Tree-Based Methods

Yuan Bian

University of Western Ontario

2022/12/01

Classification Trees

Dataset *Carseats*

- In *Carseats*, *Sales* is a continuous variable, and so we begin by recoding it as a binary variable, called *High*, which takes on a value of Yes if the *Sales* variable exceeds 8, and takes on a value of No otherwise

```r
library(tree)
library(ISLR2)
attach(Carseats)
High <- factor(ifelse(Sales <= 8, "No", "Yes"))
Carseats <- data.frame(Carseats, High)
tree.carseats <- tree(High ~ . - Sales, Carseats)
```

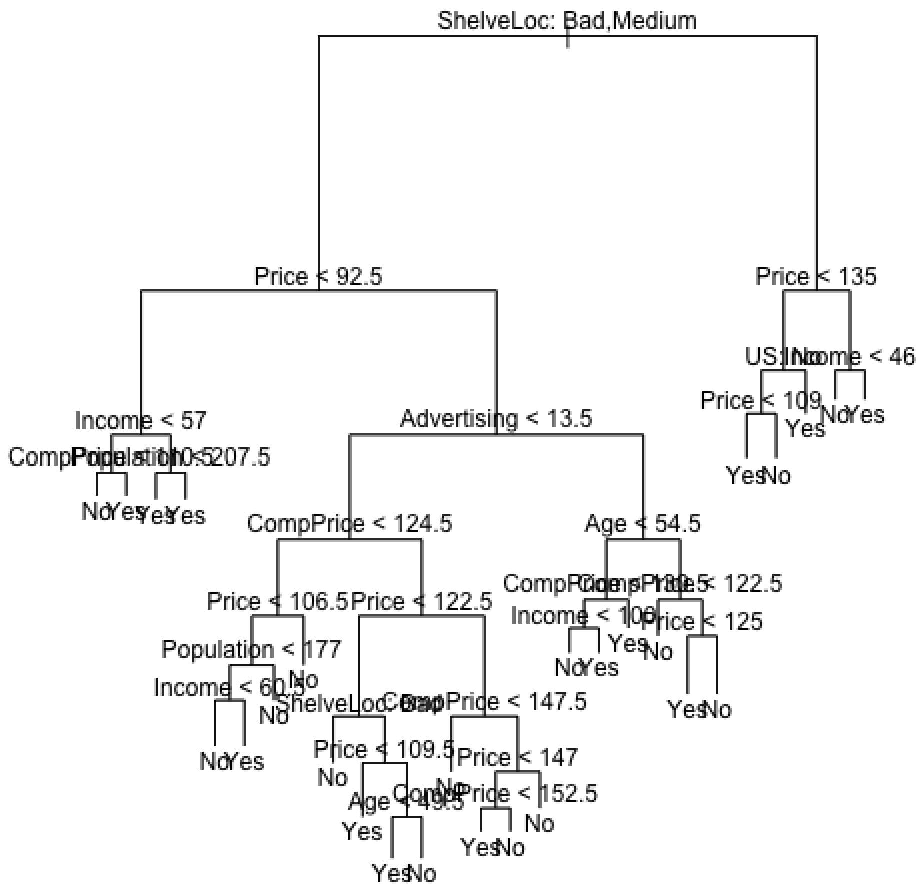
Goal: to predict *High* using all variables but *Sales*

- The summary() function lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate.

```
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
```

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



- In order to properly evaluate the performance of a classification tree, we must estimate the test error rather than simply computing the training error.

```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)
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

