Tree-Based Methods

Jasmine. Hao[1]

[1]University of Hong Kong

ECON 3225: Big Data Economics

Outline

Outline

Tree-based Method

- **Overview:** The tree-based method is a popular machine learning algorithm used for both classification and regression tasks.
- **Type of Learning:** It is a type of supervised learning, where the algorithm learns from a labeled dataset to make predictions on new, unlabeled data.
- **Functioning:** The tree-based method works by recursively partitioning the data into smaller subsets based on the values of the input features.
- **Structure:** The algorithm builds a tree structure where each internal node represents a decision based on a particular feature, and each leaf node represents a predicted outcome.
- **Goal:** The goal is to minimize impurity or error in each node, so the algorithm chooses the feature that results in the greatest reduction of impurity at each step.

Tree-based method

- Tree-based methods are simple and useful for interpretation.
- However, they are not competitive with the best supervised learning approaches in terms of predictive accuracy.
- In this chapter:
  - Tree-based methods
  - Bootstrapping
  - Bagging
  - Random forest
  - Boosting
  - Bayesian additive regression trees
- Each of the approaches involves producing multiple trees which are then combined to yield a single consensus prediction.

Hitter's Dataset Overview

Context:

- Part of the R-package ISLR.

Content:

- Originally from StatLib library, Carnegie Mellon University.
- Salary data from Sports Illustrated (20 April 1987).
- Statistics for 1986 and his career.

Format:

- 322 observations on 20 variables.
- Variables: AtBat, Hits, HmRun, Runs, RBI, Walks, Years, CAtBat, CHits, CHmRun, CRuns, CRBI, CWalks, League, Division, PutOuts, Assists, Errors, Salary, NewLeague.

Hitter's Dataset Variables

- **AtBat**: Times at bat in 1986.
- **Hits**: Number of hits in 1986.
- **HmRun**: Home runs in 1986.
- **Runs**: Number of runs in 1986.
- **RBI**: Runs batted in in 1986.
- **Walks**: Number of walks in 1986.
- **Years**: Years in major leagues.
- **CAtBat**: Times at bat in career.
- **CHits**: Hits in career.
- **CHmRun**: Home runs in career.
- **CRuns**: Runs in career.
- **CRBI**: Runs batted in career.
- **CWalks**: Walks in career.
- **League**: League at end of 1986 (A/N).
- **Division**: Division at end of 1986 (E/W).
- **PutOuts**: Put outs in 1986.
- **Assists**: Assists in 1986.
- **Errors**: Errors in 1986.
- **Salary**: Salary in 1987 (thousands).
- **NewLeague**: League at start of 1987 (A/N).

[Figure 8.1] Tree

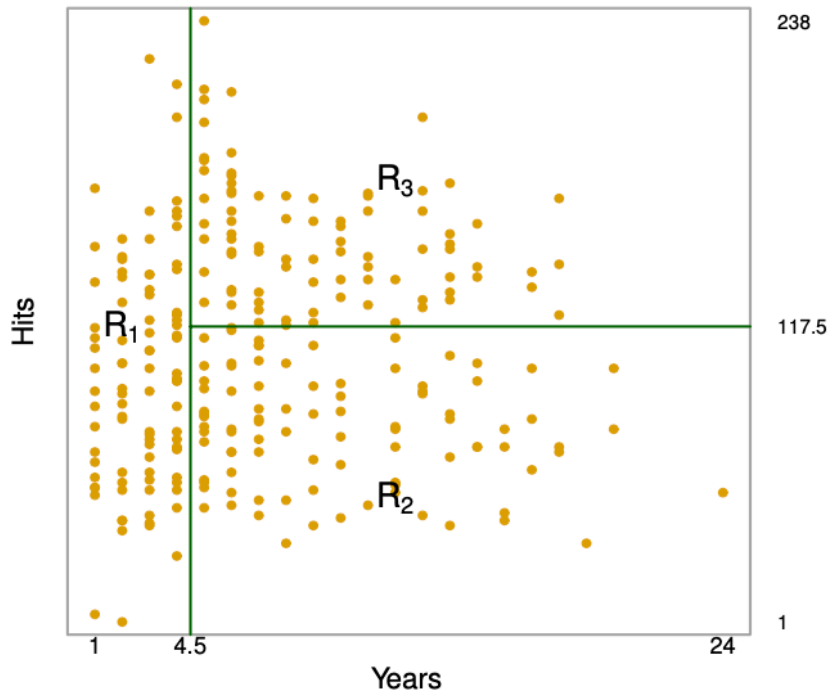
Tree Construction Example

- **Data:** For the Hitters dataset, a regression tree is built for predicting the log salary of a baseball player, based on the number of years he has played in the major leagues and the number of hits he made in the previous year.
  - **Node Label:** 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 $X_j \geq t_k$.
- **Tree Split:** 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.
  - **Leaf Number:** The number in each leaf is the mean of the response for the observations that fall there.

