TREE-BASED METHODS

(Part 1)
Regression and Classification Trees

盧信銘 台大資管系

Introduction

- Tree-based methods can be used to perform
  - Regression
  - Classification
- The basic idea of tree-based method is to segment or stratify space into simple regions.
- The rules to split can be summarized efficiently via a tree structure
  - Often-referred to as the decision-tree methods
  - Usually partitions feature space recursively
- Cf. Chapter 8 of “An Introduction to Statistical Learning with Applications in R.”

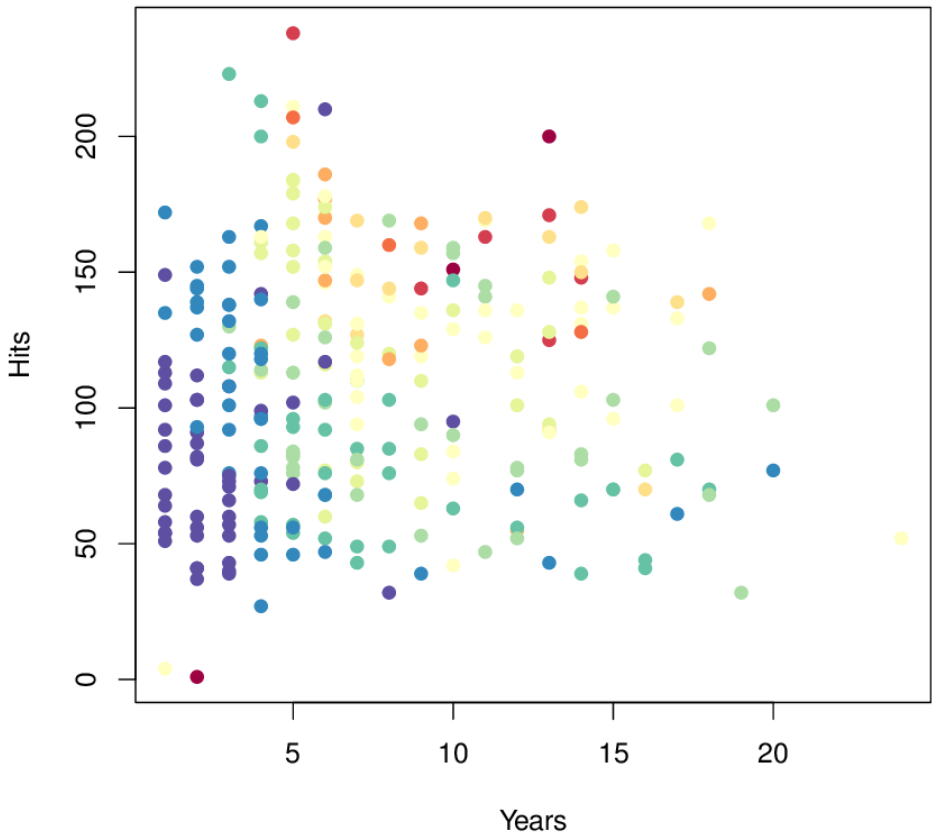
Pros and Cons

- Tree-based methods are simple to interpret
- However, its performance is often not as good as other approaches
- To improve performance, we can adopt
  - Bagging
  - Random forests
  - Boosting
- Improve performance by combining prediction from a large number of trees
  - At the expense of interpretability

Baseball Salary Data: How would you stratify it?

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

Hitter Data



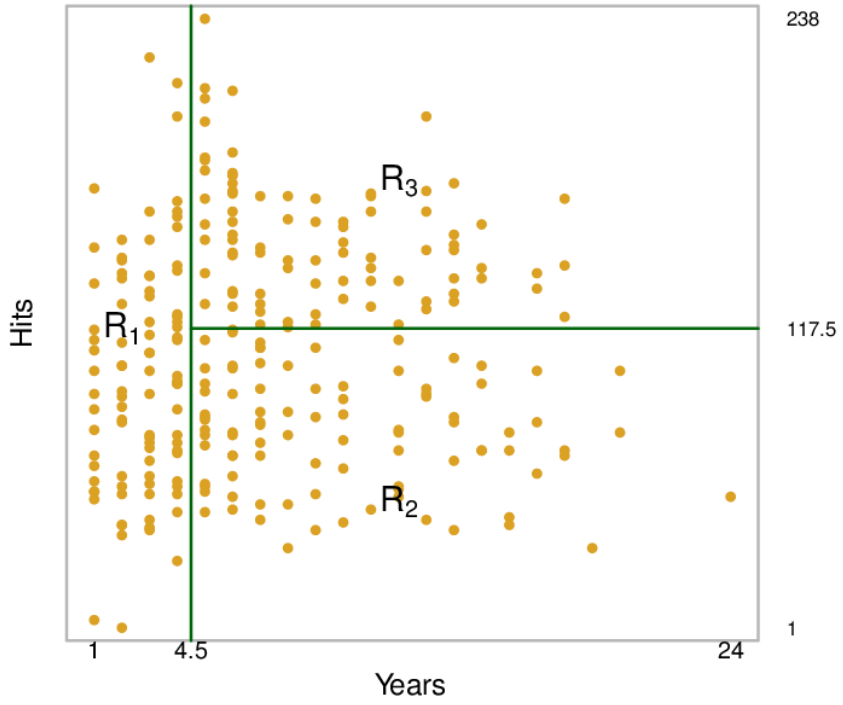
Decision Tree for the data

Details of previous figures

- 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: $R_1 = \{X \mid \texttt{Years} < 4.5\}$, $R_2 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} < 117.5\}$, and $R_3 = \{X \mid \texttt{Years} >= 4.5, \texttt{Hits} >= 117.5\}$.