```
         High.test
tree.pred  No Yes
      No  104  33
      Yes  13  50
```

```r
(104 + 50) / 200
```

```
[1] 0.77
```

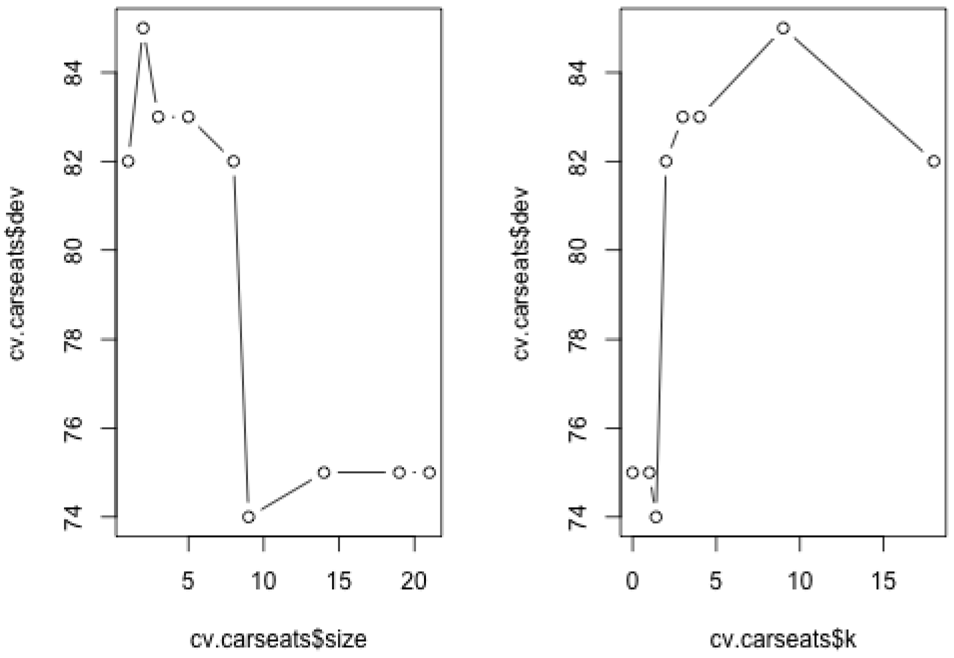
- Next, we perform cross-validation in order to determine the optimal level of tree complexity.
- We use the argument FUN = prune.misclass to indicate that we want the classification error rate to guide the cross-validation and pruning process.

```
set.seed(7)
cv.carseats <- cv.tree(tree.carseats, FUN = prune.misclass)
cv.carseats
```

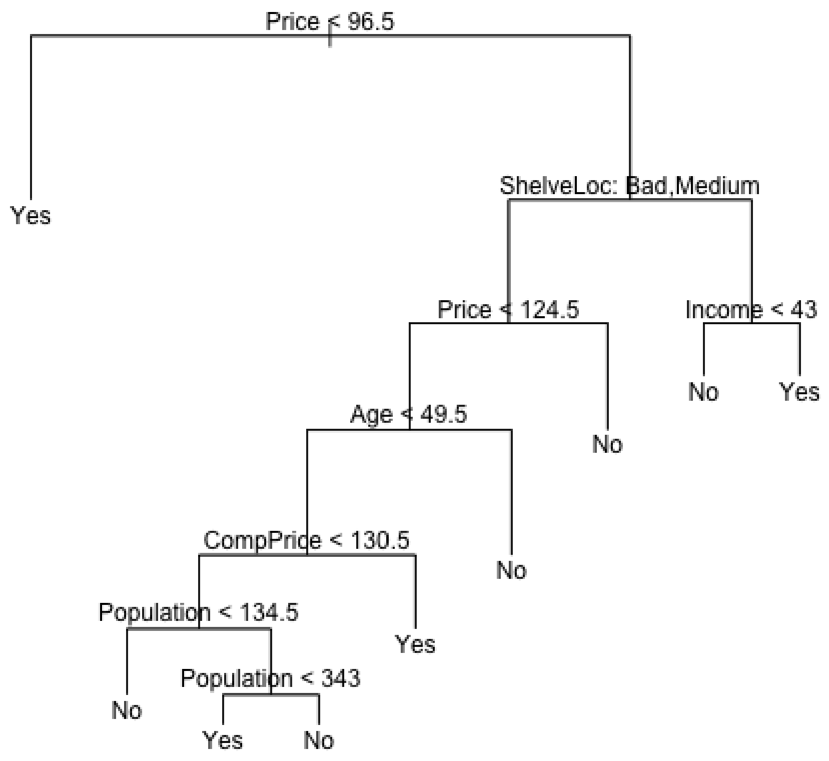
```
$size
[1] 21 19 14  9  8  5  3  2  1

$dev
[1] 75 75 75 74 82 83 83 85 82

$k
[1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

- Despite its name, *dev* corresponds to the number of cross-validation errors.