Predicting Salaries with Regression Trees

- **Data Set:** The Hitters data set is utilized to predict a baseball player's Salary based on Years (the number of years he has played in the major leagues) and Hits (the number of hits he made in the previous year).
- **Data Preparation:** Observations missing Salary values are removed and Salary is log-transformed to attain a more typical bell-shaped distribution.
- **Splitting Rules:** The following three regions, also known as terminal nodes or leaves, are determined based on the splitting rules:
  - $R1 = \{X | Years < 4.5\}$,
  - $R2 = \{X | Years >= 4.5, Hits < 117.5\}$,
  - $R3 = \{X | Years >= 4.5, Hits >= 117.5\}$.

[Figure 8.2]Three-region Partition



- Figure 8.2 illustrates the regions as a function of Years and Hits.
  - The predicted salaries for these three groups are $1,000 \times e^{5.107} = 165,174$, $1,000 \times e^{5.999} = 402{,}834$, and $1,000 \times e^{6.740} = 845,346$ respectively.

Tree Structure: Terminal Nodes, Internal Nodes, and Branches

- **Terminal Nodes:** The regions R1, R2, and R3 are known as terminal nodes or leaves of the tree, following the tree analogy.
- **Tree Orientation:** As depicted in Figure 8.1, decision trees are typically drawn *upside down*, with the leaves at the bottom of the tree.
- **Internal Nodes:** The points along the tree where the predictor space is split are referred to as internal nodes.
- **Internal Node Example:** In Figure 8.1, the two internal nodes are indicated by the conditions Years<4.5 and Hits<117.5.
- **Branches:** The segments of the tree that connect the nodes are referred to as branches.

Interpreting the Regression Tree

- **Tree Interpretation:** The regression tree in Figure 8.1 might be interpreted as follows:
  - Years is the most critical factor in determining Salary, with less experienced players earning lower salaries than more experienced players.
  - The number of hits in the previous year seems to play a minor role in salary for less experienced players.
  - For players with five or more years in major leagues, the number of hits in the previous year does impact the salary, with more hits correlating to higher salaries.
- **Simplification:** The regression tree in Figure 8.1 might be an oversimplification of the true relationship between Hits, Years, and Salary.
- **Advantages:** It is easier to interpret and provides a nice graphical representation, making it advantageous over other types of regression models.

Prediction via Feature Space Stratification

- **Building a Regression Tree:** The process involves two main steps:
  1. The predictor space — the set of possible values for $X_1, X_2, ..., X_p$ — is divided into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$.
  2. For each observation falling into the region R_j, we make the same prediction: the mean of the response values for the training observations in R_j.
- **Example:** Suppose in Step 1 we obtain two regions, R_1 and R_2, with response means of 10 and 20, respectively, for the training observations.
  - Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

How do we construct the regions?

- **Region Construction:**
  - We select to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and ease of interpretation.
- **Goal:** Minimize the Residual Sum of Squares (RSS) given by
  - $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})$ where $\hat{y}_{R_j}$ is the mean response for the training observations within the j-th box.
- **Challenge:** It is computationally infeasible to consider every possible partition of the feature space into J boxes.

Recursive Binary Splitting [1]

- We use a top-down, greedy approach known as recursive binary splitting.
- **Top-down approach:**
  - Begins at the top of the tree where all observations belong to a single region.
  - Successively splits the predictor space, indicated by two new branches further down the tree.
- The approach is **greedy** because:
  - At each step of the tree-building, the best split is made at that particular step rather than looking ahead for a split that will lead to a better tree in some future step.

