Statistical Learning

Lecture 08a - Trees

ANU - RSFAS

Last Updated: Tue Apr 26 15:42:31 2022

Tree-based Methods

- Here we describe tree-based methods for regression and classification.
- These involve stratifying or segmenting the predictor space into a number of simple regions.
- Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision-tree methods.

Pros and Cons

- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss bagging, random forests, and boosting.
  - These methods grow multiple trees which are then combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy, at the expense of some loss interpretation.

The Basics of Decision Trees

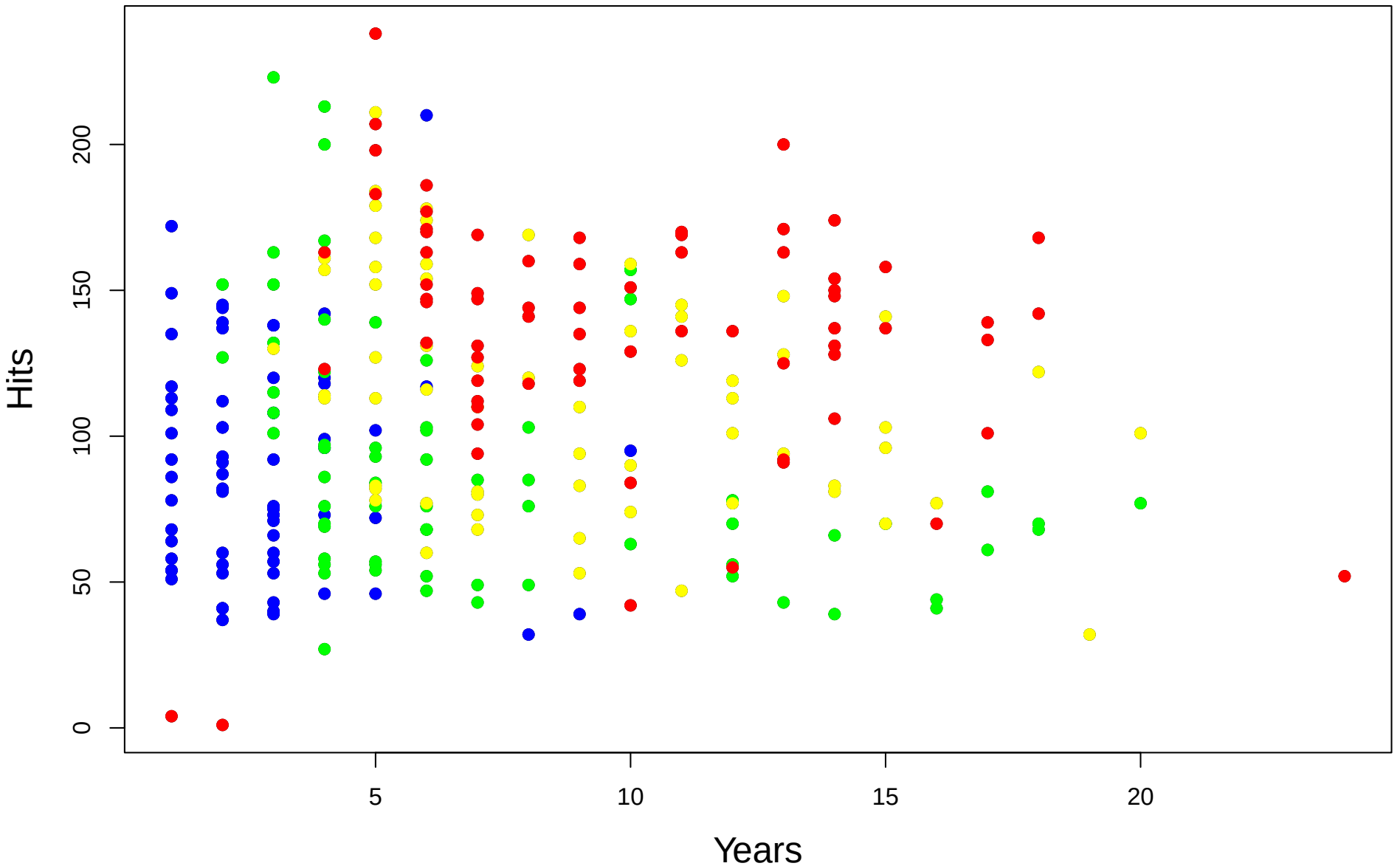
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.
- Let's look at the Baseball Salary data.
- Let's focus on **Hits**, **Years**, and **Salary** for now.
- Years: Number of years in the major leagues
- Hits: Number of hits in 1986
- Salary: 1987 annual salary on opening day in thousands of dollars

```
library(ISLR)
data("Hitters")
Hit.na <- na.omit(Hitters)
attach(Hit.na)
```

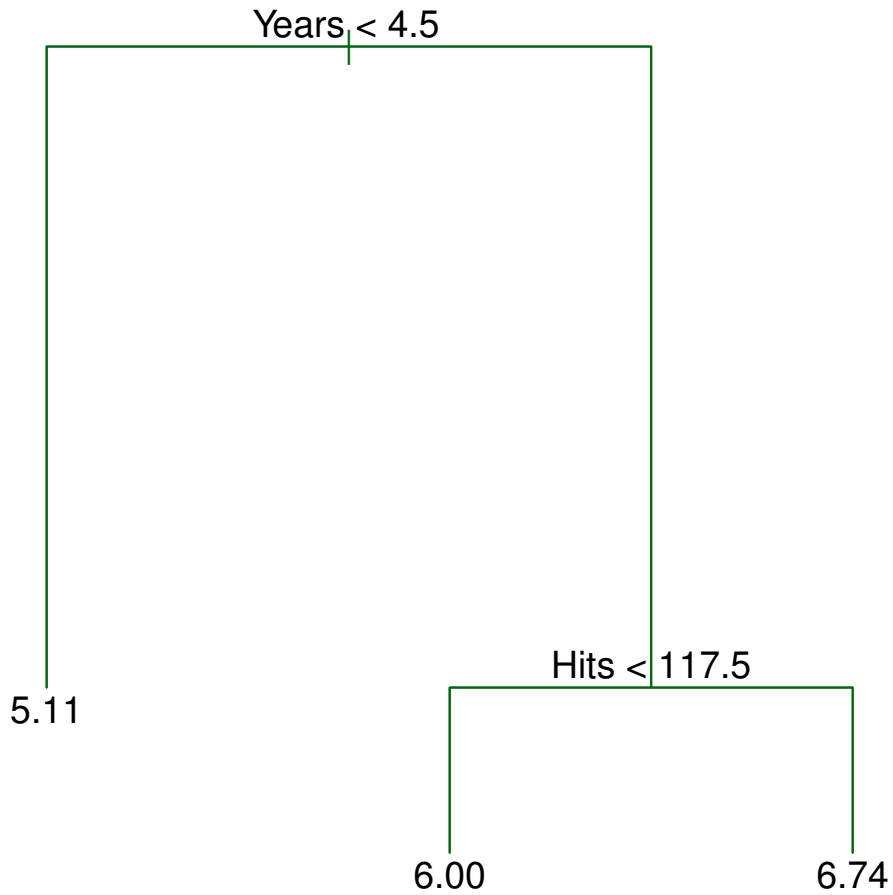
```
summary(cbind(Hits, Years, Salary))
```

```
##       Hits            Years            Salary      
##  Min.   :  1.0   Min.   : 1.000   Min.   :  67.5  
##  1st Qu.: 71.5   1st Qu.: 4.000   1st Qu.: 190.0  
##  Median :103.0   Median : 6.000   Median : 425.0  
##  Mean   :107.8   Mean   : 7.312   Mean   : 535.9  
##  3rd Qu.:141.5   3rd Qu.:10.000   3rd Qu.: 750.0  
##  Max.   :238.0   Max.   :24.000   Max.   :2460.0
```

- Salary is color-coded from low (blue, green) to high (yellow,red)



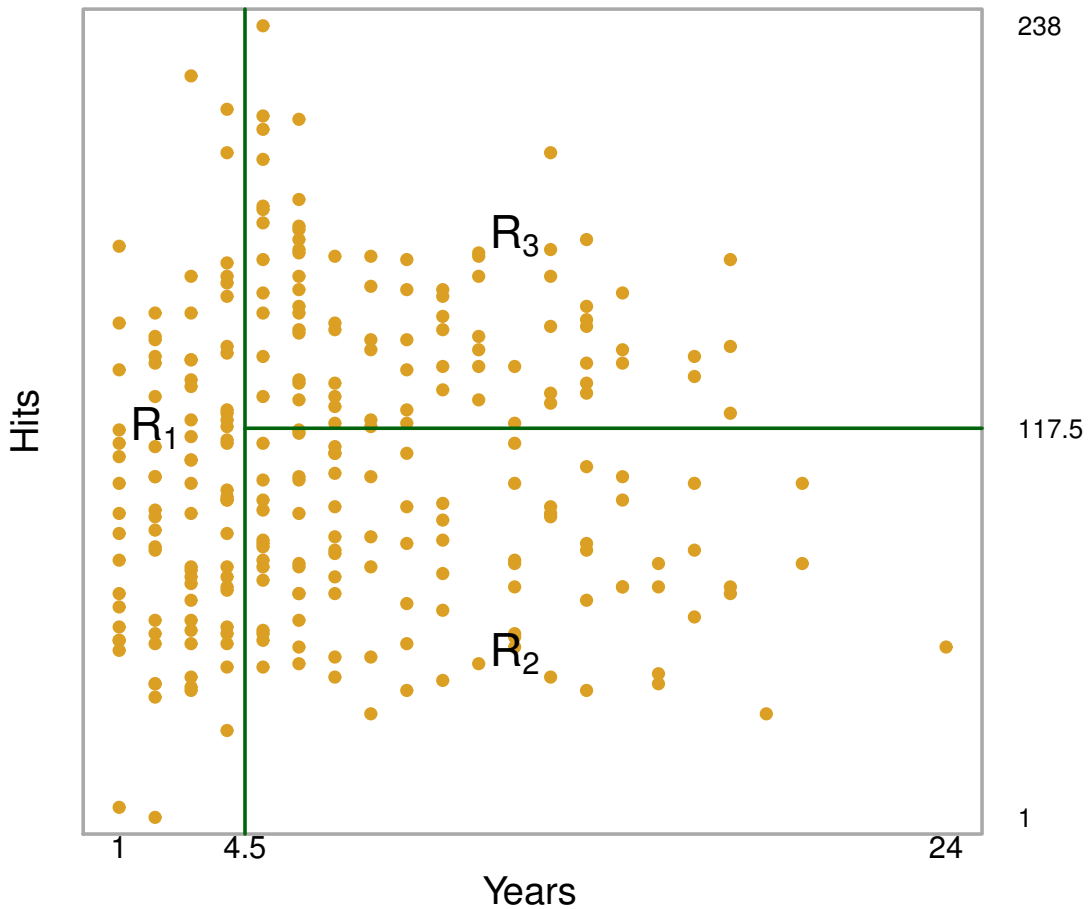
Decision Tree for These Data



- The tree has two internal nodes and three terminal nodes, or leaves.
- The number in each leaf is the mean of the response for the observations that fall there.

- Overall, the tree stratifies or segments the players into three regions of predictor space:

- $R_1 = \{X | \textit{Years} < 4.5\}$, $R_2 = \{X | \textit{Years} \geq 4.5, \textit{Hits} < 117.5\}$, $R_3 = \{X | \textit{Years} \geq 4.5, \textit{Hits} \geq 117.5\}$



Interpretation of Results

- **Years** is the most important factor in determining **Salary**, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of **Hits** that he made in the previous year seems to play little role in his Salary.
- But among players who have been in the major leagues for five or more years, the number of **Hits** made in the previous year does affect **Salary**, and players who made more **Hits** last year tend to have higher salaries.
- Surely an over-simplification, but compared to a regression model, it is easy to display, interpret and explain

Details of the Tree-Building Process

1. We divide the predictor space — that is, the set of possible values for $X_1, X_2, \ldots, X_p$ — into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.

2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j .

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.

- The goal is to find boxes $R_1, R_2, \ldots, R_J$ that minimize the RSS, given by:

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j^{th} box.

- Unfortunately, it is computationally unfeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a top-down, greedy approach that is known as recursive binary splitting.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

- We first select the predictor X_j and the cut-point s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.

