8 Trees

By: Udit (based on ISLR)

Setup

- **tree** package for Trees.
- **randomForest** package for Random Forests.
- **gbm** package for Gradient Boosted Machines.
- **BART** package for Bayesian Additive Regression Trees.

```
library(ISLR2)
library(tree)
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
library(BART)
```

```
## Loading required package: nlme
```

```
## Loading required package: nnet
```

```
## Loading required package: survival
```

```
attach(Carseats)
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

```
dim(Carseats)
```

```
## [1] 400  11
```

Decision Tree (classification)

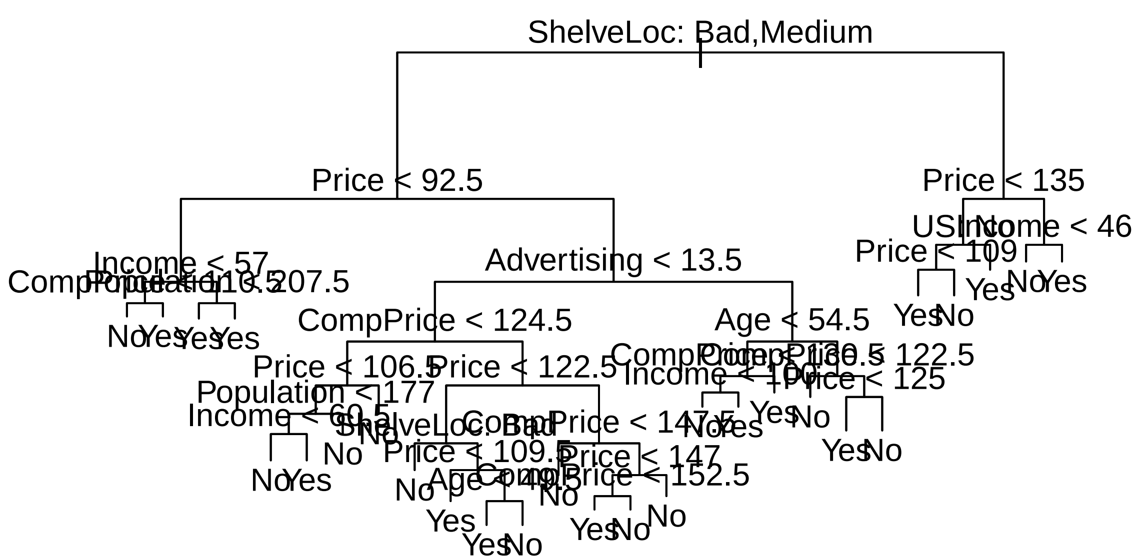
```
High = ifelse(Sales<=8,"No","Yes")
Cars = data.frame(Carseats, High=as.factor(High))
names(Cars)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"          "High"
```

```
#summary(Cars)
tree.car = tree(High~.-Sales, data=Cars)
summary(tree.car)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Cars)
## 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
```

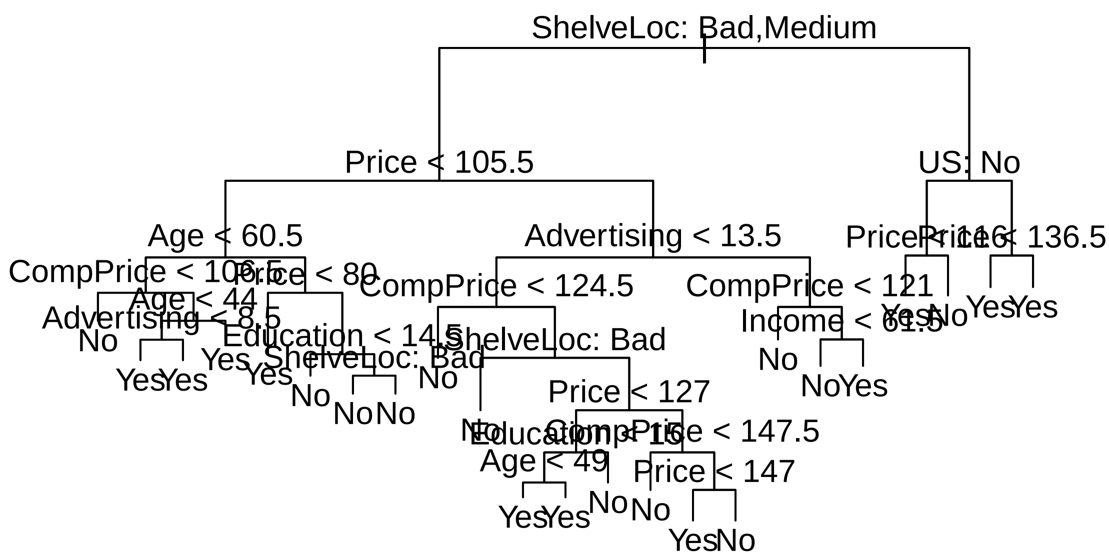
```
# plot tree
plot(tree.car); text(tree.car, pretty=0)
```




```
# display all details
tree.car
```

```
## 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 ) *
```

```r
# Checking performance using train/ test split
set.seed(1011)
train = sample(1:nrow(Cars), 250)
tree.car = tree(High~.-Sales, data=Cars, subset=train)
plot(tree.car); text(tree.car, pretty=0)
```




```r
# Predict & confusion matrix
tree.pred = predict(tree.car, Cars[-train,], type="class")
table(tree.pred, Cars[-train,]$High)
```

