

# Lecture 14: Decision Trees

BIOS635

02/25/2020

# 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

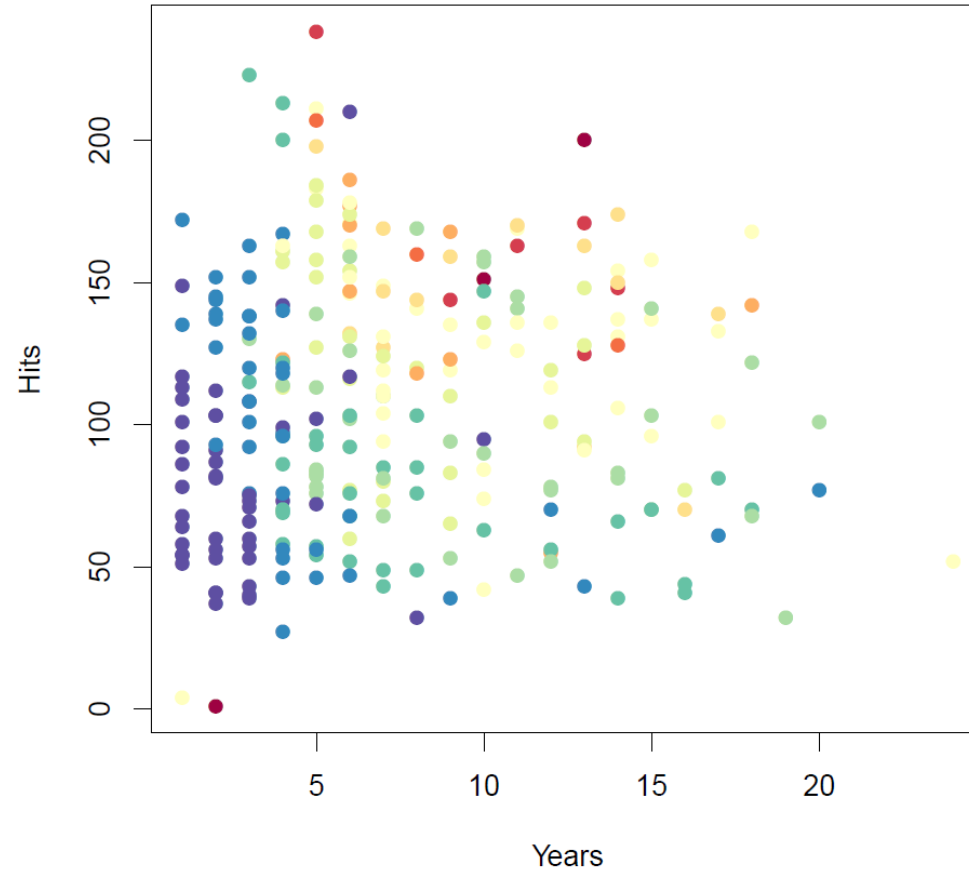
- Tree-based methods are simple and useful for interpretation.
- However they typically are not competitive with the best supervised learning approaches in terms of prediction accuracy.
- Hence we also discuss *bagging*, *random forests*, and *boosting*. These methods grow multiple trees which are then 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