Terminology for 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 $X_1, X_2, \ldots, X_p$ — 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.

More Details of Tree-Building

- 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 $R_1, \ldots, R_J$ 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 jth box.

More Details of Tree-Building

- 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.

More Details of Tree-Building

- 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.
- Consider splitting variable *j* at point s,
$$R_1(j,s) = \{X|X_j \leq s\}, \qquad R_2(j,s) = \{X|X_j > s\}.$$
- We then solve c_1, c_2 by:
$$\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2$$
- Clearly $c_1 = average(y_i|x_i \in R_1(j,s))$,
$c_2 = average(y_i|x_i \in R_2(j,s))$
- Fix *j*, search over possible s for the best s (the midpoint between two consecutive X_j)

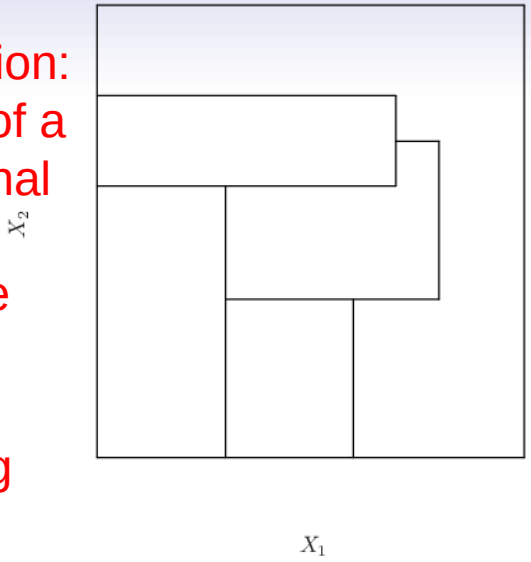
More Details of Tree-Building

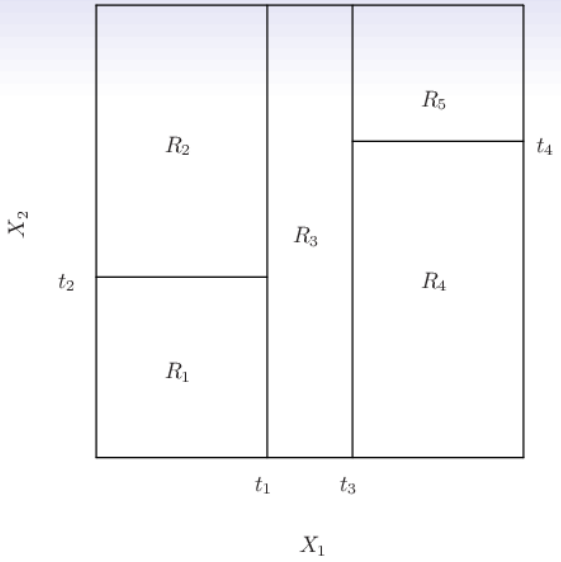
- Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. 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.

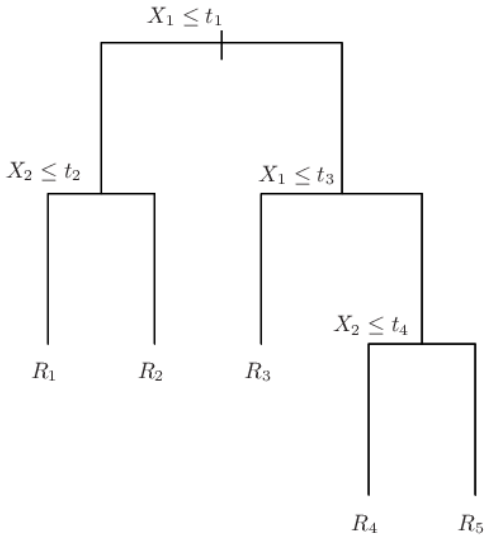
Data Generation: The partition of a two-dimensional feature space that cannot be generated by recursive binary splitting

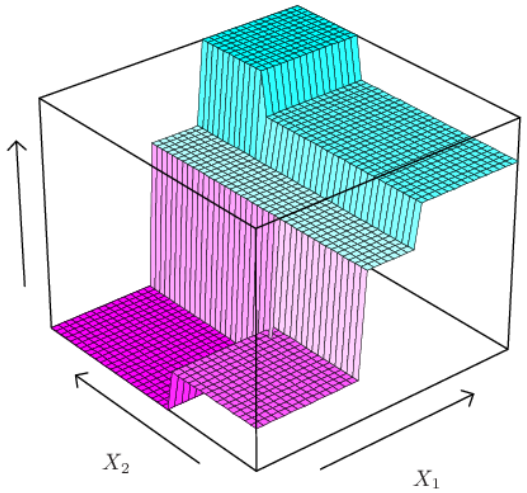




The output of recursive binary splitting

A tree corresponding to the top-right partition





The partition and response surface of the recursive binary splitting

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. *Why?*
- 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.
- 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 α. For each value of α 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 mth terminal node, and $\hat{y}_{R_m}$ is the mean of the training observations in R_m.