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



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



- How well does this pruned classification tree perform on the test set?

```
tree.pred <- predict(prune.carseats, Carseats.test, type = "class")
table(tree.pred, High.test)
```

```
         High.test
tree.pred No  Yes
      No  97  25
      Yes 20  58
```

```
(97 + 58) / 200
```

```
[1] 0.775
```

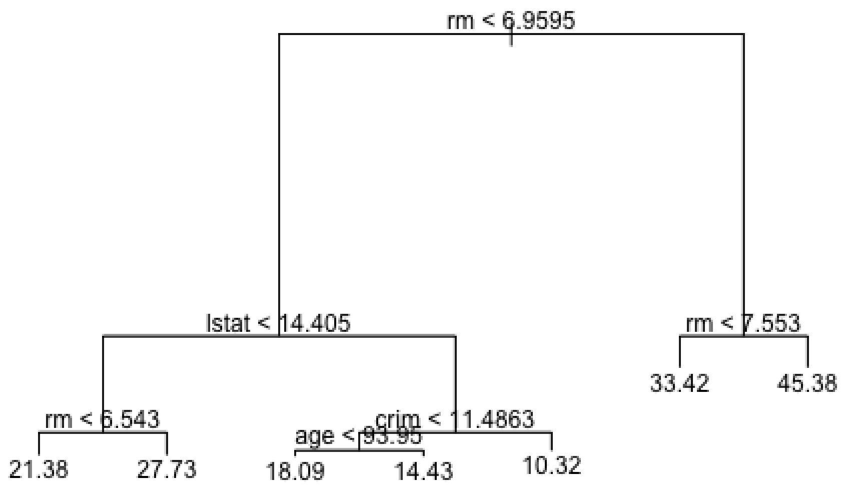
Regression Trees

Dataset *Boston*

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

```
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
```

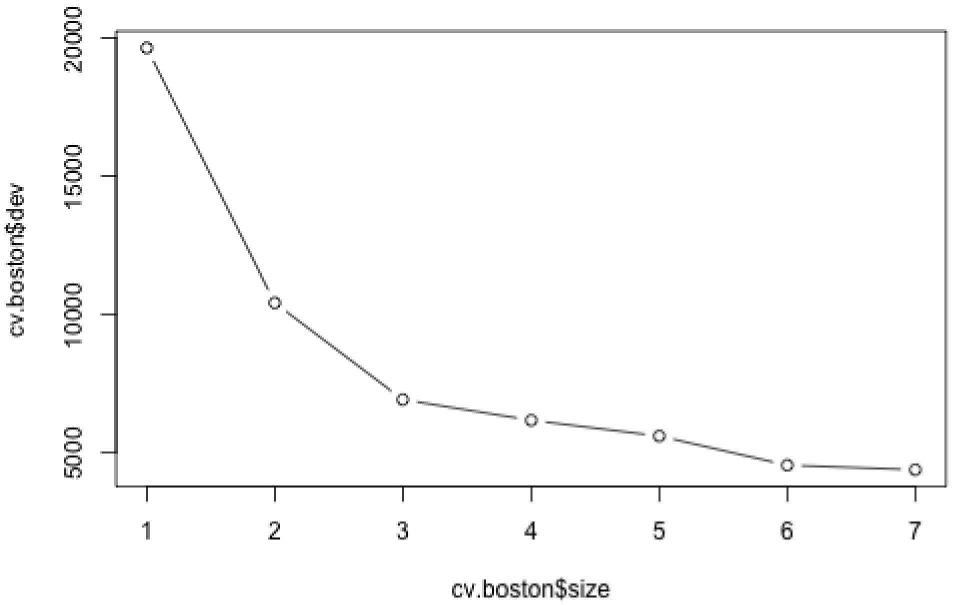
```r
plot(tree.boston)
text(tree.boston, pretty = 0)
```



- *lstat* measures the percentage of individuals with lower socioeconomic status, while *rm* corresponds to the average number of rooms.
- The tree indicates that larger values of *rm*, or lower values of *lstat*, correspond to more expensive houses.

