Classification and Regression Tree, CART

Jason

2015 年 7 月 30 日

```
library(ISLR); library(MASS); library(tree); library(ggplot2)
library(reshape2); library(randomForest); library(gbm)
```

Regression Tree

```
#Data in MASS package
data(Boston)
str(Boston)
```

```
'data.frame':   506 obs. of  14 variables:
 $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
 $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
 $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
 $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
 $ rm     : num  6.58 6.42 7.18 7 7.15 ...
 $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
 $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
 $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
 $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
 $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
 $ black  : num  397 397 393 395 397 ...
 $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
 $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
Bostree <- tree(medv ~ ., data=Boston, subset=train)
Bostree
```

```
node), split, n, deviance, yval
      * denotes terminal node

 1) root 253 20890.0 22.67
   2) lstat < 9.715 103  7765.0 30.13
     4) rm < 7.437 89  3310.0 27.58
       8) rm < 6.7815 61  1995.0 25.52
        16) dis < 2.6221 5   615.8 37.40 *
        17) dis > 2.6221 56   610.3 24.46
          34) rm < 6.4755 31   136.4 22.54 *
          35) rm > 6.4755 25   218.3 26.84 *
       9) rm > 6.7815 28   496.6 32.05 *
     5) rm > 7.437 14   177.8 46.38 *
```

```
   3) lstat > 9.715 150  3465.0 17.55
     6) lstat < 21.49 120  1594.0 19.16
      12) lstat < 14.48 62   398.5 21.04 *
      13) lstat > 14.48 58   743.3 17.16 *
     7) lstat > 21.49 30   311.9 11.10 *
```

```r
summary(Bostree)
```

```
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
```

In the Regression tree, the deviance is defined as below (i.e. RSS):

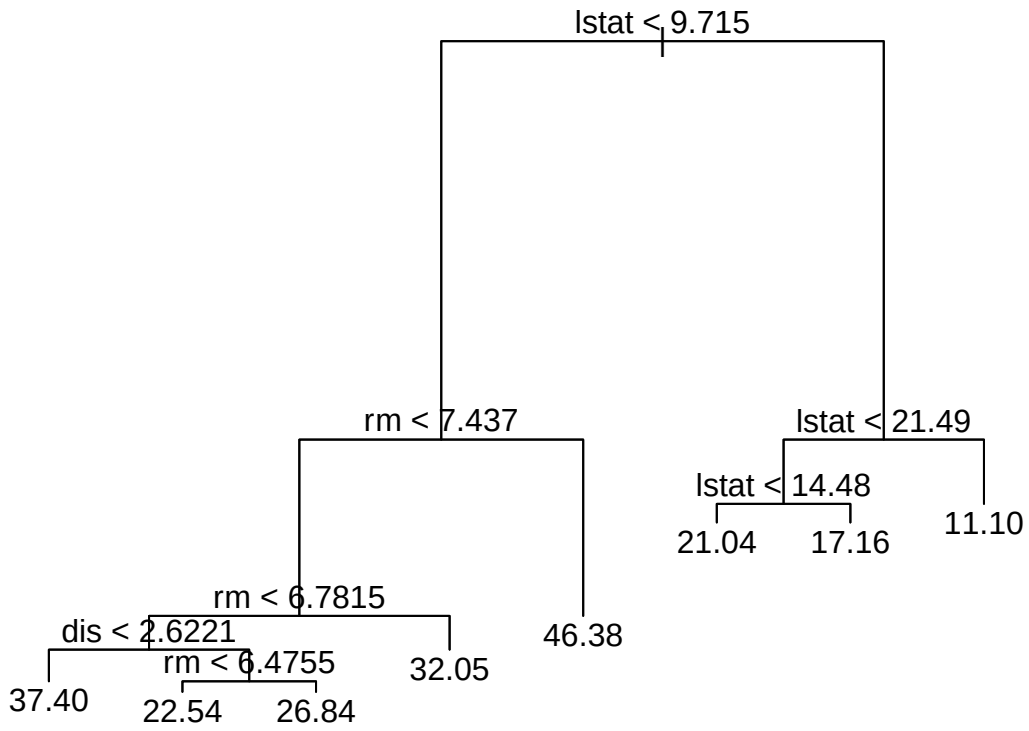
$$D_{node} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

It measures the node impurity (disorder).

When fitting a tree, we take a *top down, greedy* approach, known as *recursive binary splitting*. Start from the top of the tree, in each step, we try to make a *best* split according to the criteria such as regression deviance.

We should illustrate more about criteria here. First, we can decided whether make a split by the decreasing amount of RSS (In percentage, `minsdev`). On the other hand, we can also make the minimum number of the node after cut or child node (`minicut`) or minimum number of the node before cut or parent node (`minisize`).
You can use `?tree.control` in R to see detailed explaination.

```r
plot(Bostree)
text(Bostree, pretty=0)
```



The height of tree of each node reflect the quantity of the deviance.

```
str(cv.tree)
```

```
function (object, rand, FUN = prune.tree, K = 10, ...)
```

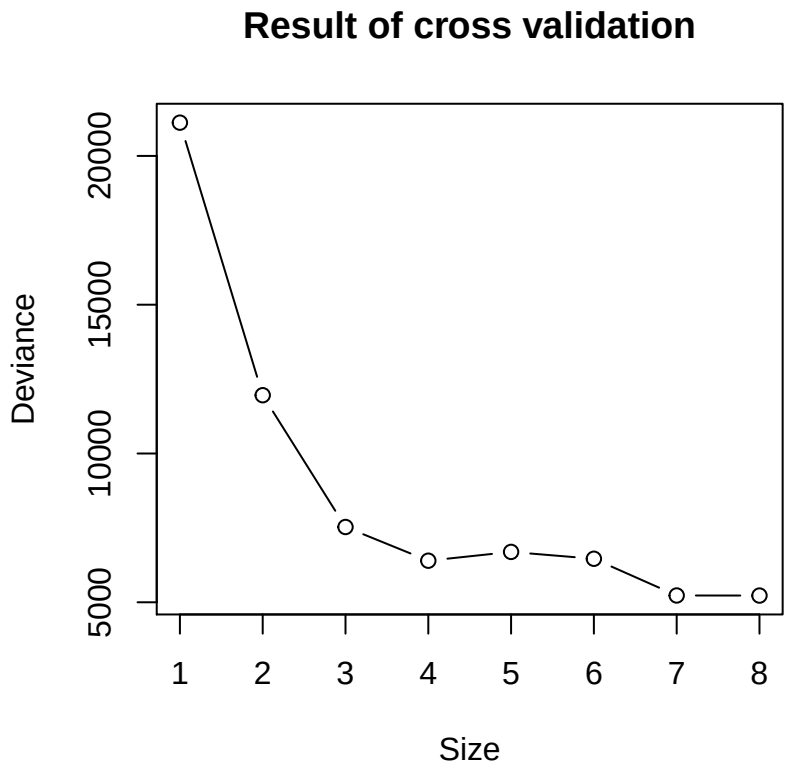
Tree prune is also a very important stage in tree model. If the tree is too large, the model may overfit the data. Therefore, a smaller tree will be more preferable because of the lower variance and easier interpretation at the cost of some bias.