```
##
## tree.pred No Yes
##       No  58  20
##       Yes 27  45
```

```r
(45+58)/150  # ~69%
```

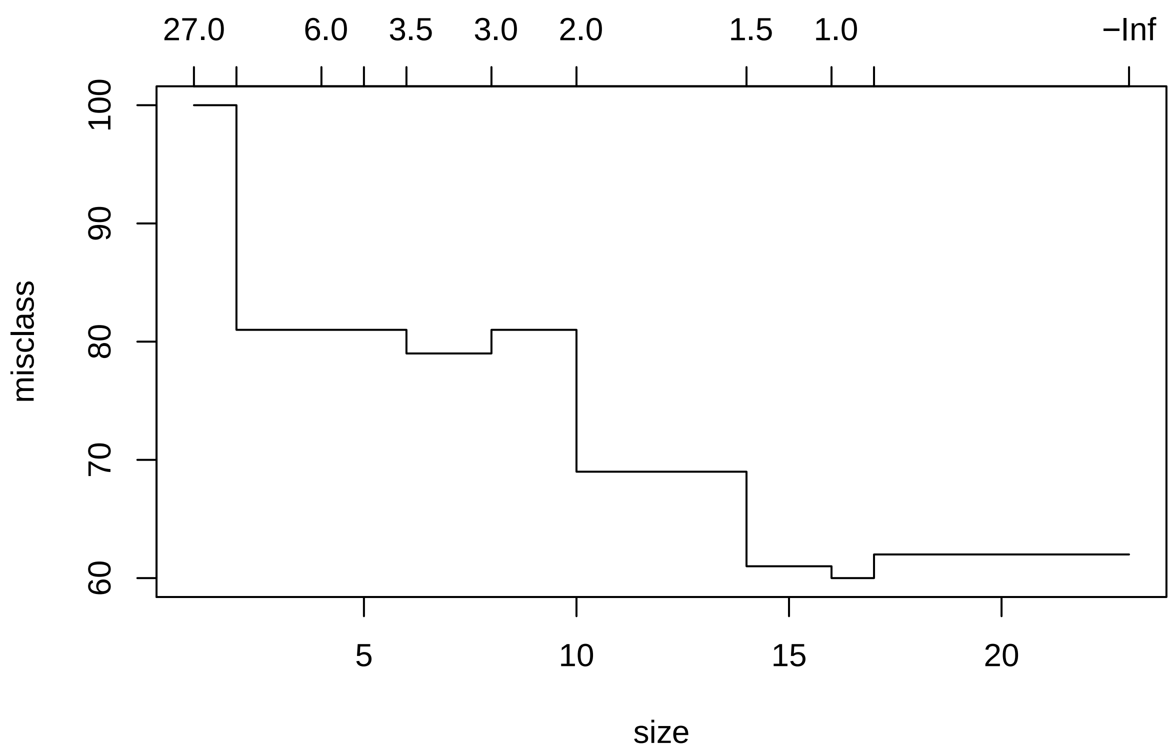
```
## [1] 0.6866667
```

```r
# Pruning Tree using CV - based on classification rate error
cv.car = cv.tree(tree.car, FUN=prune.misclass)
cv.car       # dev here means number of CV error
```

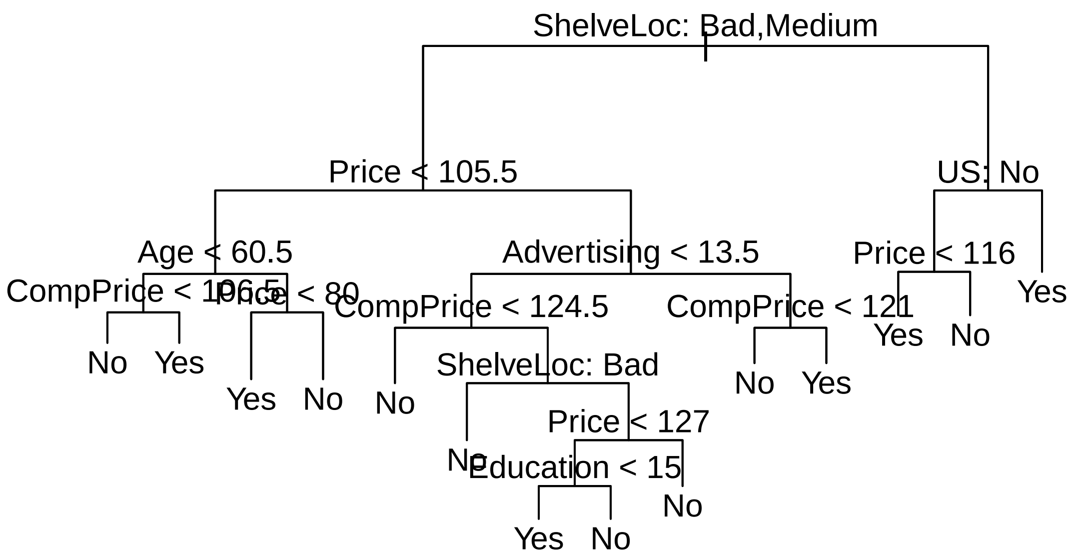
```
## $size
##  [1] 23 17 16 14 10  8  6  5  4  2  1
##
## $dev
##  [1]  62  62  60  61  69  81  79  81  81  81 100
```

```
##
## $k
##  [1] -Inf  0.0  1.0  1.5  2.0  3.0  3.5  5.0  6.0  7.0 27.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.car) # 13 terminal nodes appear to give best fit
```




```
# Pruning for 13 terminal nodes
prune.car = prune.misclass(tree.car, best=13)
plot(prune.car); text(prune.car, pretty=0)
```




```r
# Evaluate tree on test data
prune.pred = predict(prune.car, Cars[-train,], type="class")
table(prune.pred, Cars[-train,]$High)
```

