Predictive Modeling

Chapter 8: Regression Trees and Rule-Based Models

STA 6543

The University of Texas at San Antonio

Overview

- Part I: General Strategies
- Part II: Regression Models
  - Chapter 6: Linear Regression and Its Cousins
  - Chapter 7: Nonlinear Regression Models
  - Chapter 8: Regression Trees and Rule-Based Models
- Part III: Classification Models
  - Chapter 12: Discriminant Analysis and Other Linear Classification Models
  - Chapter 13: Nonlinear Classification Models
  - Chapter 14: Classification Trees and Rule-Based Models

Tree-based regression models

- A motivating example about the baseball player salary data
- Various types of tree-based models
  - Basic tree-based models (e.g., single trees, model trees)
  - Bagged trees
  - Random forest
  - Boosted trees
  - Cubist
- R demonstrations for solubility data

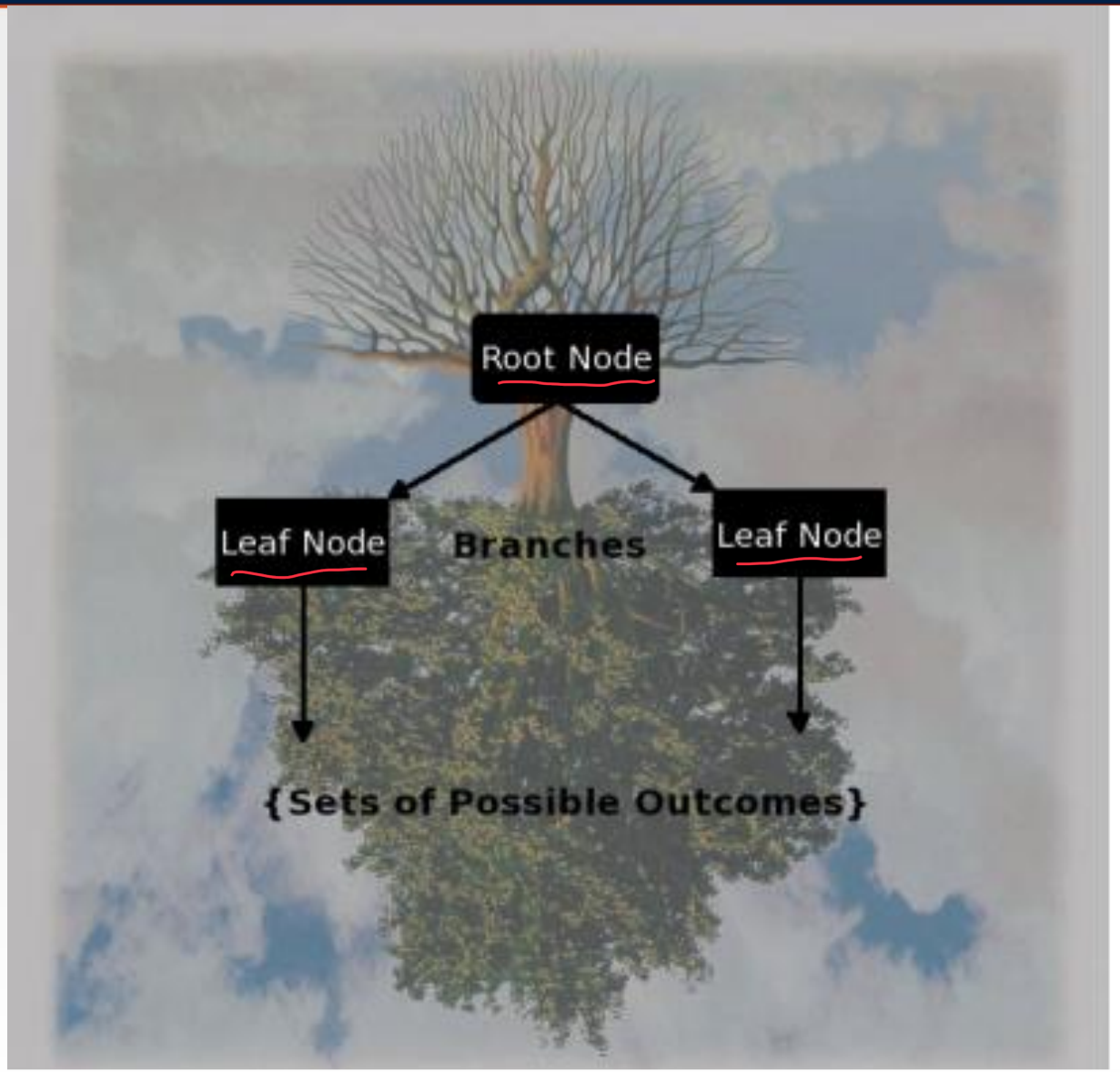
Basic tree-based methods

Chapter 8-Part II Regression Trees and Rule-Based Models

What is tree?

What is tree?

What is tree?


Root Node
Leaf Node
Branches
Leaf Node
{Sets of Possible Outcomes}

Overview

- These involve *stratifying* or *segmenting* the predictor space into a number of simple regions.
- Tree-based methods are useful for interpretation.
- They typically are not competitive with the best supervised learning approaches, such as those seen in Chapters 6 and 7, in terms of prediction accuracy.
- To improve prediction accuracy, we introduce *bagging, random forests, and boosting,* at the expense of some loss in interpretation.
- Decision trees can be applied to both *regression (Chapter 8)* and *classification* problems (Chapter 14).