One way to prune tree is described as above that we grow the tree as long as the split cause a moderate decreasing of deviance beyond a threshold. However, it would be too short-sighted. Normally, we will grow a big tree and then prune it. *Cost complexity pruning* or *Weakest link prune* can help us to do this. We add a nonnegative constant α in the deviance term to give penalty to those subtree with many nodes. The function is shown below:

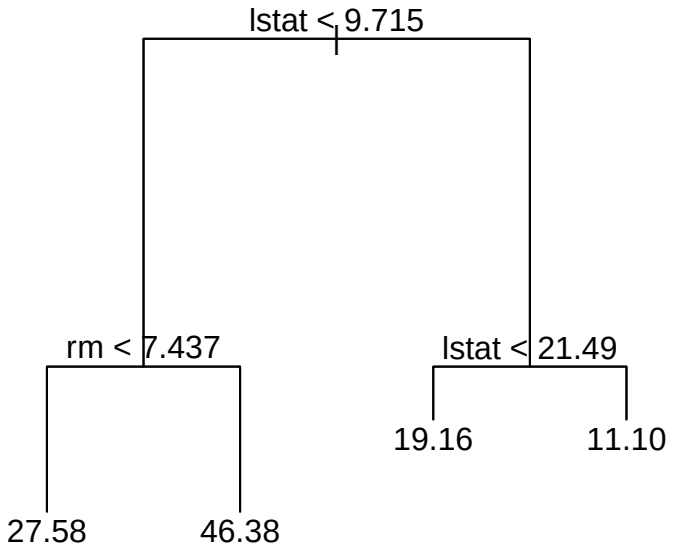
$$D_T = \sum_{m=1}^{|T|} \sum_{x_i \in y_{R_m}} (y_i - 2\bar{y}_{R_m})^2 + \alpha|T|$$

Like chosing the λ in lasso, we can use cross validation to decide the alpha.

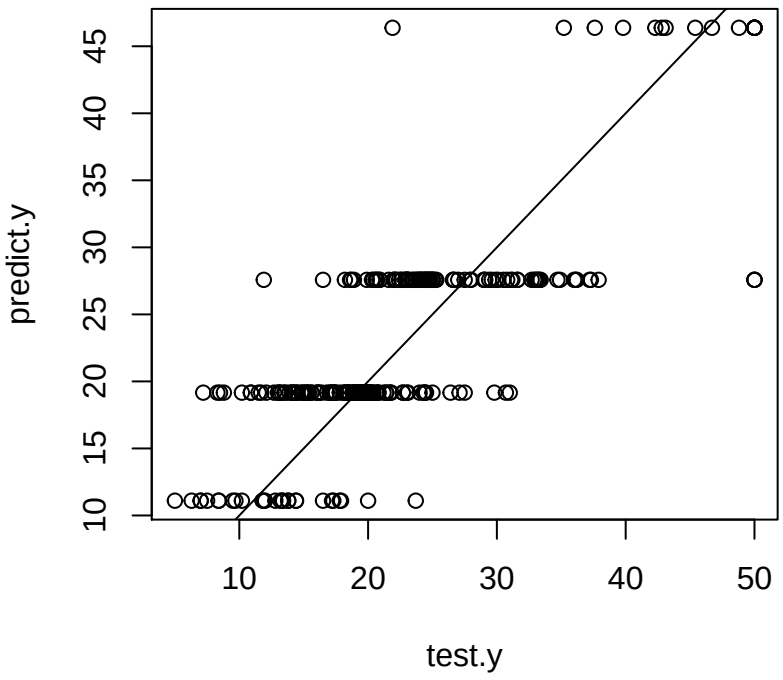
```r
cv.result <- cv.tree(Bostree)
plot(cv.result$size, cv.result$dev, type="b",
     main="Result of cross validation",
     xlab="Size", ylab="Deviance")
```




```r
fit <- prune.tree(Bostree, best=4)
plot(fit)
text(fit, pretty=0)
```




```
test.y <- Boston$medv[-train]
test.x <- Boston[-train, ]
predict.y <- predict(fit, newdata=test.x)
plot(test.y, predict.y)
abline(0, 1)
```




```
mean((test.y - predict.y)^2)
```

```
[1] 32.22697
```

Classification

In the Classification tree, the deviance is defined as below:

$$D_{node} = -2\sum_{k=1}^{m} n_k log(\frac{n_k}{n}) = -2[\sum_{k=1}^{m} n_k log(n_k) - nlog(n)]$$

In classification tree, the measure of disorder have several ways such as classification error rate, Gini Index and cross-entropy.

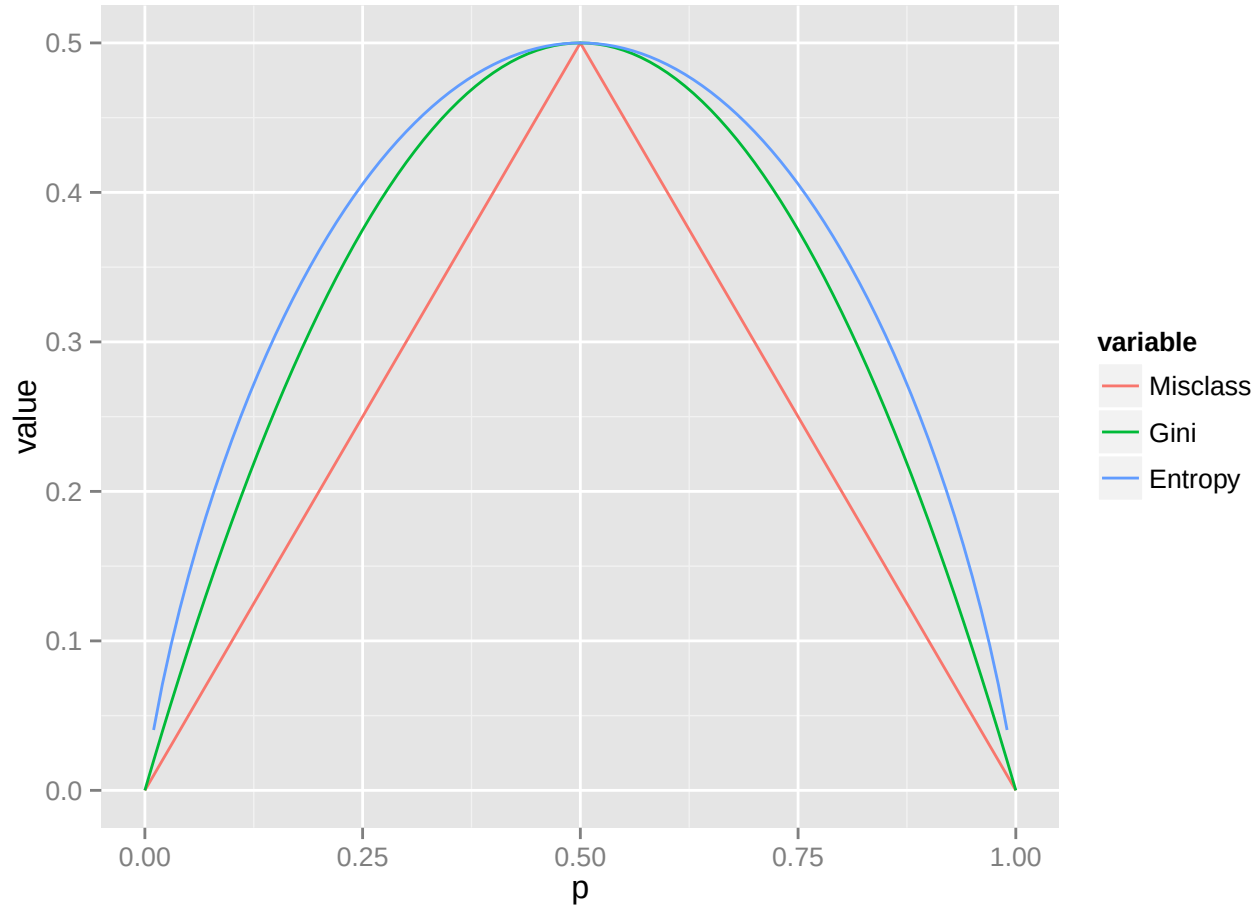
```
misclass <- function(x){
  min(x, 1 - x)
}
gini <- function(x){
  2*x*(1 - x)
}
entropy <- function(x){
  -x*log(x) - (1 - x)*log(1 - x)
}
```

```r
p <- seq(0, 1, by=0.01)

demo <- data.frame(p=p, Misclass=sapply(p, misclass),
                   Gini=gini(p), Entropy=entropy(p)/log(2)*0.5)
demo2 <- melt(demo, id="p")

ggplot(demo2) +
geom_line(aes(x=p, y=value, class=variable, col=variable))
```

```
Warning: Removed 2 rows containing missing values (geom_path).
```