```
##
## prune.pred No Yes
##        No  59  19
##        Yes 26  46
```

```r
(46+59)/150   #~70%, similar performance but shallower tree
```

```
## [1] 0.7
```

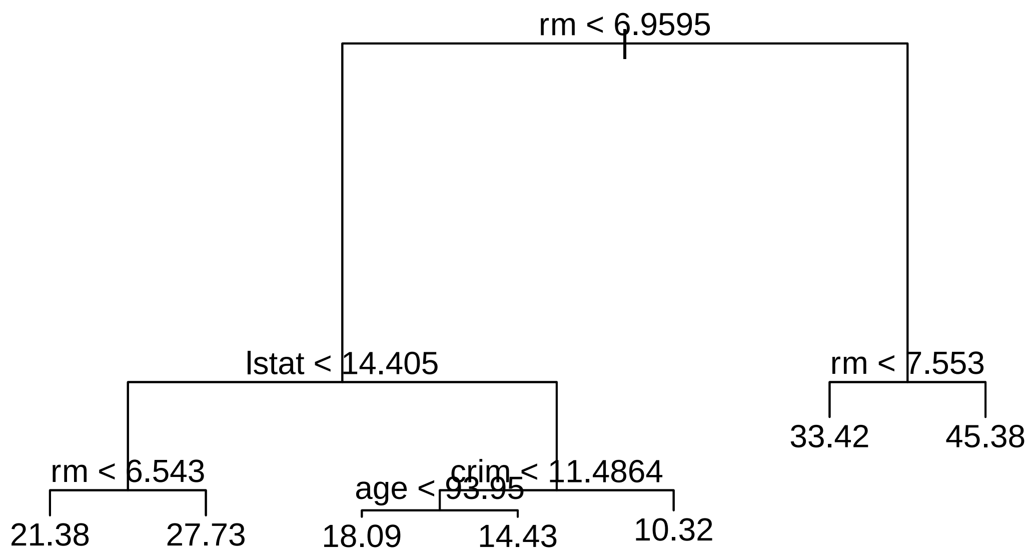
Regression Tree (quantitative)

```r
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston = tree(medv~., data=Boston, subset=train)
summary(tree.boston)   #deviance = sum of squared errors
```

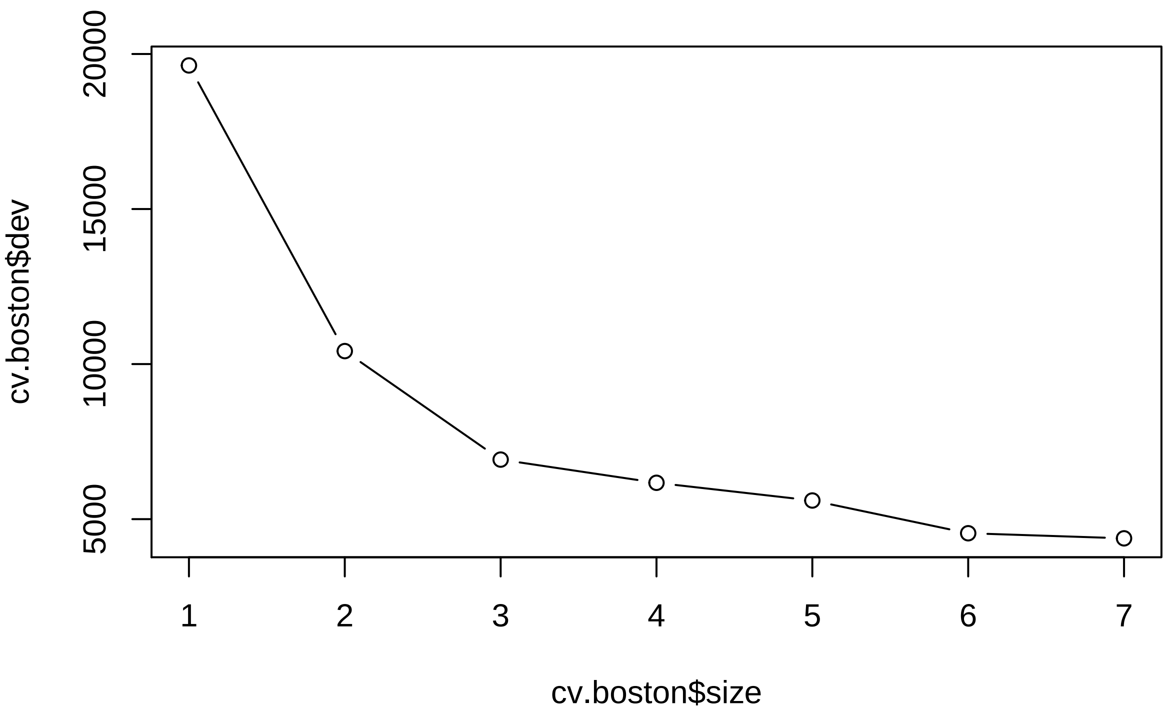
```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
```

```
## Distribution of residuals:
##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