A motivating example: baseball player salary data

- The Hitters data set is provided in the R package ISLR
- We use this data to predict a baseball players *Salary* based on Years (the number of years that he has played in the major leagues) and Hits (the number of hits that he made in the previous year)

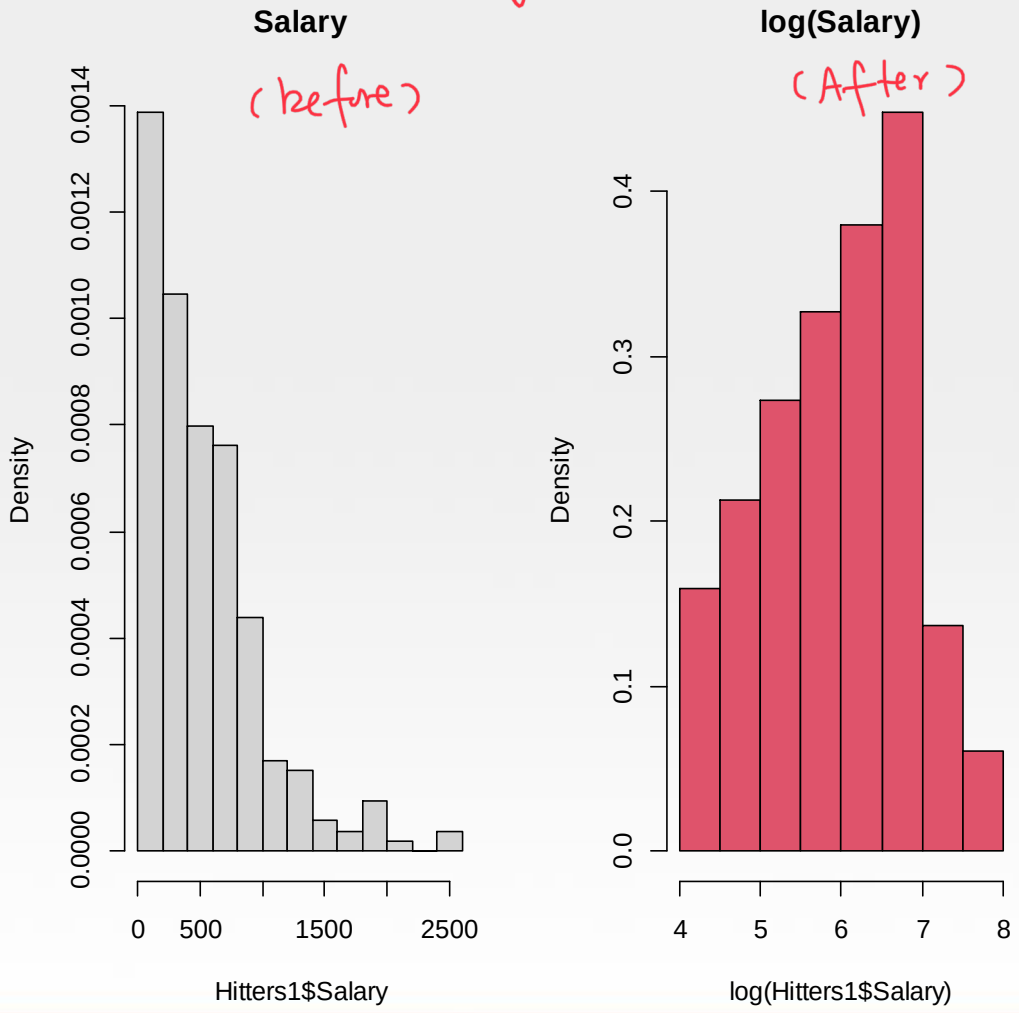
Response: Salary

predictor: Years and Hits

Data preprocessing

"mice" to impute the data with missing values

- Remove observations that are missing Hitters or Salary values.
- Log-transform Salary so that its distribution has more of a typical bell-shape.



