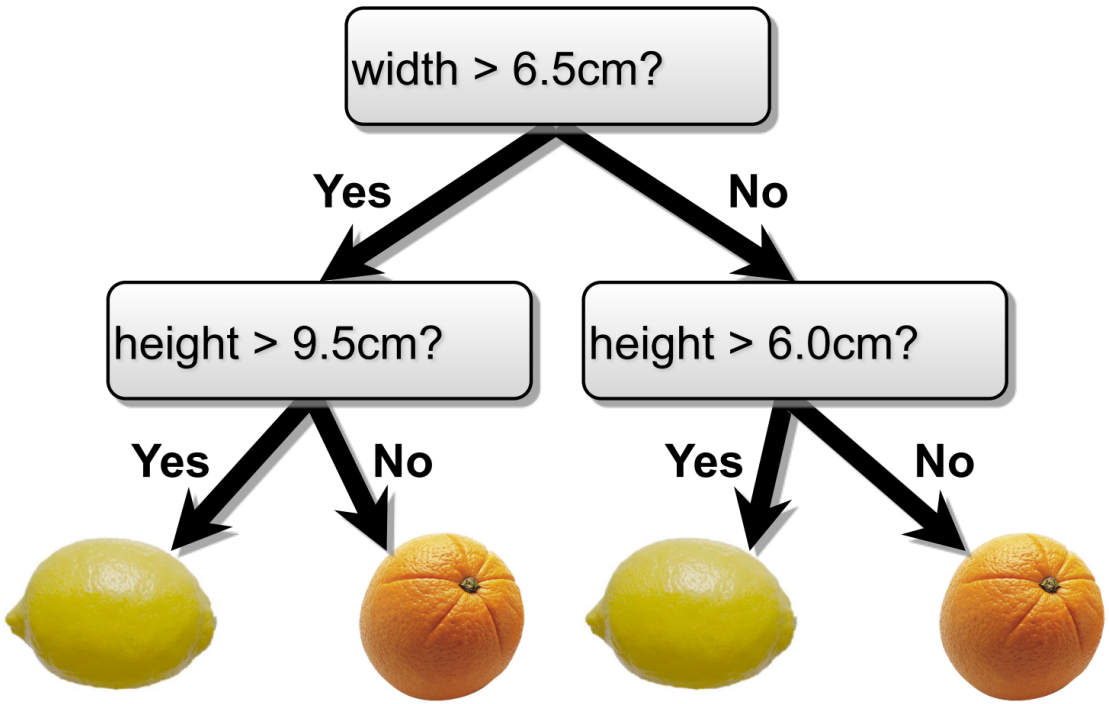
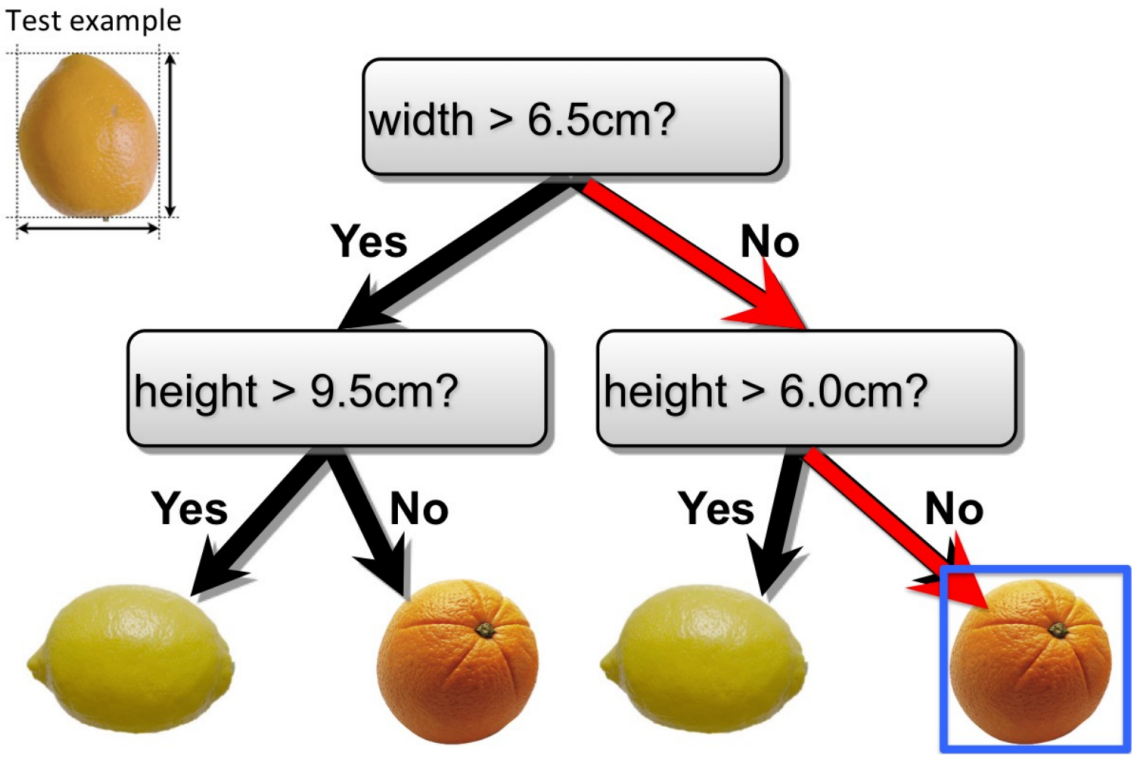
An Illustrative Example

- Internal nodes test attributes
- Branching is determined by attribute value
- Leaf nodes are outputs (predictions)