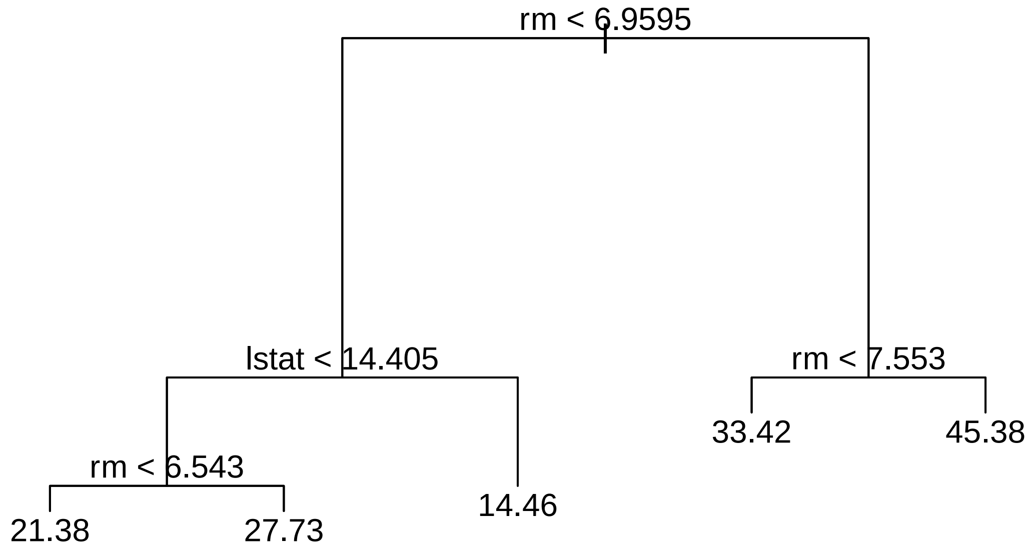
```
plot(tree.boston); text(tree.boston, pretty=0)
```



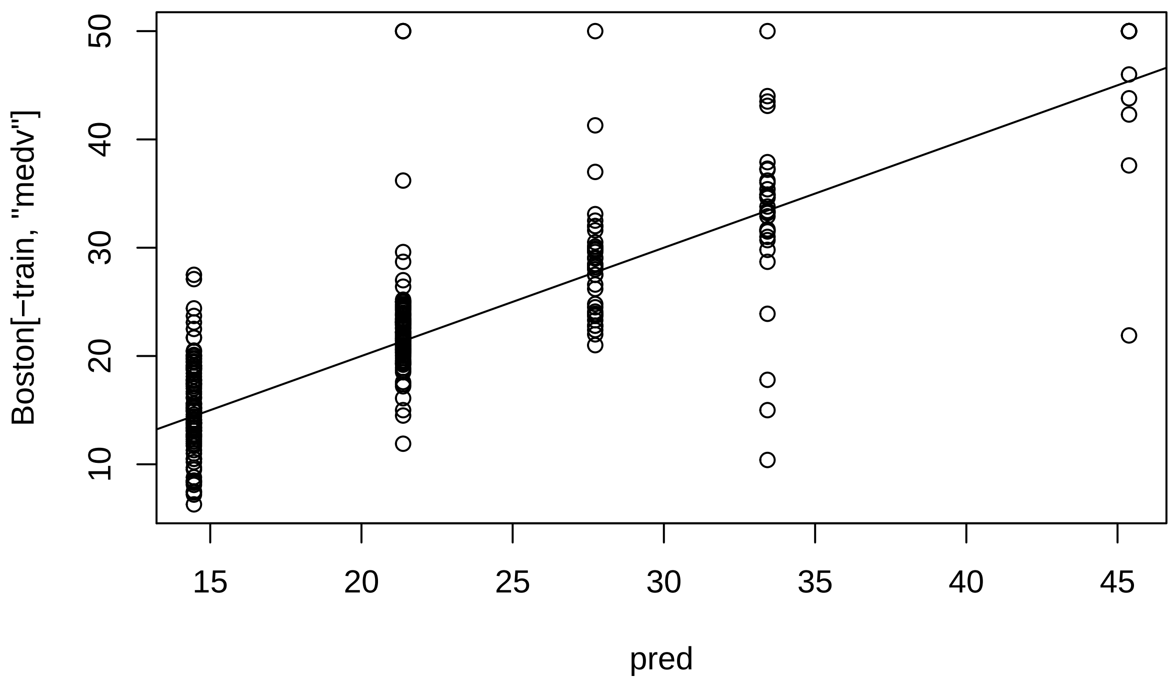

```
# pruning
cv.boston = cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type="b")
```




```
prune.boston = prune.tree(tree.boston, best=5)
plot(prune.boston); text(prune.boston, pretty=0)
```



```r
# making predictions
pred = predict(prune.boston, Boston[-train,])
plot(pred, Boston[-train, "medv"])
abline(0,1)
```




```r
sqrt(mean((pred-Boston[-train,"medv"])^2))  # ~$6000 error
```

```
## [1] 5.991746
```

```r
# fitting a larger tree
tree.boston.deep = tree(medv~., data=Boston, subset=train,
                   control=tree.control(nobs=length(train), mindev=0))
summary(tree.boston.deep)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train, control = tree.control(nobs = length(train),
##     mindev = 0))
## Variables actually used in tree construction:
## [1] "rm"      "lstat"   "indus"   "age"     "nox"     "dis"     "ptratio"
## [8] "tax"     "crim"
## Number of terminal nodes:  41
## Residual mean deviance:  5.542 = 1175 / 212
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -8.140  -1.200   0.000   0.000   1.087  12.860
```

Random Forest & Bagging

Bagging (bootstrap aggregating) is a special case of Random Forest, when all variables are available for selection at each split.