- whether pruning the tree will improve performance?

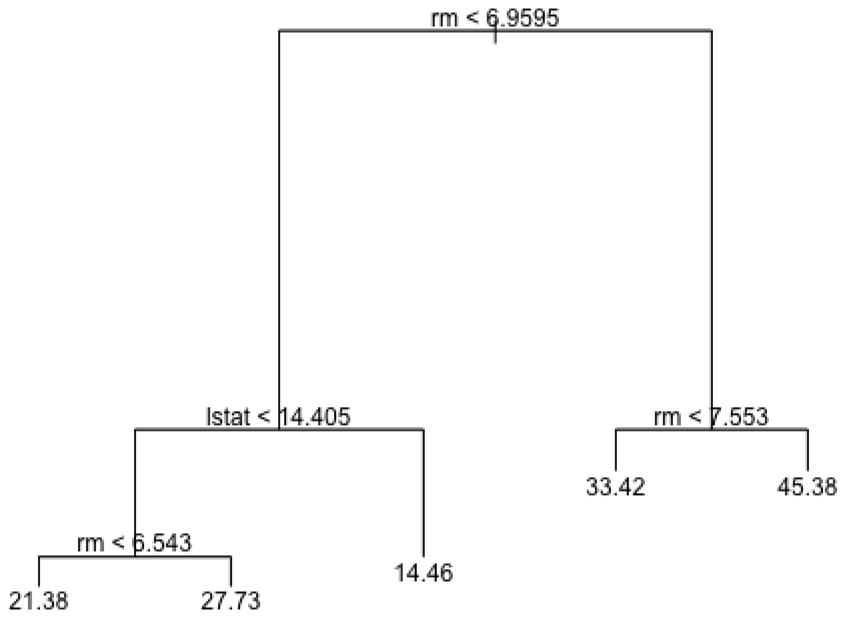
```r
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type = "b")
```



- In this case, the most complex tree under consideration is selected by cross-validation.

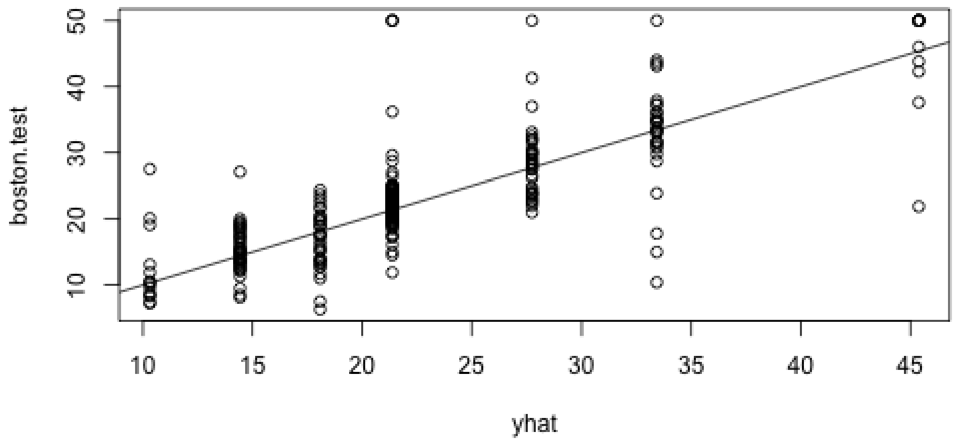
- However, if we wish to prune the tree, we could do so as follows, using the prune.tree() function:

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



- In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

```
yhat <- predict(tree.boston, newdata = Boston[-train, ])
boston.test <- Boston[-train, "medv"]
boston.test <- Boston[-train, "medv"]
plot(yhat, boston.test); abline(0, 1)
```



```
mean((yhat - boston.test)^2)
```

```
[1] 35.28688
```