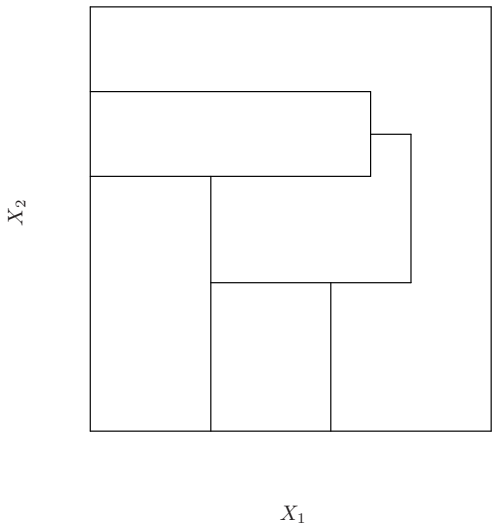
- Next, we repeat the process, looking for the best predictor and best cut-point in order to split the data further so as to minimize the RSS within each of the resulting regions.

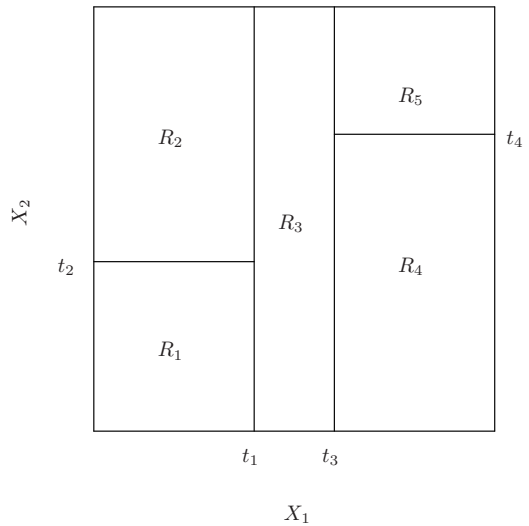
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions.

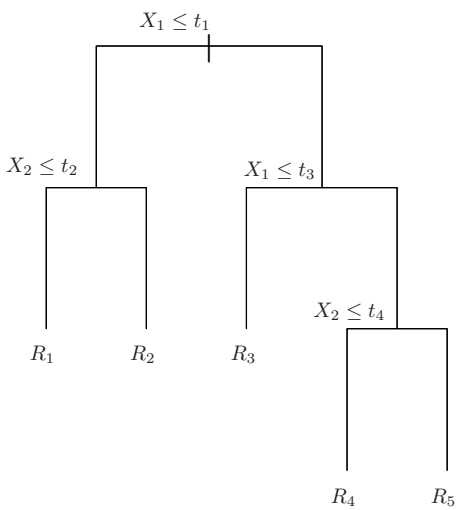
- Again, we look to split one of these three regions further, so as to minimize the RSS.

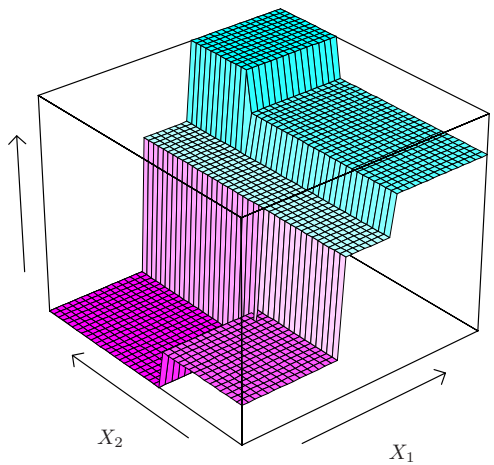
- The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

- A five-region example of this approach.









Predictions

- We predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Pruning a Tree

- The process described above may produce good predictions on the training set, but is likely to over fit the data, leading to poor test set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, R_2, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too short-sighted:
  - a seemingly worthless split early on in the tree might be followed by a very good split.

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree
- Cost complexity pruning — also known as weakest link pruning — is used to do this
- We consider a sequence of trees indexed by a non-negative tuning parameter α.
- For each value of α there corresponds a subtree $T \subset T_0$ such that

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

is as small as possible.

- We just have another penalty term.
- $|T|$ indicates the number of terminal nodes of the tree T.
- R_m is the rectangle (i.e. the subset of predictor space) corresponding to the m^{th} terminal node.
- $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Choosing the Best Subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- We select an optimal value $\hat{\alpha}$ using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$.

Summary: Tree Algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

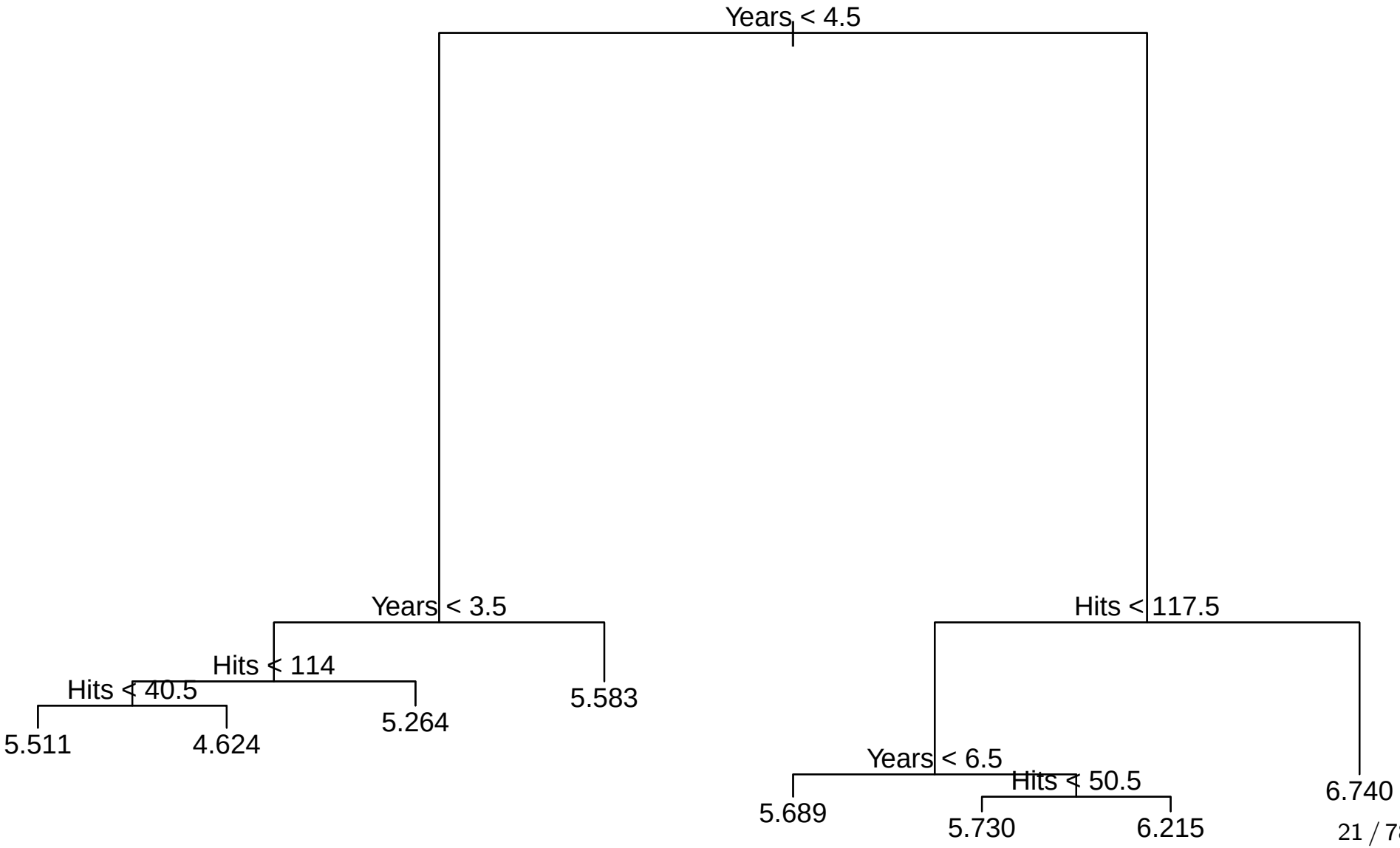
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. For each $k = 1, \ldots, K$:

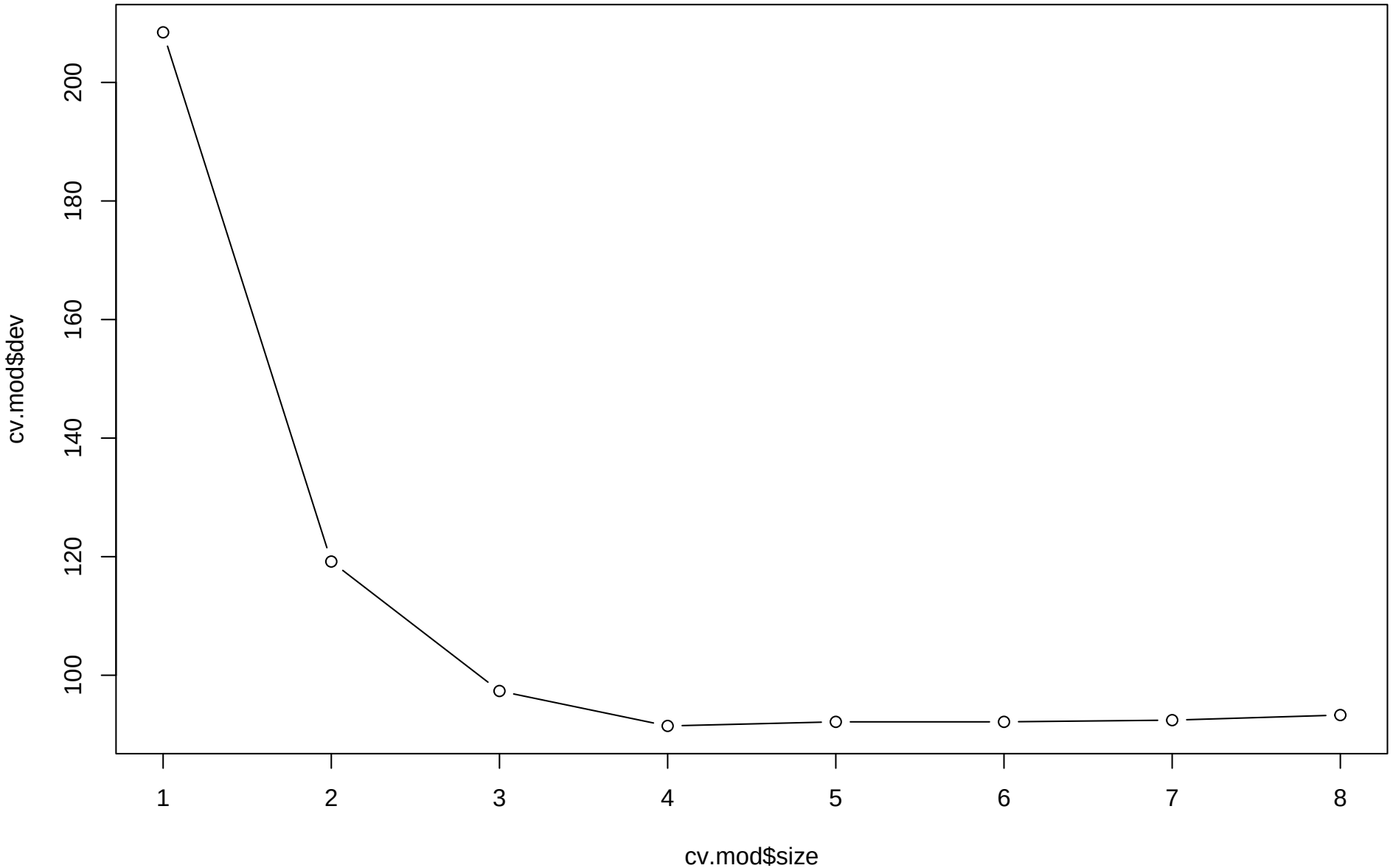
   - Repeat Steps 1 and 2 on the $\left(\frac{K-1}{K}\right)^{th}$ fraction of the training data, excluding the k^{th} fold.
   - Evaluate the mean squared prediction error on the data in the left-out k^{th} fold, as a function of α.
   - Average the results, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