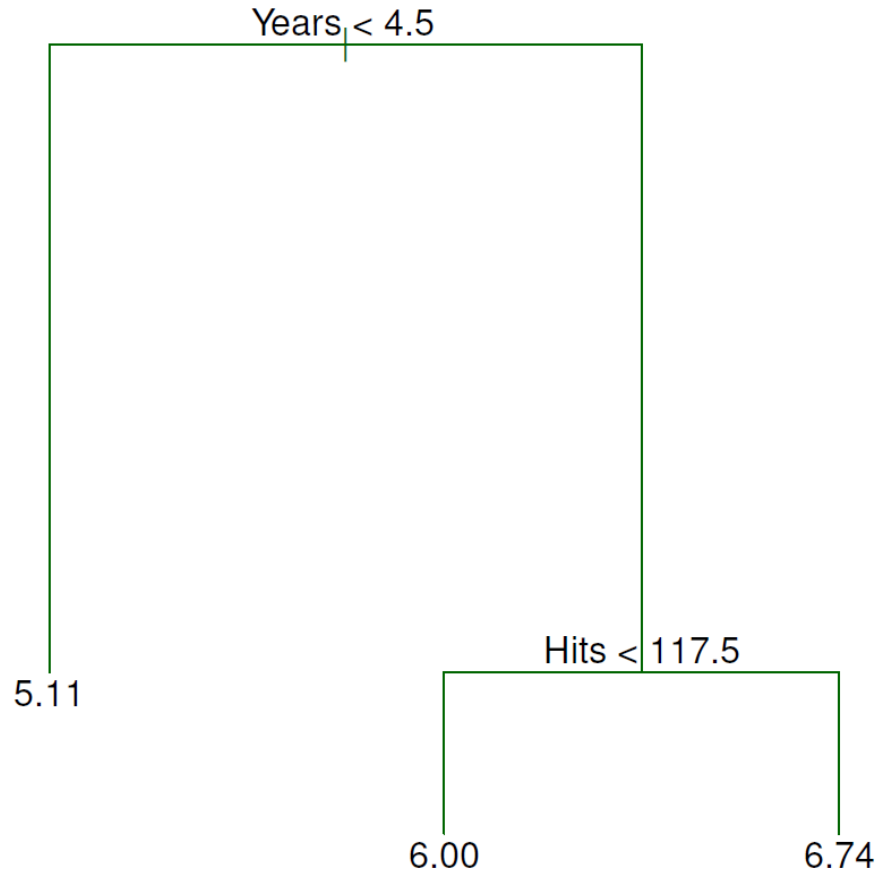
- Decision trees can be applied to both regression and classification problems.
- We first consider regression problems, and then move on to classification.

# Baseball salary data: how would you stratify it?

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



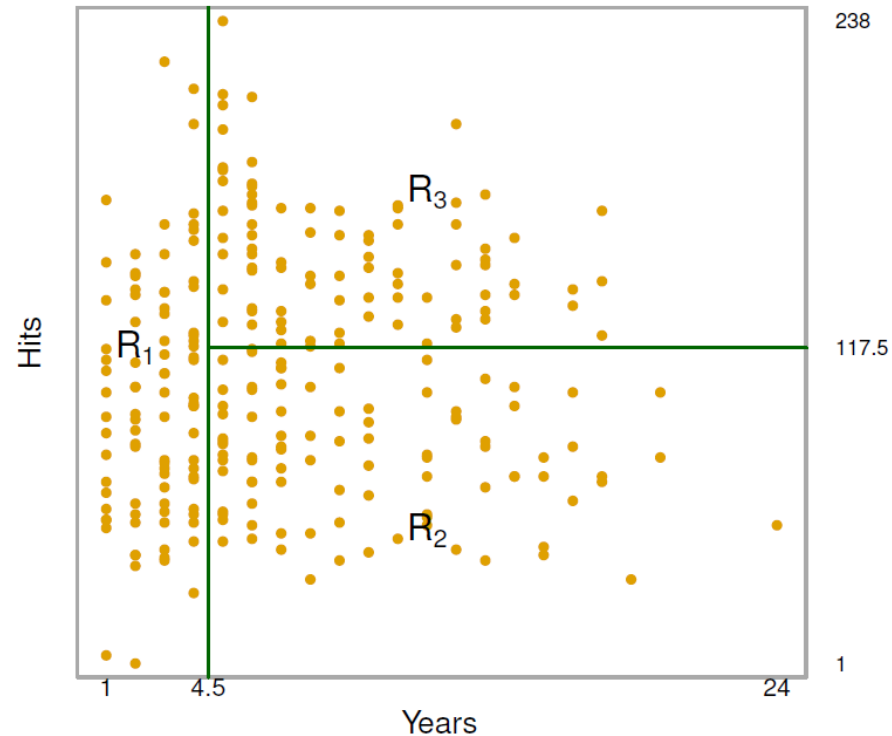
# Decision tree for these data



- For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.
- At a given internal node, the label (of the form  $X_j < t_k$ ) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to  $X_j \geq t_k$ . For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to **Years**<4.5, and the right-hand branch corresponds to **Years**>=4.5.
- 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.