Weakest Link Pruning

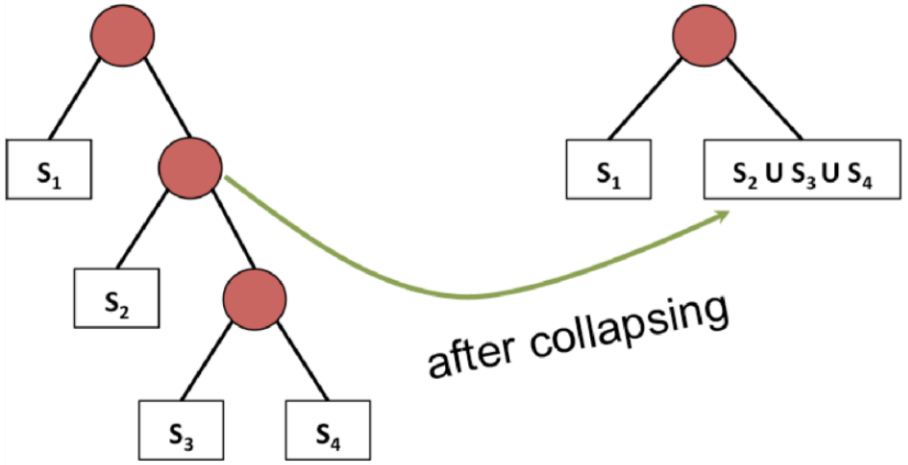
- Terminology
  - T is a tree; t is a node in a tree
  - $|T|$ is the number of leaf nodes in tree T
  - T_t is the sub-tree rooted at node t (inclusive); $T_{root} = T$
  - $R(T)$ is a measure of the error on the training subset hold by leaf nodes in T

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

  - $R(T)$ can be replaced with entropy or misclassification rate in classification tree

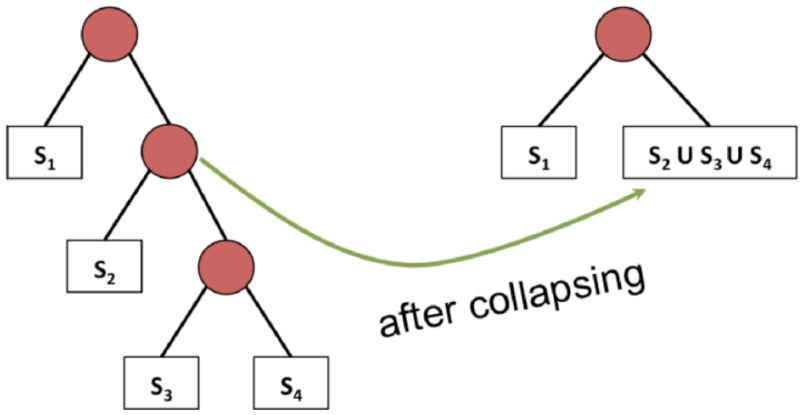
Collapsing a Node

- Let T_{full} be the decision tree from the learning algorithm
  - Leaves are rectangles of feature space (or subsets of training data for classification tree)
- To make it smaller, we will be collapsing internal nodes of the tree
  - In the picture: s_1, s_2, s_3, s_4 are rectangles of feature space



Collapsing a node (Cont'd.)

- By the nature of the algorithm, collapsing a node t always increases the error that point downward
  - Use $R(t)$ to denote the error on a tree with a single node
    - This node is obtained as a result of collapsing the subtree
  - $R(t) > R(T_t)$
    - R on $s_2 \cup s_3 \cup s_4$ is more than the sum of R on s_2, s_3, and s_4 individually.



Cost Complexity Function

- Let $\alpha \geq 0$ specifies the cost of having one leaf node in a tree
- Define the cost-complexity measure for a tree T as

$$C_\alpha(T) = R(T) + \alpha|T|$$

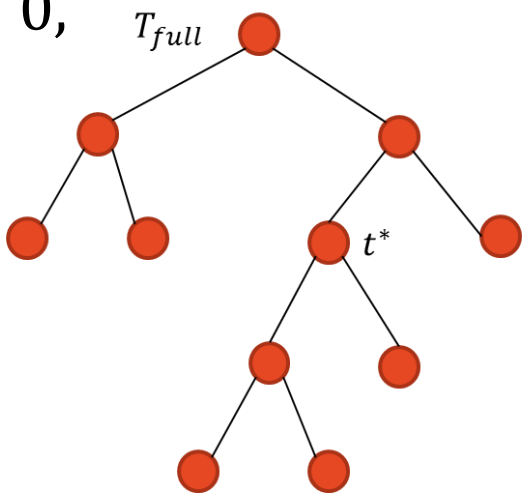
- For T_{full}, cost complexity is $R(T_{full}) + \alpha|T_{full}|$
  - If $\alpha = 0$, cost complexity will not collapse nodes in T_{full}
  - If $\alpha > 0$, $\alpha|T_{full}|$ is the complexity cost for the tree; proportional to the number of leaf nodes.
    - Remove some nodes will increase $R(T)$ but reduces $\alpha|T|$ ➔ may result in a smaller tree

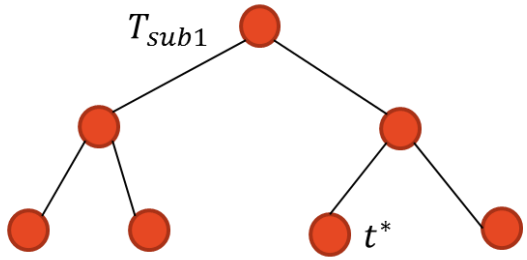
Weakest Link Pruning

- Minimal cost-complexity tree
  - Given some $\alpha \geq 0$, find a subtree T^* of T_{full} such that
$$T^* = \underset{T \text{ is subtree of } T_{full}}{\operatorname{argmin}} C_\alpha(T)$$
- To achieve minimal cost-complexity tree, we sequentially selected internal nodes to collapse the tree.
  - Starting from T_{full}
  - Choose the node such that (平均每個葉節點對誤差貢獻最少)
$$t^* = \underset{t \text{ is internal node of } T_{full}}{\operatorname{argmin}} \frac{R(t) - R(T_t)}{|T_t| - 1}$$
  - The tree so generated is called T_{sub1}
  - This node is the “weakest link” of T_{full}
  - Repeat the process on T_{sub1} until $\frac{R(t^*) - R(T_{t^*})}{|T_{t^*}| - 1} > \alpha$.