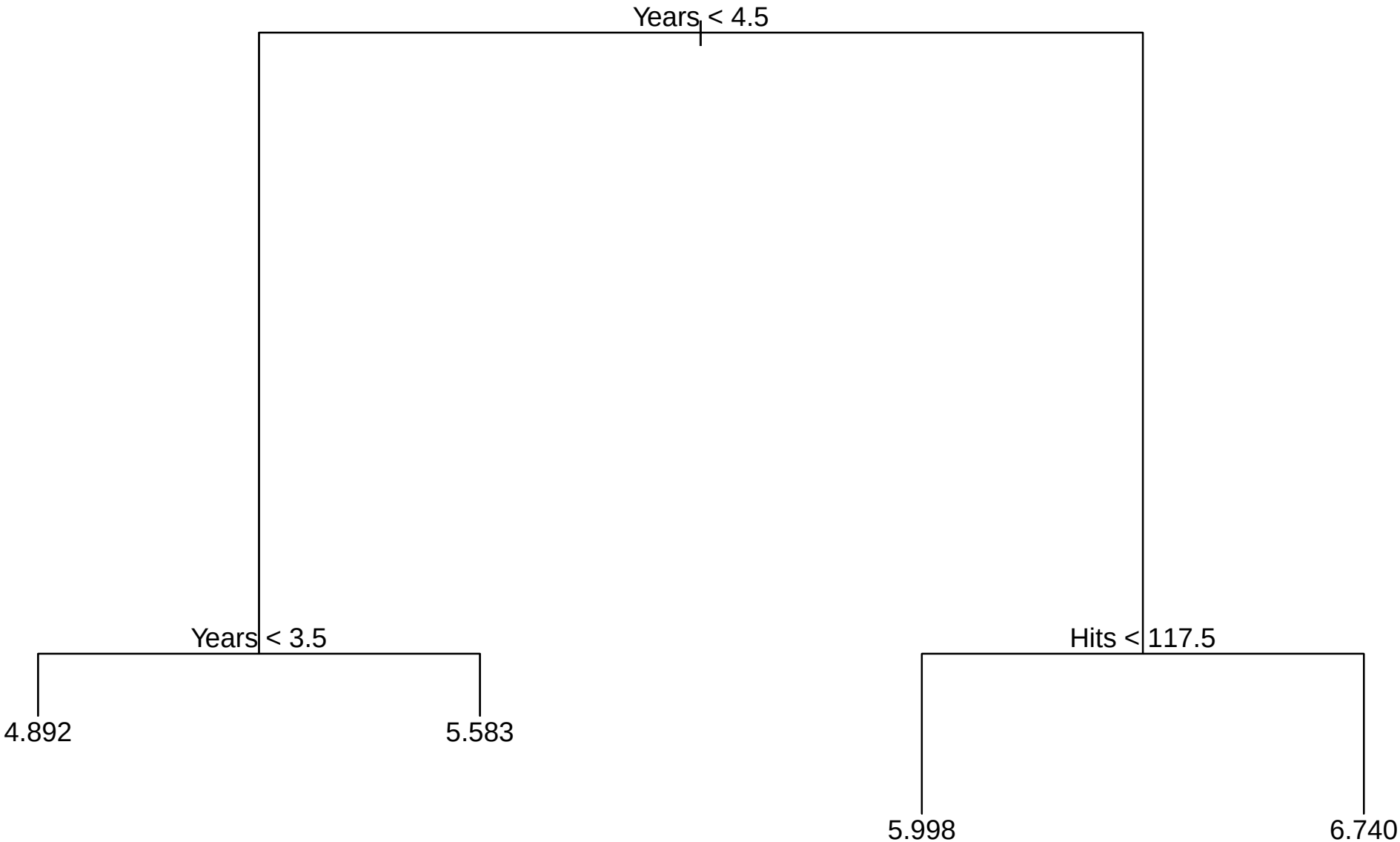
```
library(tree)
mod <- tree(log(Salary) ~  Years + Hits, data=Hit.na)
plot(mod)
text(mod, pretty=0)
```



```
cv.mod  <- cv.tree(mod, K=10)
plot(cv.mod$size, cv.mod$dev, type="b")
```



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



Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, we predict that each observation belongs to the most commonly occurring class of training in the region to which it belongs. observations in the region to which it belongs.

Details of Classification Trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the classification error rate. This is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max_k (\hat{p}_{mk})$$

Here $\hat{p}_{mk}$ represents the proportion of training observations in the m^{th} region that are from the k^{th} class.

- However classification error is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Gini Index and Deviance

- The Gini index is defined by

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

a measure of total variance across the K classes.

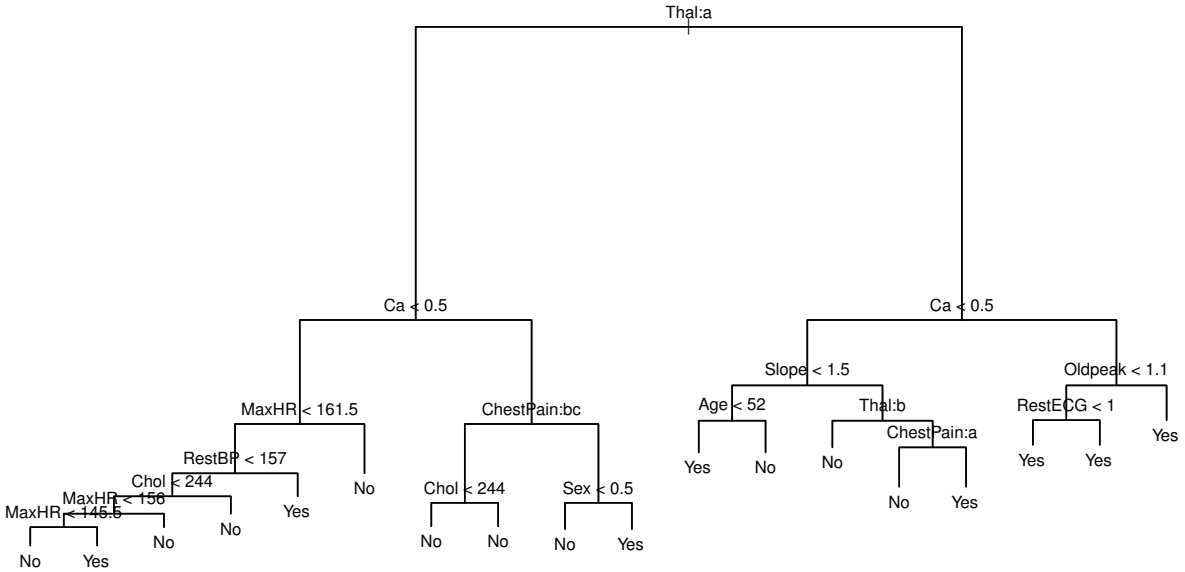
- The Gini index takes on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one.
- For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is cross-entropy, given by

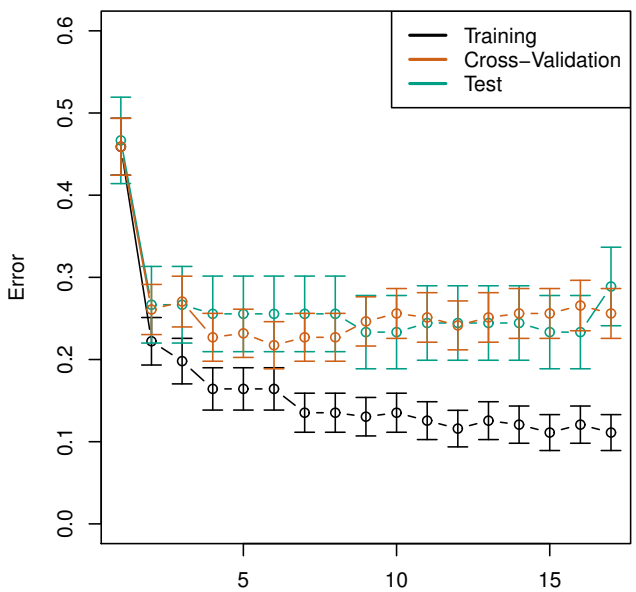
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

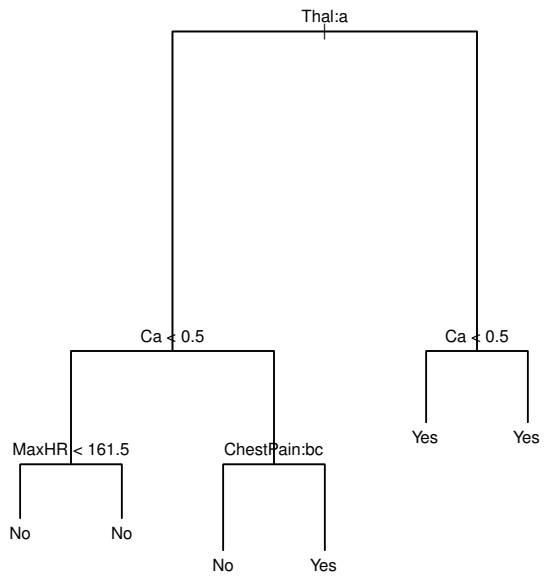
- It turns out that the Gini index and the cross-entropy are very similar numerically.

Example: Heart Data

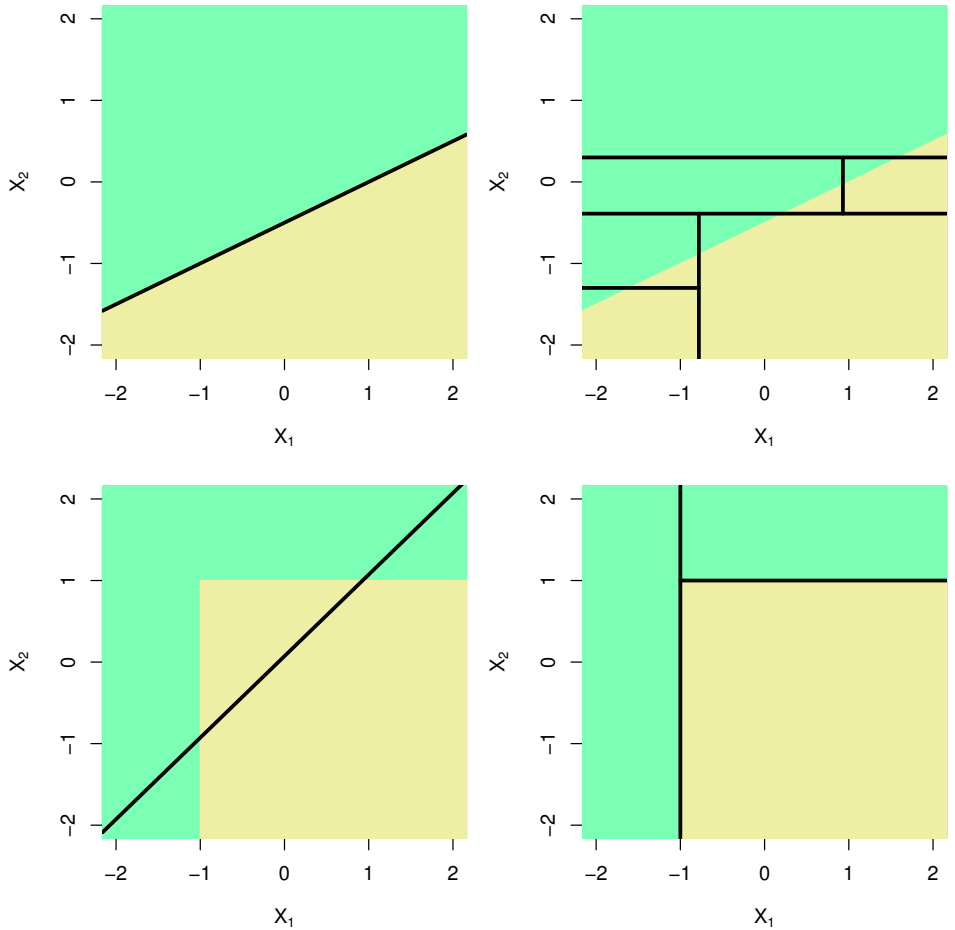
- These data contain a binary outcome HD for 303 patients who presented with chest pain.
- An outcome value of **Yes** indicates the presence of heart disease based on an angiographic test, while **No** means no heart disease.
- There are 13 predictors including **Age**, **Sex**, **Chol** (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation yields a tree with six terminal nodes.