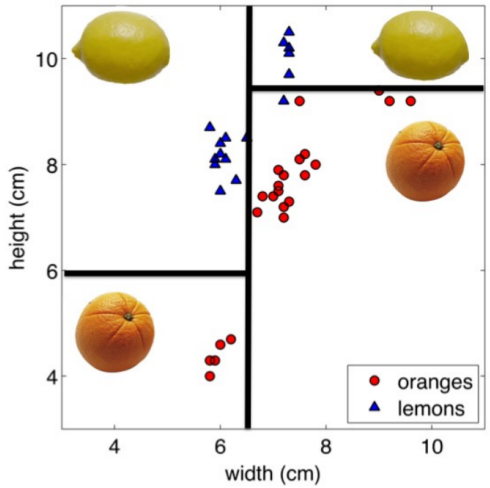


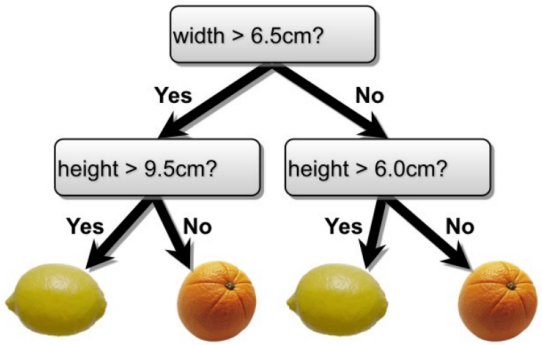
An Illustrative Example

- Decision trees make predictions by recursively splitting on different attributes according to a tree structure.



Building Decision Trees





- We divide the predictor space - that is, the set of possible values for $X_1, X_2, \cdots, X_p$ - into J distinct and non-overlapping regions, $R_1, R_2, \cdots, R_J$.
- Each path from root to a leaf defines a region R_j of input space.
- For every observation that falls into the region R_j, we make the same prediction.
- At each level, one must choose:
  1. Which variable to split.
  2. Possibly where to split it.

Building Decision Trees

- **Regression Tree**:
  - Continuous output
  - Leaf value y_j typically set to the **mean value** of the response values for the training observations in R_j.
- **Classification tree**:
  - Discrete output
  - Leaf value y_j typically set to the **most common value** of the response values for the training observations in R_j.

How do we construct the regions (Regression Tree)?

- In theory, the regions could have any shape.
- We choose to divide the space into high-dimensional rectangles, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, R_2, \cdots, R_J$ 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.
- It is computationally infeasible to consider every possible partition of the feature space into J boxes. So, what to do?

Recursive Binary Splitting

- Recursive, greedy approach to build a tree node-by-node.
- **Top-down**: Starts at the top of the tree and recursively splits the predictor space into branches; each split is indicated via two new branches further down on the tree.
- **Greedy**: Selects the best split at each step without considering future splits.

Recursive Binary Splitting

- We consider all predictors $X_1, \cdots, X_p$, and all possible values of the cutpoint s for each of the predictors.
- Then choose the predictor and cutpoint such that the resulting tree has the lowest RSS.
- For instance, suppose that we obtain two regions, R_1 and R_2.
- The pair of half-planes is defined as

$$R_1(j,s) = \{X|X_j < s\}, \quad R_2(j,s) = \{X|X_j \geq s\},$$

and we seek the value of j and s that minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2.$$

Recursive Binary Splitting

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

Building A Regression Tree: An Example

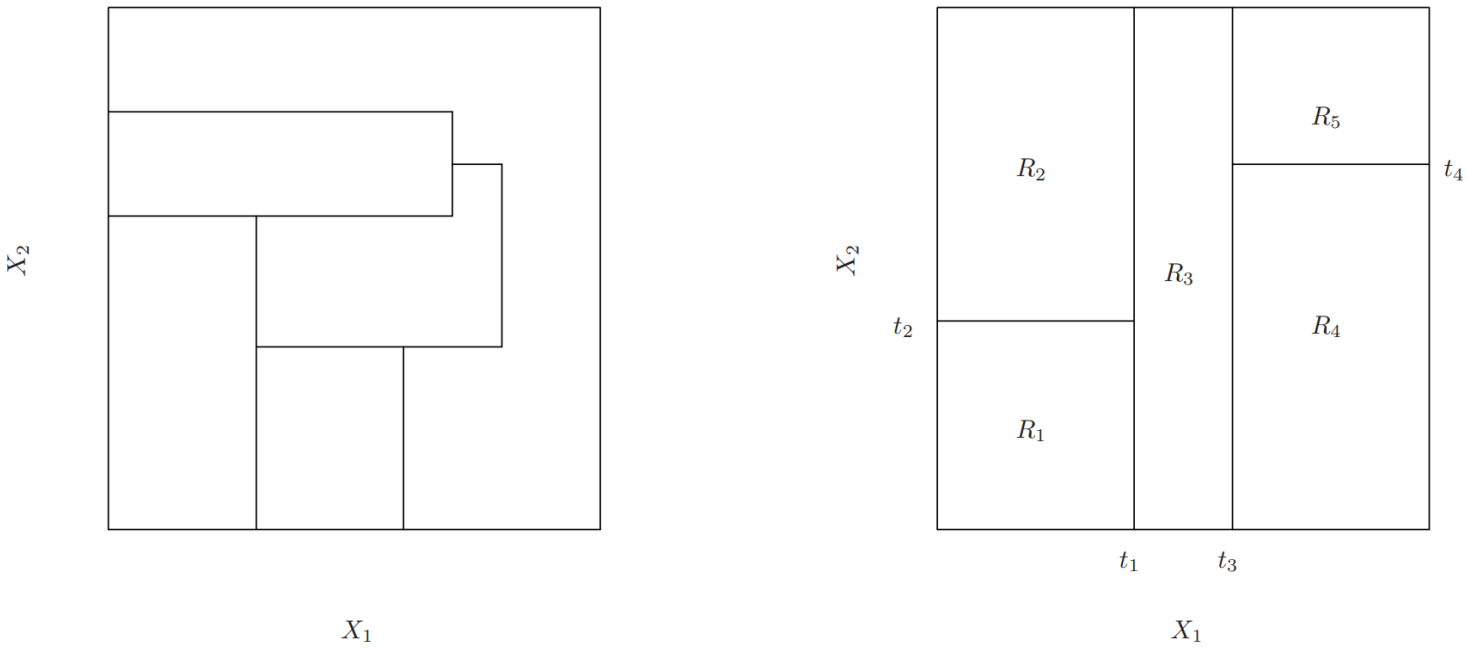


Figure: *Left*: A partition of two-dimensional feature space that could not result from recursive binary splitting. *Right*: The output of recursive binary splitting on a two-dimensional example.