Node Purity - small value indicates that a node contains mostly observations from a single class.

```
attach(Boston)
dim(Boston)    # has 13 variables, MASS package has 14 variables (+ "black")
```

```
## [1] 506  13
```

```
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "lstat"   "medv"
```

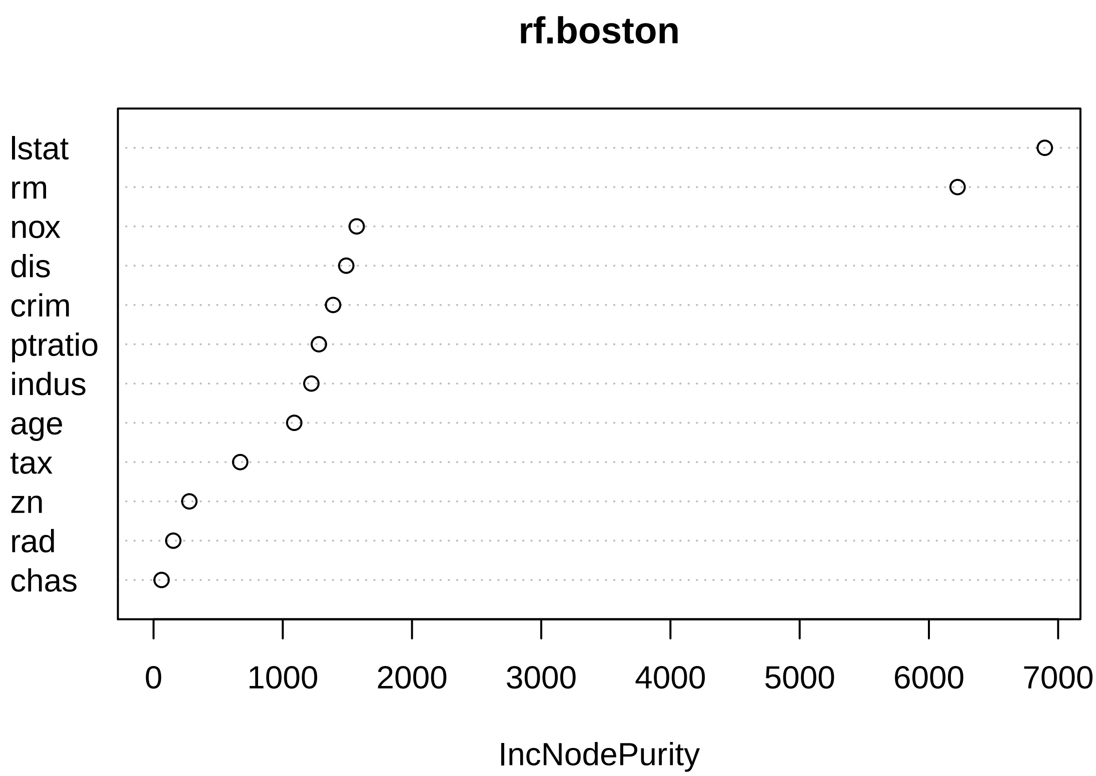
```
set.seed(101)
train = sample(1:nrow(Boston), 300)

# Random Forest
rf.boston = randomForest(medv~., data=Boston, subset=train)
rf.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.93072
##                     % Var explained: 83.14
```

```
# Variable Importance
# total decrease in node purity from that variable avg. over all trees
importance(rf.boston)
```

```
##         IncNodePurity
## crim       1389.45860
## zn          277.03195
## indus      1221.36063
## chas         62.03796
## nox        1571.99923
## rm         6221.84481
## age        1088.76010
## dis        1491.16923
## rad         152.82805
## tax         670.71948
## ptratio    1279.41833
## lstat      6896.72511
```

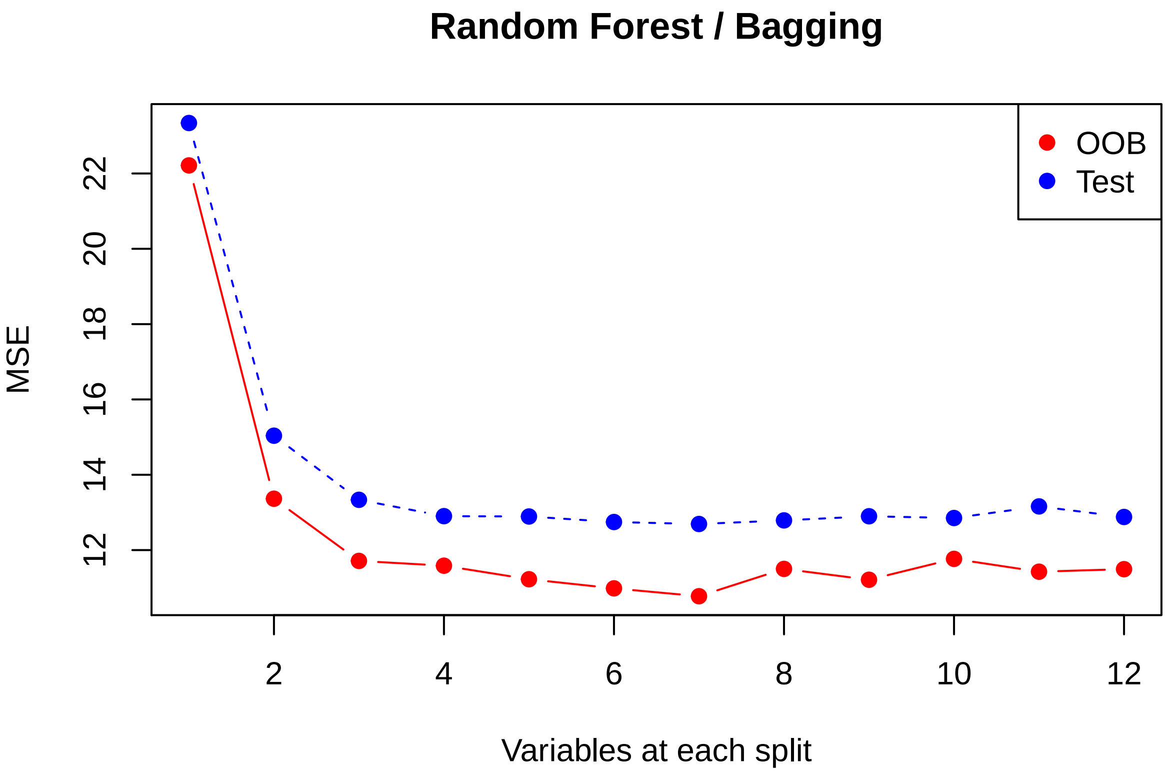
```
varImpPlot(rf.boston)
```




```r
# Tuning parameter - only 1 - number of variables tried at each split
oob.err = double(12)
test.err = double(12)
for(i in 1:12){
  fit = randomForest(medv~., data=Boston, subset=train, mtry=i, ntree=400)
  oob.err[i] = fit$mse[400]

  pred = predict(fit, Boston[-train,])
  test.err[i] = mean((Boston[-train,]$medv - pred)^2)
  cat(i," ")
}
```

```
## 1  2  3  4  5  6  7  8  9  10  11  12
```