R code

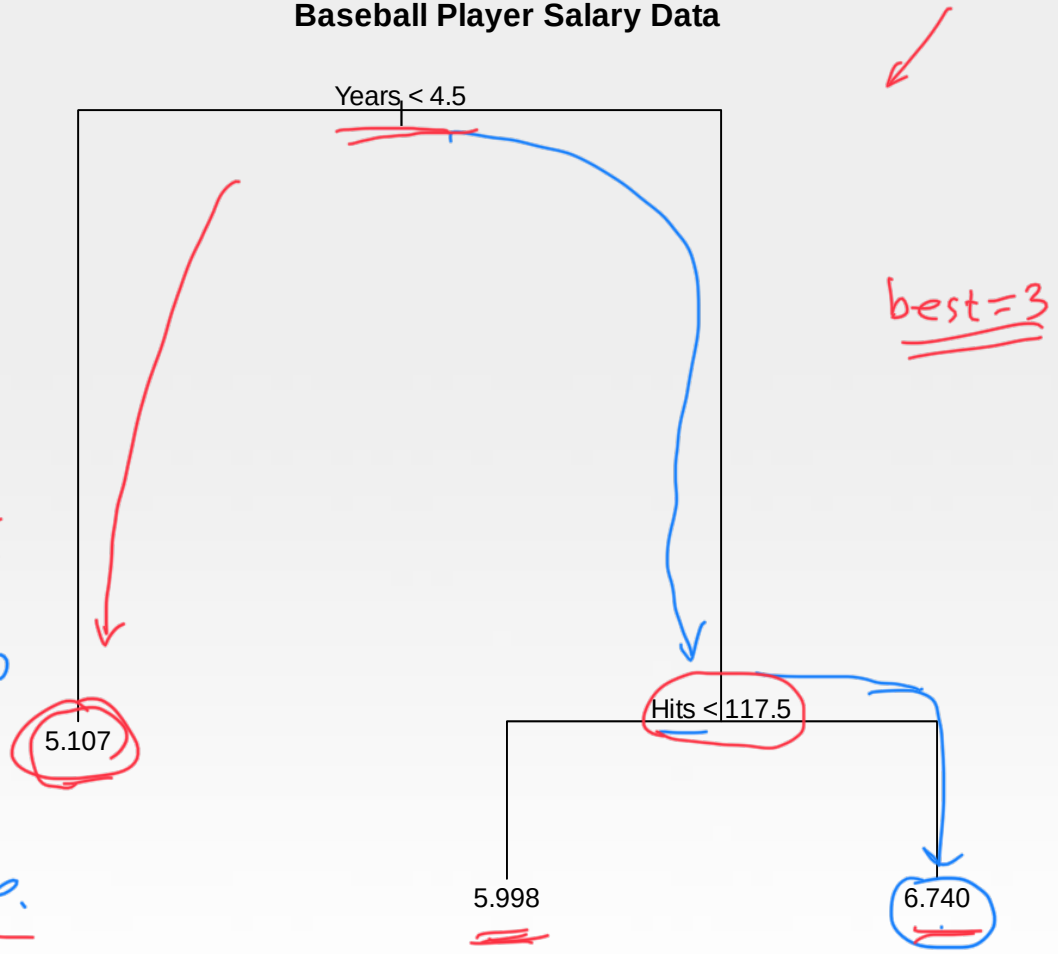
```
### Load packages
library(AppliedPredictiveModeling)
library(caret)
library(ISLR) #access the Hitters data
library(tree)

#### Example: Baseball Player Salary Data (Hitters)
Hitters1 = na.omit(Hitters) ← remove missing values
par(mfrow=c(1,2))
hist(Hitters1$Salary,prob=T, main="Salary") ← before
hist(log(Hitters1$Salary),prob=T, main="log(Salary)", col=2) ← after
dev.off()

lm ← Linear regression
sal.tree = tree(log(Salary) ~ Years + Hits, data = Hitters1)
sal.tree3 = prune.tree(sal.tree, best=3) #the number of terminal nodes is 3
plot(sal.tree3)
text(sal.tree3, pretty=0)
title("Baseball Player Salary Data")
```

A motivating example: baseball player salary data

- We want to predict a baseball player's Salary based on Years and Hits. Here log-transform is applied to Salary.



- A new player with years being 3

5.107: ⇒ Salary = $1000 × $e^{5.107}$ = $165,174

- A new player with years being 7, hitter = 200

6.74: ⇒ Salary = $1000 × $e^{6.74}$ = $845,346.

best = 3

Regression trees: baseball example

- The regression tree is easy to interpret and has a nice graphical representation.
- *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.

The three-region partition

- The tree stratifies the players into three regions:

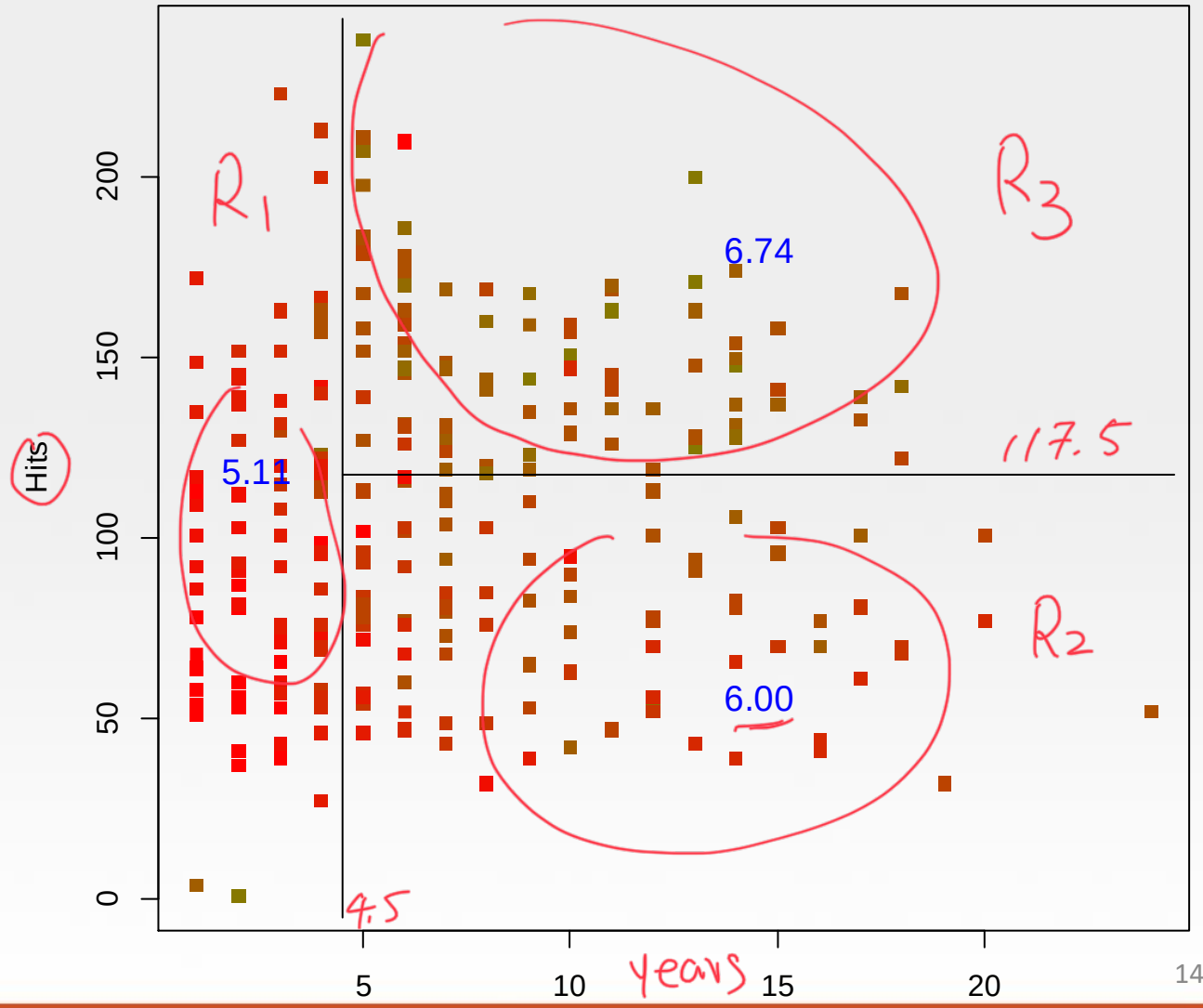
$$R_1 = \{X | Years < 4.5\},$$
$$R_2 = \{X | Years \geq 4.5, Hits < 117.5\}, \text{ and}$$
$$R_3 = \{X | Years \geq 4.5, Hits \geq 117.5\}.$$

The predicted salaries for three regions

R_1: $\$1000 \times e^{5.107} = \$165,174$

R_2: $\$1000 \times e^{6.00} = \$402,834$

R_3: $\$1000 \times e^{6.740} = 845,346.$



R code

```
#The three-region partition
rbPal <- colorRampPalette(c('red','green'))
Hitters1$Col <- rbPal(20)[as.numeric(cut(log(Hitters1$Salary),
breaks = 10))]
plot(Hitters1$Years,Hitters1$Hits,pch = 15, xlab = "Years",
ylab = "Hits", col = Hitters1$Col)
partition.tree(sal.tree3,add=T, cex = 1.2, col = "blue")
```

Building a regression tree

- We divide the predictor space—that is, the set of possible values for X_1, X_2, . . . , X_p—into J distinct and non-overlapping regions, R_1, R_2, ..., R_J .
- 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 predictors into high-dimensional *rectangles*, or boxes, for simplicity and ease interpretation of the resulting predictive model.

How to construct the regions?