Building A Regression Tree: An Example

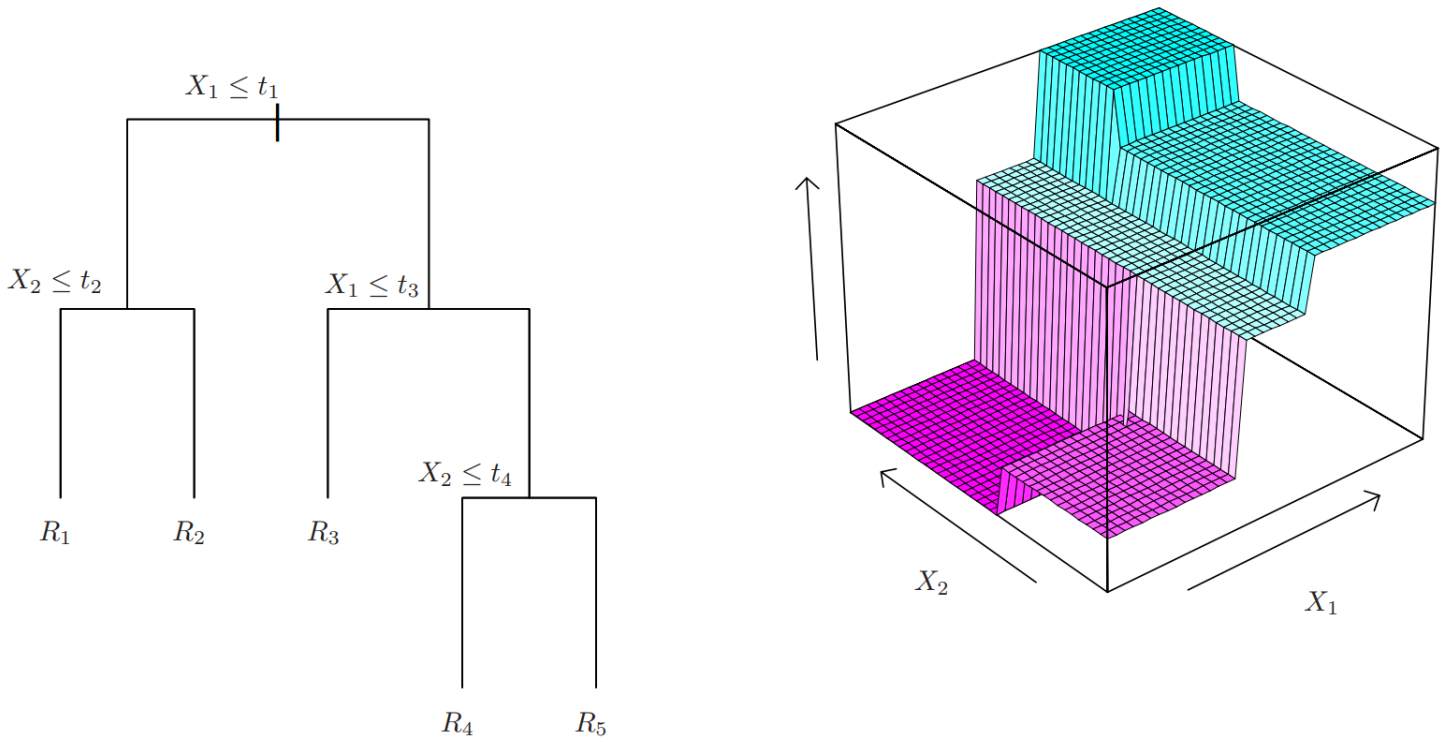


Figure: *Left*: A tree corresponding to the partition in previous figure. *Right*: A perspective plot of the prediction surface corresponding to that tree.

Building a Regression Tree: Tree Pruning

- Growing a large tree can overfit, harming test set performance.
- Smaller trees reduce variance and improve interpretability but add slight bias. But, How?

Building a Regression Tree: Tree Pruning

- Growing a large tree can overfit, harming test set performance.
- Smaller trees reduce variance and improve interpretability but add slight bias. But, How?
- Stopping splits based on a high RSS reduction threshold creates small trees but may miss better splits later, that is, a split that leads to a large reduction in RSS later on.

Building a Regression Tree: Tree Pruning

- Growing a large tree can overfit, harming test set performance.
- Smaller trees reduce variance and improve interpretability but add slight bias. But, How?
- Stopping splits based on a high RSS reduction threshold creates small trees but may miss better splits later, that is, a split that leads to a large reduction in RSS later on.
- A better approach: grow a large tree (T_0), then **prune** it to create an optimal **subtree**.

Cost Complexity Pruning

- Prunes the tree by balancing fit and complexity.
- Uses a tuning parameter α to penalize trees with many terminal nodes.
- Minimizes:

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

- Here $|T|$ indicates the number of terminal nodes of the tree T.
- As α increases, trees become smaller by systematically pruning branches.
- Produces a sequence of nested subtrees for different α values.