Recursive Binary Splitting [2]

- To perform recursive binary splitting:
  - First, select the predictor X_j and the cutpoint s such that splitting the predictor space into two regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in RSS.
  - For any j and s, we define the pair of half-planes $R_1(j,s) = \{X|X_j < s\}$ and $R_2(j,s) = \{X|X_j \geq s\}$.
  - We seek the value of j and s that minimize the equation $\sum_{i:x_i \in R_1(j,s)}(y_i - \hat{y}_{R_1}) + \sum_{i:x_i \in R_2(j,s)}(y_i - \hat{y}_{R_2})$.

Recursive Binary Splitting [3]

- The process is repeated, looking for the best predictor and best cutpoint to split the data further to minimize the RSS within each of the resulting regions.
- Instead of splitting the entire predictor space, one of the two previously identified regions is split, resulting in three regions.
- The process continues, further splitting one of these three regions to minimize the RSS.
- Once the regions $R_1, ..., R_J$ have been created, the response for a given test observation is predicted using the mean of the training observations in the region to which that test observation belongs.

[Figure 8.3] A five-region example

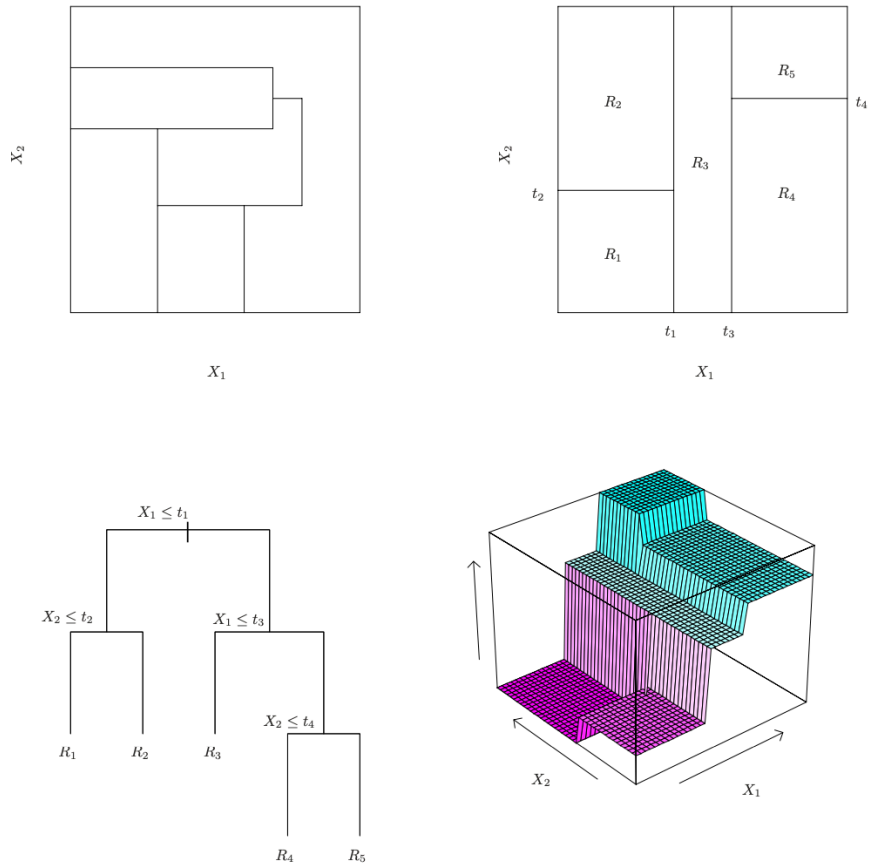


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A perspective plot of the prediction surface corresponding to that tree.*

Tree Pruning [1]

- **Overfitting Issue:** The described process might overfit the data, leading to poor test set performance.
  - A more complex tree may have good predictions on the training set but is likely to overfit the data.
  - A simpler tree with fewer regions may lead to lower variance and better interpretation at the expense of a slight increase in bias.
- **Threshold Approach:** Building the tree only so long as the decrease in RSS exceeds a certain threshold results in smaller trees, but this approach can be short-sighted.
  - A seemingly insignificant split early on could lead to a substantial reduction in RSS later.
- **Better Strategy:** It is more effective to grow a large tree and then prune it to obtain a more suitable subtree.