Weakest Link Pruning (Cont'd.)

- Which is higher? $C_\alpha(T_{full})$ or $C_\alpha(T_{sub1})$?
- $C_\alpha(T_{sub1}) = R(T_{sub1}) + \alpha|T_{sub1}|$
  $= \mathrm{R}(\mathrm{T_{full}}) - \mathrm{R}(\mathrm{T_{t^*}}) + \mathrm{R}(\mathrm{t^*}) + \alpha\left(|T_{full}| - |T_{t^*}| + 1\right)$
  $= \left(R(T_{full}) + \alpha|T_{full}|\right) + \left(\mathrm{R}(\mathrm{t^*}) - \mathrm{R}(\mathrm{T_{t^*}})\right) - \alpha(|T_{t^*}| - 1)$
  $= C_\alpha(T_{full}) + \left(\mathrm{R}(\mathrm{t^*}) - \mathrm{R}(\mathrm{T_{t^*}})\right) - \alpha(|T_{t^*}| - 1)$
- That is,
  $C_\alpha(T_{sub1}) - C_\alpha(T_{full}) = \left(\mathrm{R}(\mathrm{t^*}) - \mathrm{R}(\mathrm{T_{t^*}})\right) - \alpha(|T_{t^*}| - 1)$
- Since $\mathrm{R}(\mathrm{t^*}) - \mathrm{R}(\mathrm{T_{t^*}}) > 0$,
- $C_\alpha(T_{sub1}) < C_\alpha(T_{full})$
  if $\frac{R(t^*) - R(T_{t^*})}{|T_{t^*}| - 1} < \alpha$





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 $k = 1, \ldots, K$:
   - 3.1 Repeat Steps 1 and 2 on the $\frac{K-1}{K}$th fraction of the training data, excluding the kth fold.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth 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.

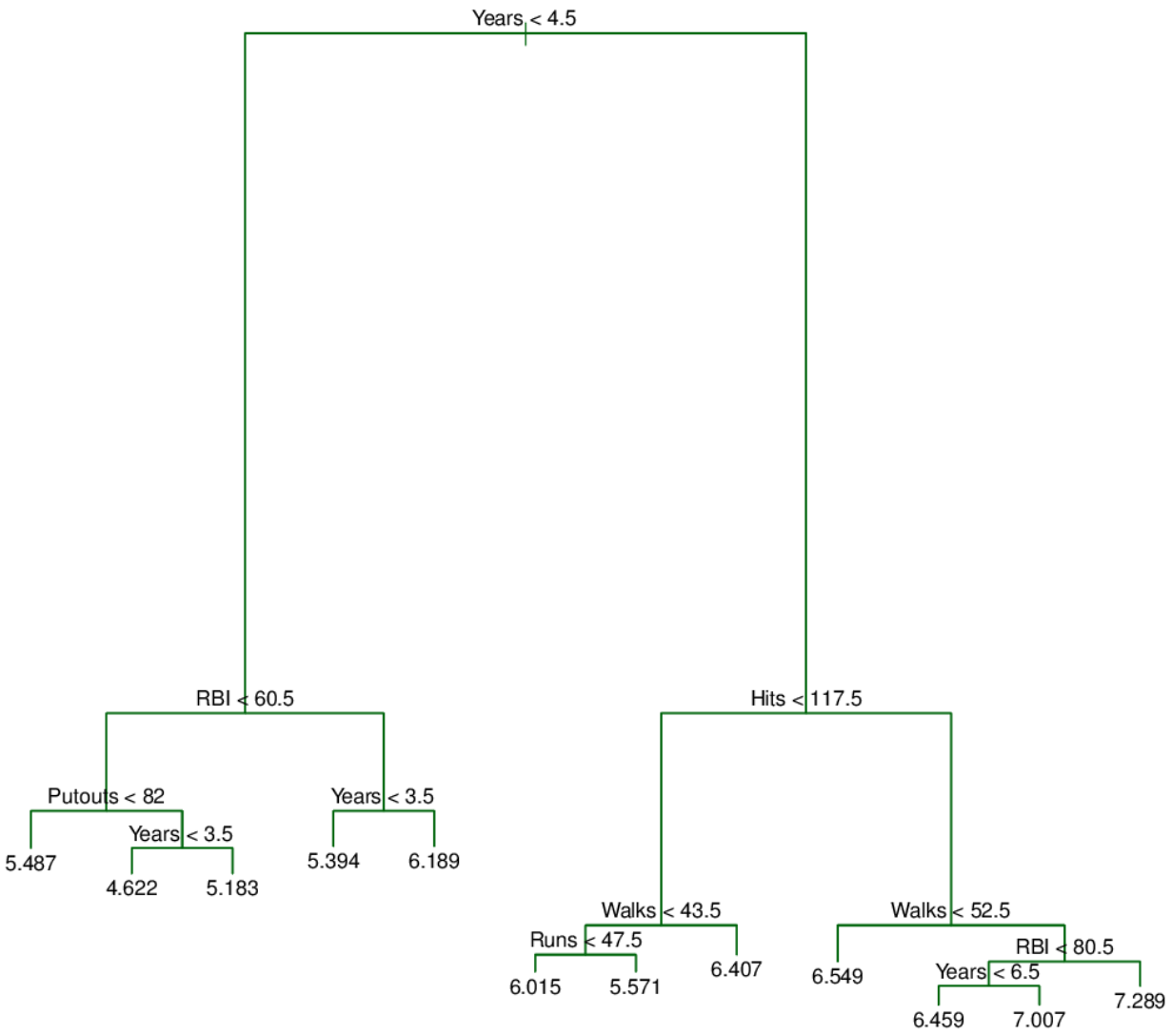
Baseball Example Continued

- 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 α in 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 α.