Building a Regression Tree: Algorithm

Algorithm 8.1 *Building a Regression Tree*

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. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

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

An Example

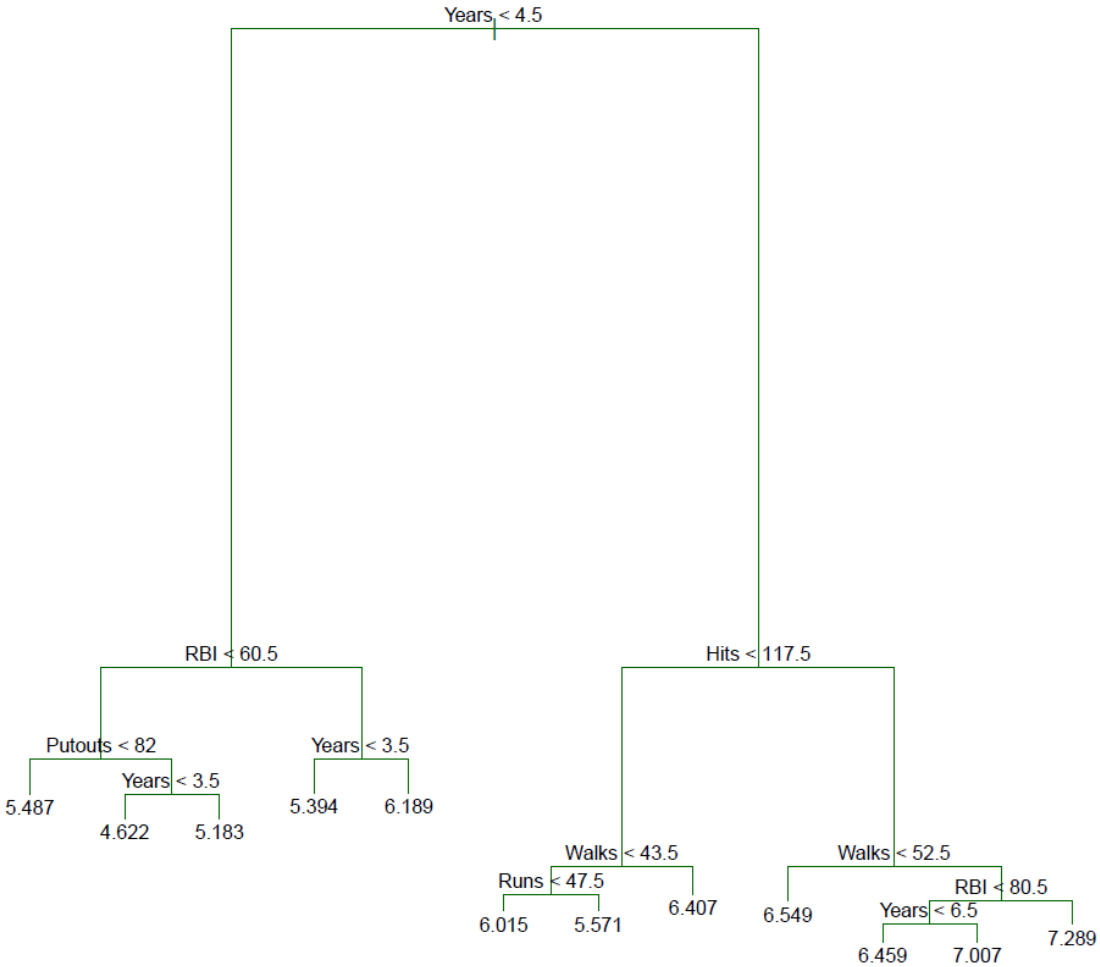
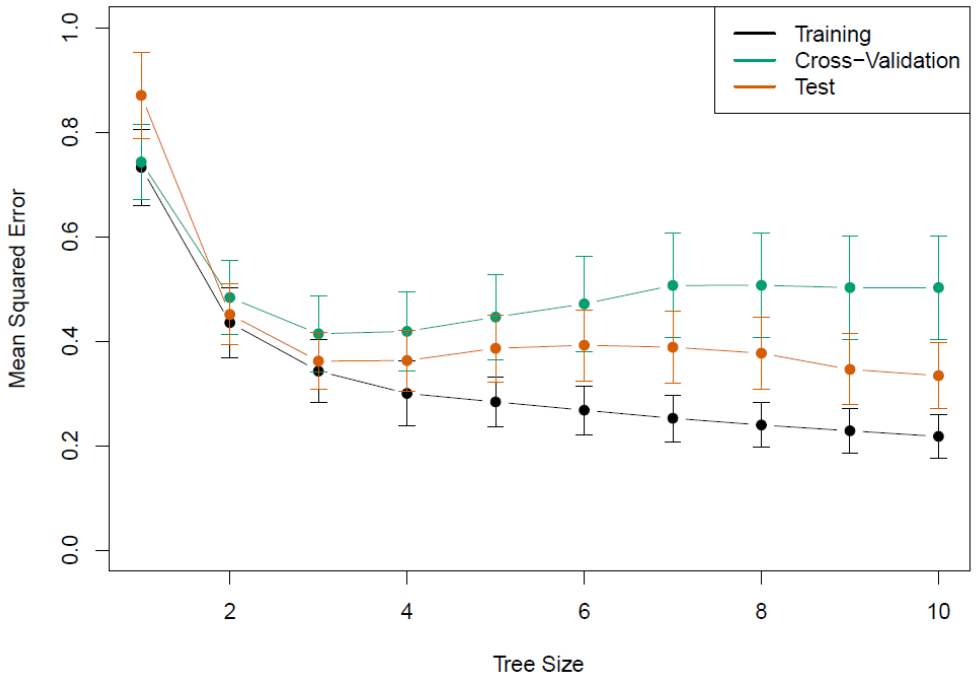
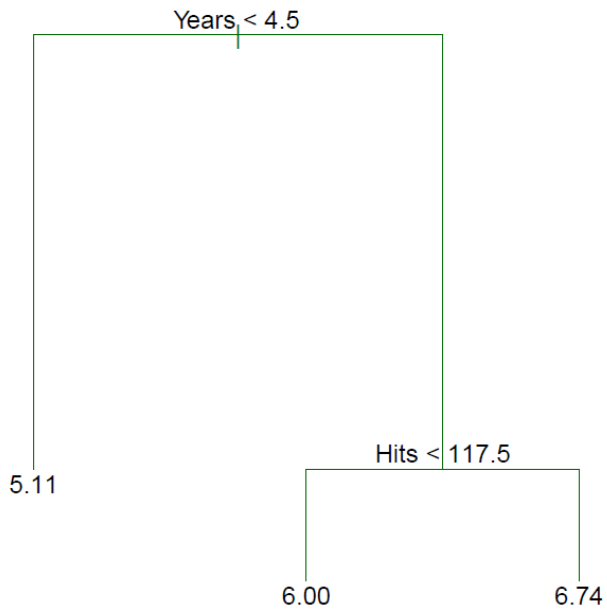