- The goal is to find boxes R_1, R_2, ..., R_J that minimize

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

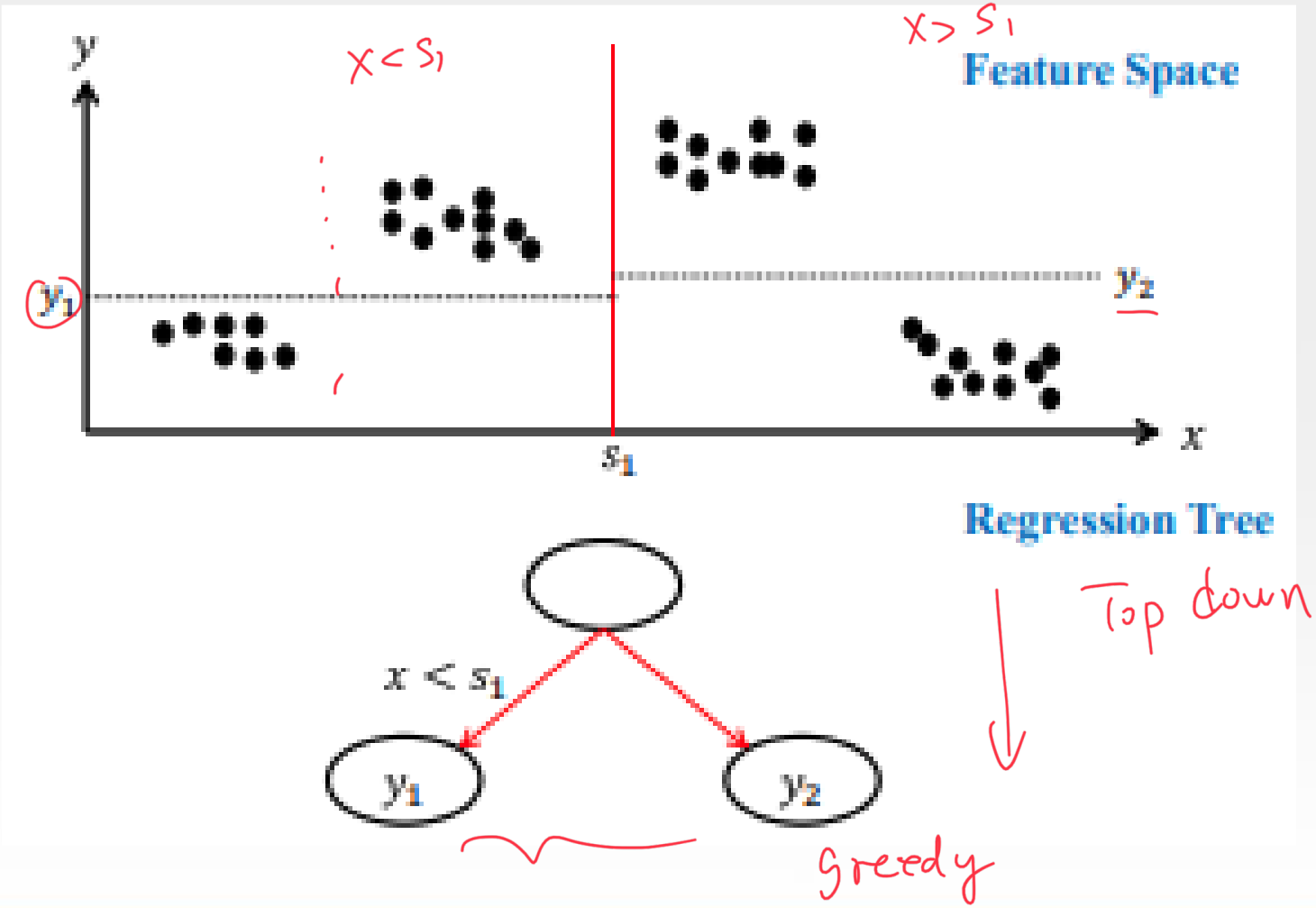
where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- The top-down, greedy approach is employed.

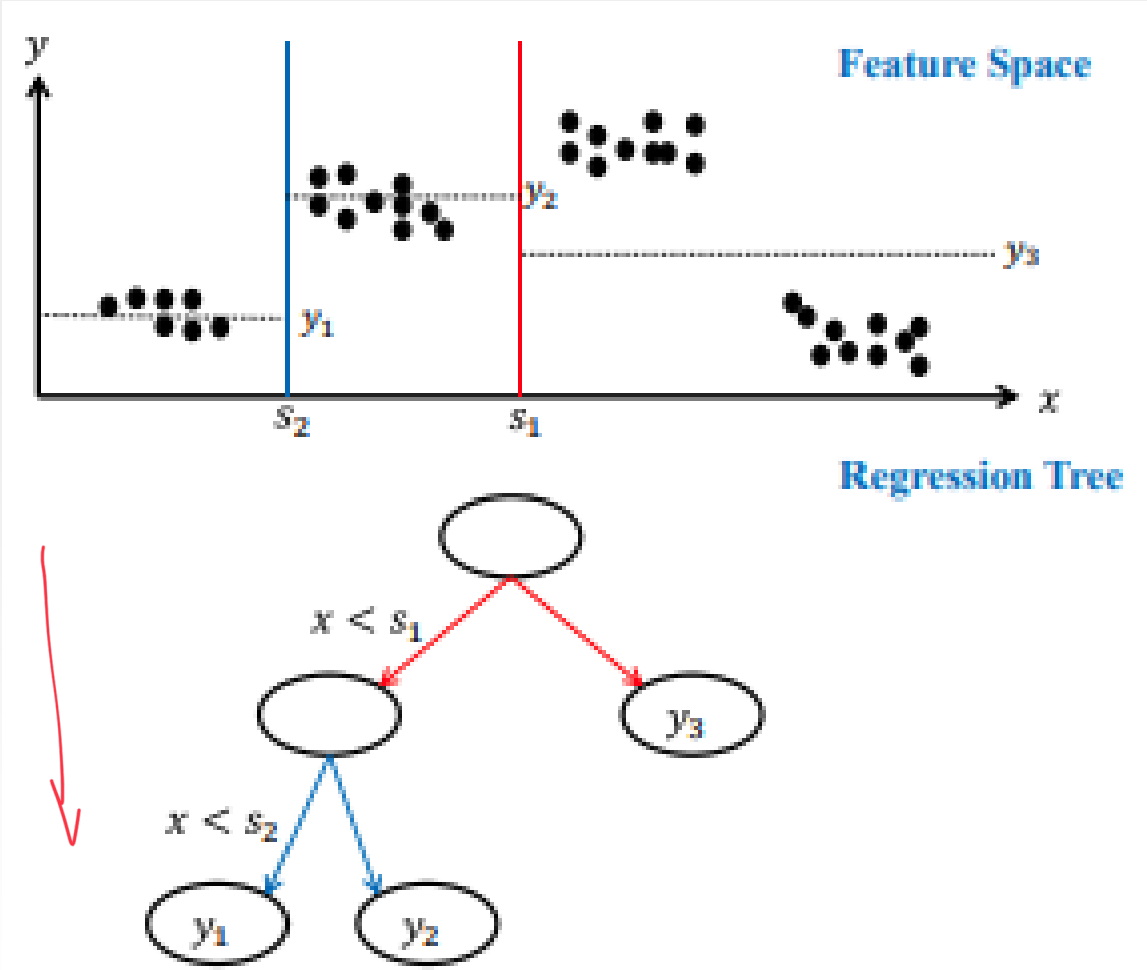
The top-down, greedy approach

- It is known as *recursive binary splitting*.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- 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.