# Results

- Overall, the tree stratifies or segments the players into three regions of predictor space:  $R_1 = \{X \mid \text{Years} < 4.5\}$ ,  $R_2 = \{X \mid \text{Years} \geq 4.5, \text{Hits} < 117.5\}$ , and  $R_3 = \{X \mid \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$ .



# Terminology of trees

- In keeping with the *tree* analogy, the regions  $R_1$ ,  $R_2$ , and  $R_3$  are known as *terminal nodes*
- Decision trees are typically drawn *upside down*, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as *internal nodes*
- In the hitters tree, the two internal nodes are indicated by the text **Years**<4.5 and **Hits**<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  $X_1, X_2, \dots, X_p$  — into  $J$  distinct and non-overlapping regions,  $R_1, R_2, \dots, R_J$ .
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  $R_j$ .

# More details of the tree-building process

- 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  $R_1, \dots, R_J$  that minimize the RSS, given by

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

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

# More details of the tree-building process

- Unfortunately, it is computationally infeasible 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.

# More details of the tree-building process

- Unfortunately, it is computationally infeasible 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.
- 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.

# Details, cont'd

- We first select the predictor  $X_j$  and the cutpoint  $s$  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.

# Details, cont'd

- We first select the predictor  $X_j$  and the cutpoint  $s$  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 cutpoint 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.

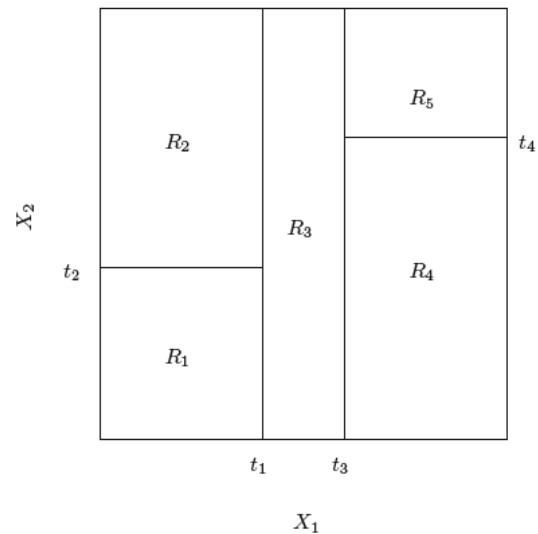
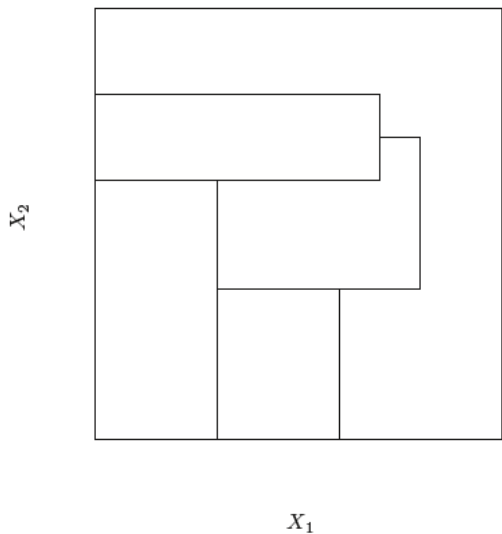
# Details, cont'd

- We first select the predictor  $X_j$  and the cutpoint  $s$  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 cutpoint 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.



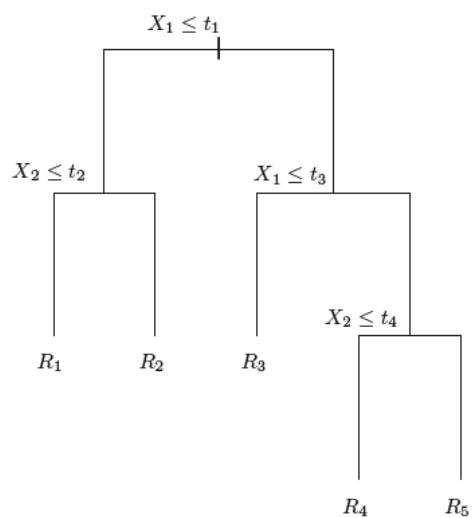
# Predictions

- 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.
- A five-region example of this approach is shown in the next slide.

