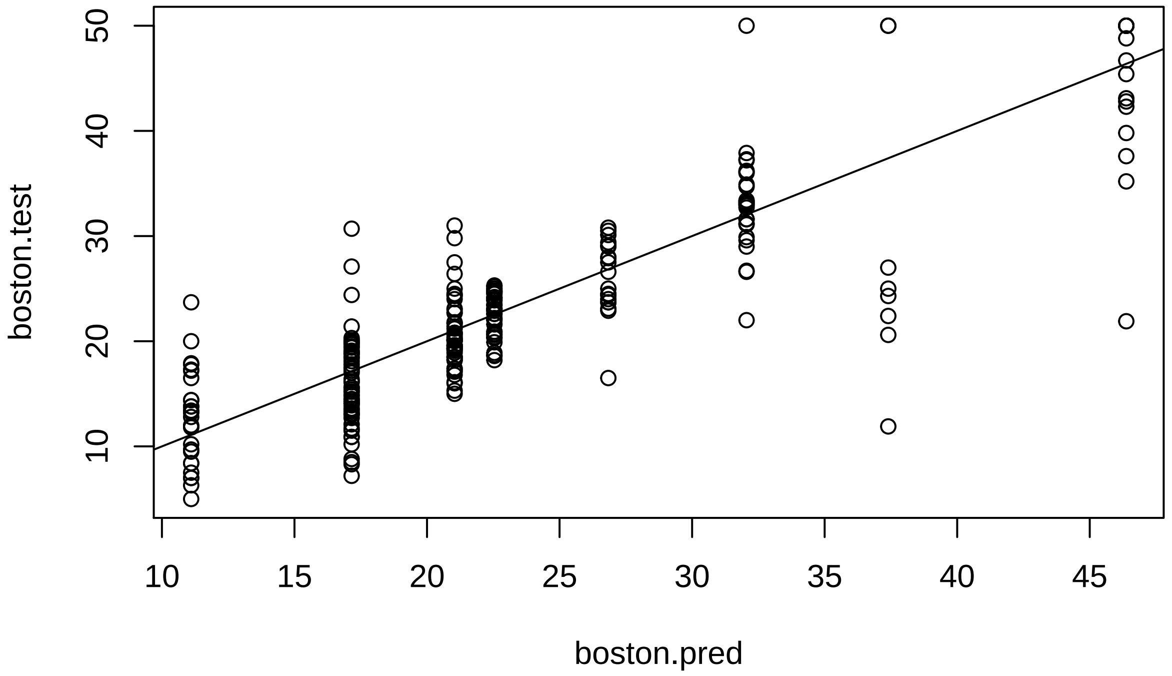
It looks like the tree of size 7 has the lowest cross-validation error. So, we prune the tree to have 7 terminal nodes.

```r
prune.boston <- prune.tree(tree.boston, best = 7)
plot(prune.boston)
text(prune.boston, pretty = 0)
```



Now we assess the performance of the tree on the left-out test set.

```
boston.test = Boston[-train, "medv"]
boston.pred <- predict(tree.boston, newdata = Boston[-train, ])

#Plot the predicted against the truth
plot(boston.pred, boston.test)
abline(0, 1)
```



```
#Calculate the MSPE on the test set
mean((boston.pred - boston.test)^2)
```

```
## [1] 25.04559
```

Random Forests

In this section, we apply random forests to the *Boston* data using the *RandomForest* package in R. First, we install and load the needed library.

```
install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Installing package into '/Users/jdwilson4/Library/R/3.4/library'
## (as 'lib' is unspecified)

##
## The downloaded binary packages are in
##  /var/folders/hm/8gnvskgx0rb1c11fzmz7sgf82j1yqg/T//RtmpfGU1gX/downloaded_packages
```

```
library(randomForest, quietly = TRUE)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.
```

The primary function that we will use is the *randomForest()* function. This funciton follows the same syntax as the *tree()* function above; except, we will need to determine how many predictors to use for each tree. This argument is given by *mtry*. By default, the *randomForest()* function uses $m = p/3$ for regression trees and $m = \sqrt(p)$ for classification trees. Furthermore, the *ntree* argument specifies the number of trees to be generated. You should consider choosing the number of trees based on performance of the random forest.