Example of regression trees



Example of regression trees

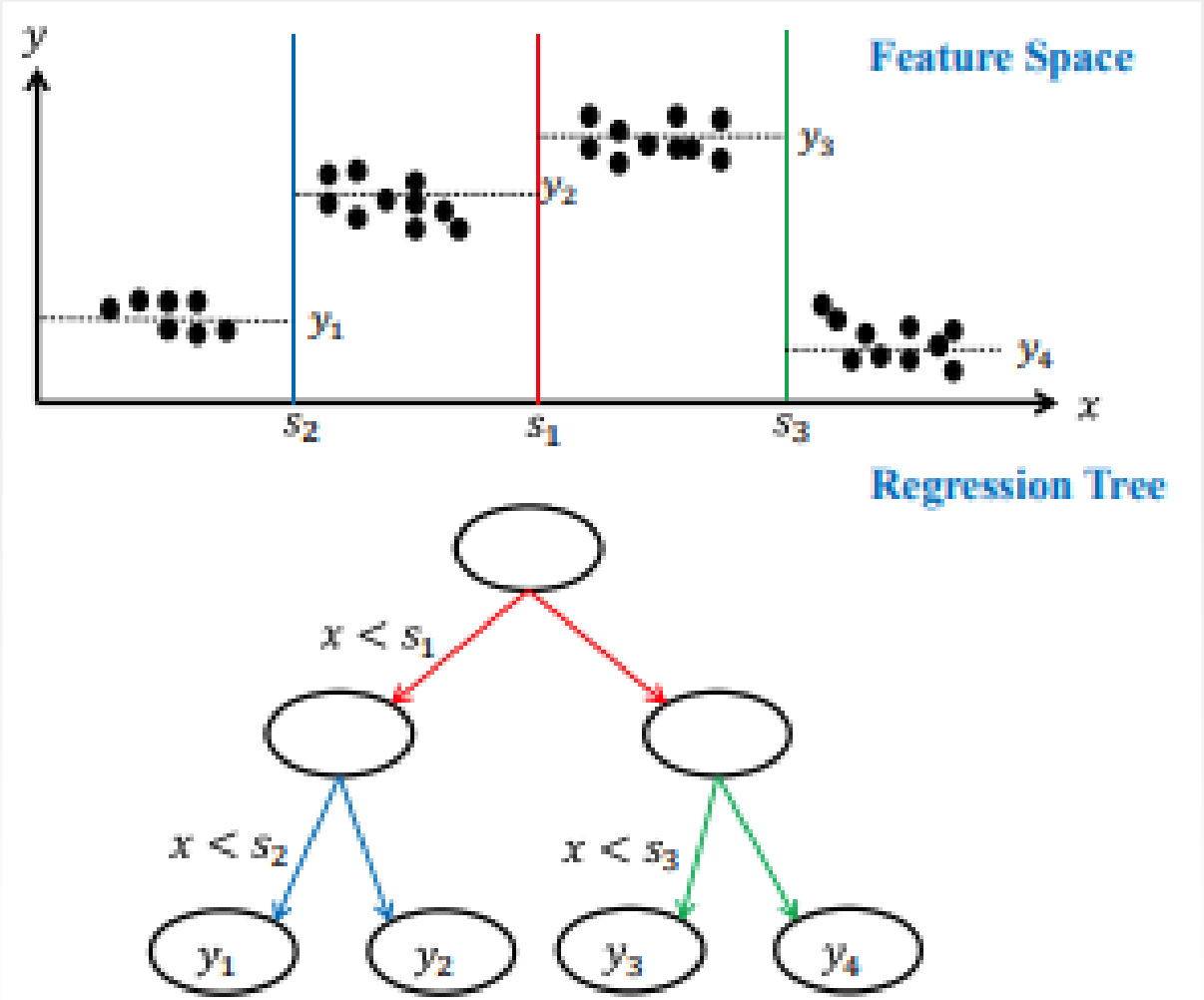


Top-down

Example of regression trees

- When should tree growing be stopped?