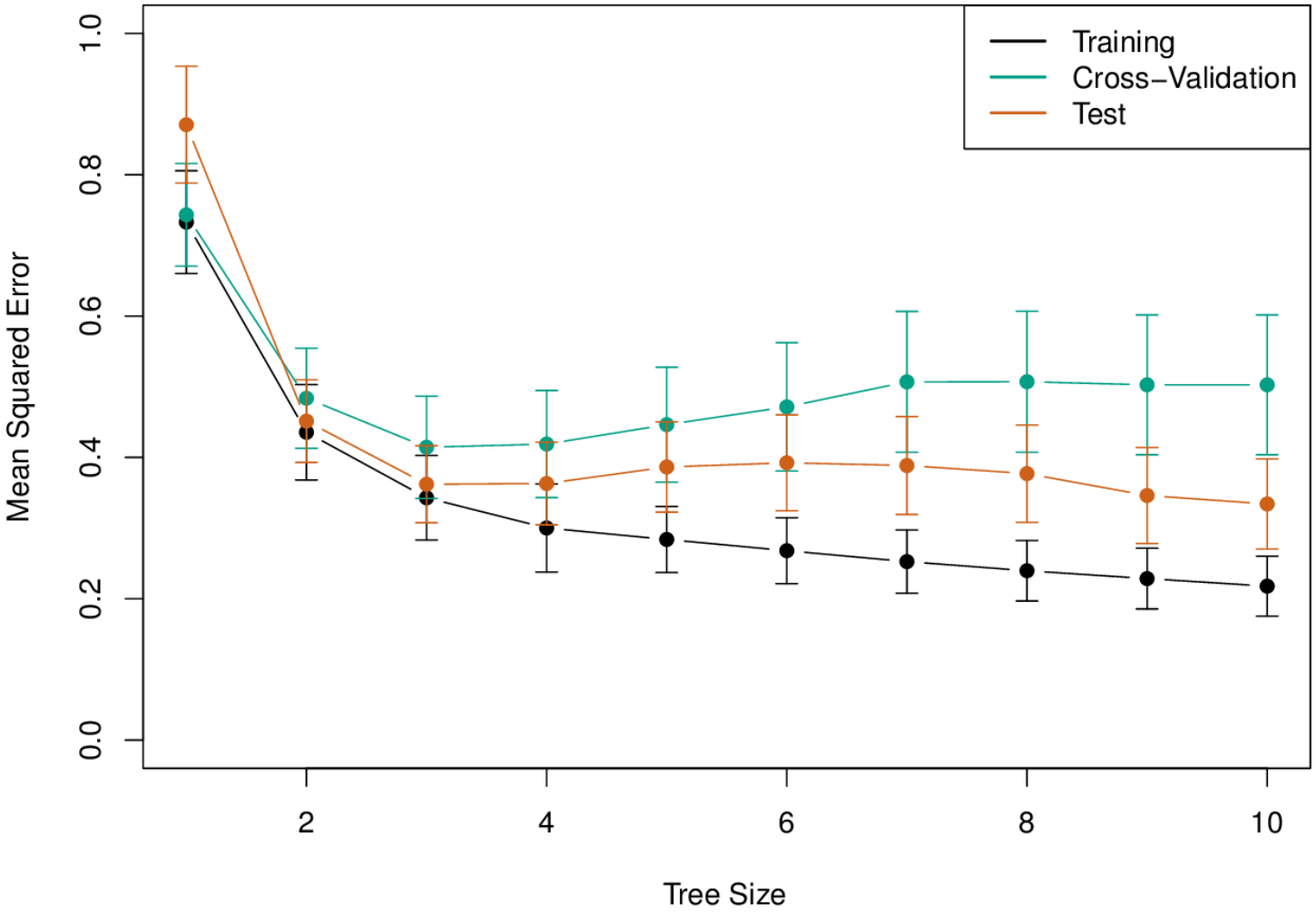
Baseball Example Continued

- Randomly split the data into a training (132 data points) and test (131 data points) sets.
- Create a unpruned tree using training data (see next slide).
- Further split the training data into six fold, and conduct cross-validation to compute the cross-validation errors using different α (which maps to different tree size).
- For each α, train the model using the whole training data and test on test data to obtain test error.
- Note: We do this so that we can have test error on different test size. In most cases, you can pick a tree size from cross validation results, and just compute the test error using this particular setting.

Baseball Example (Unpruned Tree)



Baseball Example (Cont'd.)



CLASSIFICATION TREES

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.

- For features with continuous values, we use recursive binary splitting to grow a tree.
- For features with discrete values, two approaches are possible:
  - Binary split: Pick a value and split the tree according to whether the feature equal to this value (text book use this approach).
  - Or, we split a node according to all unique values of that feature (also common in decision trees).