Figure: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown

An Example



(a)



(b)

Figure: Regression tree analysis for the Hitters data

Classification Trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.

Classification Trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- A natural alternative to RSS is the **classification error rate**.
- **classification error rate**: 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 mth region that are from the kth class.
- classification error is not sufficiently **sensitive** for tree-growing.

Classification Trees

- 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 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 **entropy**, given by

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

- Since $0 \leq \hat{p}_{mk} \leq 1$, it follows that $0 \leq -\hat{p}_{mk} \log \hat{p}_{mk}$.
- The entropy will take on a small value if the mth node is pure.
- When building a classification tree, either the Gini index or the entropy are typically used to evaluate the quality of a particular split.

Building a Classification Tree: An Example

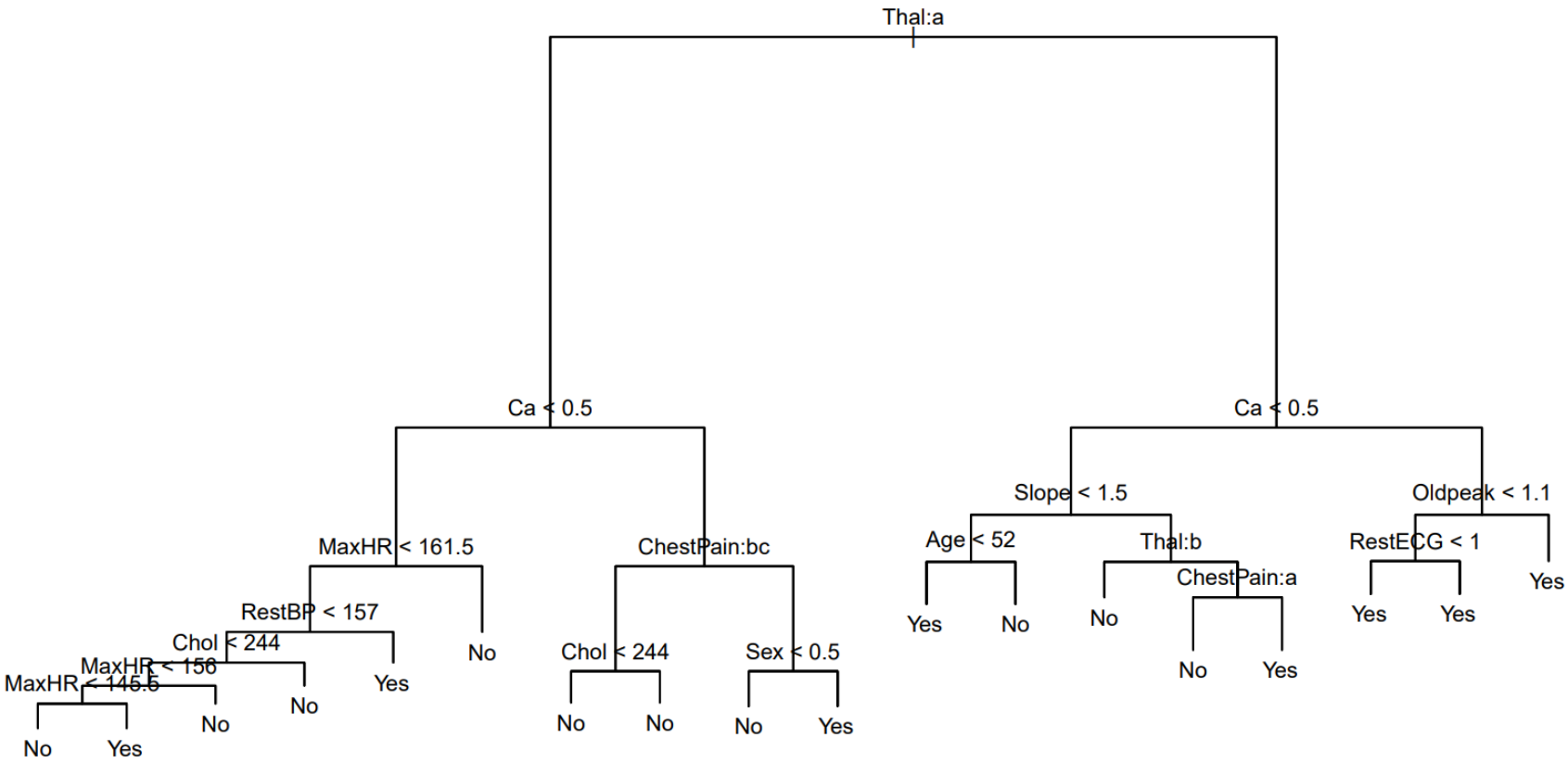


Figure: The unpruned tree

Building a Classification Tree: An Example

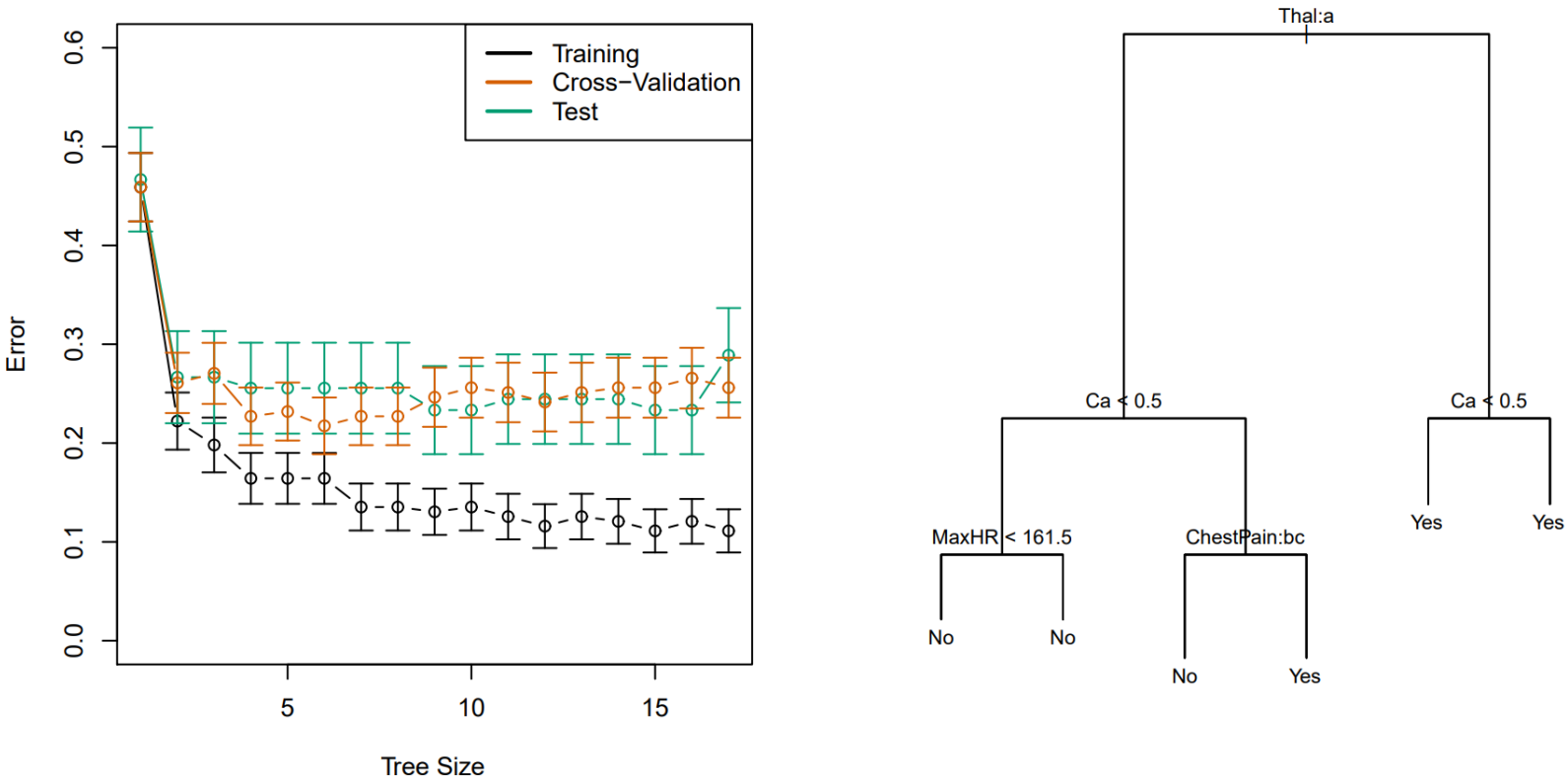


Figure: *Left*: Cross-validation error, training, and test error, for different sizes of the pruned tree. *Right*: The pruned tree corresponding to the minimal cross-validation error