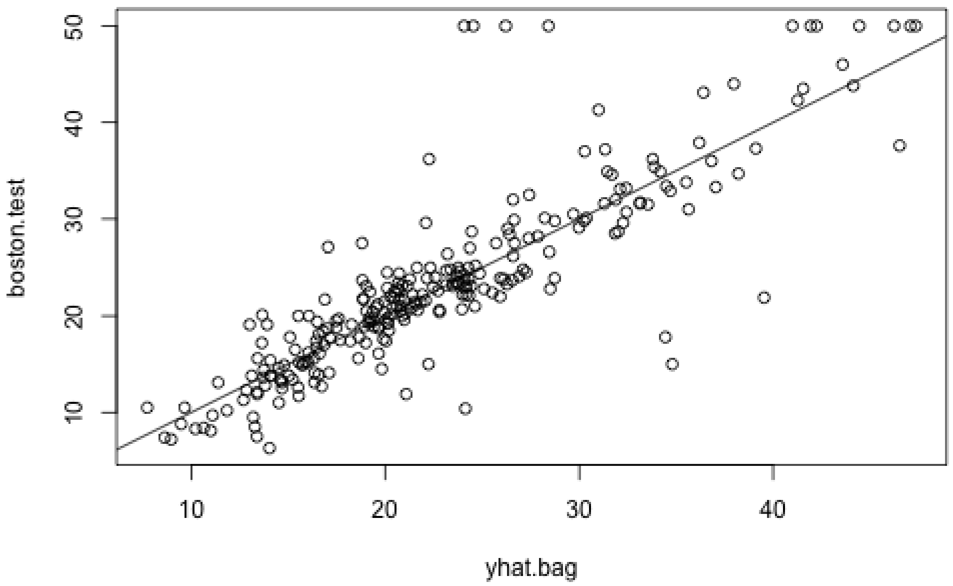
Bagging and Random Forests

- Bagging is simply a special case of a random forest with $m = p$.
- The argument $mtry = 12$ indicates that all 12 predictors should be considered for each split of the tree

```r
library(randomForest)
set.seed(1)
bag.boston <- randomForest(medv ~ ., data = Boston,
    subset = train, mtry = 12, importance = TRUE)
```

- How well does this bagged regression tree perform on the test set?

```
yhat.bag <- predict(bag.boston, newdata = Boston[-train, ])
plot(yhat.bag, boston.test); abline(0, 1)
```



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

```
[1] 23.41916
```

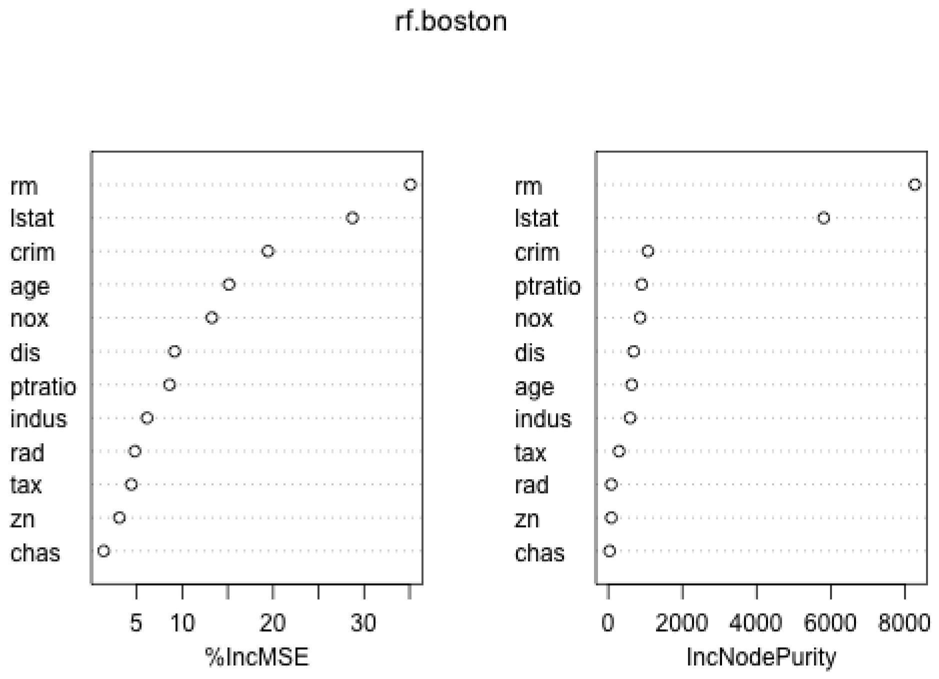
- Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the *mtry* argument.
- By default, randomForest() uses $p/3$ variables when building a random forest of regression trees, and $\sqrt{p}$ variables when building a random forest of classification trees.
- Here we use mtry = 6.

```
set.seed(1)
rf.boston <- randomForest(medv ~ ., data = Boston,
    subset = train, mtry = 6, importance = TRUE)
yhat.rf <- predict(rf.boston, newdata = Boston[-train, ])
mean((yhat.rf - boston.test)^2)
```

```
[1] 20.06644
```