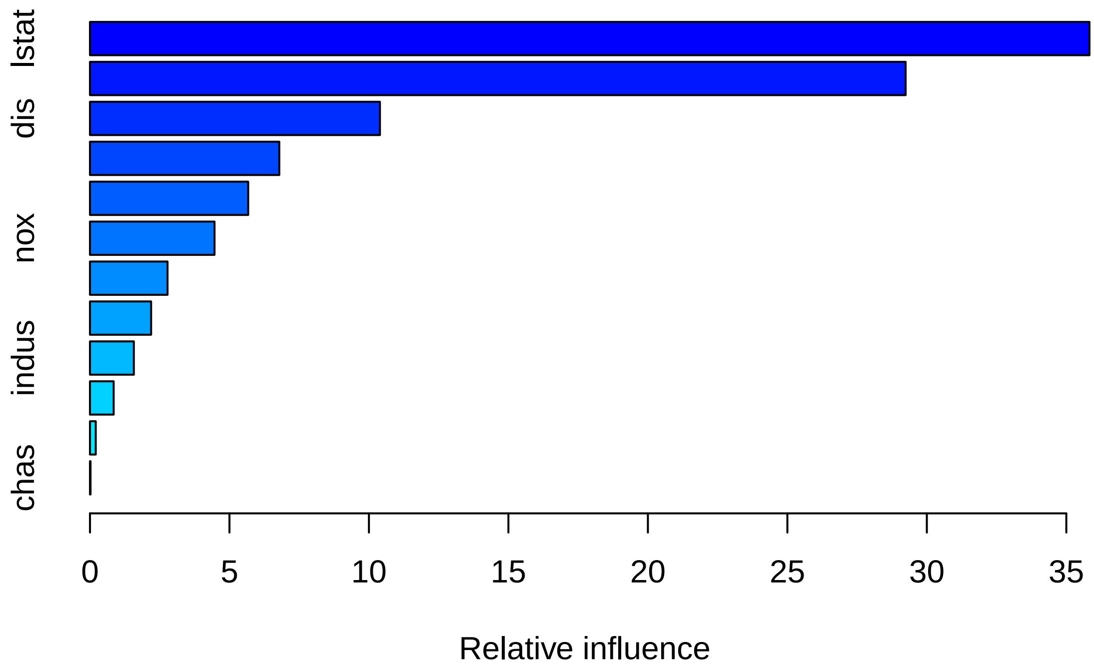
```r
# plot - 4 appears to be a good choice
matplot(1:12, cbind(test.err, oob.err), pch=19, col=c("red","blue"), type="b",
        ylab="MSE", xlab="Variables at each split", main="Random Forest / Bagging")
legend("topright", legend=c("OOB", "Test"), pch=19,col=c("red","blue"))
```



Boosting

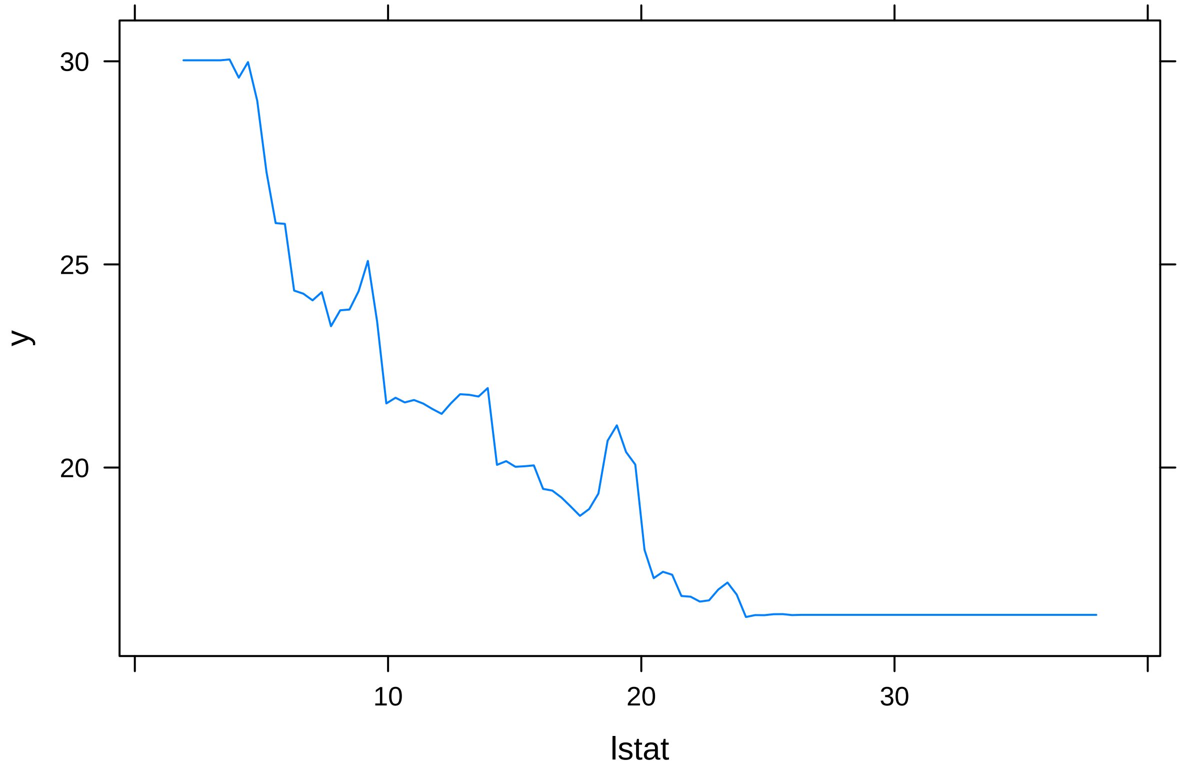
Slow learning based on lots of shallow trees. Unlike random forests, no bootstrapping is done, instead each new tree fits on updated residuals.
Interaction depth defines depth of tree and is a *tuning parameter* along with **shrinkage**.

```
# "gaussian" - regression; "bernoulli" - classification
boost.boston = gbm(medv~., data=Boston[train,], distribution="gaussian",
                   n.trees=10000, shrinkage=0.01, interaction.depth=4)

# variable importance plot
summary(boost.boston)
```