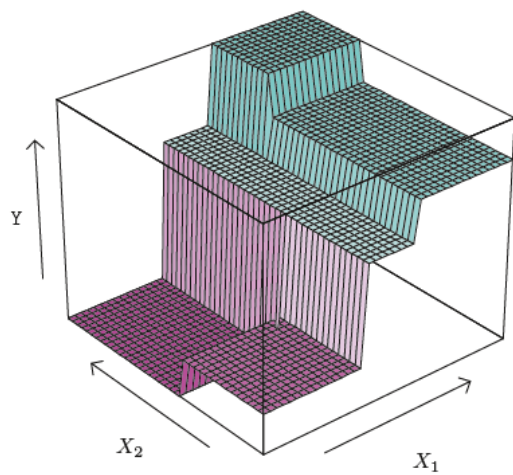


*Top Left:* A partition of two-dimensional feature space that could not result from recursive binary splitting.

*Top Right:* The output of recursive binary splitting on a two-dimensional example.



*Bottom Left:* A tree corresponding to the partition in the top right panel.



*Bottom Right:* A perspective plot of the prediction surface corresponding to that tree.

# Pruning a tree

- 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.

# Pruning a tree

- 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  $R_1, \dots, R_J$ ) might lead to lower variance and better interpretation at the cost of a little bias.

# Pruning a tree

- 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  $R_1, \dots, R_J$ ) 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.

# Pruning a tree

- 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  $R_1, \dots, R_J$ ) 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 — that is, a split that leads to a large reduction in RSS later on.

# Pruning a tree, cont'd

- 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 nonnegative tuning parameter  $\alpha$ . For each value of  $\alpha$  there corresponds a subtree  $T \subset T_0$  such that

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

is as small as possible. Here  $|T|$  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, and  $\hat{y}_{R_m}$  is the mean of the training observations in  $R_m$ .

# Choosing the best subtree

- The tuning parameter  $\alpha$  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  $\alpha$ .

# 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  $\alpha$ .
3. Use K-fold cross-validation to choose  $\alpha$ . For each  $k = 1, \dots, K$ :
  - 3.1 Repeat Steps 1 and 2 on the  $\frac{K-1}{K}$ th fraction of the training data, excluding the  $k$ th fold.
  - 3.2 Evaluate the mean squared prediction error on the data in the left-out  $k$ th fold, as a function of  $\alpha$ .

Average the results, and pick  $\alpha$  to minimize the average error.