Tree Pruning [2]

- **Pruning Challenge:** How to determine the optimal pruning approach?
- **Goal:** Select a subtree that yields the lowest test error rate.
  - Cross-validation or the validation set approach can be used to estimate the test error for a given subtree.
  - Estimating the cross-validation error for every possible subtree would be burdensome due to the vast number of potential subtrees.
  - **Solution:** We need a method to select a small set of subtrees for consideration.

Cost Complexity Pruning

- The approach is also known as **weakest link pruning**
- as we increase α from zero, branches get pruned from the tree in a nested and predictable fashion, so obtaining the whole sequence of subtrees as a function of α is easy.
  - We can select a value of α using a validation set or using cross-validation.
  - We then return to the full data set and obtain the subtree corresponding to α.

Building a Regression Tree

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.

[Figure 8.4] Tree Pruning

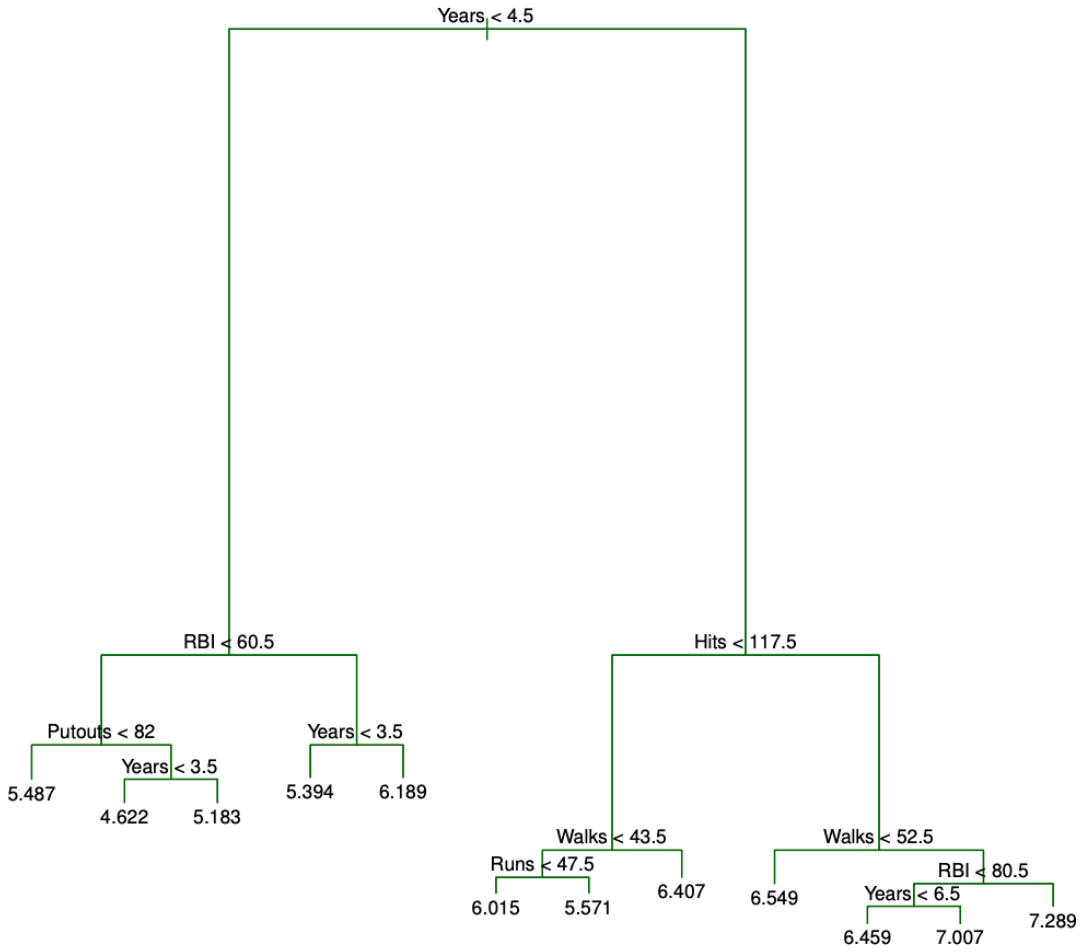


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

[Figure 8.5] Regression Tree

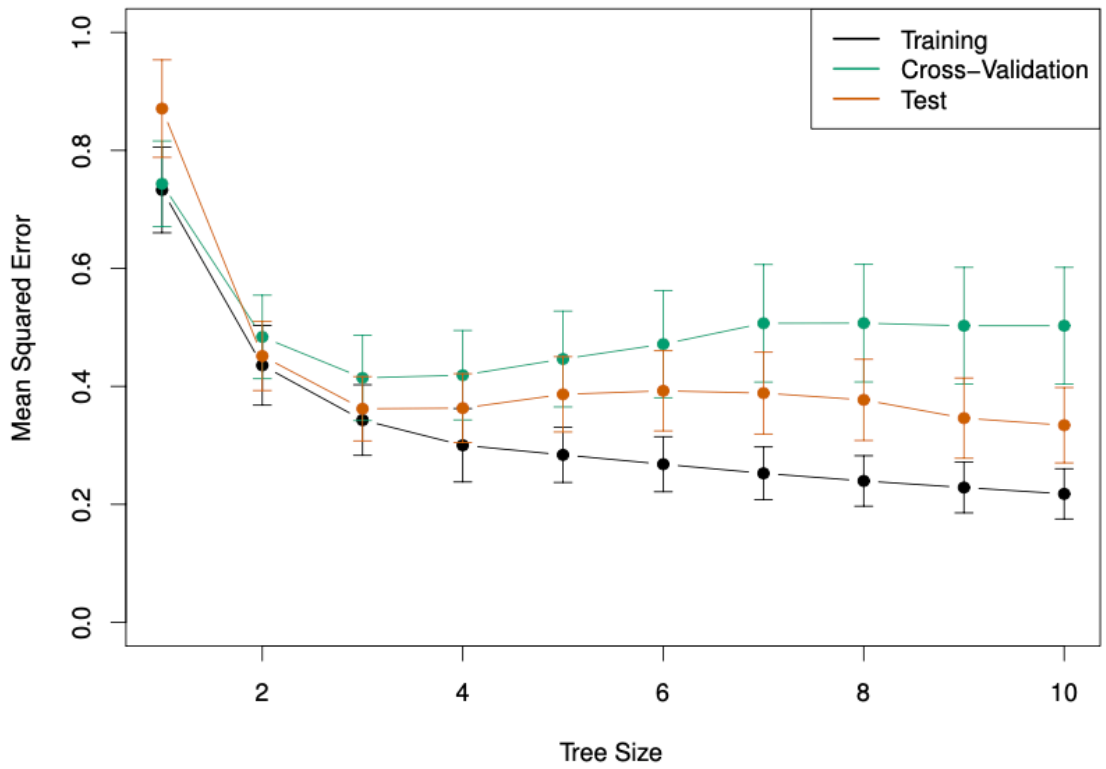


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

Regression Tree Fitting and Pruning

- **Data and Figures:** Figures 8.4 and 8.5 display the results of fitting and pruning a regression tree on the Hitters data, using nine features.
  - **Dataset Split:** The data set was randomly divided into halves, yielding 132 observations in the training set and 131 in the test set.
  - **Tree Building:** A large regression tree was built on the training data, and α in equation (8.4) was varied to create subtrees with different numbers of terminal nodes.
  - **Cross-Validation:** Six-fold cross-validation was performed to estimate the cross-validated MSE of the trees as a function of α.
- **Unpruned Tree:** The unpruned regression tree is shown in Figure 8.4.
  - **Error Curves:** The green curve in Figure 8.5 shows the CV error as a function of the number of leaves, while the orange curve indicates the test error. Standard error bars around the estimated errors are also shown.
  - **Training Error Curve:** For reference, the training error curve is shown in black. The CV error is a reasonable approximation of the test error.

Classification Tree

- **Overview:** A classification tree is similar to a regression tree, but it is used to predict a qualitative response instead of a quantitative one.
- **Prediction:**
  - In a regression tree, the predicted response for an observation is given by the mean response of the training observations in the same **terminal node**.
  - In a classification tree, we predict that each observation belongs to the most **commonly occurring class** of training observations in its region.
- **Interpretation:** When interpreting a classification tree, we are often interested in both the class prediction for a particular terminal node region and the class proportions among the training observations that fall into that region.