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Slope < 1.5
Oldpeak < 1.1
Age < 52
Thal:b
RestECG < 1
RestBP < 157
Chol < 244
Chol < 244
Sex < 0.5
ChestPain:a
MaxHR < 156
MaxHR < 145.5
No
Yes


Training
Cross−Validation
Test
Error
0.0
0.1
0.2
0.3
0.4
0.5
0.6
5
10
15


Thal:a
Ca < 0.5
Ca < 0.5
MaxHR < 161.5
ChestPain:bc
Yes
No

Trees Versus Linear Models - Classification



- Top Row: True linear boundary; Bottom row: true non-linear boundary.
- Left column: linear model; Right column: tree-based model

Advantages and Disadvantages of Trees

- [A] Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- [A] Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- [A] Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- [A] Trees can easily handle qualitative predictors without the need to create dummy variables.
- [D] Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

Modeling Predictions for the US Supreme Court

Competing Approaches to Predicting Supreme Court Decision Making - Andrew D. Martin, Kevin M. Quinn, Theodore W. Ruger, and Pauline T. Kim

> *Employing two different methods, we attempted to predict the outcome of every case pending before the Supreme Court during its October 2002 term and compared those predictions to the actual decisions. One method used a statistical forecasting model based on information derived from past Supreme Court decisions [classification tree]. The other captured the expert judgments of legal academics and professionals.*

Figure 1
Estimated classification tree for Justice O'Connor for forecasted nonunanimous cases

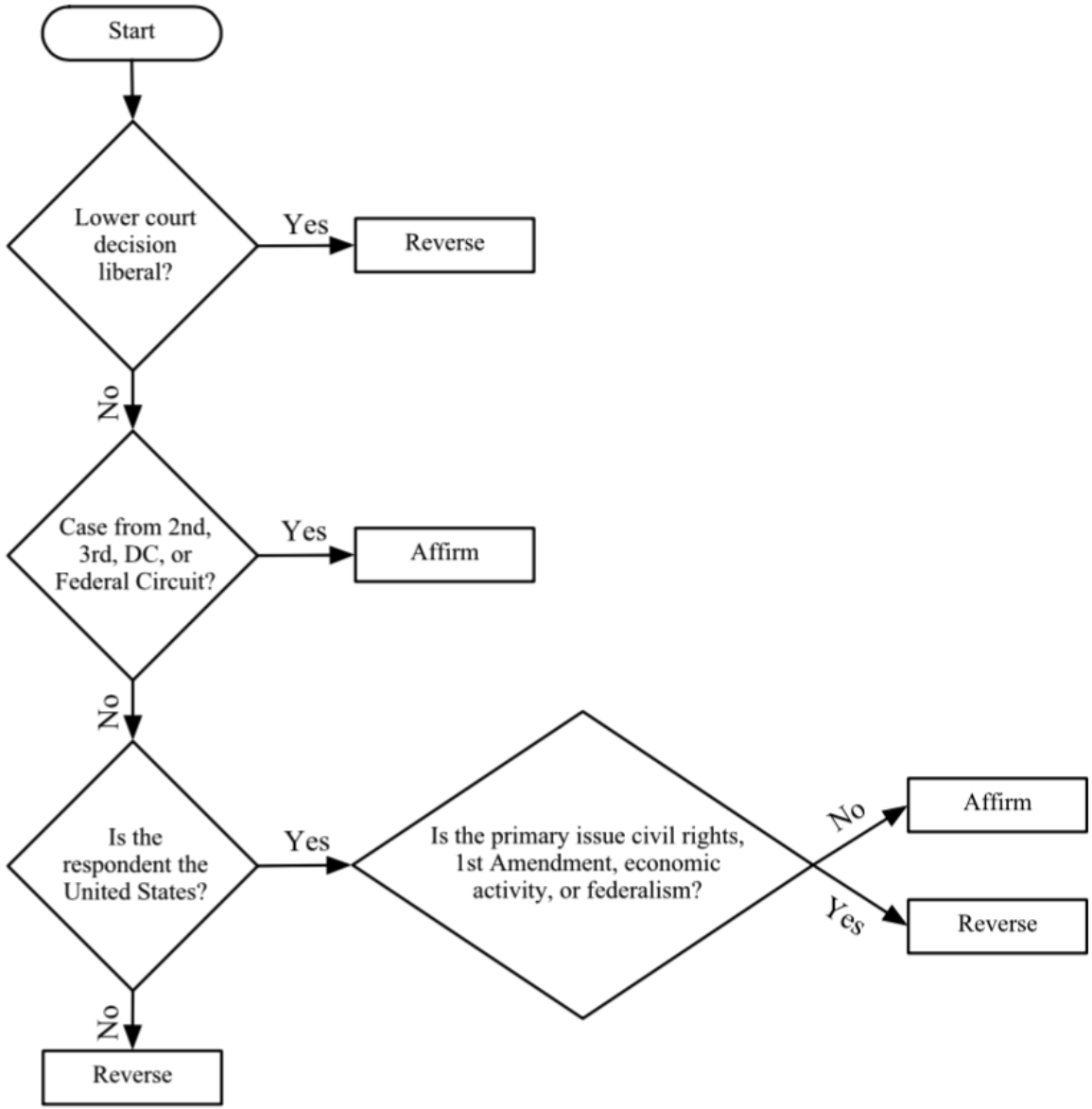


Table 1
Model and expert forecasts of case outcome for decided cases

	Case outcome forecast		
	Correct	Incorrect	Total
Model	51 (75.0%)	17 (25.0%)	68 (100.0%)
Experts	101 (59.1%)	70 (40.9%)	171 (100.0%)

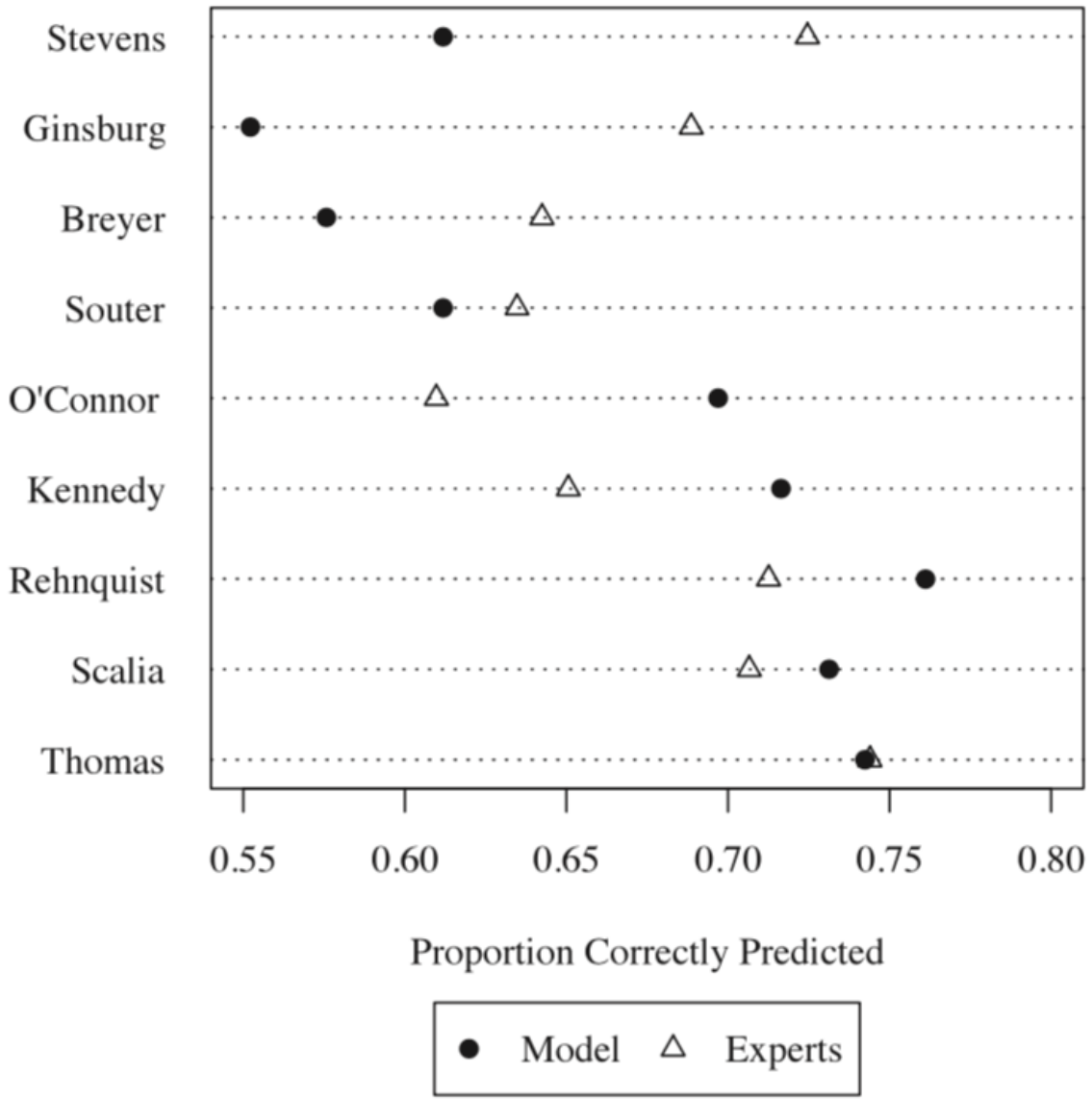
Note: Table is based on 68 cases. The unit of analysis is the case-prediction. Row percentages are in parentheses. The estimated (conditional maximum likelihood) odds ratio is 2.073 ($p = 0.025$, Fisher's exact test).

Table 2
Model and expert forecasts of individual justice's votes for decided cases

	Justice vote forecast		
	Correct	Incorrect	Total
Model	400 (66.7%)	200 (33.3%)	600 (100.0%)
Experts	1015 (67.9%)	479 (32.1%)	1494 (100.0%)

Note: Table is based on 67 cases. The unit of analysis is the justice-case-prediction. Row percentages are in parentheses. Some justices did not vote on some cases, and are thus not included. The estimated (conditional maximum likelihood) odds ratio is 0.943 ($p = 0.571$, Fisher's exact test).

Figure 2
Model and expert forecasts of votes for decided cases (n = 67), by justice.



Bagging

- Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.

- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.

- In other words, averaging a set of observations reduces variance. Of course, this is not practical because we generally do not have access to multiple training sets.

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets.
- We then train our method on the b^{th} training set in order to get $\hat{f}^{*b}(x)$. The prediction at a point x.
- We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

- This is called bagging.

Bagging Regression and Classification Trees

- Regression trees: We just apply the algorithm as described.
- For classification trees: for each test observation we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

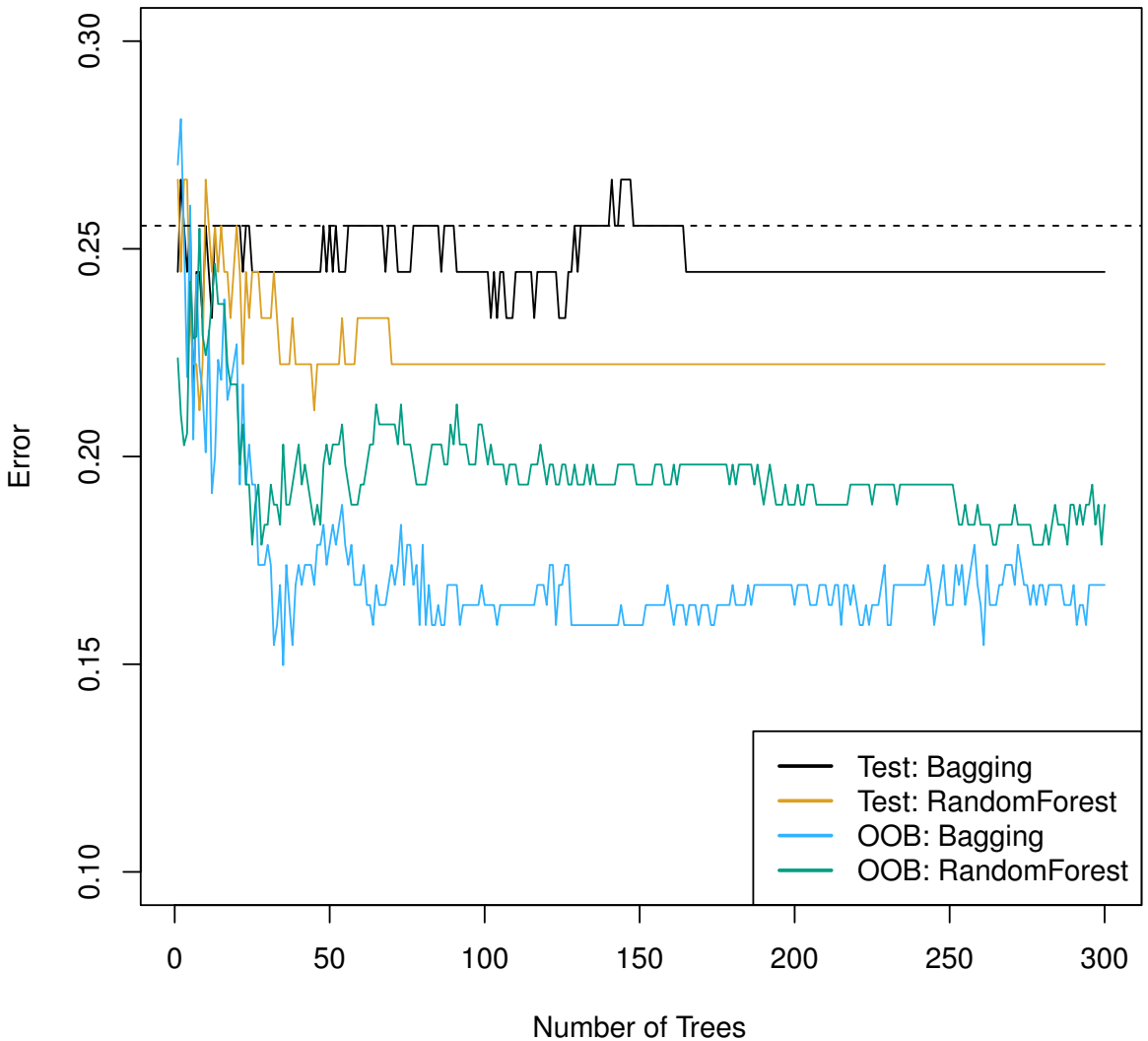
Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the i^{th} observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the i^{th} observation, which we average.