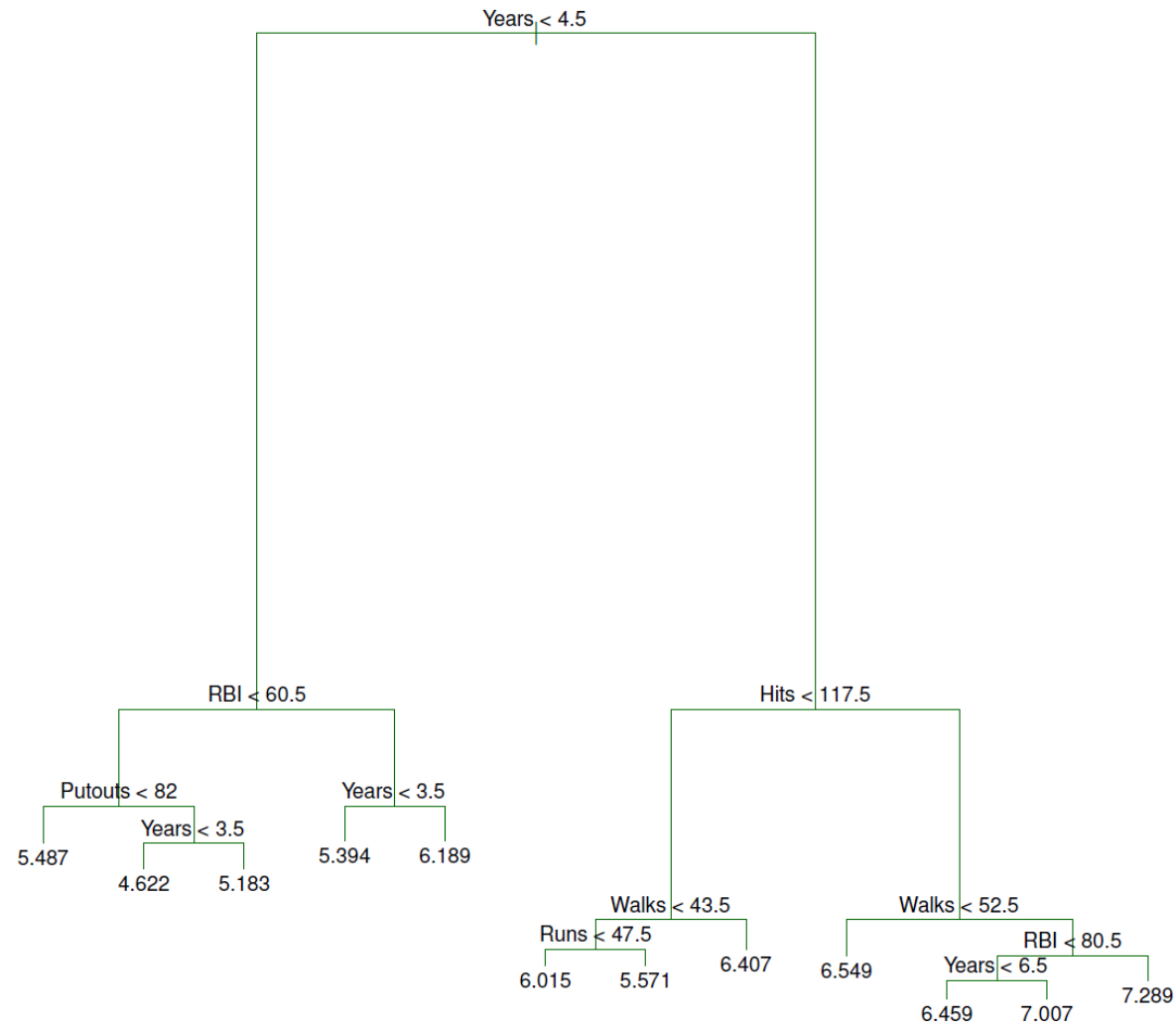
# Summary: tree algorithm

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  $\alpha$ .
3. Use K-fold cross-validation to choose  $\alpha$ . For each  $k = 1, \dots, K$ :
  - 3.1 Repeat Steps 1 and 2 on the  $\frac{K-1}{K}$ th fraction of the training data, excluding the  $k$ th fold.
  - 3.2 Evaluate the mean squared prediction error on the data in the left-out  $k$ th fold, as a function of  $\alpha$ .Average the results, and pick  $\alpha$  to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of  $\alpha$ .