An important feature of random forests is its ability to measure the *importance* of predictors using measurements of

a) the reduction in MSPE
b) the increase in node purity / response purity

Below, we fit a random forest of regression trees on the *Boston* data using all 13 predictors and 500 trees (Note: this is in fact **bagging** since we are using all 13 predictors).

```
set.seed(1)
bag.boston <- randomForest(medv ~ ., data = Boston, subset = train, mtry = 13, importance = TRUE, ntree
bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      ntree = 500, subs
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 11.15723
##                     % Var explained: 86.49
```

```
#Prediction on test set
yhat.bag <- predict(bag.boston, newdata = Boston[-train,])

#Measure MSPE
mean((yhat.bag - boston.test)^2)
```

```
## [1] 13.50808
```

Now we fit a random forest using $\sqrt{13} \approx 4$ predictors. We then investigate its performance on the test set.

```r
set.seed(1)
rf.boston <- randomForest(medv ~ ., data = Boston, subset = train, mtry = 4, importance = TRUE)
#Prediction on test set
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])

#Measure MSPE
mean((yhat.rf - boston.test)^2)
```

```
## [1] 11.6076
```

We now use the *importance()* function to get an idea of the importance of each predictor in the random forest. This function gives two statistics that are calculated on the out-of-bag samples:

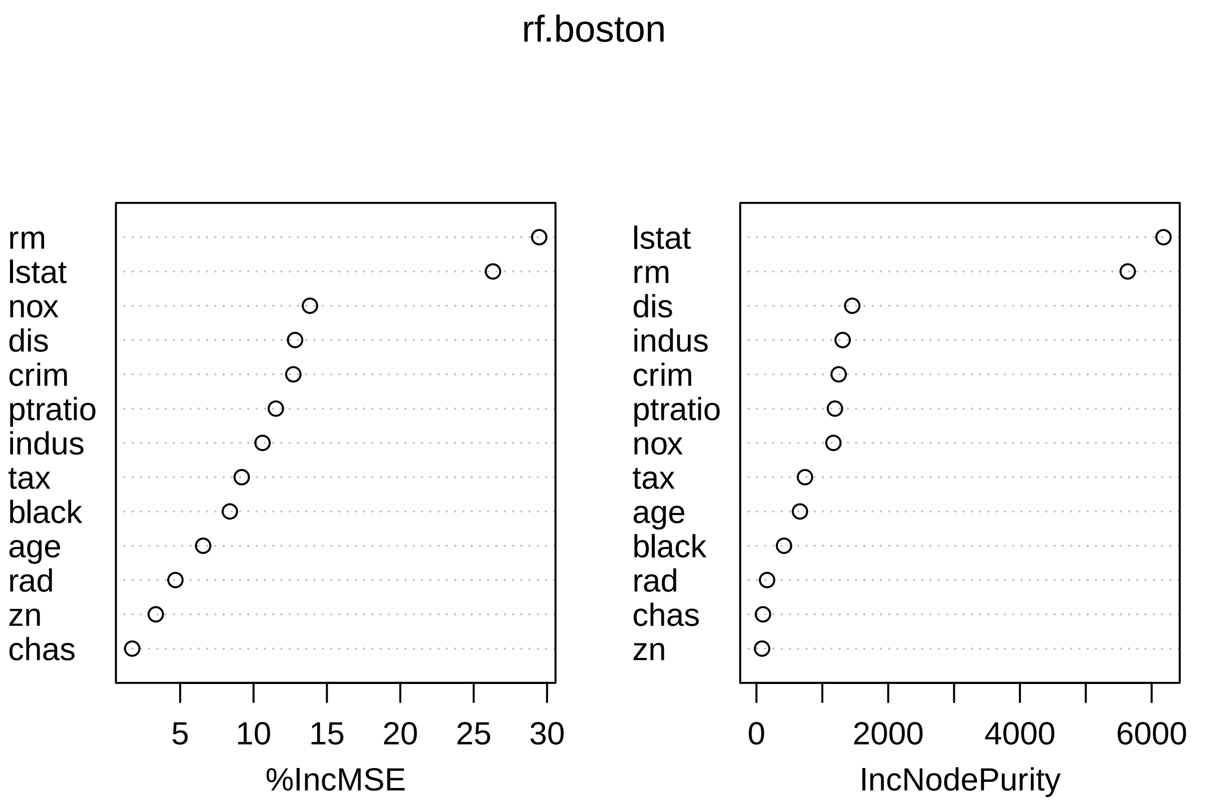
a) *IncMSE*: the average increase in MSPE when the given variable is not included in the model
b) *IncNodePurity*: the average increase in node purity when there is a split on this variable in any decision tree.

So, variables with large values of each of these statistics can be deemed *important*.

```r
#Calculate the importance of each predictor
importance(rf.boston)
```

```
##            %IncMSE IncNodePurity
## crim     12.712371    1247.87271
## zn        3.340046      84.97987
## indus    10.611878    1308.70340
## chas      1.733390      98.53459
## nox      13.843947    1168.44941
## rm       29.465247    5636.39110
## age       6.565646     660.34542
## dis      12.830180    1453.81733
## rad       4.679142     161.38834
## tax       9.195031     736.79049
## ptratio 11.521584    1191.12896
## black     8.385275     418.42770
## lstat    26.313510    6179.11950
```

```r
#Plot these importance measures
varImpPlot(rf.boston)
```



From the above results, we see that the predictors *lstat* and *rm* are by far the most important variables.