Details of Classification Trees

- In the classification problem, RSS cannot be used as the criterion to make splits.
- A natural alternative to RSS is the classification error rate.
- Classification error rate is 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.
  - In practice, we use either entropy of gini index.

Gini Index and Entropy Reduction

- 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

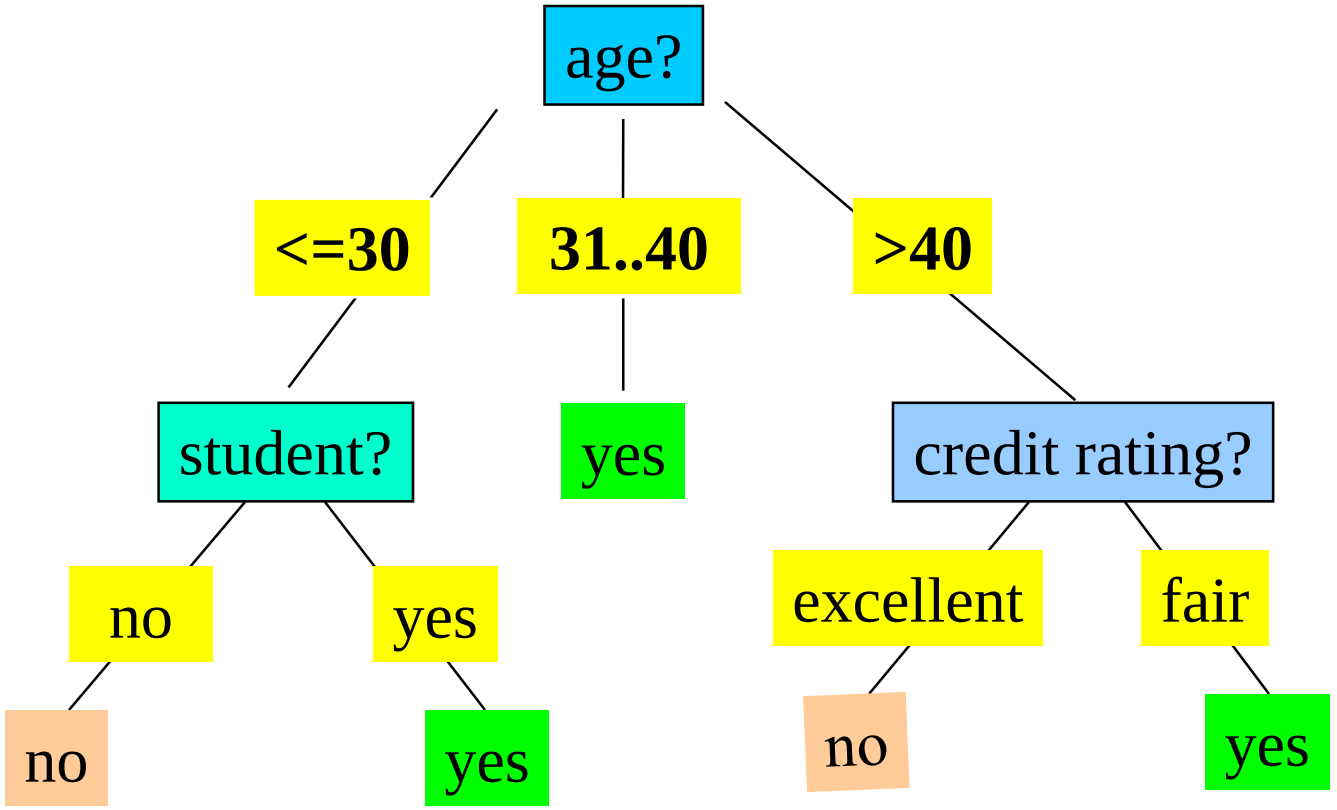
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

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

Decision Tree Induction: Training Dataset

age	income	student	credit_rating	buys_iphone
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for "*buys_iphone*"



Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they can be (1) discretized in advance, or (2) attempt binary partition at every possible value)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no data points left (or data points left is less than the predefined threshold)

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Entropy of D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Conditional Entropy of D given A:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times I(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_iphone = "yes"
- Class N: buys_iphone = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

$\frac{5}{14} I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes's and 3 no's. Hence