When growing a tree, Gini and entropy will be more sensitive that classification error rate. For instance, suppose that we have 400 observation in two group, (400, 400), if one split creats nodes as region1 (300, 100) and region2 (100, 300), while another split creats nodes as region1 (200, 400) and region2 (200, 0), then gini and entropy will choose the one will lower impurity, which is the latter split.

In first split, $p_{11} = 0.75 = p_{22}$ and $p_{12} = 0.25 = p_{21}$. In second split, $p_{11} = 0.33$, $p_{12} = 0.67$, $p_{21} = 1$ and $p_{22} = 0$.

1. Misclassification: both classification error rate is 0.25.

2. Gini: Split1 => node1 0.75× 0.25 + 0.25× 0.75 = **0.375**, node2 0.25× 0.75 + 0.75× 0.25 = **0.375**. Split2 => node1 0.33× 0.67 + 0.67× 0.33 = **0.4422**, node2 0× 1 + 1× 0 = **0**.
3. entropy: Split1 => node1 -0.75× log(0.75)- 0.25× log(0.25) = **0.5623**, node2 -0.25× log(0.25) - 0.75× log(0.75) = **0.5623**. Split2 => node1 -0.33× log(0.33) 0 - 0.67× log(0.67) = **0.6342**, node2 -1× log(1) - 0× log(0) = **0**.

```
data(Carseats)
str(Carseats)
```

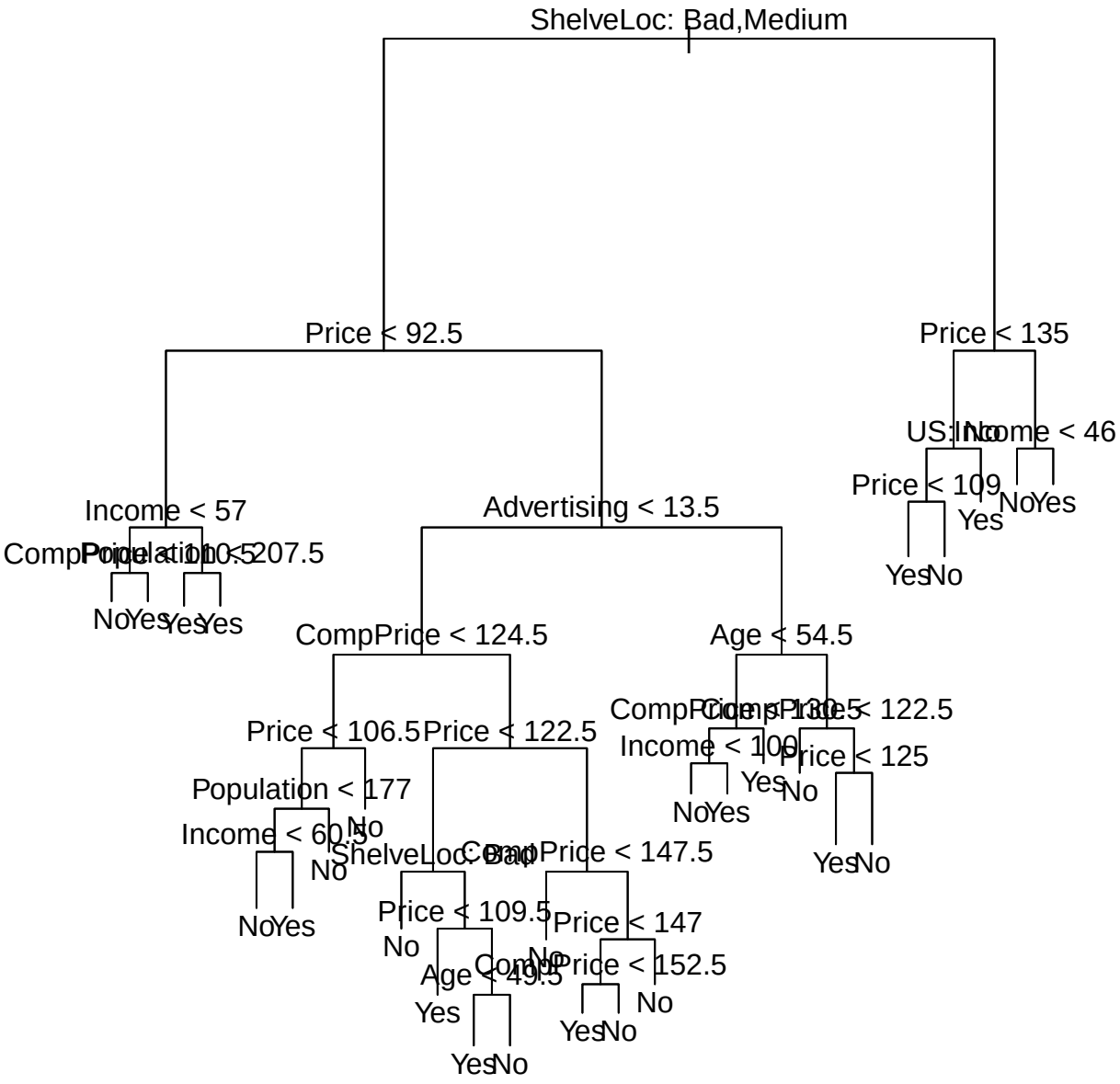
```
'data.frame':   400 obs. of  11 variables:
 $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
 $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
 $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
 $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
 $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
 $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
 $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
 $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
 $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
 $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
 $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