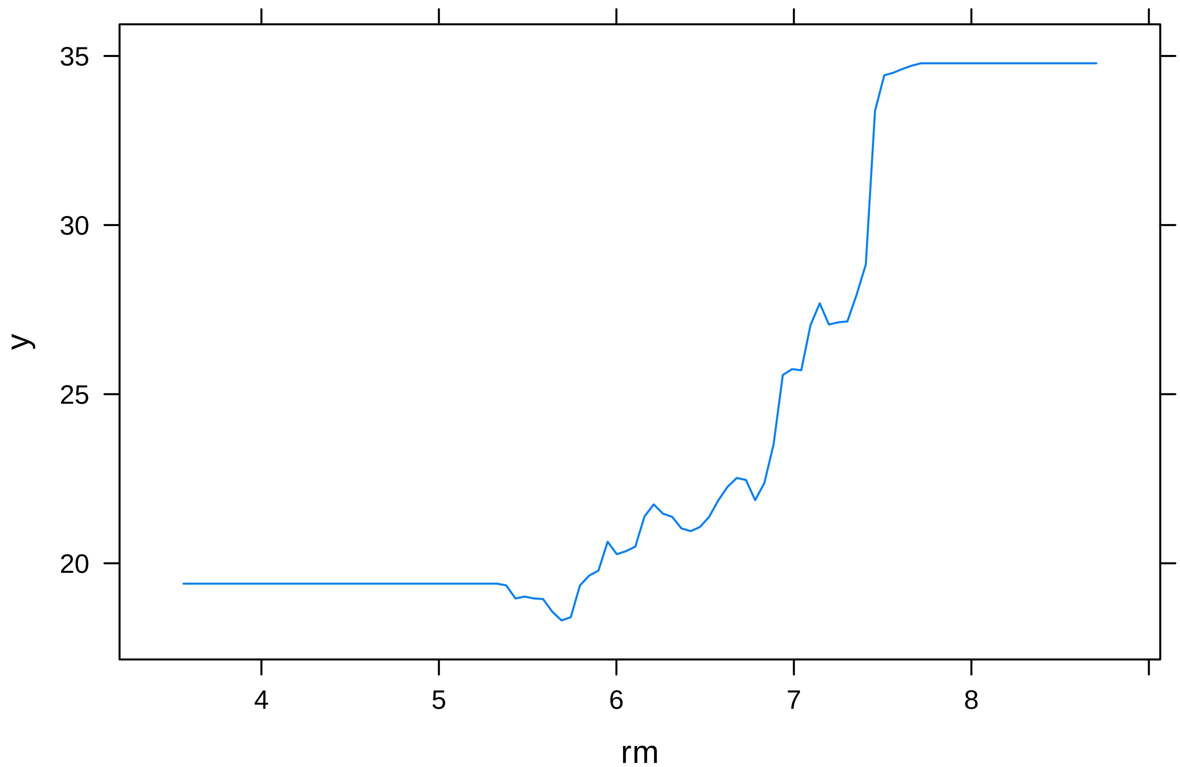



```
##             var     rel.inf
## lstat     lstat 35.82917216
## rm           rm 29.23864389
## dis         dis 10.39453021
## crim       crim  6.78604568
## age         age  5.67145821
## nox         nox  4.46378272
## ptratio ptratio  2.77996610
## tax         tax  2.19110269
## indus     indus  1.57136620
## rad         rad  0.84875894
## zn           zn  0.20622806
## chas       chas  0.01894515
```

```
# Partial Dependence Plots
plot(boost.boston, i="lstat")  # price falls with increase in lower status of pop
```




```
plot(boost.boston, i="rm")      # price increases with number of rooms
```



```r
# Performance
boost.pred = predict(boost.boston, Boston[-train,],n.trees=10000)
sqrt(mean((Boston[-train,"medv"]-boost.pred)^2))  # ~$3500 error
```

```
## [1] 3.591466
```