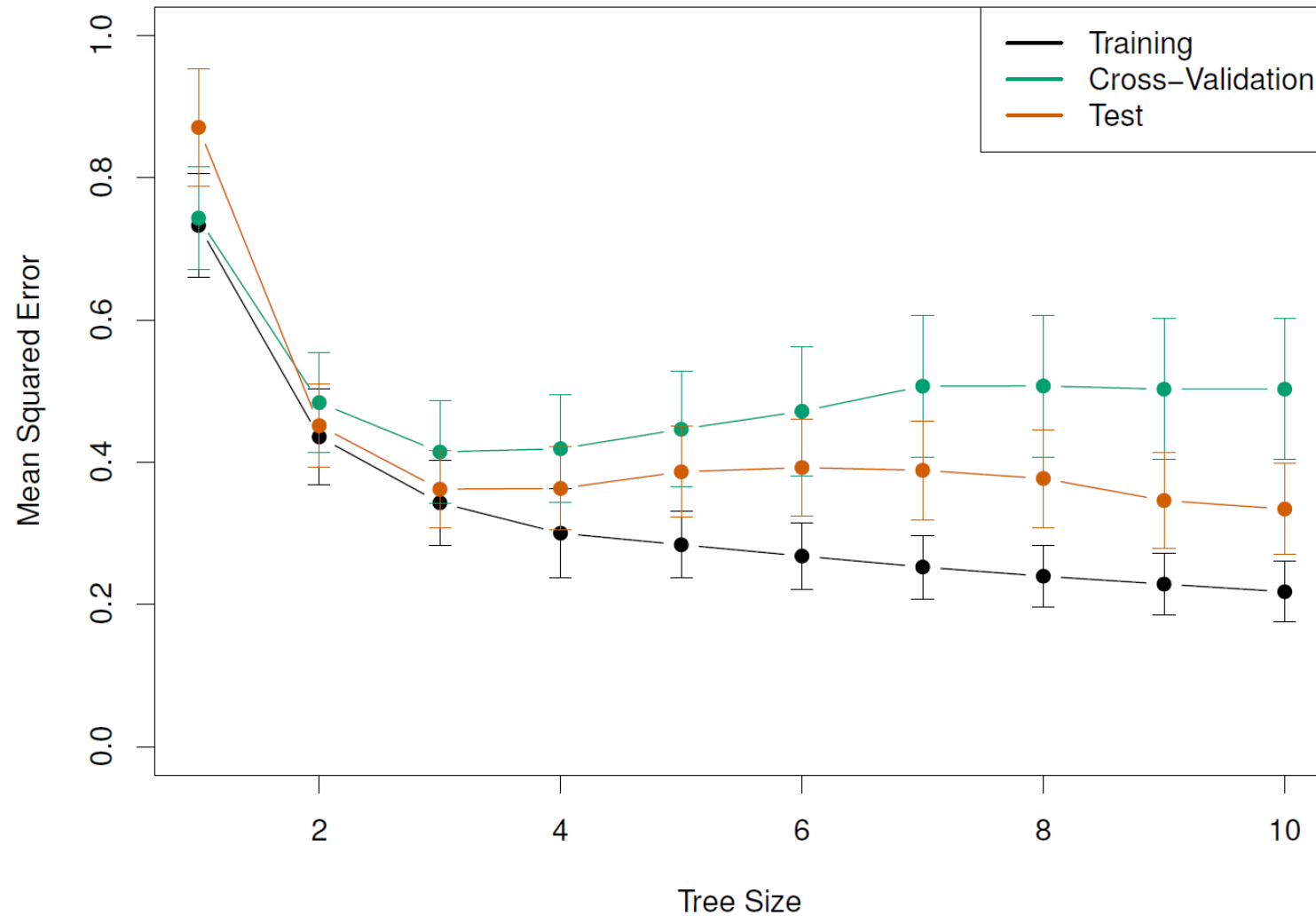
# Baseball example cont'd

- First, we randomly divided the data set in half, yielding 132 observations in the training set and 131 observations in the test set.
- We then built a large regression tree on the training data and varied  $\alpha$  in order to create subtrees with different numbers of terminal nodes.
- Finally, we performed six-fold cross-validation in order to estimate the cross-validated MSE of the trees as a function of  $\alpha$ .

# Baseball example cont'd



# Baseball example cont'd



# 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 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



# 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.

# 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 as a measure of node purity

- 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.
- In **prediction**, you choose splits to minimize the RSS
- In **classification**, choose splits to maximize **node purity** (minimize **Gini index**)