Random Forests

- Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors.
- The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$

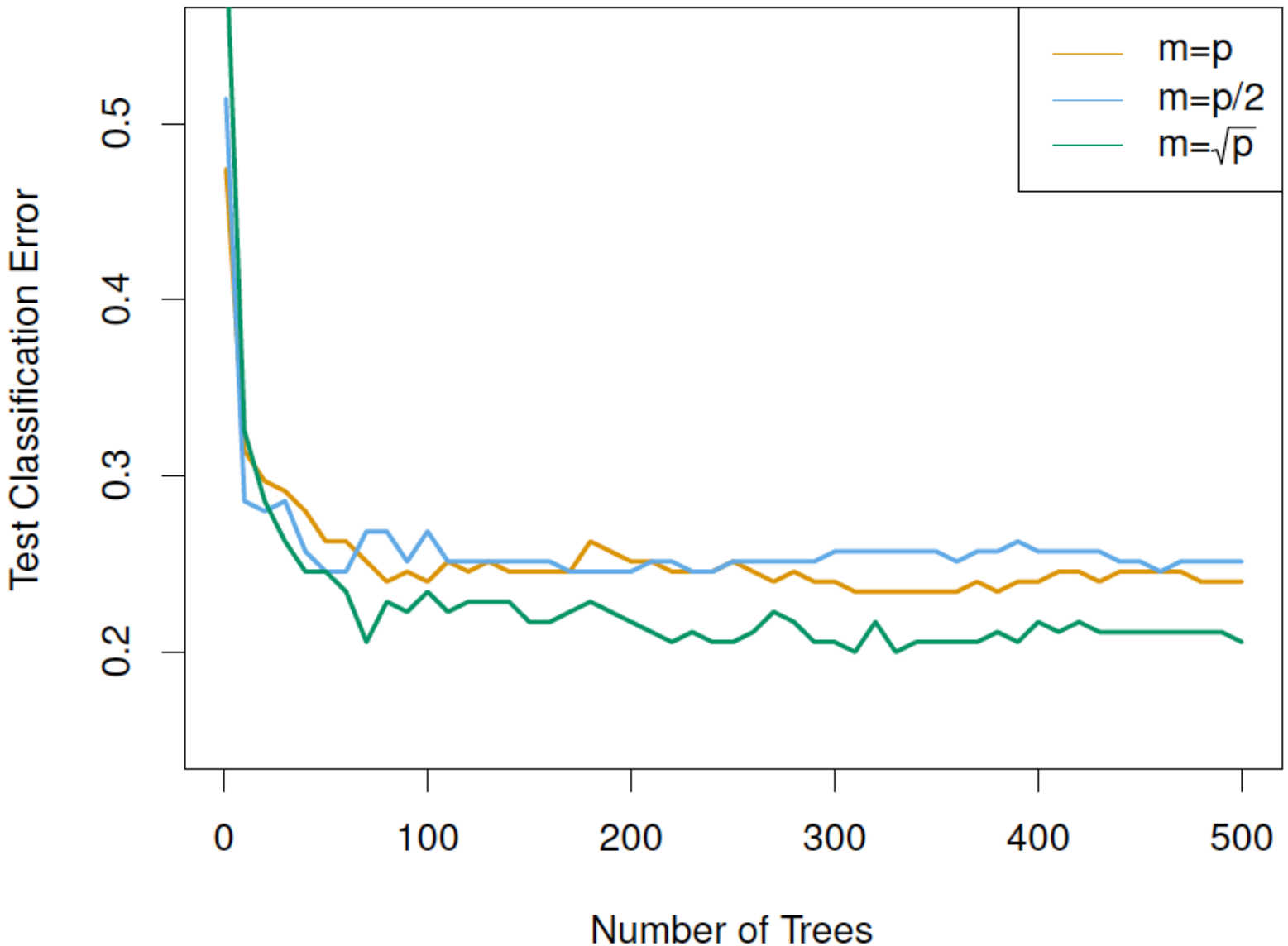
Bagging the Heart Data



- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- Random forests were applied with $m = \sqrt{p}$ (4 out of the 13 for the Heart data)
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the OOB error, which in this case is considerably lower

Example: Gene Expression Data

- Random forests was applied to a high-dimensional biological data set consisting of expression measurements of **4,718 genes** measured on tissue samples from **349 patients**.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a **qualitative label with 15 different levels**: either normal or one of 14 different types of cancer.
- Random forests was used to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.


m=p
m=p/2
m=√p
Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Boosting

- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. I will only discuss boosting for decision trees.

- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.

- Notably, each tree is built on a bootstrap data set, independent of the other trees.

- Boosting works in a similar way, except that the trees are grown sequentially: each tree is grown using information from previously grown trees.

Boosting Algorithm for Regression Trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i \quad \forall\ i$ in the training data set.

2. For $b = 1, 2, \ldots, B$, repeat:

   - Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   - Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f} \leftarrow \hat{f} + \lambda \hat{f}^b$$

   - Update the residuals:

   $$r_i \leftarrow r_i - \lambda \hat{f}^b$$

3. Output the boosted model:

$$\hat{f} = \sum_{b=1}^{B} \lambda \hat{f}^b$$

What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to fitting the data hard and potentially overfitting, the boosting approach instead learns slowly.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting for Classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex.
- Students can learn about the details in Elements of Statistical Learning, chapter 10.
- The R package gbm (gradient boosted models) handles a variety of regression and classification problems.

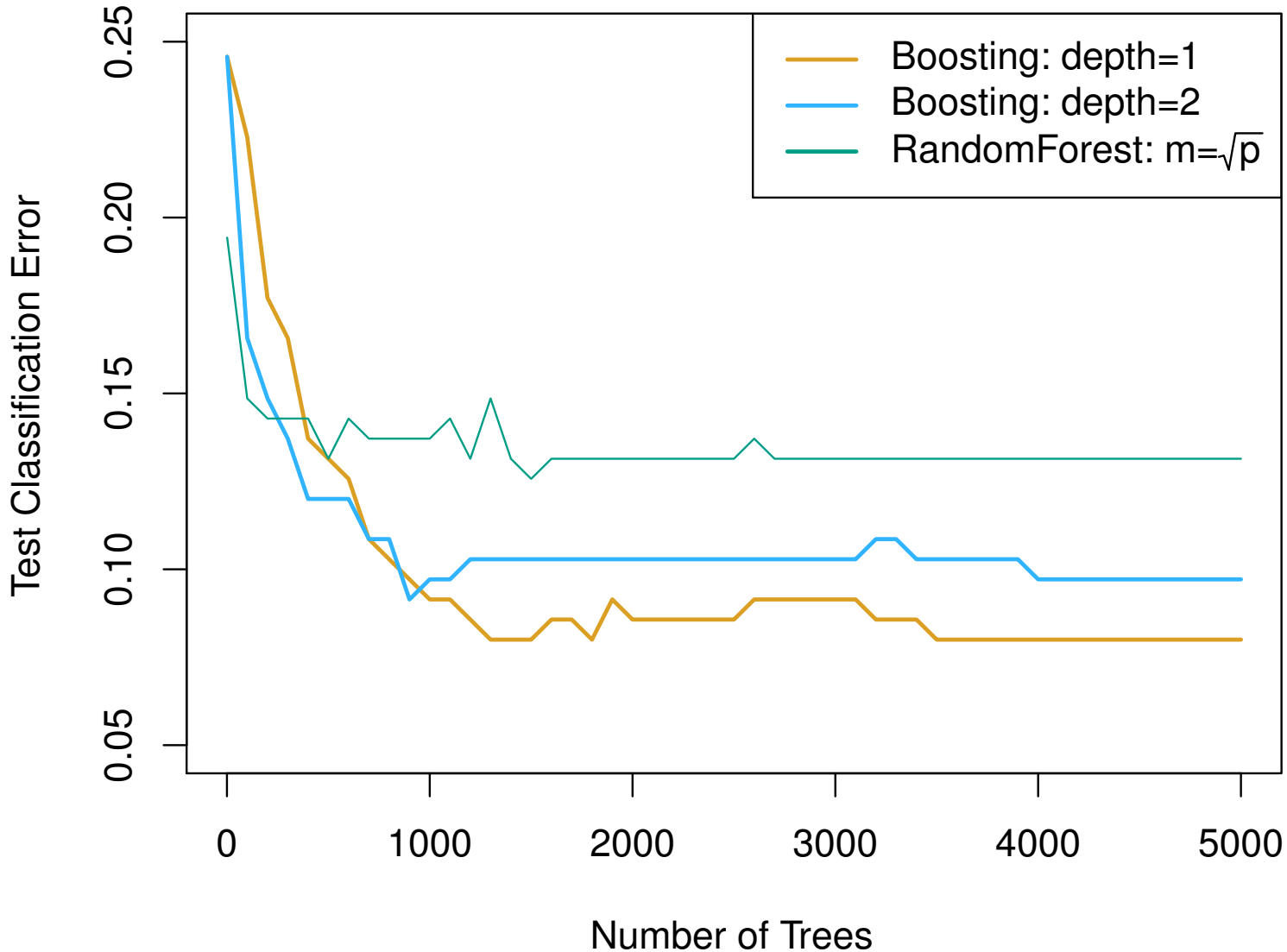
Tuning Parameters for Boosting

1. The number of trees B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.

2. The shrinkage parameter λ, a small positive number. This controls the rate at which boosting learns.
   - Typical values are 0.01 or 0.001, and the right choice can depend on the problem.
   - A very small λ can require using a very large value of B in order to achieve good performance.

3. The number of splits d in each tree, which controls the complexity of the boosted ensemble. Often d $= 1$ works well, in which case each tree is a stump, consisting of a single split and resulting in an additive model.
   - More generally d is the interaction depth, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Gene Expression Data Continued



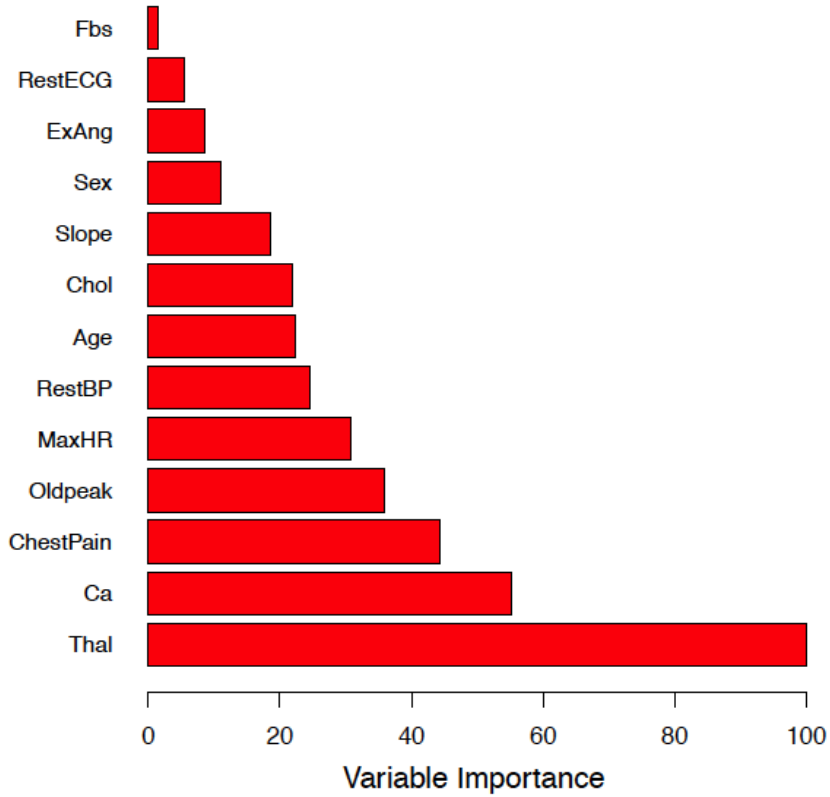
Details of Previous Figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict cancer versus normal.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$.
- Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Variable Importance Measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.

- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot for the `Heart` data

Example - Boston Data

- Housing Values in Suburbs of Boston ($n = 506$)
- Variables:
  - crim: per capita crime rate by town.
  - zn: proportion of residential land zoned for lots over 25,000 sq.ft.
  - indus: proportion of non-retail business acres per town.
  - chas: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
  - nox: nitrogen oxides concentration (parts per 10 million).
  - rm: average number of rooms per dwelling.
  - age: proportion of owner-occupied units built prior to 1940.
  - dis: weighted mean of distances to five Boston employment centers.
  - rad: index of accessibility to radial highways.
  - tax: full-value property-tax rate per $10,000.

```
library(MASS)
data(Boston)

set.seed (1)
train <- sample(1: nrow(Boston), nrow(Boston)/2)
```

```
summary(Boston)
```

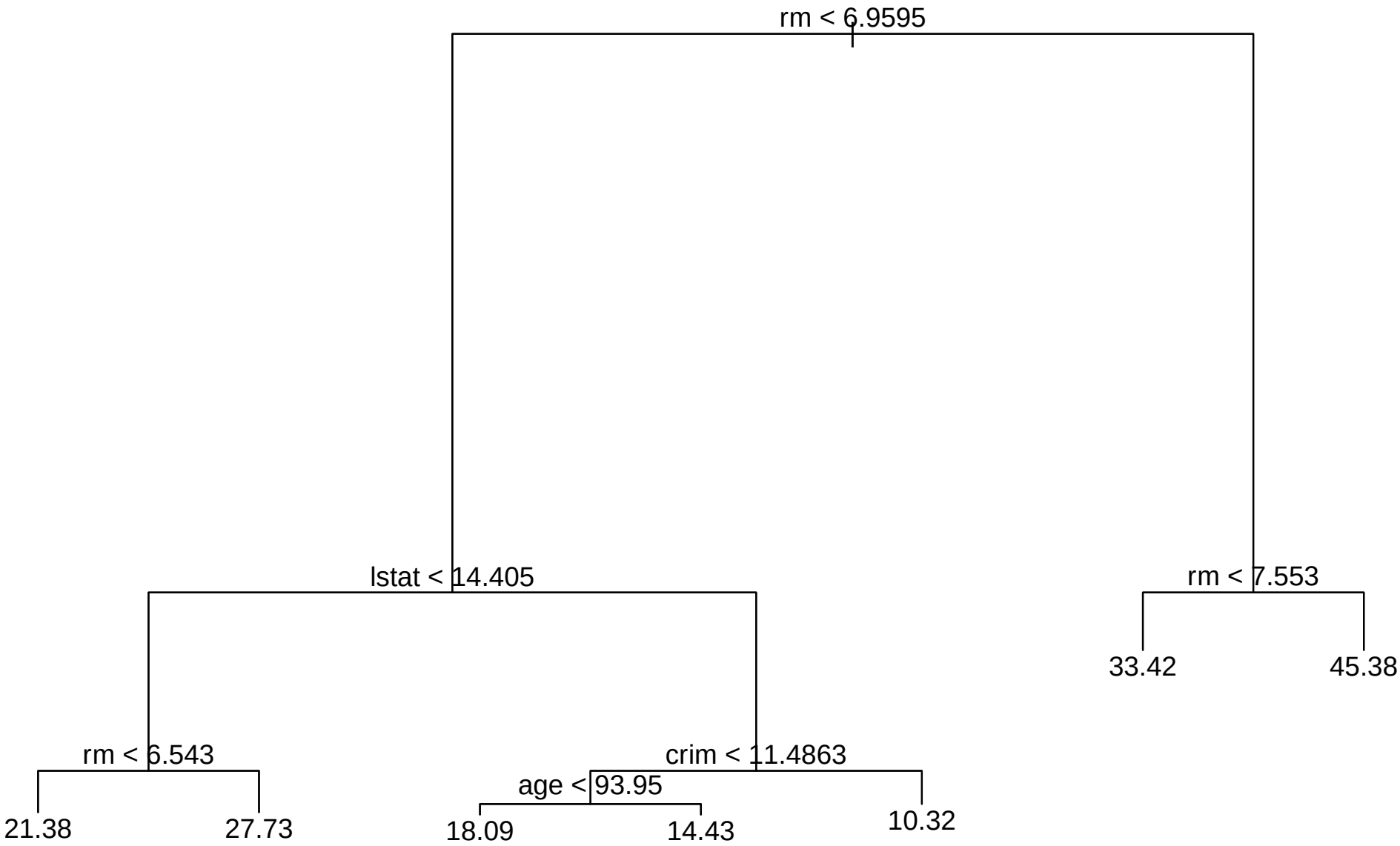
```
##       crim                zn              indus            chas        
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
##  1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
##       nox               rm              age              dis        
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
##       rad              tax           ptratio          black       
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
##      lstat            medv      
##  Min.   : 1.73   Min.   : 5.00  
##  1st Qu.: 6.95   1st Qu.:17.02  
##  Median :11.36   Median :21.20  
##  Mean   :12.65   Mean   :22.53  
##  3rd Qu.:16.95   3rd Qu.:25.00  
##  Max.   :37.97   Max.   :50.00
```

```
library (tree)
tree.boston  <- tree(medv ~ ., Boston, subset=train)
summary(tree.boston)
```

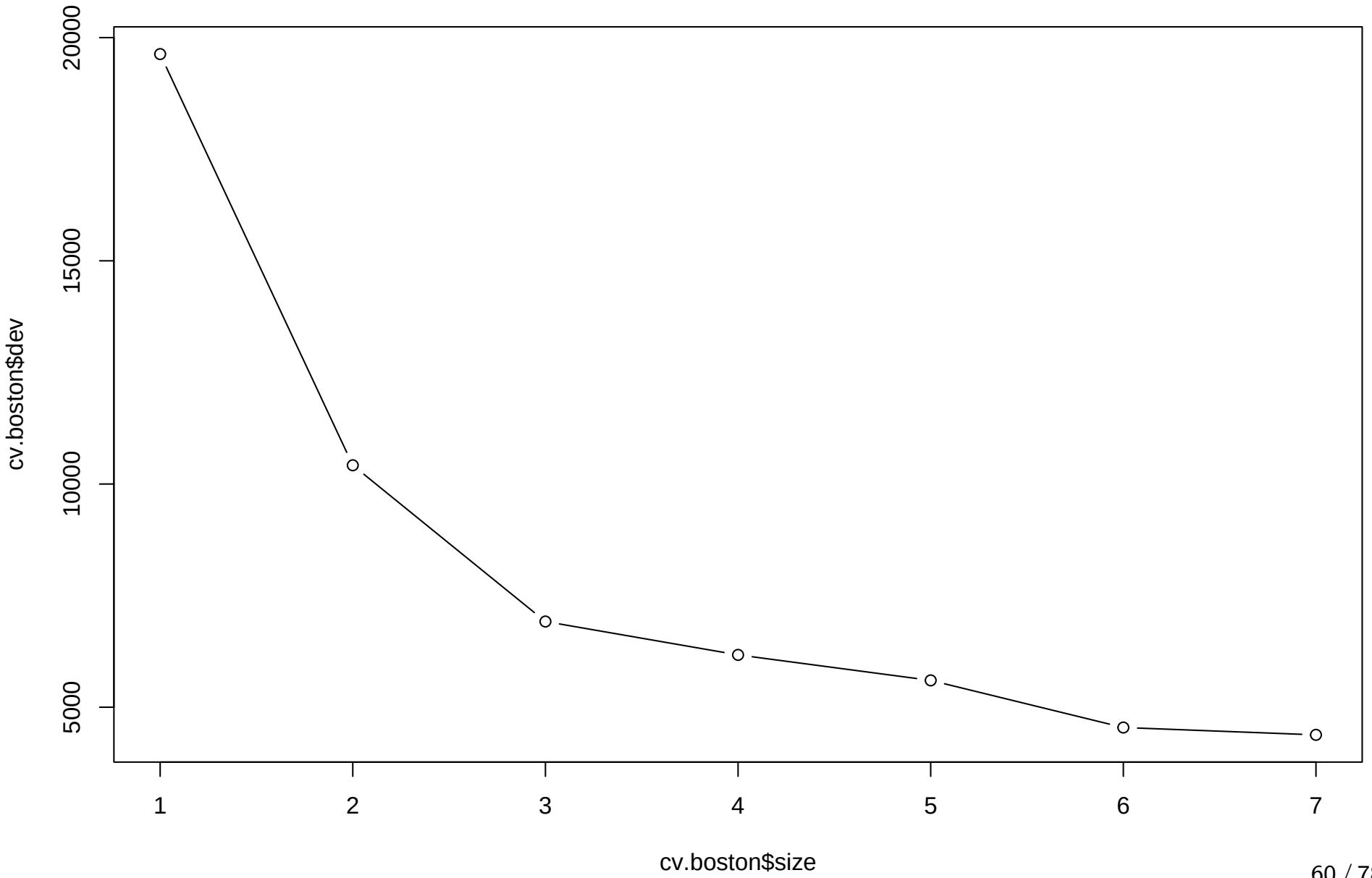
```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

- The deviance is simply the sum of squared errors for the tree.

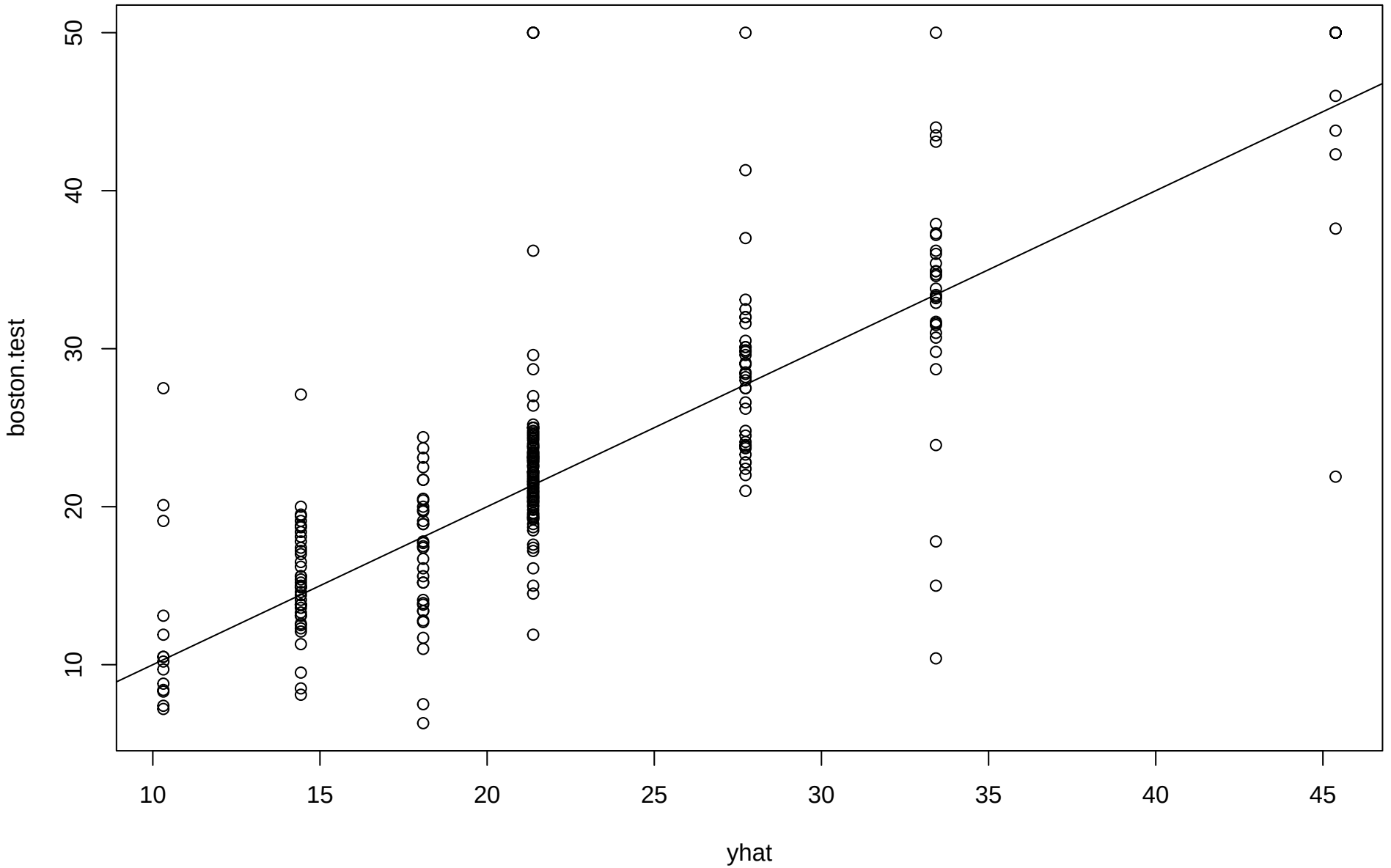
```
plot(tree.boston)
text(tree.boston, pretty=0)
```



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



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



MSE Validation/Test Set

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