Tree can grow up quickly



Revisit baseball player salary data

> summary(sal.tree)

Regression tree:

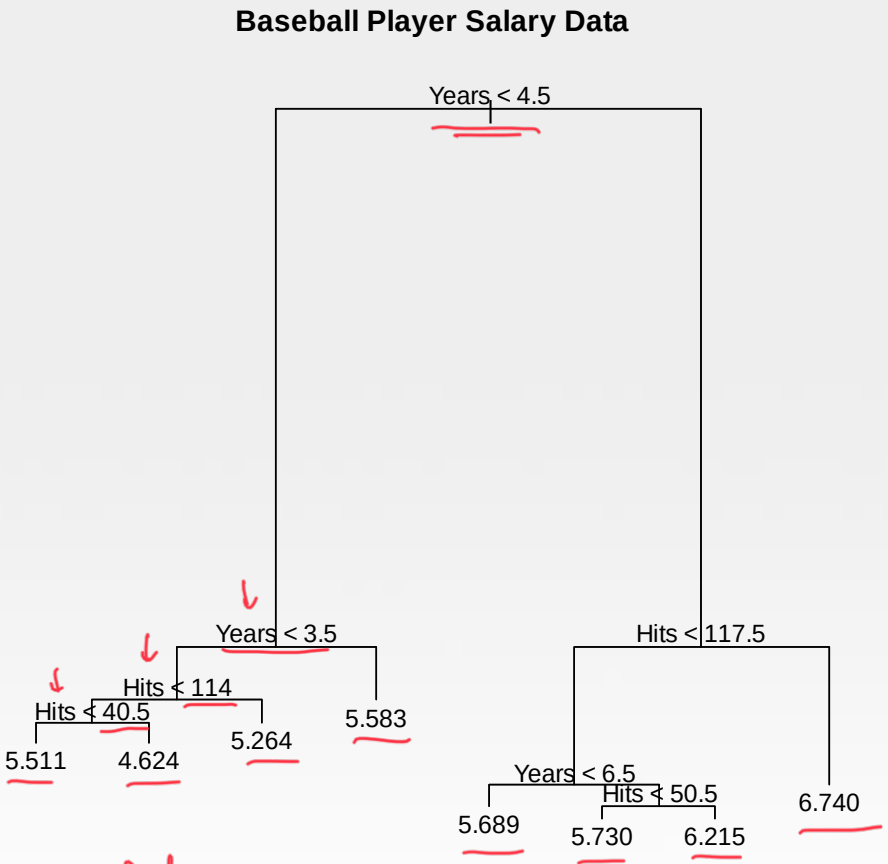
tree(formula = log(Salary) ~ Years + Hits, data = Hitters1)

Number of terminal nodes: 8

Residual mean deviance: 0.2708 = 69.06 / 255

Distribution of residuals:

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-2.2400	-0.2980	-0.0365	0.0000	0.3233	2.1520

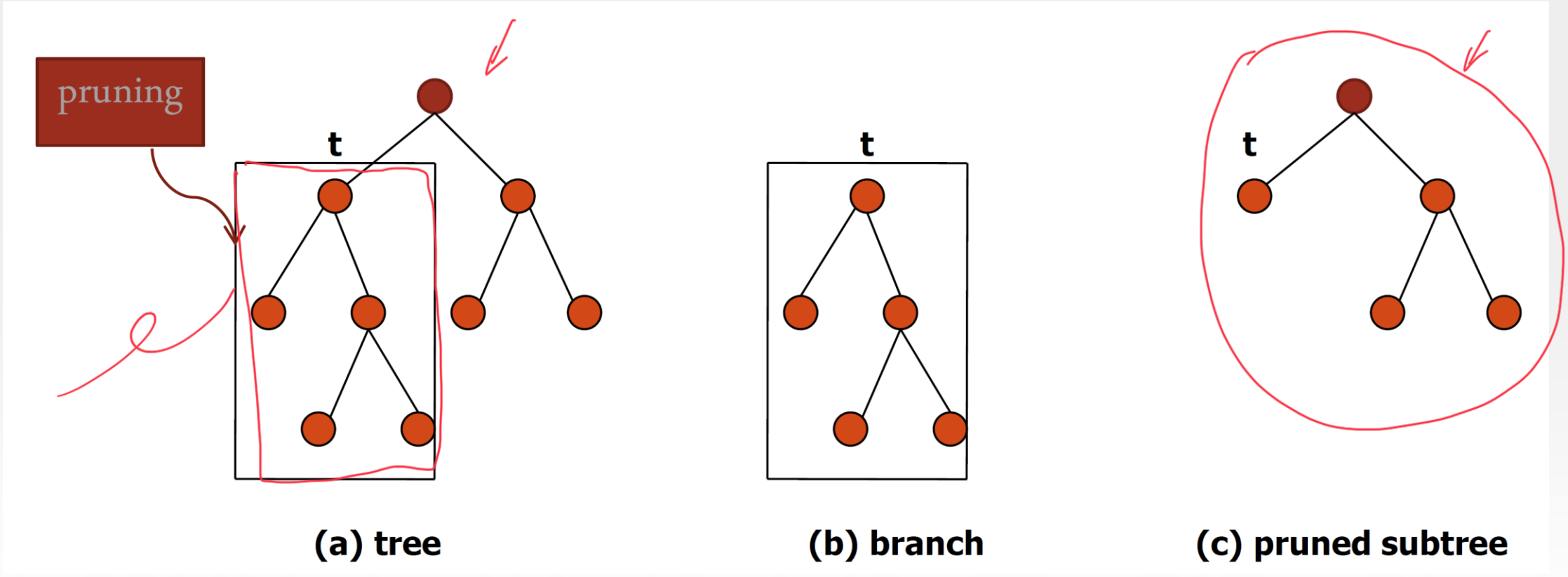


Complete tree without any pruning.