```
High <- ifelse(Carseats$Sales <= 8, "No", "Yes")
Carseats <- data.frame(Carseats, High)
```

```
Cartree <- tree(High ~ . - Sales, data=Carseats)
summary(Cartree)
```

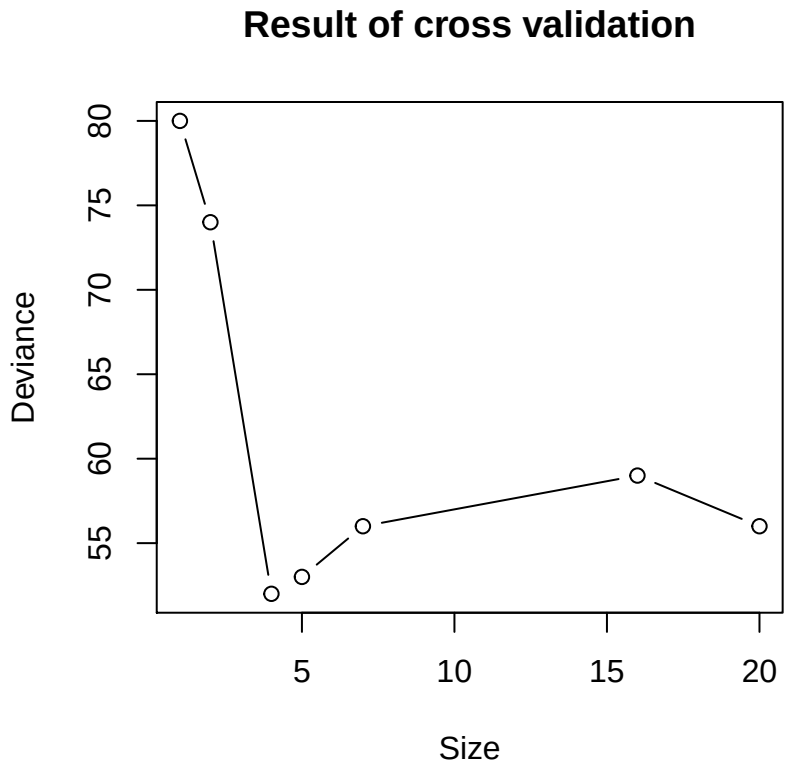
```
Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
[6] "Advertising" "Age"         "US"
Number of terminal nodes:  27
Residual mean deviance:  0.4575 = 170.7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

```
plot(Cartree)
text(Cartree, pretty=0)
```

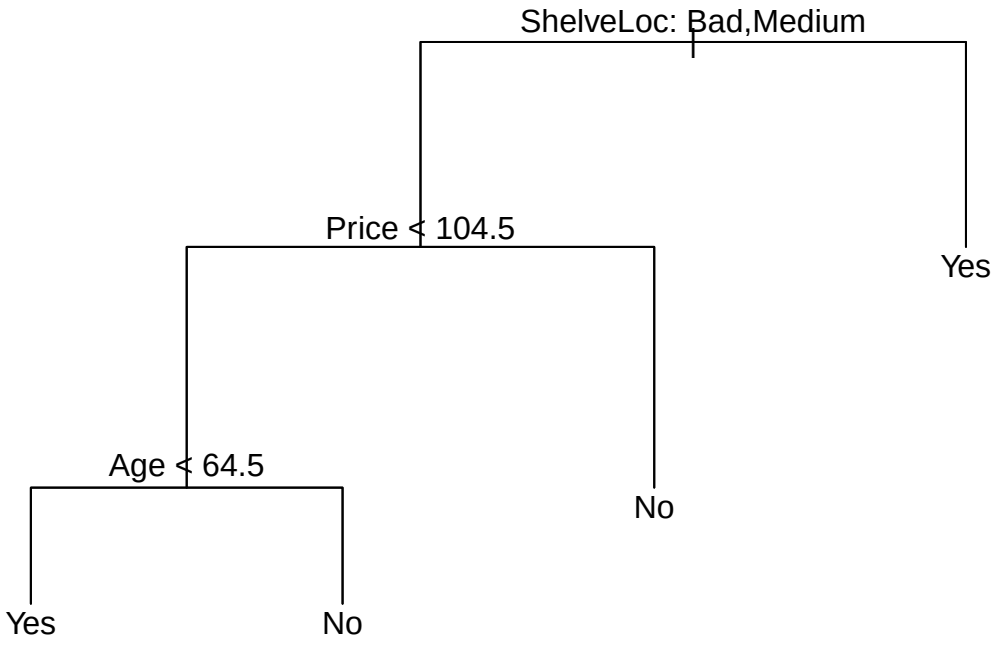



```
set.seed(1)
train <- sample(1:nrow(Carseats), nrow(Carseats)/2)
Cartree <- tree(High ~ . - Sales, data=Carseats, subset=train)
cv.result2 <- cv.tree(Cartree, FUN=prune.misclass)
```

```
plot(cv.result2$size, cv.result2$dev, type="b",
     main="Result of cross validation",
     xlab="Size", ylab="Deviance")
```




```
fit2 <- prune.misclass(Cartree, best=4)
plot(fit2)
text(fit2, pretty=0)
```




```
test.y <- High[-train]
test.x <- Carseats[-train, ]
predict.y <- predict(fit2, newdata=test.x, type="class")
table(Prediction=predict.y, True=test.y)
```

```
          True
Prediction No Yes
       No  92  29
       Yes 24  55
```

Bagging (Bootstrap Aggregation)

Tree may sometimes cause overfitting which means the model have a high variance. If we can fit model from many data sets and combine all the result, it may yield a better result. However, we do not have so many data. Therefore, bootstrap can help us somehow simulate lots of data. For regression tree, we can average the result directly. For classification tree, we will assign the observation with the majority class.