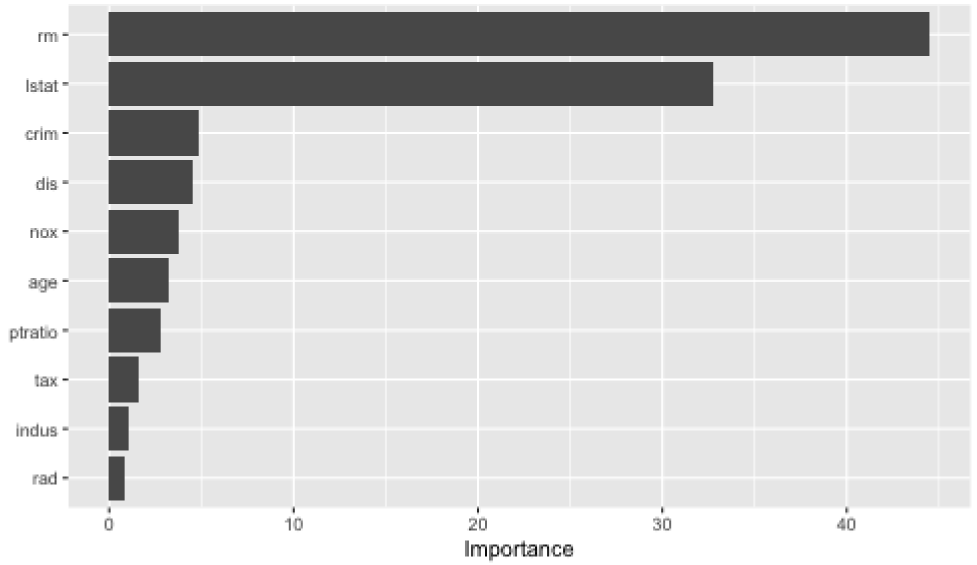
- The first measure is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is permuted.
- The second measure is a measure of the total decrease in node impurity (the training RSS for regression trees, and the deviance for classification trees) that results from splits over that variable, averaged over all trees.

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



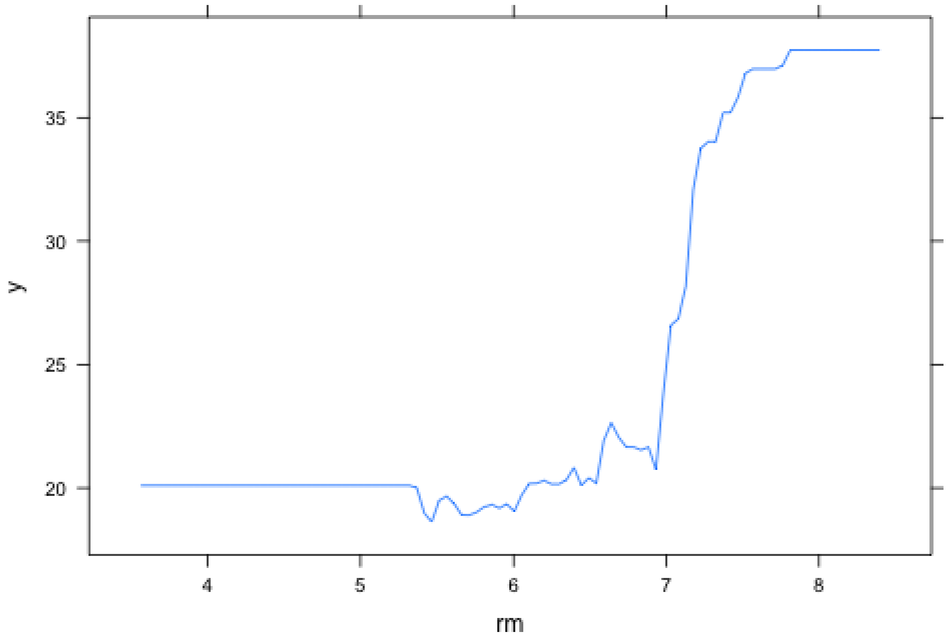
Boosting

```r
library(gbm)
set.seed(1)
boost.boston <- gbm(medv ~ ., data = Boston[train,], distribution =
                     "gaussian", n.trees = 5000, interaction.depth = 4)
library(vip)
vip(boost.boston)
```