Classification error rate

- **Growing a Classification Tree:** The process is quite similar to growing a regression tree, using recursive binary splitting.
- **Criterion for Splits:** In the classification setting, RSS cannot be used for making binary splits. A natural alternative is the **classification error rate**.
- **Classification Error Rate:** This is the fraction of training observations in a region that do not belong to the most common class: $E = 1 - \max_k(\hat{p}_{km})$
  - Here, $\hat{p}_{km}$ represents the proportion of training observations in the mth region from the kth class.

Gini Index

- **Gini Index:** The Gini index is defined by the formula:

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{8.6}$$

- It measures the total variance across the K classes.
- Takes small values if all $\hat{p}_{mk}$ are close to 0 or 1, indicating a high degree of node purity.
- **Quality of Split:** In building a classification tree, the Gini index is often used to *evaluate the quality of a particular split* because it is sensitive to node purity.

Entropy

- **Entropy:** Entropy is an alternative measure to the Gini index and is defined as:

$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}), \tag{8.7}$$

- It also takes small values when $\hat{p}_{mk}$ are close to 0 or 1.
- **Node Purity:** A small value of entropy indicates the mth node is pure.
- **Quality of Split:** Like the Gini index, entropy can be used to *evaluate the quality of a particular split* in the context of classification trees, providing an alternative way to measure node purity.

[Figure 8.6] Pruned Tree

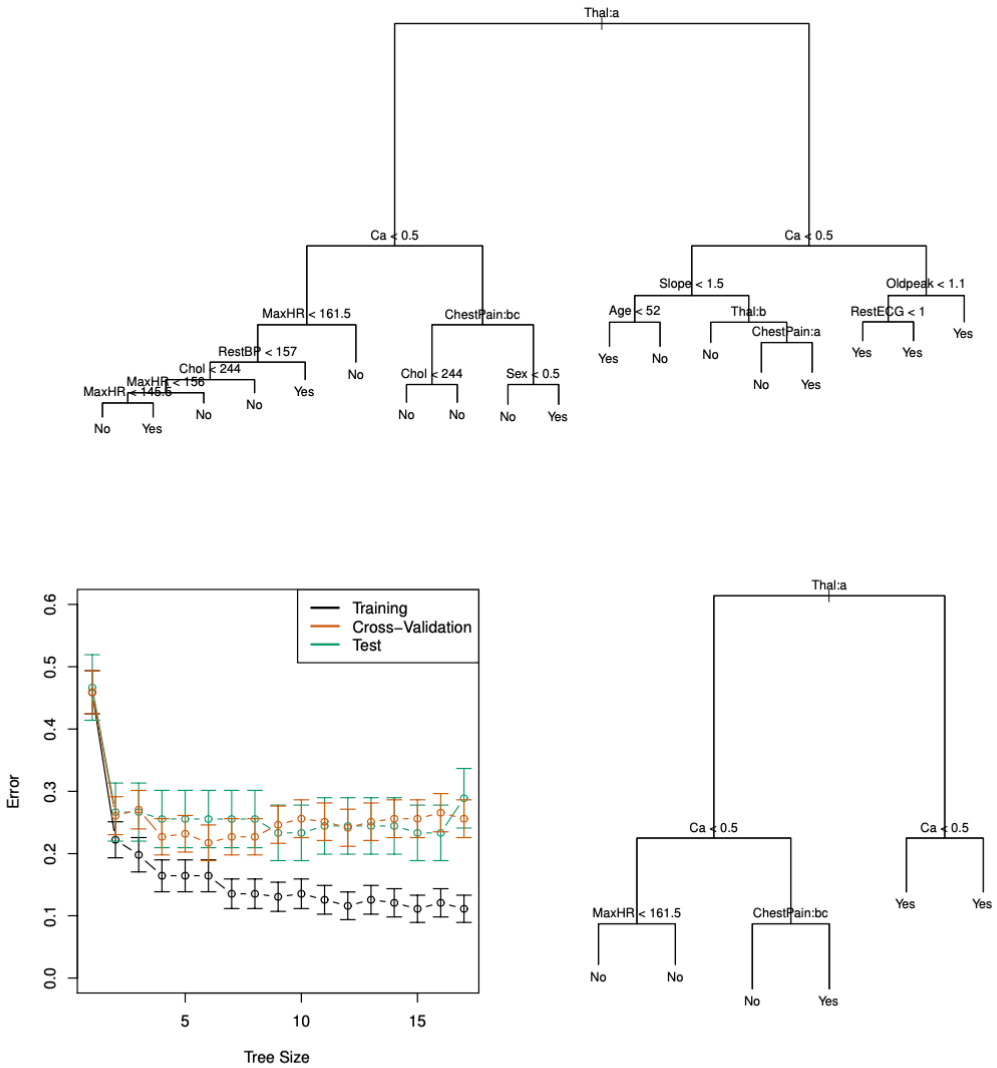


FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross-validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