As for out of bag (OOB), it is a way that we can estimate the test error. One can show that in each bootstrap simulation, there are around two-thirds observation in it. Hence, we can predict those out-of-bag observations (each may probably have one-third predictions for the total bootstrap number) and average the result. Then we can get a OOB test error estimation.

```
#Use the function randomForest in library randomForest
#Get original train index
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
test.y <- Boston$medv[-train]
test.x <- Boston[-train, ]
#Initialize bagging process
set.seed(1)
Boston.bag <- randomForest(medv ~ ., data=Boston, mtry=13,
                           subset=train, importance=TRUE)
Boston.bag
```

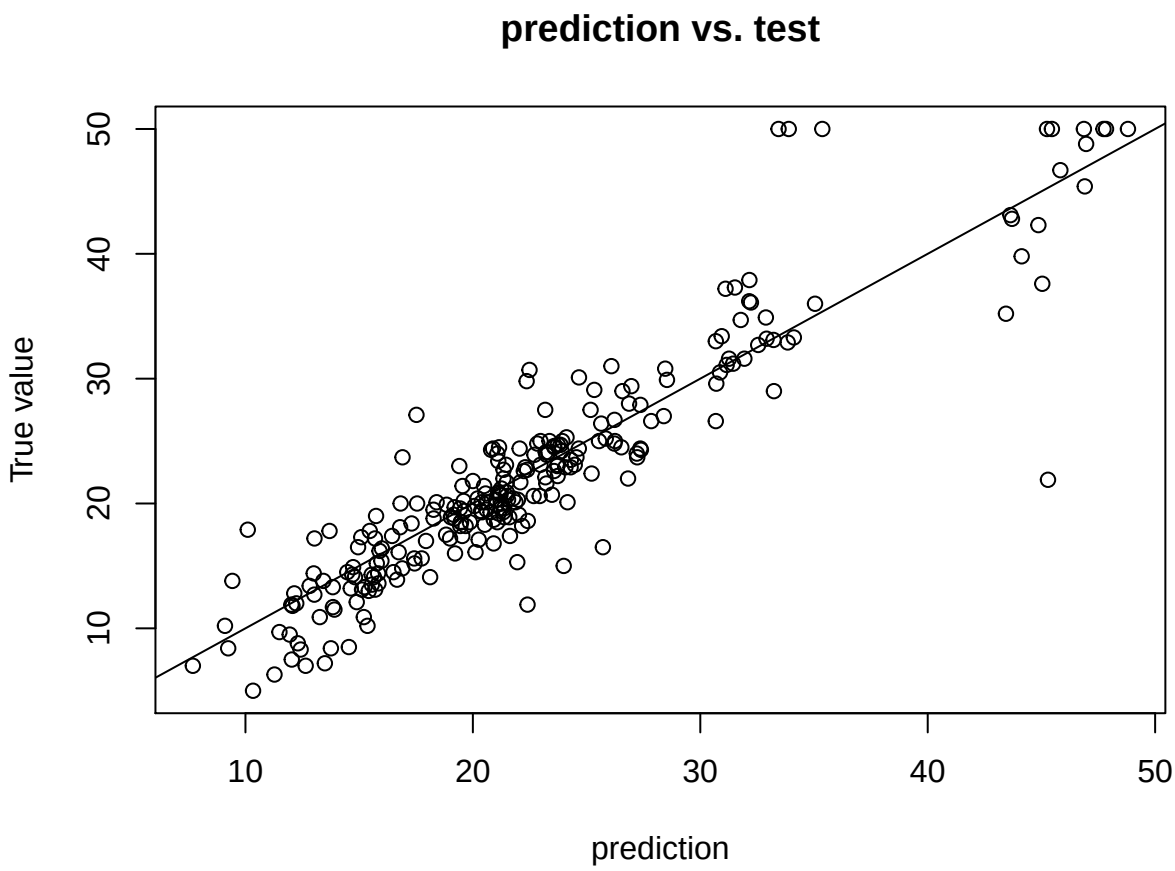
```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,     subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 11.02509
                    % Var explained: 86.65
```

```
prediction.y.bag <- predict(Boston.bag, newdata=Boston[-train, ])
mean((test.y - prediction.y.bag)^2)
```

```
[1] 13.47349
```

```
#Plot
plot(prediction.y.bag, test.y, main="prediction vs. test",
     xlab="prediction", ylab="True value")
abline(0, 1)
```



To prove that there are round two-thirds observation in each simulation data set, let's assume n observation in data set:

P(jth observation is not the first observation in Bootstrap data set)= $\frac{n-1}{n}$

P(jth observation is not the second observation in Bootstrap data set)= $\frac{n-1}{n}$

...

P(jth observation is not the nth observation in Bootstrap data set)= $\frac{n-1}{n}$

P(jth is not in the Bootstrap data set)= $(\frac{n-1}{n})^n = (1-\frac{1}{n})^n = (1+\frac{-1}{n})^n \approx e^{-1} = 0.3678794$ if n is large.

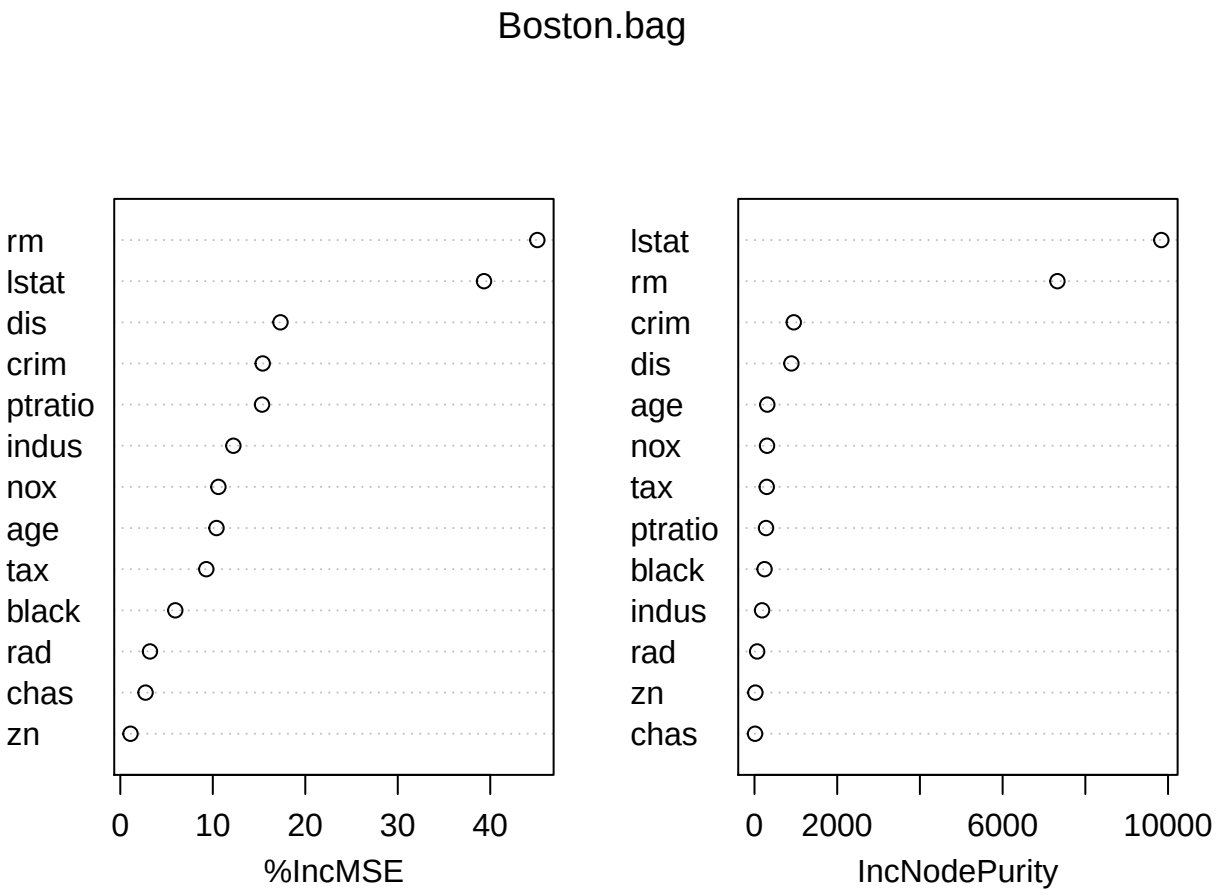
While bagging improve the prediction accuracy, it makes the model become complex and not-well explainable. But we can summarize each variable's importance by the amount they contribute. For given predictor, we can average the decreasing amount of RSS(Regression) or Gini index, cross-entropy(Classifiction). Then take the biggest as 100%, standardize each variable and get a over-all picture about which variable is much more importance than others.

```
importance(Boston.bag)
```

```
          %IncMSE IncNodePurity
crim    15.396510     950.03191
```

```
zn       1.100738      21.42389
indus   12.225351     183.14933
chas     2.726681      13.25062
nox     10.606485     302.78478
rm      45.090272    7325.33947
age     10.400796     309.19654
dis     17.315918     892.19354
rad      3.208664      64.56585
tax      9.296886     296.22083
ptratio 15.325244     279.25118
black    5.944955     243.04952
lstat   39.324555    9837.83280
```

```r
#Plot
varImpPlot(Boston.bag)
```