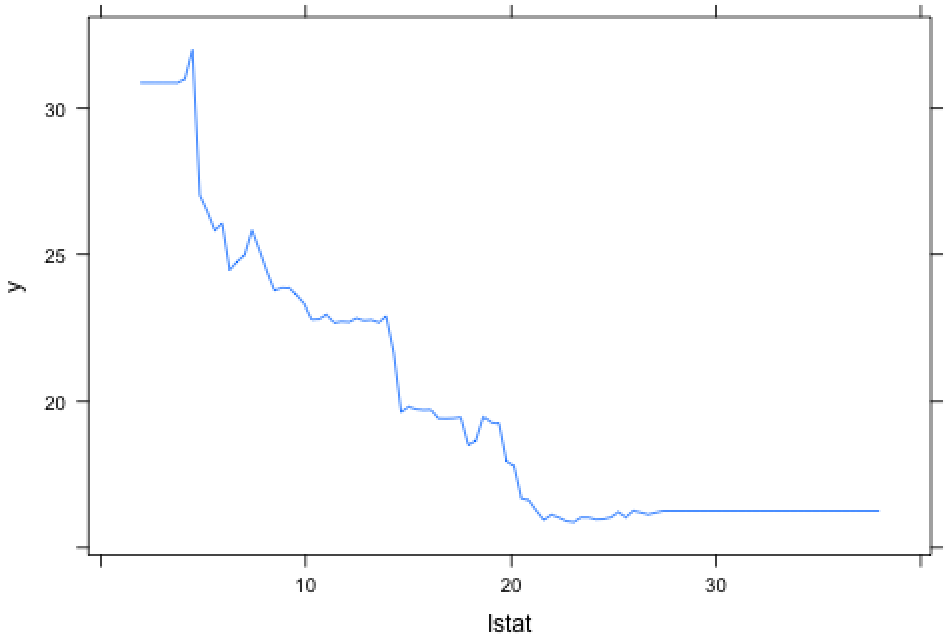


- These plots illustrate the marginal effect of the selected variables on the response after integrating out the other variables.
- In this case, as we might expect, median house prices are increasing with rm and decreasing with lstat.

```
plot(boost.boston, i = "rm")
```



```r
plot(boost.boston, i = "lstat")
```



- How well does this boosted regression tree perform on the test set?

```
yhat.boost <- predict(boost.boston,
    newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
[1] 18.39057
```

- If we want to, we can perform boosting with a different value of the shrinkage parameter λ.
- The default value is 0.001, here we take $\lambda = 0.2$. .

```
boost.boston <- gbm(medv ~ ., data = Boston[train, ],
    distribution = "gaussian", n.trees = 5000,
    interaction.depth = 4, shrinkage = 0.2, verbose = F)
yhat.boost <- predict(boost.boston,
    newdata = Boston[-train, ], n.trees = 5000)
mean((yhat.boost - boston.test)^2)
```

```
[1] 16.54778
```

Reference

James, G, Witten, D, Hastie, T, Tibshirani, R (2013) *An Introduction to Statistical Learning*. Springer, New York, Second edition.