Heart Data Set Overview

- **Data Set:** An example is shown in Figure 8.6 using the Heart data set.
- This data set contains a binary outcome HD for 303 patients who presented with chest pain.
- An outcome value of Yes indicates the presence of heart disease, while No means no heart disease.

Predictors in Heart Data Set

- There are 13 predictors in the data set, including Age, Sex, Chol (a cholesterol measurement), and other heart and lung function measurements.
- Cross-validation of this data results in a tree with six terminal nodes.

Decision Trees with Qualitative Predictors

- Decision trees can be constructed even when predictor variables are qualitative.
- For instance, in the Heart data, some predictors, such as Sex, Thal (Thallium stress test), and ChestPain, are qualitative.
- A split on one of these variables assigns some of the qualitative values to one branch and the remaining to the other branch.

Splits on Qualitative Variables

- **Introduction:**
  - In Figure 8.6, internal nodes demonstrate the splitting of qualitative variables within the classification tree.
- **Example - Thal:**
  - The primary internal node splits on *Thal*, with *Thal:a* indicating the left branch has observations where *Thal* is normal.
  - The right branch includes observations with fixed or reversible defects.
- **Example - ChestPain:**
  - A subsequent split on *ChestPain:bc* further down the left side of the tree delineates observations with the second and third values of the *ChestPain* variable on the left branch.
- **Implications:**
  - These splits categorize patients by specific characteristics of *Thal* and *ChestPain*, which are crucial for the classification process.

Surprising Splits and Node Purity

- **Observation:**
  - In Figure 8.6, some splits result in terminal nodes with identical predicted outcomes, which may seem counterintuitive at first glance.
- **Example - RestECG Split:**
  - The split on *RestECG<1* is particularly unexpected. Both resulting terminal nodes predict a "Yes" response, irrespective of the *RestECG* value.
- **Rationale for the Split:**
  - This split is made to enhance *node purity*, demonstrating that the goal isn't solely to decrease the classification error.
  - Such splits can improve the model's ability to generalize by focusing on the purity of the data within the nodes, which might be indicative of more subtle data structures.

Importance of Node Purity

- **Significance of Node Purity:**
  - Node purity is a critical factor in the effectiveness of decision trees.
  - A high purity in terminal nodes increases the confidence in the predictions made for test observations falling within these regions.
- **Certainty in Prediction:**
  - Observations in the right-hand leaf from the *RestECG<1* split are highly likely to have a response value of Yes.
  - Conversely, observations in the left-hand leaf also lean towards a Yes response, but with less certainty.
- **Impact on Gini Index and Entropy:**
  - Increased node purity is directly related to improvements in the Gini index and entropy measures.
  - These metrics are preferred over the classification error rate as they are more sensitive to the distribution of classes within the nodes.

Tree v.s. LinearModels

- Regression and classification trees have a very different flavor from the more classical approaches for regression and classification.
  - Linear regression assumes a model of the form
    - $$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{8.8}$$
  - Regression trees assume a model of the form

$$f(X) = \sum_{m=1}^{M} c_m \mathbb{1}_{X \in R_m} \tag{8.9}$$

Model Selection Overview

- The choice of model depends on the problem at hand.
- The relationship between the features and the response is critical in determining the model choice.
- This can either be linear or non-linear and complex.

Choosing Between Linear Models and Decision Trees

When to Use Linear Models

- The relationship between features and response is linear.
- Linear structures in the data are well captured by methods such as linear regression.
- They can be more effective than methods like regression trees when this linear premise holds true.

When to Use Decision Trees

- The relationship between features and response is non-linear and complex.
- They can model interactions that linear models cannot easily capture.
- Decision trees may provide better performance in these scenarios.
- See Figure 8.7 for an example.

Assessing Model Performance

- The relative performances of tree-based and classical approaches can be assessed by estimating the test error.
- This estimation can be done using either cross-validation or the validation set approach.