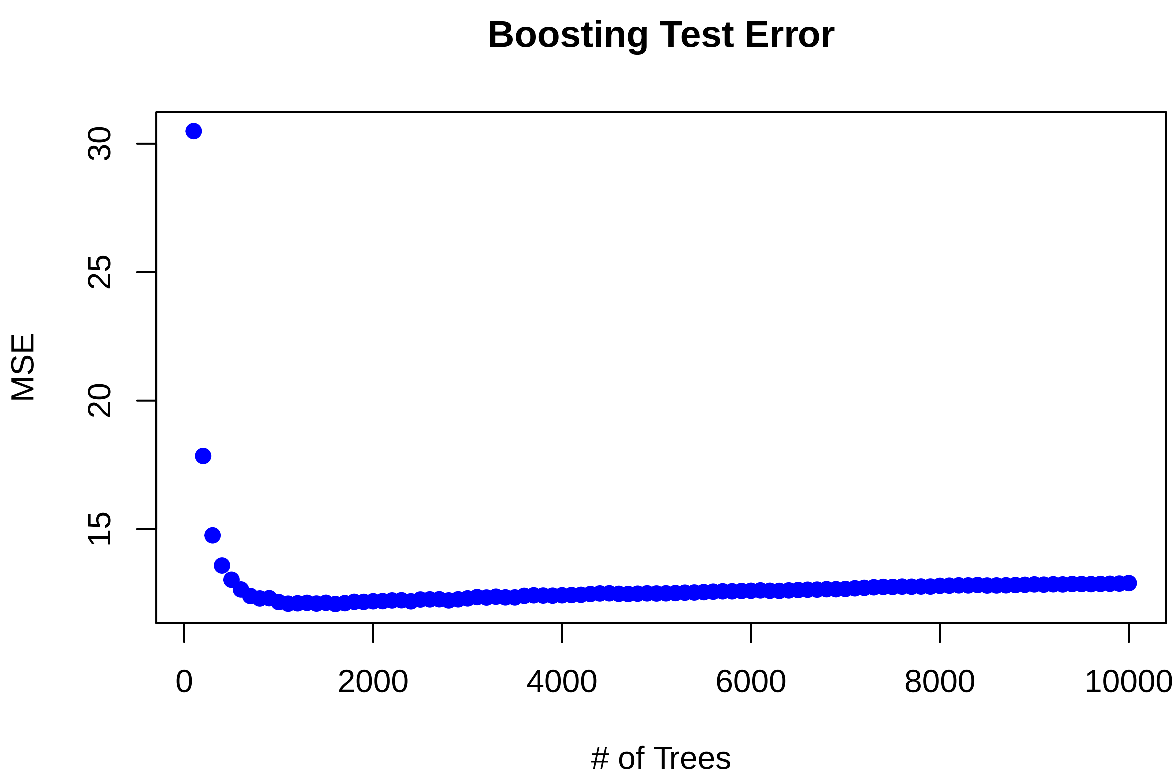
```r
# Test performance as number of trees
n.trees = seq(100,10000,100)
predmat = predict(boost.boston, newdata=Boston[-train,], n.trees=n.trees)
dim(predmat) # 206 observations, 100 number of trees
```

```
## [1] 206 100
```

```r
boost.err = apply((predmat-Boston[-train,]$medv)^2, 2, mean)
length(boost.err) # 100
```

```
## [1] 100
```

```r
plot(n.trees, boost.err, pch=19, ylab="MSE", xlab="# of Trees",
     main="Boosting Test Error", col="blue")
abline(h=min(test.err), col="red")
```




```r
# Using different value of 'lambda'
boost.boston2 = gbm(medv~., data=Boston[train,], distribution="gaussian",
                    n.trees=10000, shrinkage=0.2, interaction.depth=4)
# Performance
boost.pred2 = predict(boost.boston2, Boston[-train,],n.trees=10000)
sqrt(mean((Boston[-train,"medv"]-boost.pred2)^2))  # ~$3600 error
```

```
## [1] 3.64889
```

Bayesian Additive Regression Trees

```r
x <- Boston[,1:12]
y <- Boston[,"medv"]

xtrain = x[train,]
ytrain = y[train]
xtest  = x[-train,]
ytest  = y[-train]

set.seed(1)
bartfit = gbart(xtrain, ytrain, x.test=xtest)
```

```
## *****Calling gbart: type=1
## *****Data:
## data:n,p,np: 300, 12, 206
## y1,yn: -2.772333, -2.472333
## x1,x[n*p]: 0.066170, 12.260000
## xp1,xp[np*p]: 0.006320, 5.640000
## *****Number of Trees: 200
## *****Number of Cut Points: 100 ... 100
## *****burn,nd,thin: 100,1000,1
## *****Prior:beta,alpha,tau,nu,lambda,offset: 2,0.95,0.795495,3,4.37401,22.0723
## *****sigma: 4.738655
## *****w (weights): 1.000000 ... 1.000000
## *****Dirichlet:sparse,theta,omega,a,b,rho,augment: 0,0,1,0.5,1,12,0
## *****printevery: 100
##
## MCMC
## done 0 (out of 1100)
## done 100 (out of 1100)
## done 200 (out of 1100)
## done 300 (out of 1100)
## done 400 (out of 1100)
## done 500 (out of 1100)
## done 600 (out of 1100)
## done 700 (out of 1100)
## done 800 (out of 1100)
## done 900 (out of 1100)
## done 1000 (out of 1100)
## time: 5s
## trcnt,tecnt: 1000,1000
```

```r
bart.pred = bartfit$yhat.test.mean
sqrt(mean((ytest-bart.pred)^2))   #~$3400 error
```

```
## [1] 3.448037
```

```r
# How many times each variable appeared in the collection of trees.
ord = order(bartfit$varcount.mean, decreasing=T)
bartfit$varcount.mean[ord]
```

```
##     rad     nox   lstat      rm     tax     age ptratio   indus    chas      zn
##  24.831  24.708  22.041  20.123  19.932  19.013  18.756  18.157  18.142  16.578
##     dis    crim
##  15.161  11.441
```