Trees Versus Linear Models

- Linear regression assumes a model of the form

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j,$$

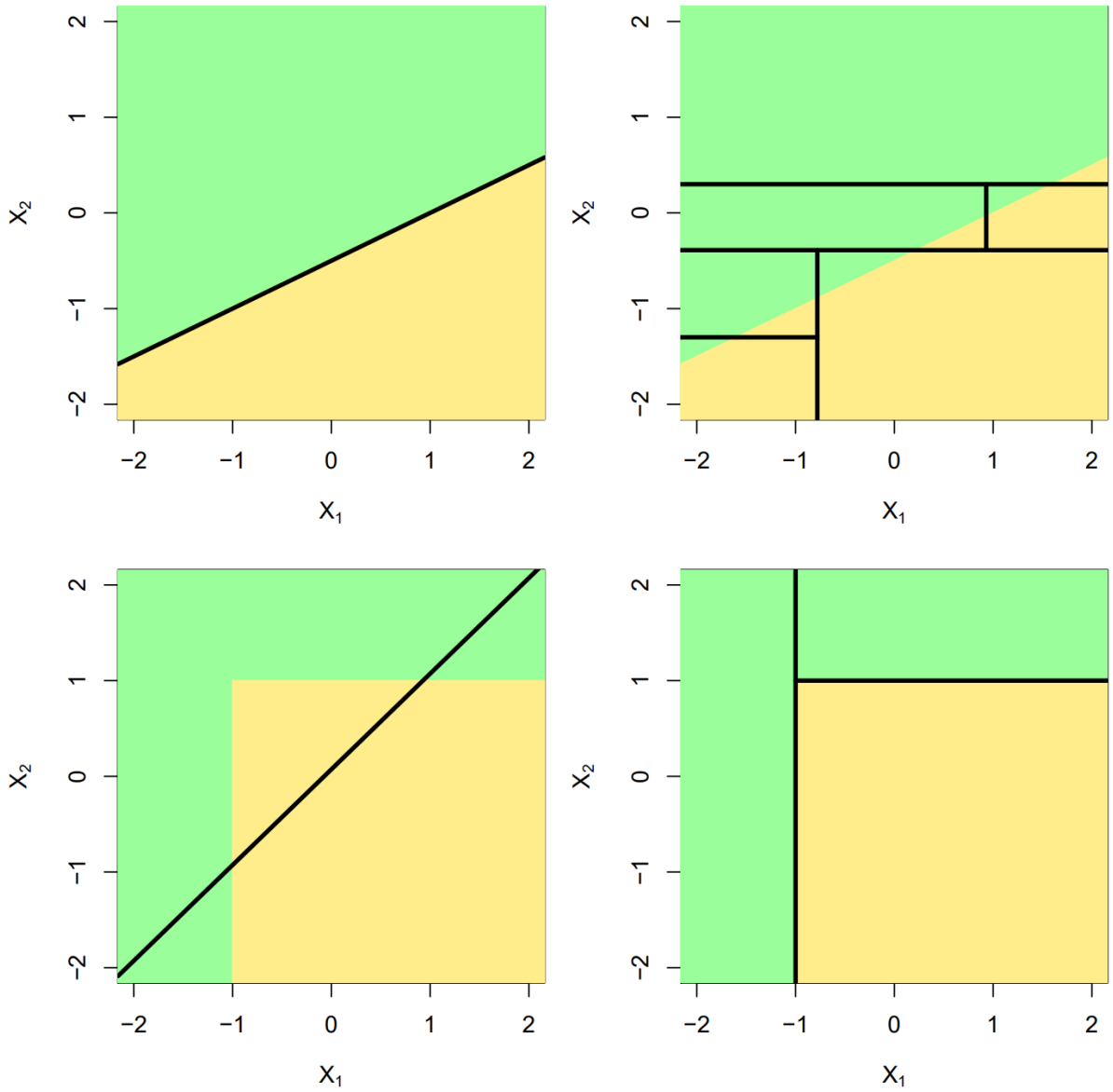
whereas regression trees assume a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \cdot 1_{(X \in R_m)},$$

where $R_1, \cdots, R_M$ represent a partition of feature space.

- When the features and response have a linear relationship, linear regression works better than regression trees.
- For highly nonlinear and complex relationships, decision trees outperform classical methods.

Trees Versus Linear Models: An illustrative example



Advantages and Disadvantages of Decision Trees

- **Advantages**:
  - Trees are very easy to explain to people (easier than linear regression).
  - Decision trees more closely mirror human decision-making than do the regression and classification approaches.
  - 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.
- **Disadvantages**:
  - Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.
  - Trees can be very non-robust. A small change in the data can cause a large change in the final estimated tree.

Table of Contents

Why Ensemble Learning?

- **Improved Performance:** Combining multiple models reduces errors and increases robustness.
- **Enhanced Generalization:** Multiple predictors capture complex relationships better than any single one.
- **Increased Stability:** Reduces risk by mitigating biases present in individual models.

Learning Objectives:

- Identify a variety of ensemble techniques.
- Explore tree-based methods (e.g., Random Forests, Extra Trees) and boosting variants (AdaBoost, Gradient Boosting).
- Gain practical insights with Scikit-Learn implementations.

Benefits and Challenges

- **Benefits:**
  - Increases accuracy through variance reduction.
  - Provides robustness to noise and outliers.
  - Often easier to generalize to unseen data.
- **Challenges:**
  - More complex training and tuning process.
  - Increased computational cost.
  - Risk of reduced interpretability (unless specific techniques are used).

Bagging

- **Ensemble methods** combine multiple simple models, known as **weak learners**, to create a powerful model.
- **Weak learners** individually produce mediocre predictions.
- Decision trees suffer from **high variance**.
- **Bagging** (Bootstrap Aggregation) reduces variance by averaging predictions from several models.
- Averaging improves model reliability:

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

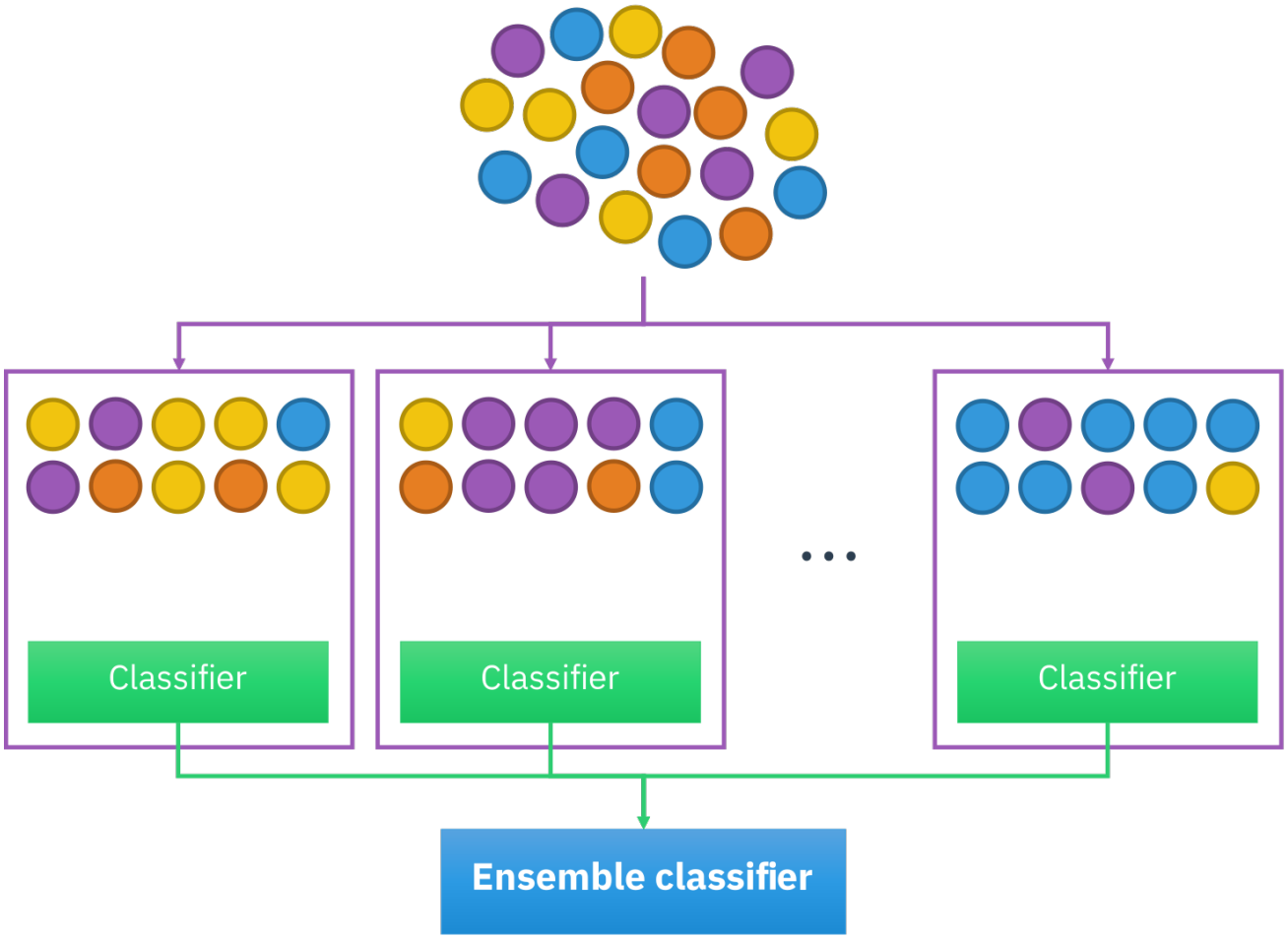
Bagging

- But we generally do not have access to multiple training sets.
- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- Bagging creates B bootstrapped datasets and trains a model on each, producing $\hat{f}^{*b}(x)$.
- The predictions are averaged to reduce variance:

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

- For regression trees, bagging involves constructing B trees from bootstrapped sets and averaging their predictions.

Bagging



Original Data

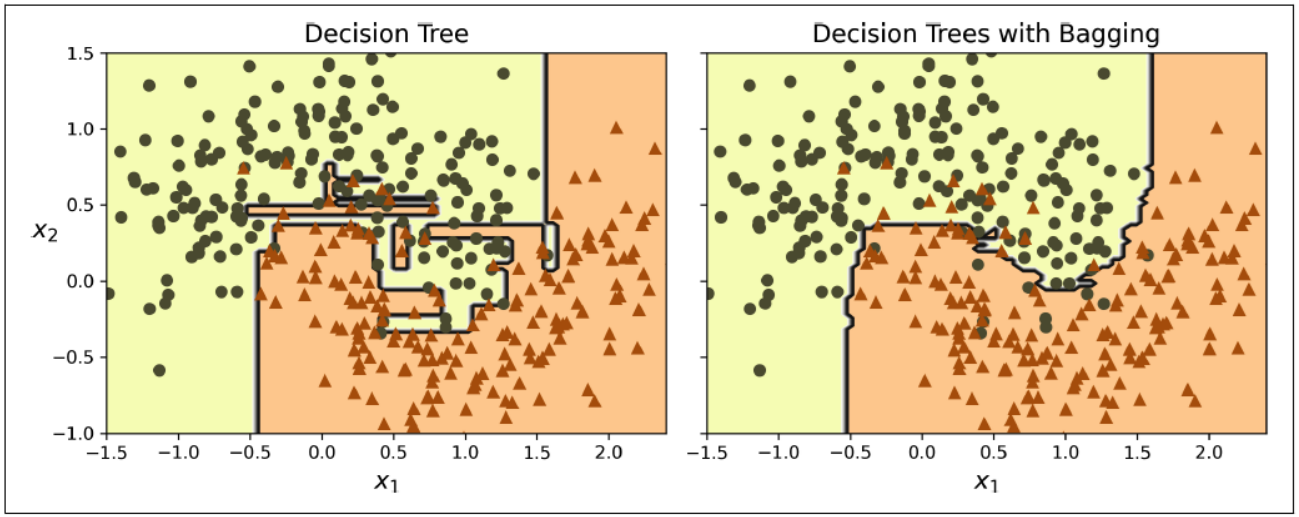
Bootstrapping

Aggregating

Bagging

Bagging

- For classification, each test observation is predicted by B trees, and the final prediction is the **majority vote**.
- Bagging reduces the test error rate compared to a single tree.
- The number of trees, B, is not critical and large values of B do not cause overfitting.
- On average, each bagged tree uses about two-thirds of the observations.



Performance measures using Bagging

The remaining one-third of unused observations are called **out-of-bag** (OOB) observations.

OOB Error: Estimates test error using unused observations (1/3 per tree).

- Tree-based methods provide a natural metric of feature importance.
- **Mechanisms:**
  - *Impurity Reduction:* Assessment based on metrics such as Gini impurity or entropy decrease.
  - *Weighted Contributions:* Importance is weighted by the number of samples reaching each node.