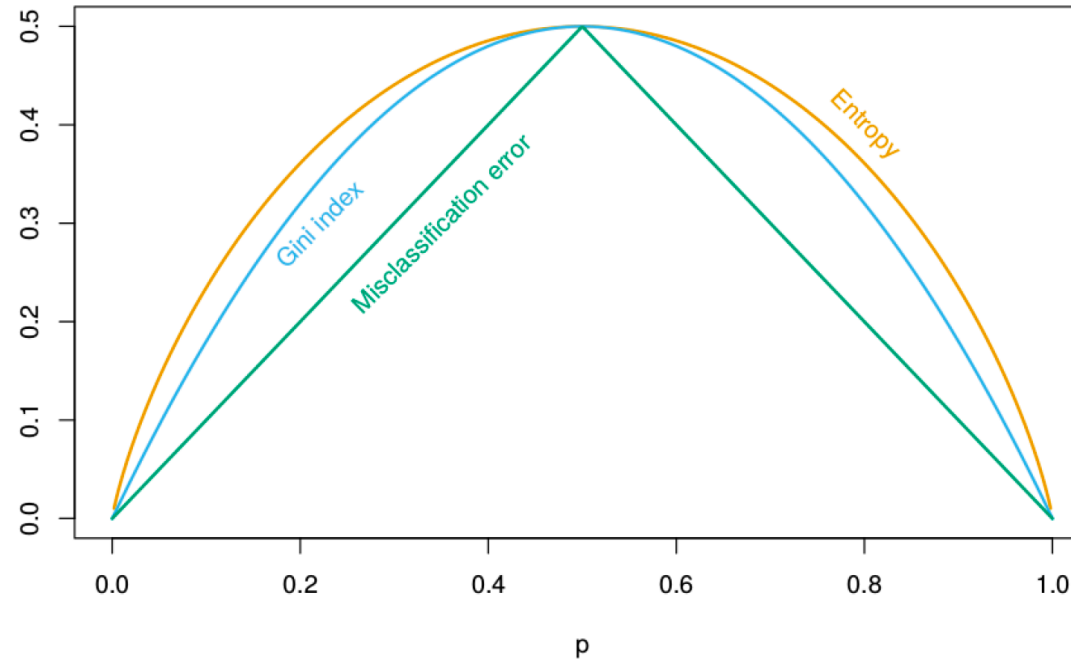
# Entropy as a measure of node purity

- 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.

# Why not minimize the classification error?

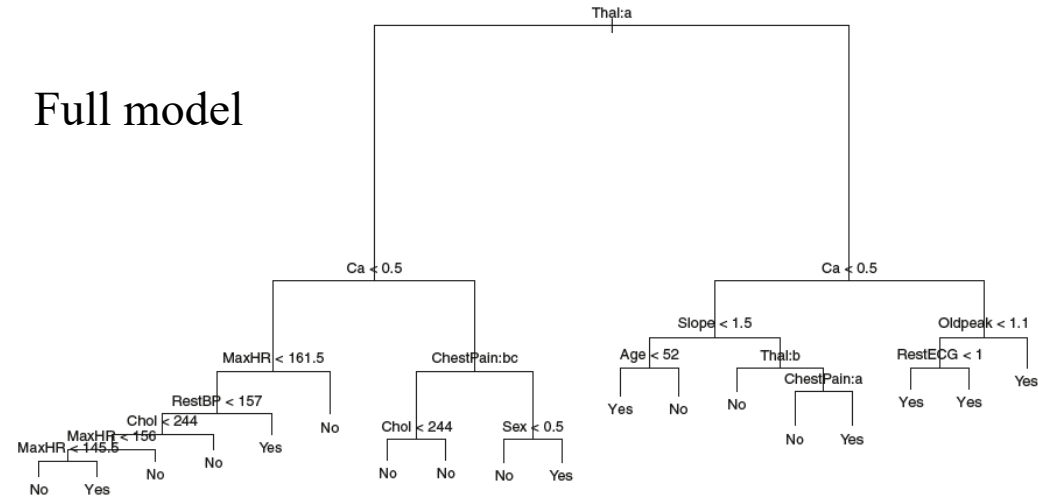


- Misclassification rate is poor at pushing for really **pure nodes**
- With **Gini**: going from  $\hat{p}_{mk} = 0.8$  to  $\hat{p}_{mk} = 0.9$  is better than going from  $\hat{p}_{mk} = 0.5$  to  $\hat{p}_{mk} = 0.6$
- With **Misclassification error**, these are considered equal improvements