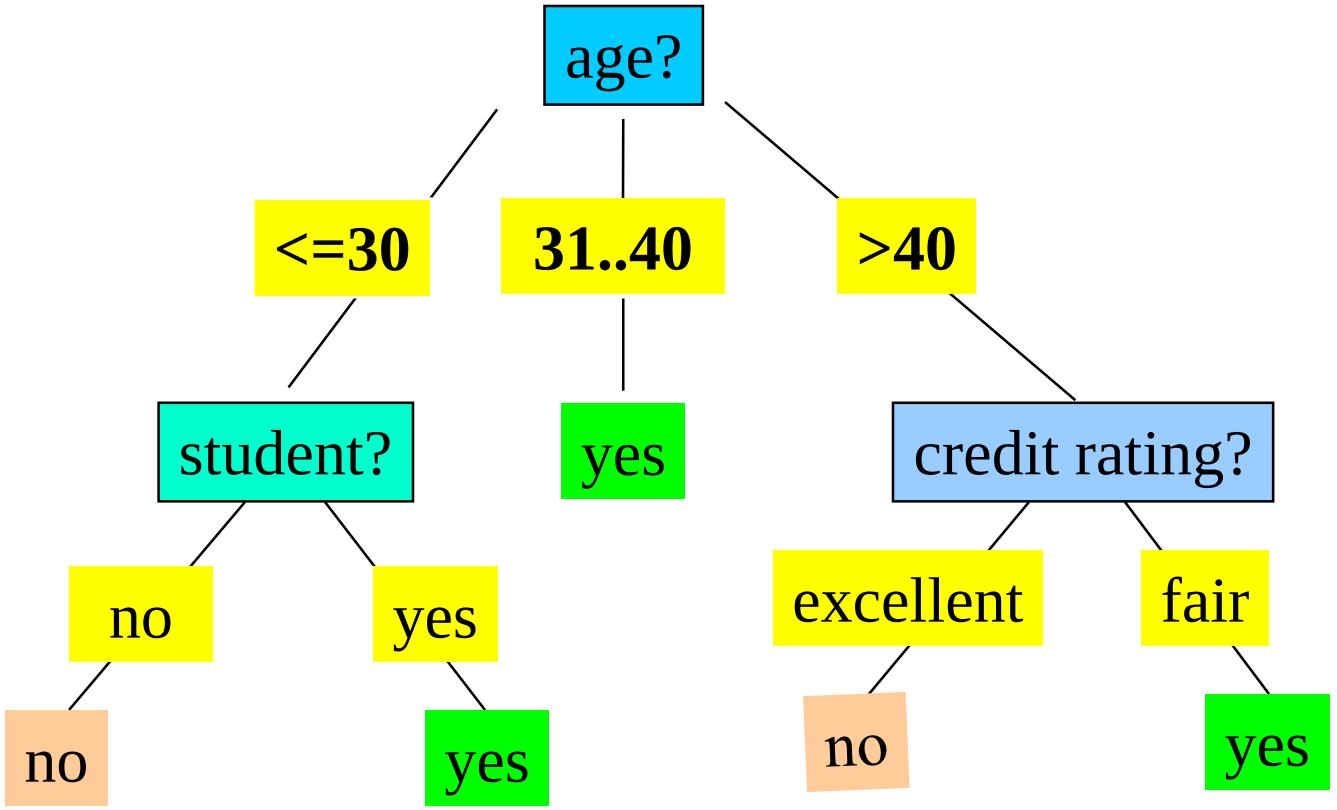
$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

age	income	student	credit_rating	buys_iphone
<=30	high	no	fair	no
<=30	high	no	excellent	no
31..40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31..40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31..40	medium	no	excellent	yes
31..40	high	yes	fair	yes
>40	medium	no	excellent	no

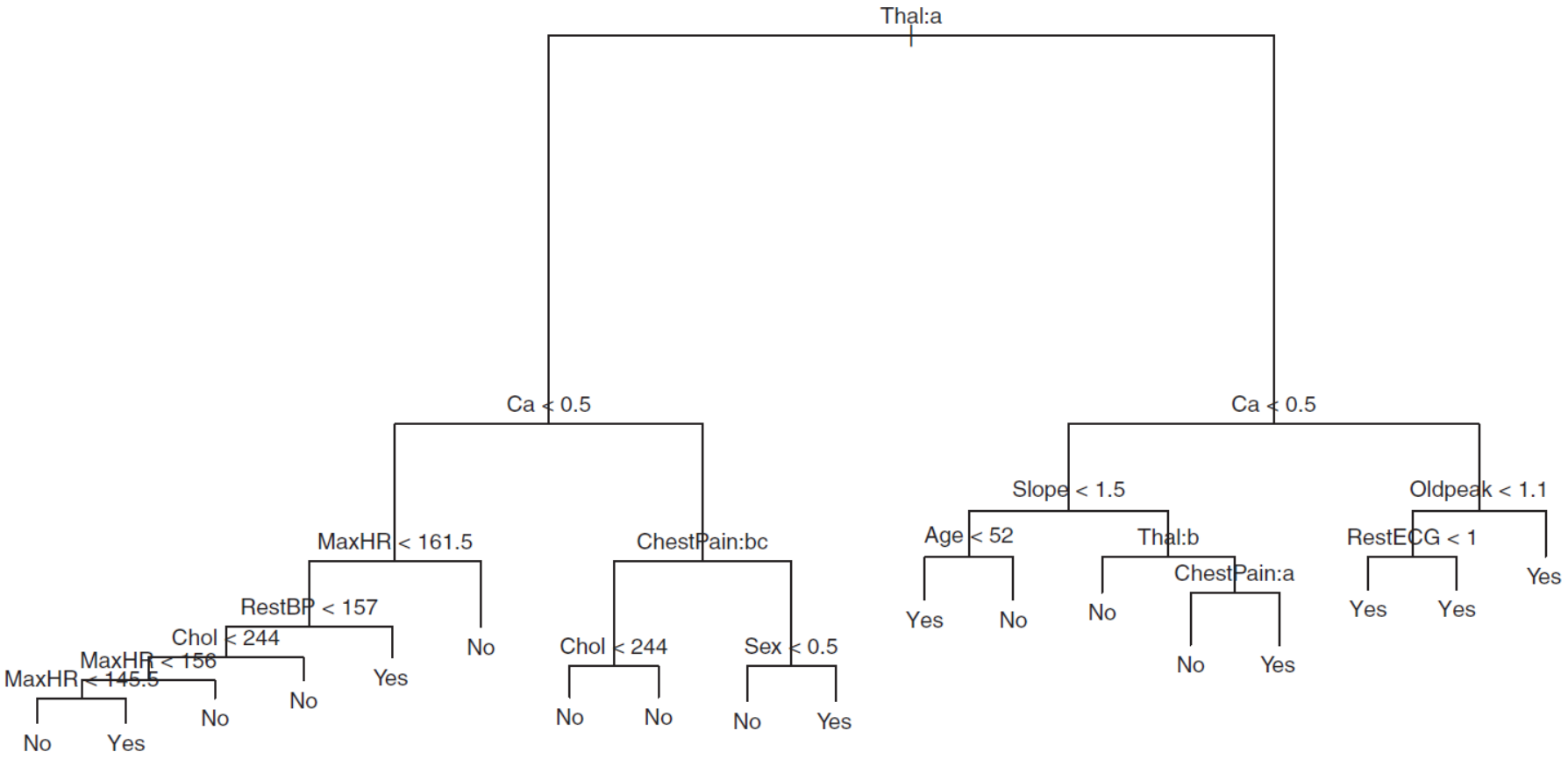
Output: A Decision Tree for "*buys_iphone*"