Tree pruning

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias. (Bias - Variance trade off)
- A common strategy to help lower down the number of candidate subtrees is *cost–complexity tuning* by Breiman et al. (1984) .

Tree pruning



Tree pruning

- The goal of this process is to find a "right-sized tree" that has the smallest error rate. To do this, we penalize the error rate using the size of the tree

$$\text{RSS}_{c_p} = \text{RSS} + c_p * (\#\text{Terminal Nodes})$$

where c_p is called the *complexity parameter*. For a specific value of c_p , we find the smallest pruned tree that has the lowest penalized error rate.

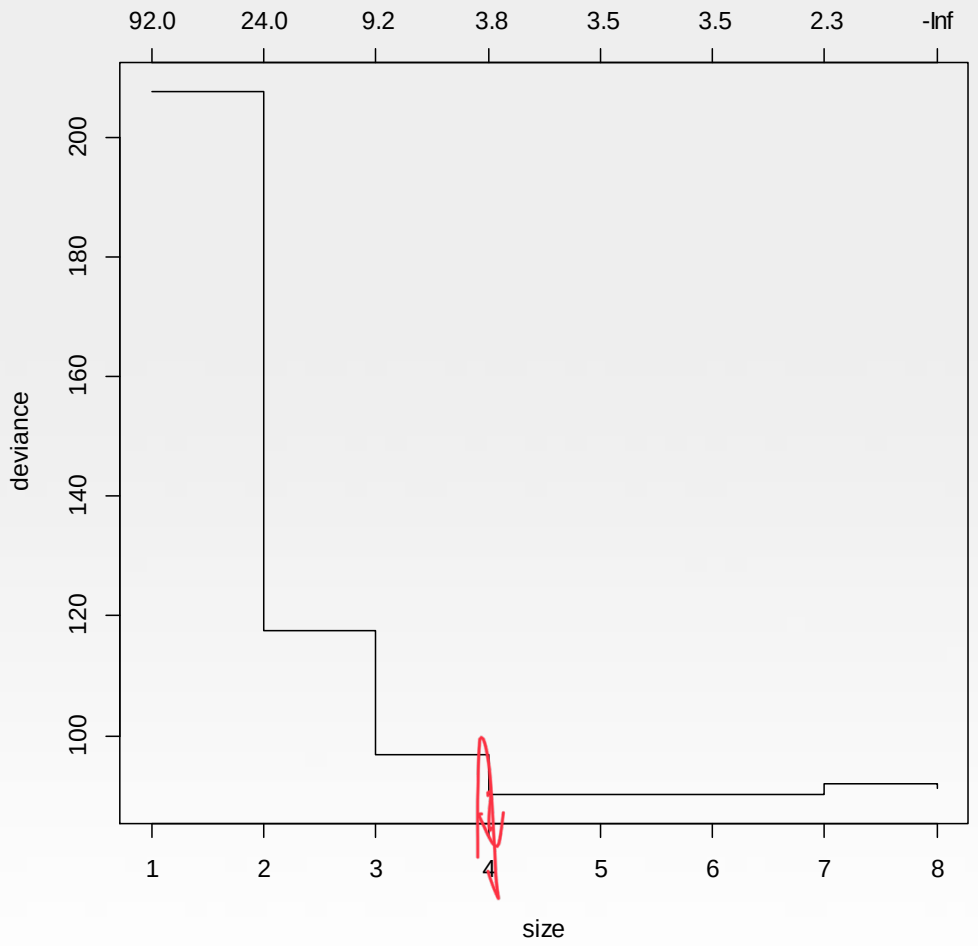
- The model can be tuned by choosing the value of the complexity parameter c_p associated with the smallest possible RMSE value.

Cross-validation and pruning

- The tree package contains function cv.tree for pruning trees by cross-validation.
- The function cv.tree does *k*-fold cross-validation (default is 10)

Prune the tree using cv.tree

- The optimal prune size is 4.



R code

```
sal.tree = tree(log(Salary) ~ Years + Hits, data = Hitters1)
summary(sal.tree)
sal.tree
plot(sal.tree)
text(sal.tree, pretty=0)
title("Baseball Player Salary Data")

#Pruning a tree by cv
set.seed(1)
sal.tree0 = tree(log(Salary) ~ Years + Hits, data = Hitters1)
my.tree.seq = cv.tree(sal.tree0)
plot(my.tree.seq)
opt.trees = which(my.tree.seq$dev == min(my.tree.seq$dev))
# Positions of
# optimal (with respect to error) trees
  min(my.tree.seq$size[opt.trees])
```

The rpart function in R

- CART (Classification and Regression Trees) is developed by Breiman, Friedman, Olshen and Stone
  - CART is the trademarked name of a particular software implementation of these ideas
  - tree() has been used for R routines
- Hence, Recursive PARTitioning (rpart) was chosen
  - rpart has now become more common than the original and more descriptive "cart"
- An introduction of rpart() can be found [here]
- We will look at the solubility data using rpart()

Advantages and disadvantages of trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- Unfortunately, trees generally do not have the same level of predictive accuracy as other regression approaches seen in this book.
- However, by aggregating many decision trees, using methods like *bagging*, *random forests*, *boosting*, and *cubist*, the predictive performance of trees can be substantially improved.