```
## [1] 35.28688
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.1.2
```

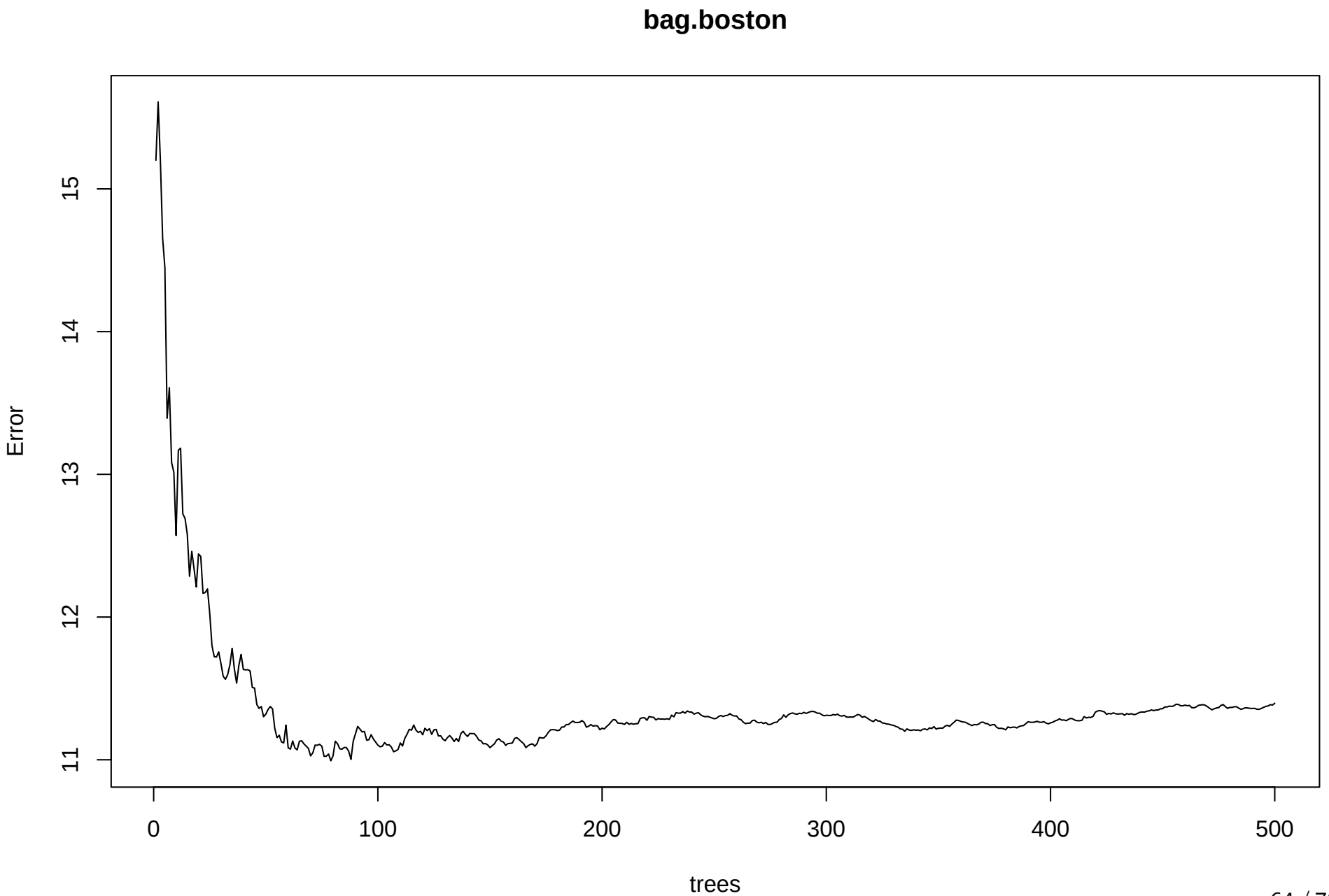
```
## randomForest 4.7-1
```

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

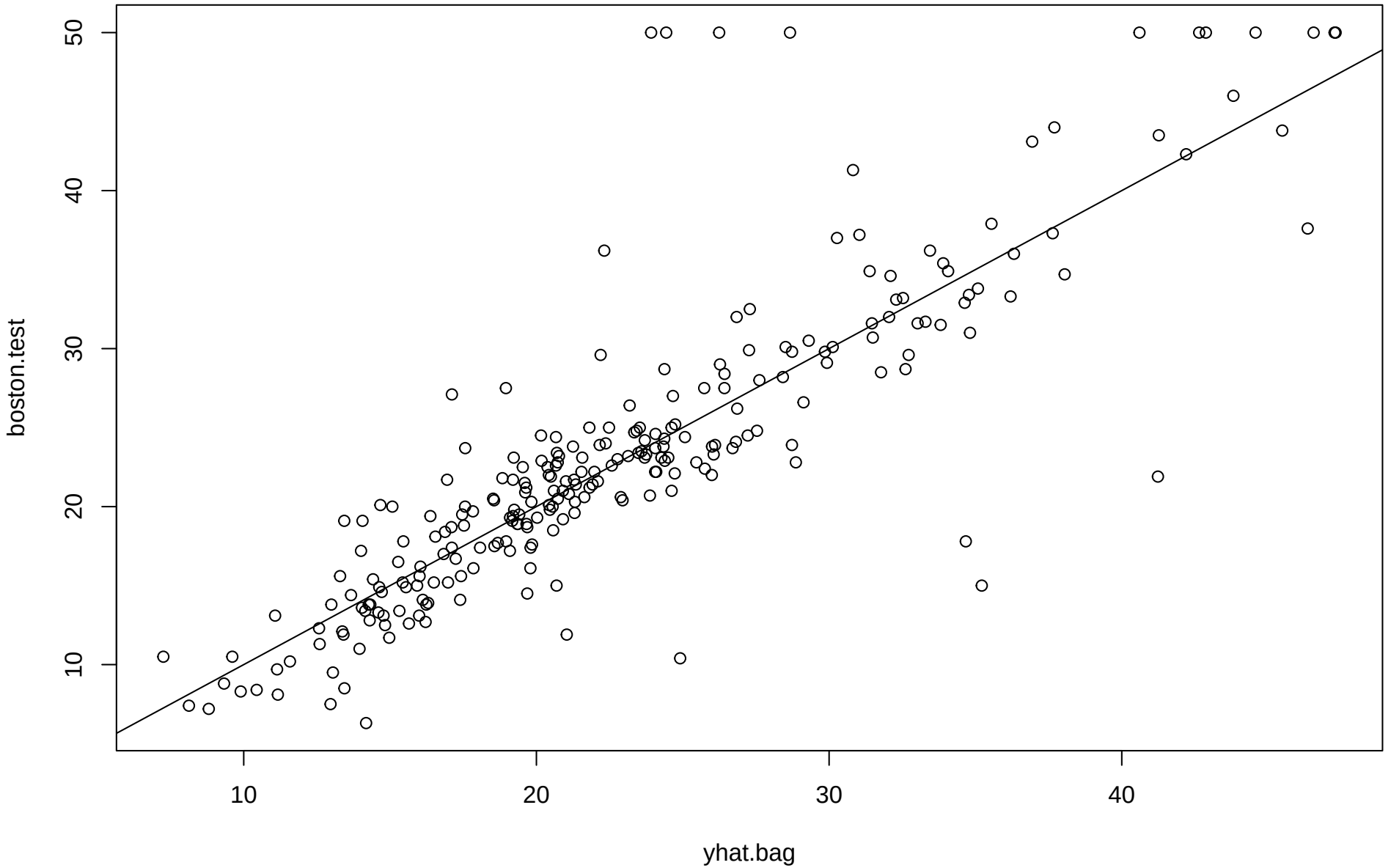
```
set.seed(1)
bag.boston <- randomForest(medv ~., data=Boston, subset=train,
                           mtry=13, importance=TRUE)
bag.boston
```

```
##
## 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: 11.39601
##                     % Var explained: 85.17
```

```
plot(bag.boston)
```



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



MSE Validation/Test Set

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

```
## [1] 23.59273
```

```
set.seed(1)
bag.boston <- randomForest(medv ~., data=Boston,
                           subset=train,
                           mtry=13, ntree =25)

bag.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = 13,
##                Type of random forest: regression
##                      Number of trees: 25
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 12.41382
##                     % Var explained: 83.85
```

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

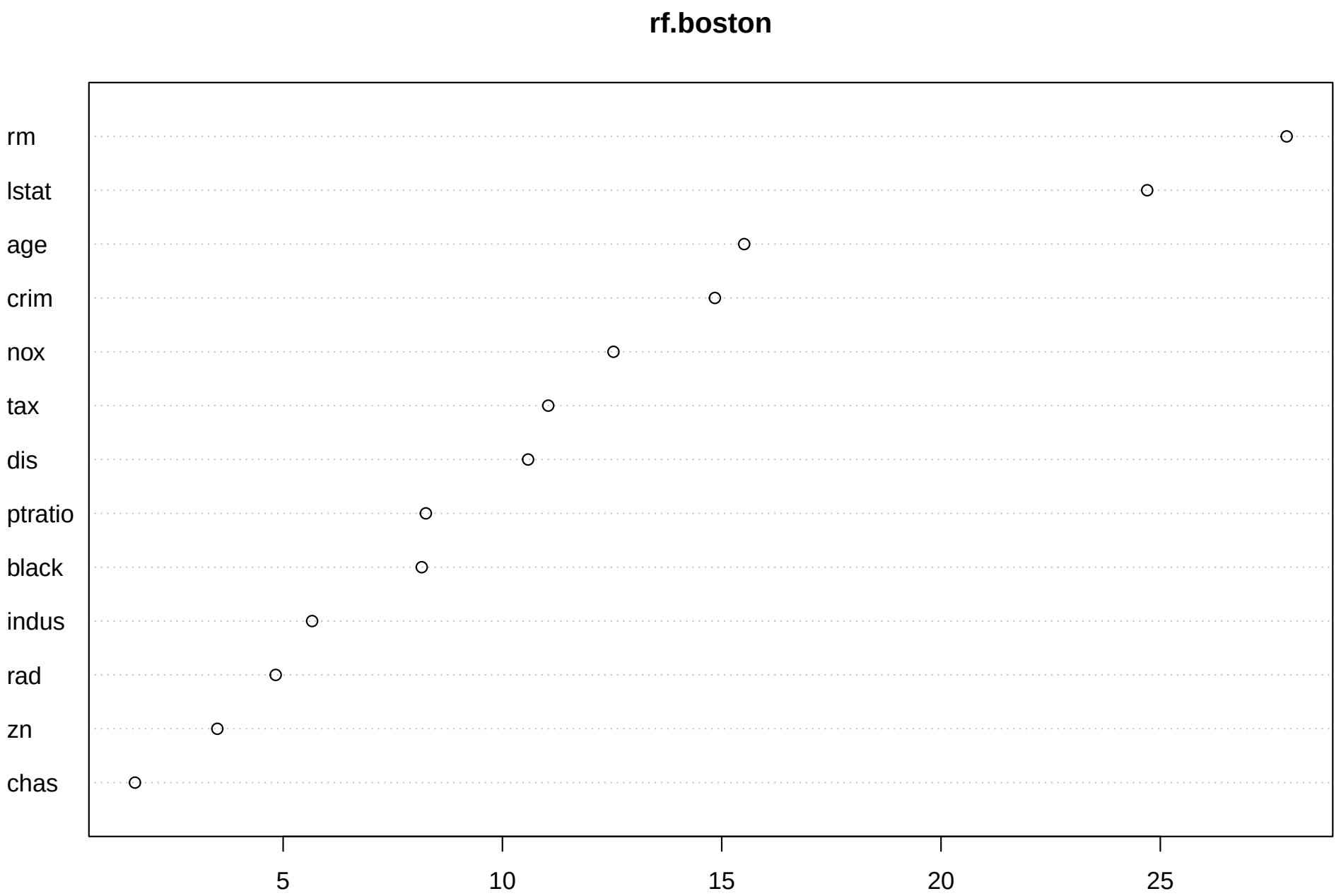
```
## [1] 23.46398
```

Random Forest