Random Forest

For random Forest, it is similar to bagging, but, when we consider each split, we random choose a set of m predictors. It can be kind of offset the effect of dominant variables and lead the result of decorrelating. Normally, m may be $\frac{p}{2}$ or $\frac{p}{3}$ or $\sqrt{p}$

```r
set.seed(1)
Boston.rf <- randomForest(medv ~ ., subset=train, mtry=6,
                          importance=TRUE, data=Boston)
Boston.rf
```

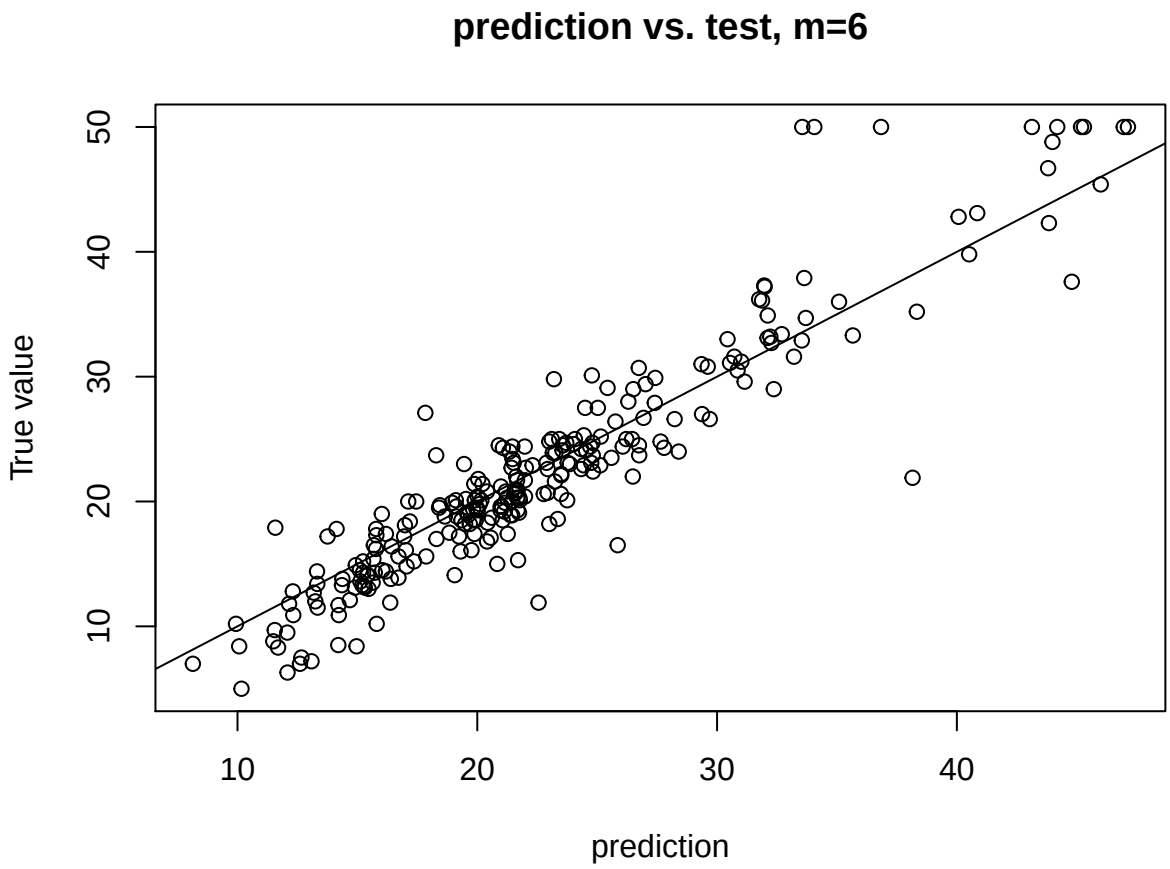
```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 6, importance = TRUE,     subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 6

          Mean of squared residuals: 12.09928
                    % Var explained: 85.35
```

```r
prediction.y.rf <- predict(Boston.rf, newdata=test.x)
mean((test.y - prediction.y.rf)^2)
```

```
[1] 11.48022
```

```r
#Plot
plot(prediction.y.rf, test.y, main="prediction vs. test, m=6",
     xlab="prediction", ylab="True value")
abline(0, 1)
```



Try m = $\sqrt{p}$

```
set.seed(1)
Boston.rf <- randomForest(medv ~ ., subset=train, mtry=4,
                          importance=TRUE, data=Boston)
Boston.rf
```