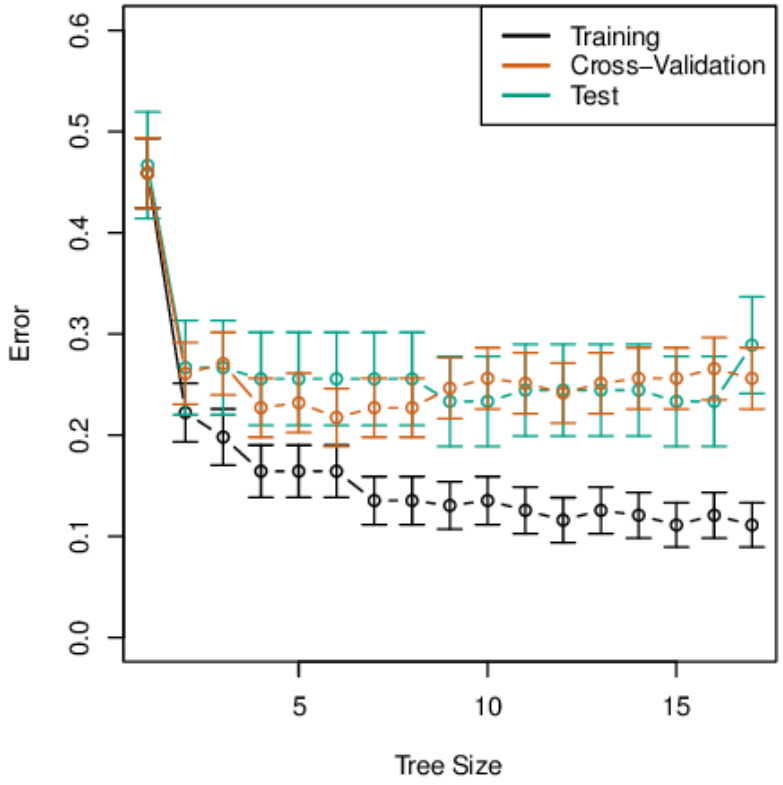
Example: Heart Data

- 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. See next figure.

Classification Tree (Full)

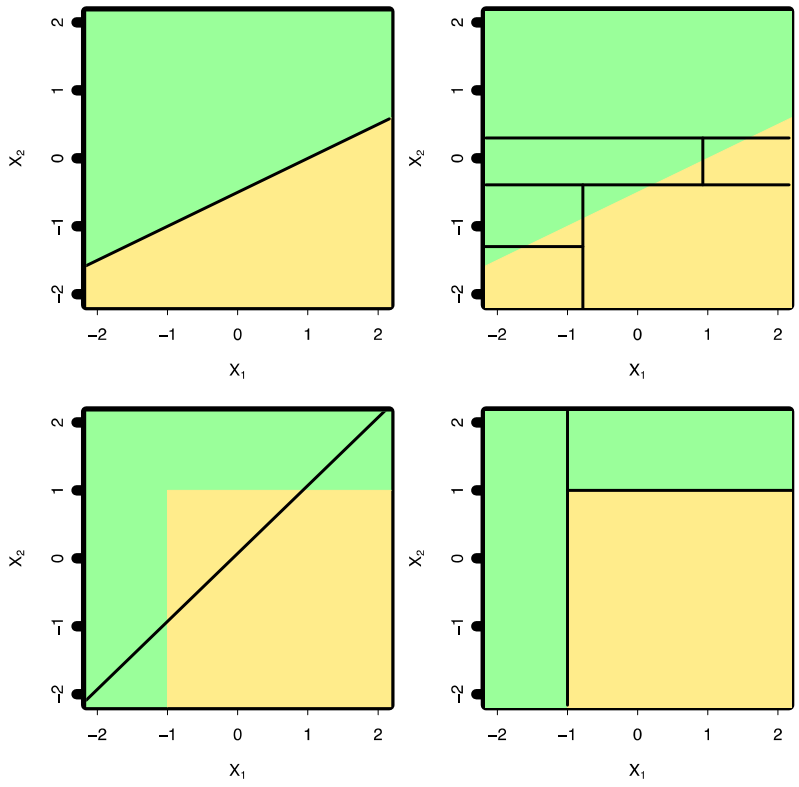


Classification Tree (Pruned)



Trees vs. Linear Model: Classification Example

- Top row: the true decision boundary is linear
  - Left: linear model (good)
  - Right: decision tree
- Bottom row: the true decision boundary is non-linear
  - Left: linear model
  - Right: decision tree (good)



Advantages and Disadvantages of Trees

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ 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 some of the other regression and classification approaches seen in this book.

However, by aggregating many decision trees, the predictive performance of trees can be substantially improved. We introduce these concepts next.