```
rf.boston <- randomForest(medv ~., data=Boston,
                          subset=train, mtry=4,
                          importance=TRUE)
yhat.rf <- predict(rf.boston, newdata=Boston[-train ,])
mean((yhat.rf - boston.test)^2)
```

```
## [1] 18.31342
```

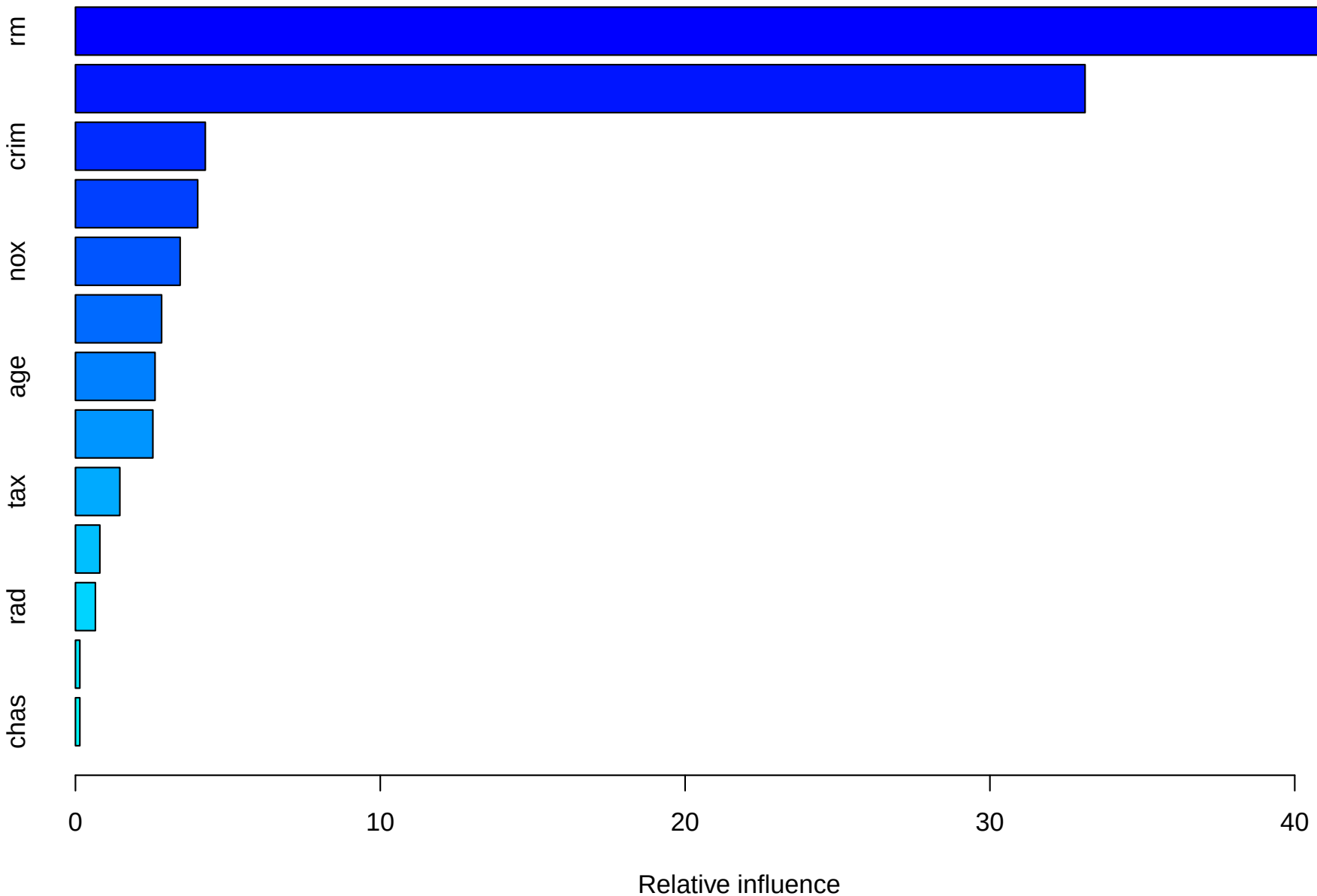
```
varImpPlot(rf.boston, type=1)
```



Boosting

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

```
summary(boost.boston)
```

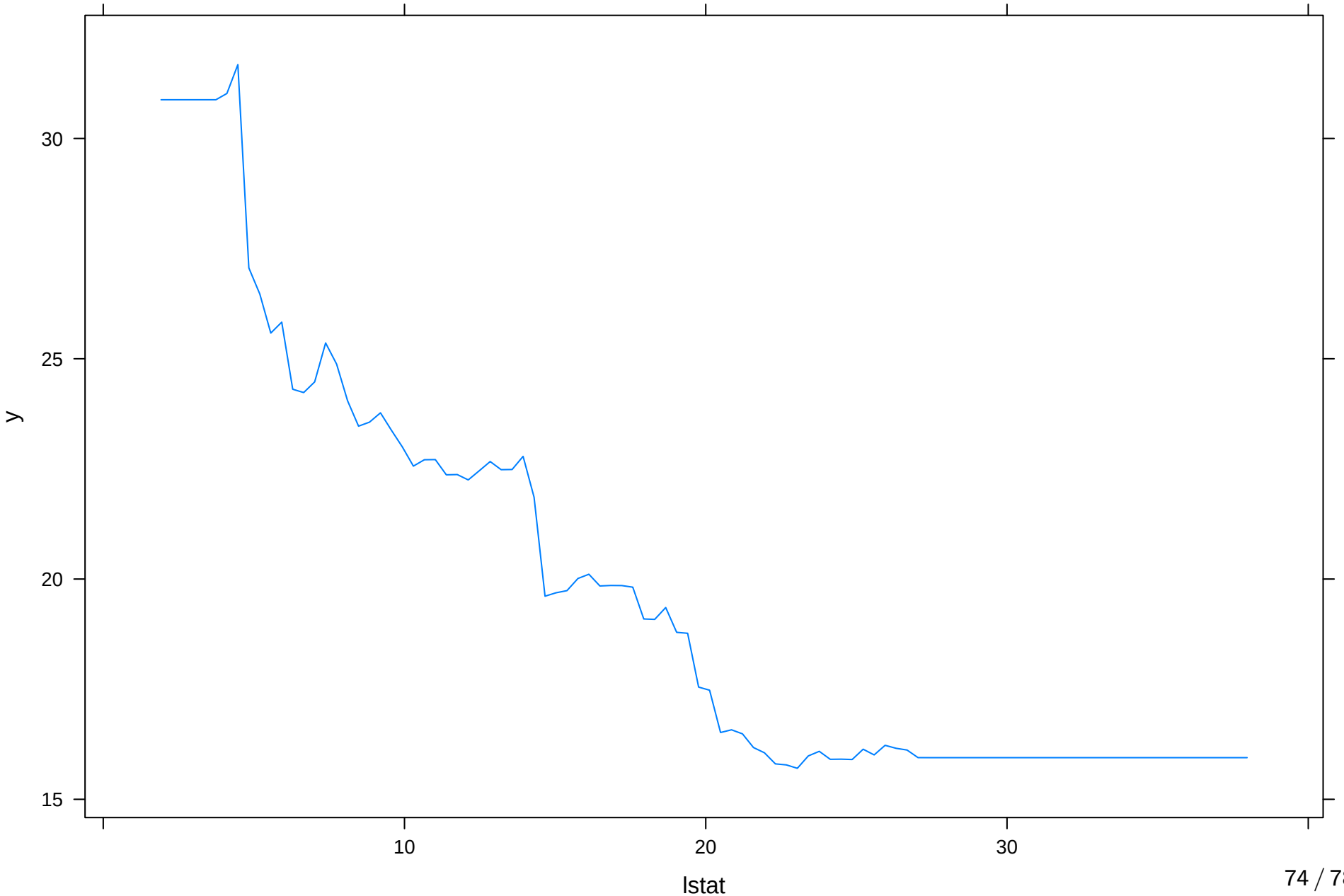



```
##                var    rel.inf
```

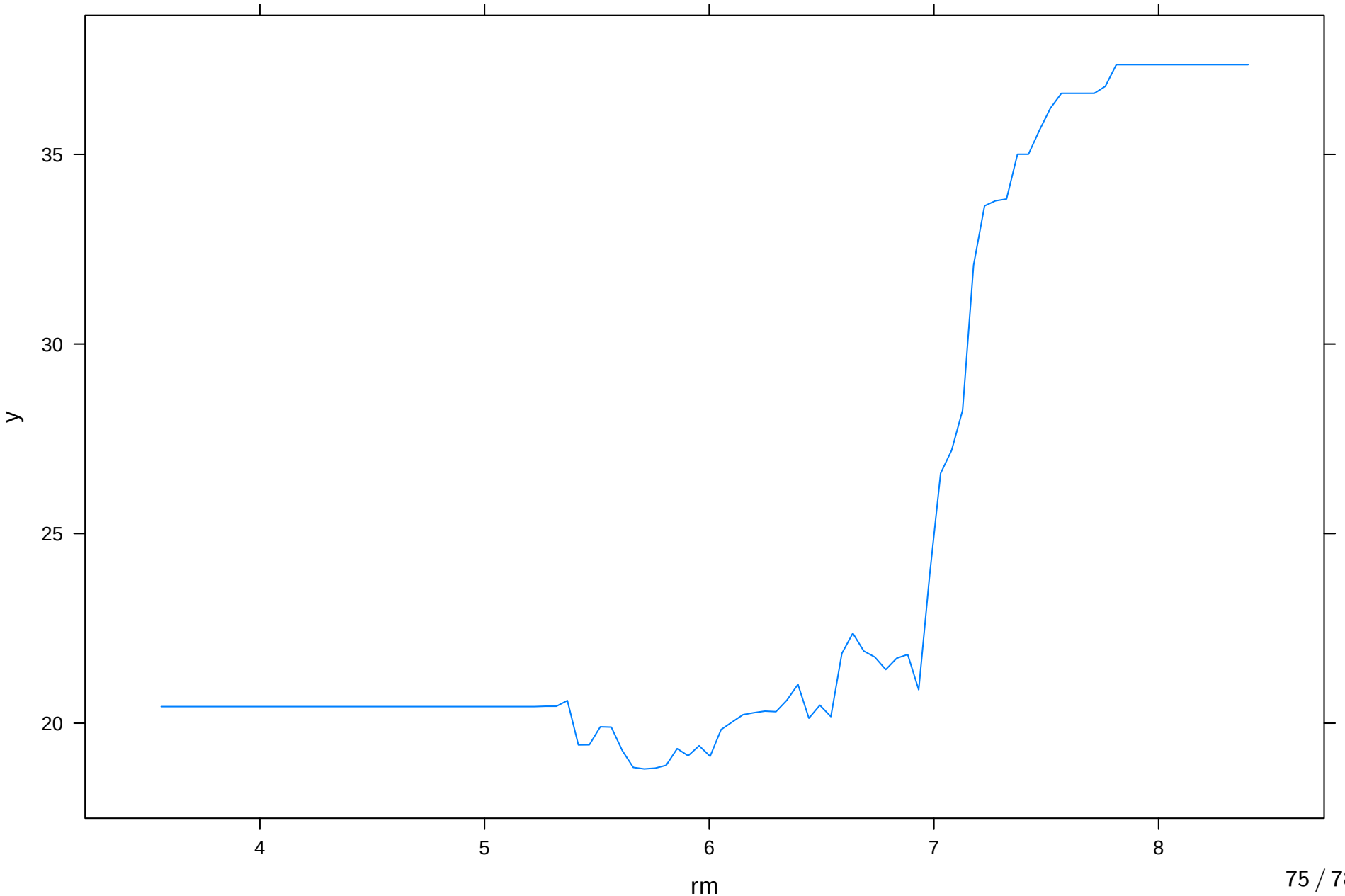
Partial Dependence Plots

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

Partial Dependence Plots



Partial Dependence Plots



Partial Dependence Plots

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

MSE Test/Validation

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

```
## [1] 18.84709
```

Summary

- Decision trees are simple and interpretable models for regression and classification.
- However they are often not competitive with other methods in terms of prediction accuracy (not always the case).
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods – random forests and boosting – are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.