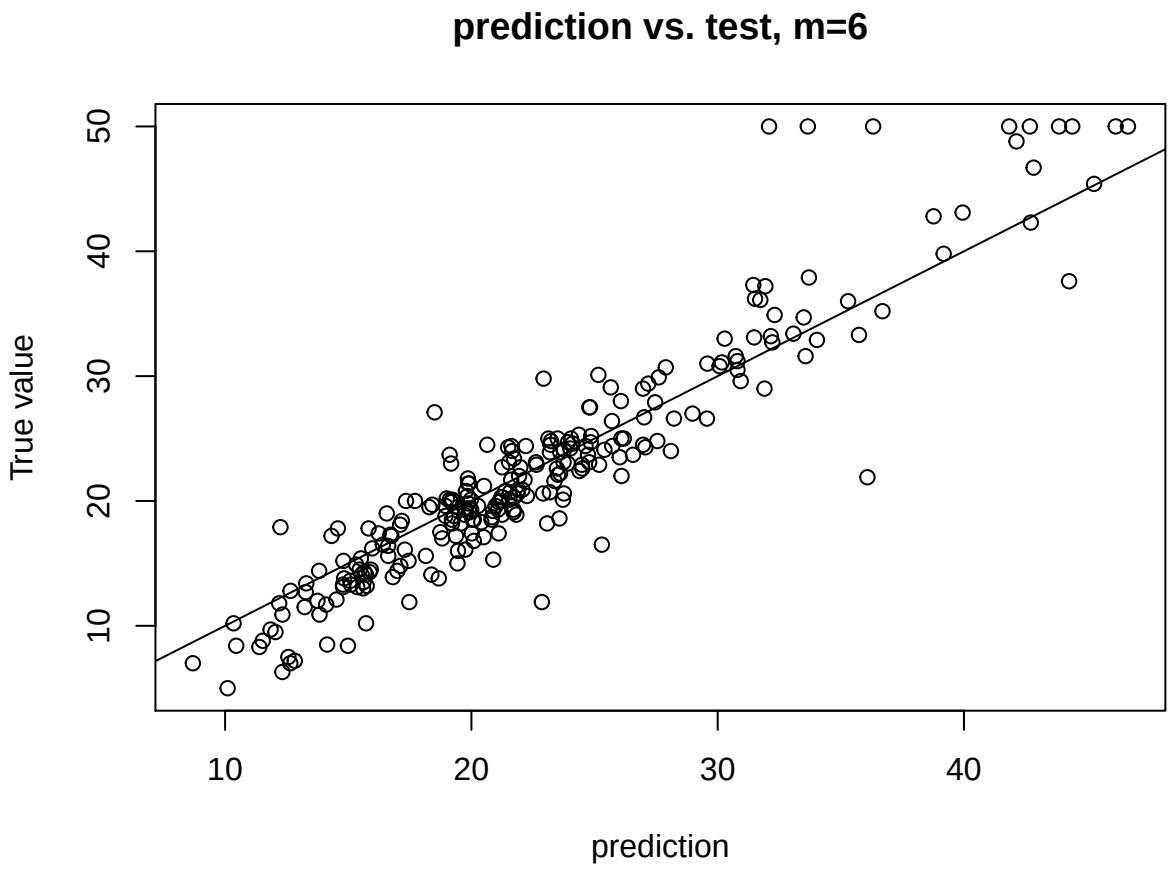
```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 4, importance = TRUE,     subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 4

          Mean of squared residuals: 12.09408
                    % Var explained: 85.36
```

```
prediction.y.rf <- predict(Boston.rf, newdata=test.x)
mean((test.y - prediction.y.rf)^2)
```

```
[1] 11.71882
```

```r
#Plot
plot(prediction.y.rf, test.y, main="prediction vs. test, m=6",
     xlab="prediction", ylab="True value")
abline(0, 1)
```



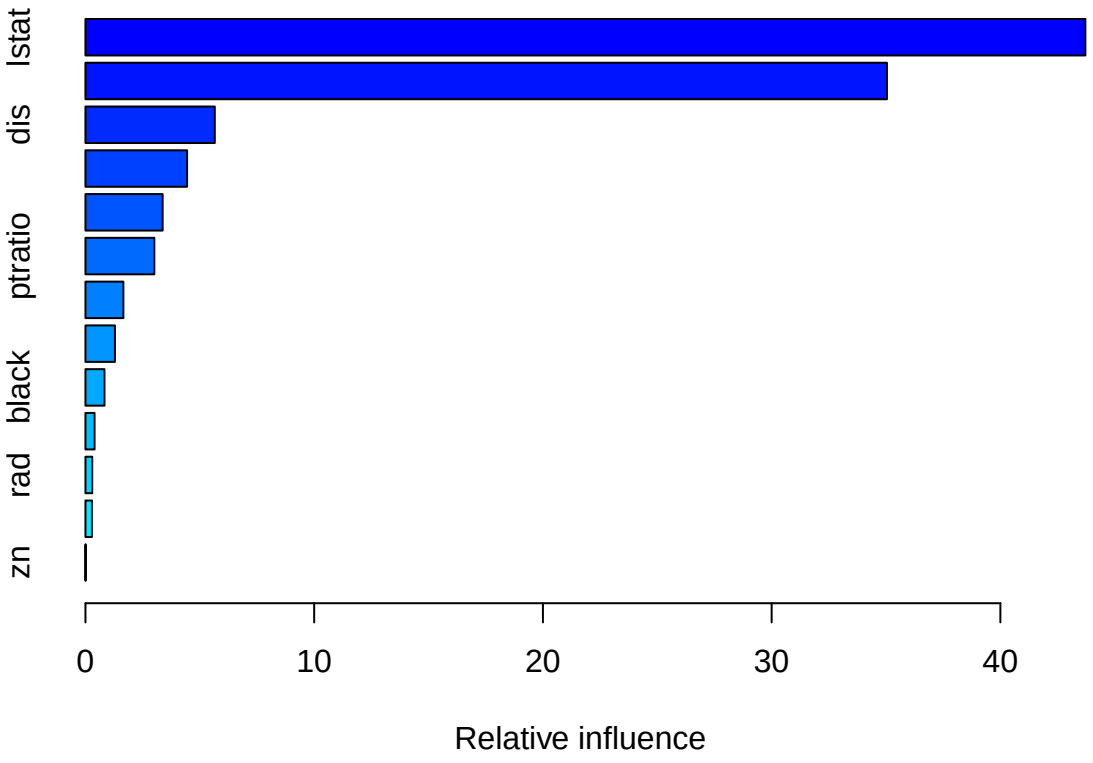
Boosting

Boosting is another way to enhacne the performance of tree. Its idea is to grow the tree slowly. After each split, we would compute their residual as our new response variable and fit a tree. It is a adaptive model. There are three parameters we should determine when using boosting. First, the number of tree, B. Because we do not want to overfit the data. Therefore we would not fit a big tree. We determine B by cross validation.

Second, the shrinkage parameterm, λ. It control the speed that boosting learns. Normally, λ is 0.01 or 0.001.