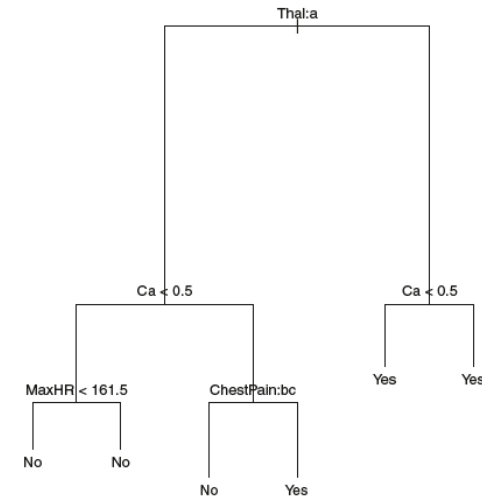
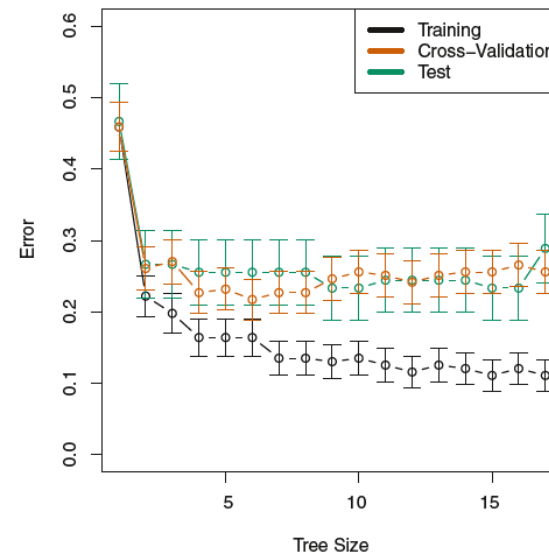
# Example: heart data

- 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.

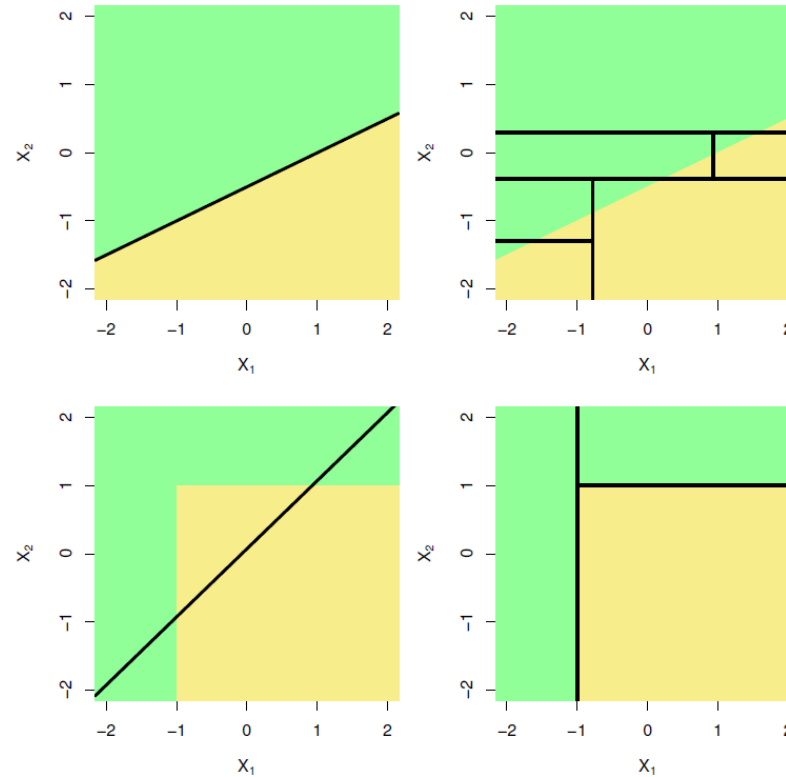
## Full model



Cross validation yields a tree with six terminal nodes.



# Trees v.s. linear models



Top Row: True linear boundary; Bottom row: true non-linear boundary.

Left column: linear model; Right column: tree-based model



# Good things about trees

- Trees are the most easily **interpretable** method we've talked about in this class
  - You can explain a decision tree to even your least technical colleague
- Arguably, trees more closely *mirror human decision-making*
- Trees are easy to **display graphically**.
  - You can print off your tree and easily obtain predictions by hand
- Trees can handle **qualitative predictors** and **ordered qualitative predictors** without needing to create dummy variables
- Trees handle **missing values** very nicely, without throwing out observations
  - When a value is missing, they split on a **surrogate variable**: E.g., if a user's *years of job experience* is missing, they'll split on an optimally chosen correlated variable like *age*.

# Why are decision trees poor predictors?

- Decision trees tend to have high variance. A small change in the training data can produce big changes in the estimated Tree.