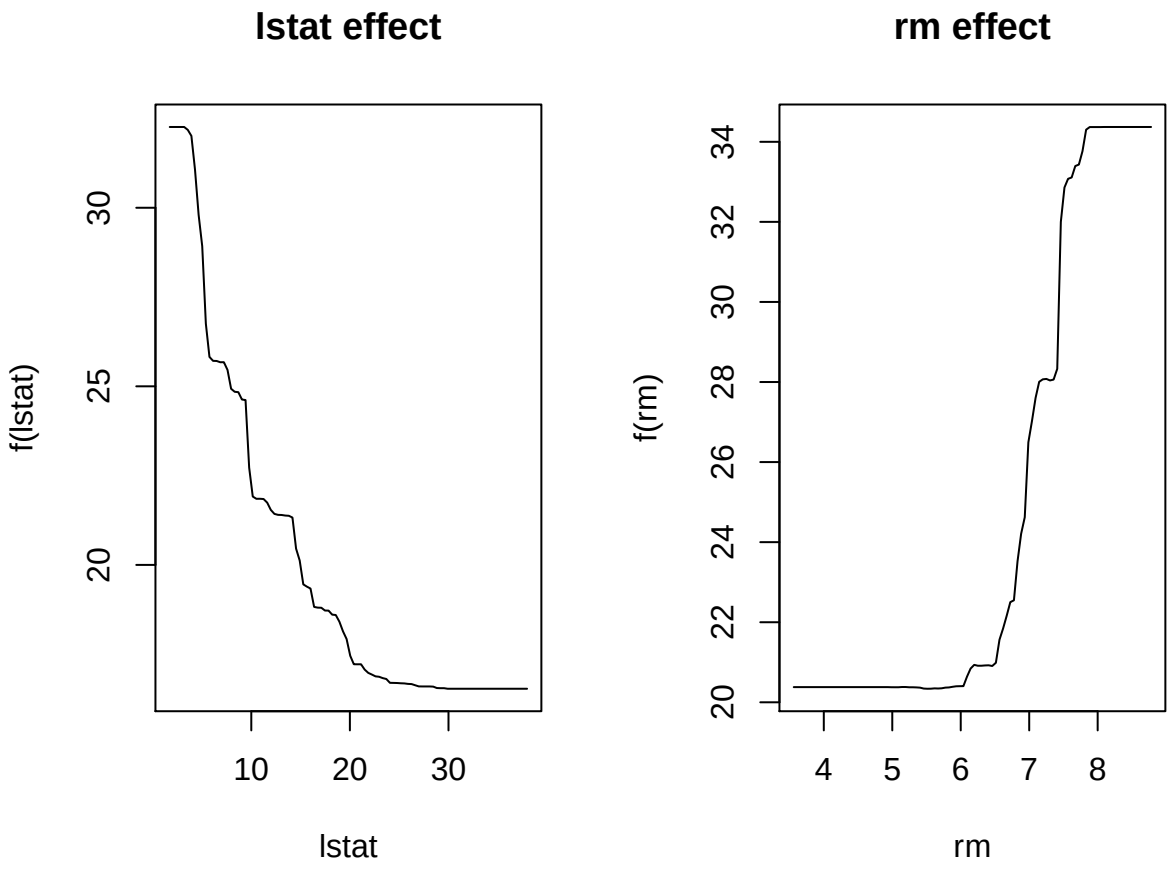
Third, the number of split in each tree, d. When we fit the tree, we can decide the depth of each one. Often d equal to 1 works well.

```r
set.seed(1)
Boston.boost <- gbm(medv ~ ., data=Boston, distribution="gaussian",
                    n.tree=5000, interaction.depth=4)
summary(Boston.boost)
```




```
            var      rel.inf
lstat     lstat 43.715813222
rm           rm 35.040223124
dis         dis  5.654810734
nox         nox  4.442533265
crim       crim  3.374343012
ptratio ptratio  3.009831539
age         age  1.657292679
tax         tax  1.288764835
black     black  0.830886519
chas       chas  0.397043555
rad         rad  0.295978908
indus     indus  0.283961554
zn           zn  0.008517056
```

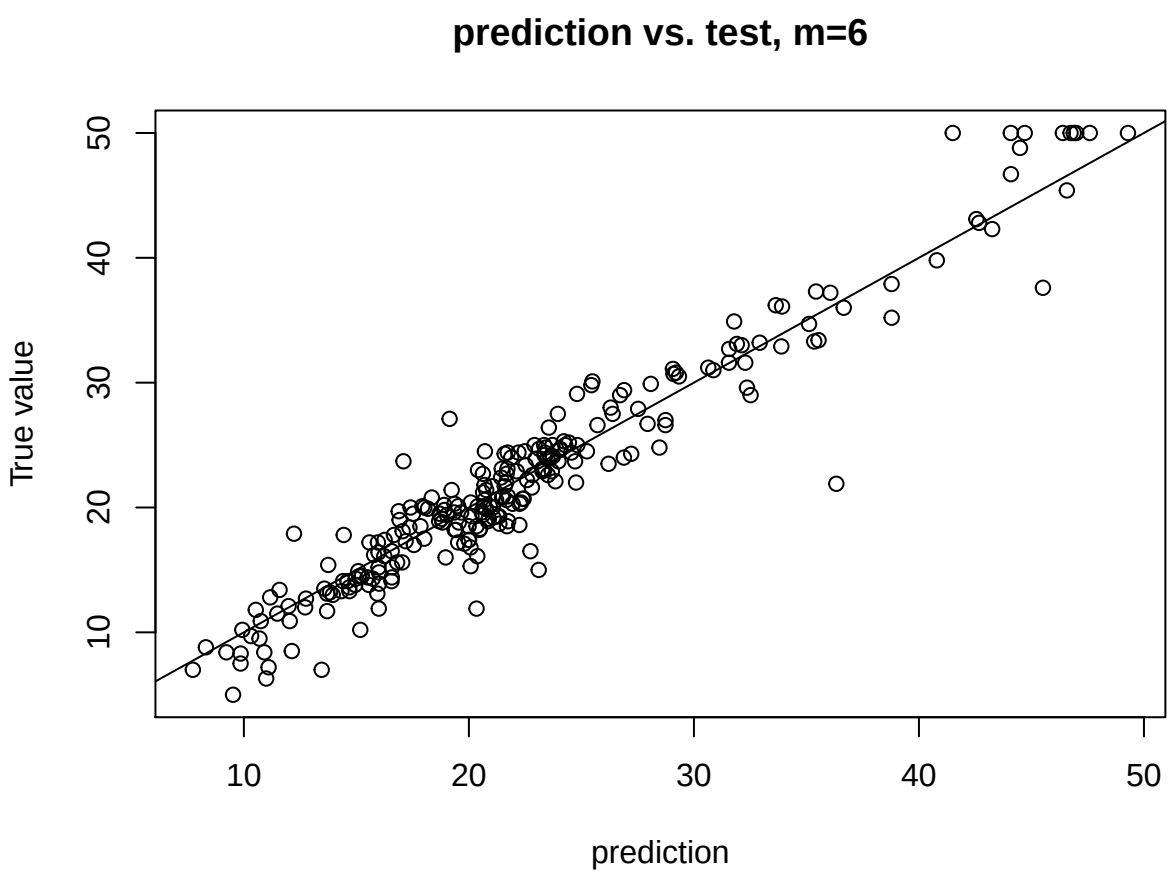
```
par(mfrow=c(1, 2))
plot(Boston.boost, i="lstat", main="lstat effect")
plot(Boston.boost, i="rm", main="rm effect")
```




```
prediction.y.boost <- predict(Boston.boost, newdata=test.x,
                              n.tree=5000)
mean((test.y - prediction.y.boost)^2)
```

```
[1] 6.410153
```

```
#Plot
plot(prediction.y.boost, test.y, main="prediction vs. test, m=6",
     xlab="prediction", ylab="True value")
abline(0, 1)
```


prediction vs. test, m=6
True value
prediction
10
20
30
40
50