[Figure 8.7] Tree v.s. Linear

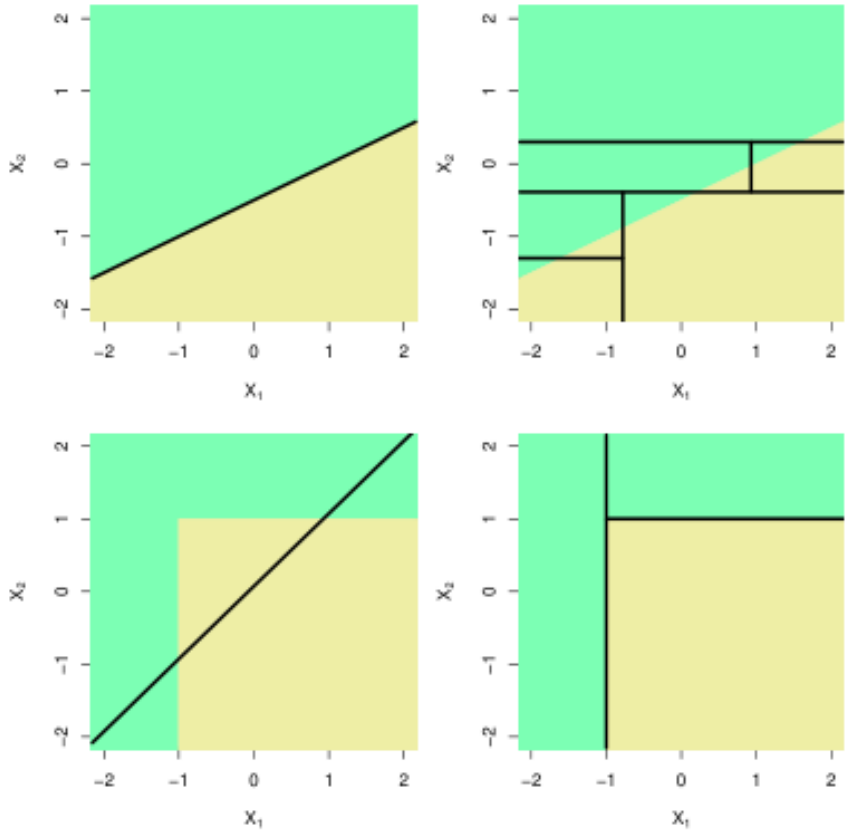


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: *Here the true decision boundary is non-linear. Here a linear model is unable to capture the true decision boundary (left), whereas a decision tree is successful (right).*

Pros and Cons

- Pros
  - Trees are very **easy to explain** to people.
    - 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,
    - 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.
- Cons
  - trees generally do not have the same level of predictive **accuracy** as some of the other regression and classification approaches
  - Additionally, trees can be very **non-robust**.
    - a small change in the data can cause a large change in the final estimated tree.

Outline

The Bootstrap

- **Overview:**
  - Reference: Chapter 5.2
  - The bootstrap is a versatile and robust statistical tool designed to assess uncertainty in estimators and statistical methods.
- **Application Example:**
  - It can, for instance, be used to calculate standard errors for coefficients in a linear regression model.
  - While in linear regression standard errors are readily available (Chapter 3), the bootstrap provides a general approach applicable in more complex scenarios.
- **Advantage of the Bootstrap:**
  - The bootstrap's real strength is its applicability to a vast array of models, particularly those where variability is not straightforward to measure.
  - It fills a gap by providing an estimate of variability for methods where statistical software may not offer this information by default.

[Example] Bootstrap

- In this section we illustrate the bootstrap on a toy example in which we wish to determine the best investment allocation under a simple model.
- Suppose we wish to invest fixed amount of money in financial assets X and Y
  - where X and Y are random quantities
  - invest a fraction α of money in X and invest the rest of money in Y
- Minimize the total risk
  - min $\mathrm{Var}(\alpha X + (1-\alpha)Y)$
  - We can show $\alpha = \frac{\sigma_X^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$
- However, in reality$\sigma_X^2, \sigma_Y^2, \sigma_{XY}$ unknown
- Need to estimate $\hat{\sigma}_X^2, \hat{\sigma}_Y^2, \hat{\sigma}_{XY}$ and then estimate the value $\hat{\alpha}$

[Figure 5.9] Bootstrap

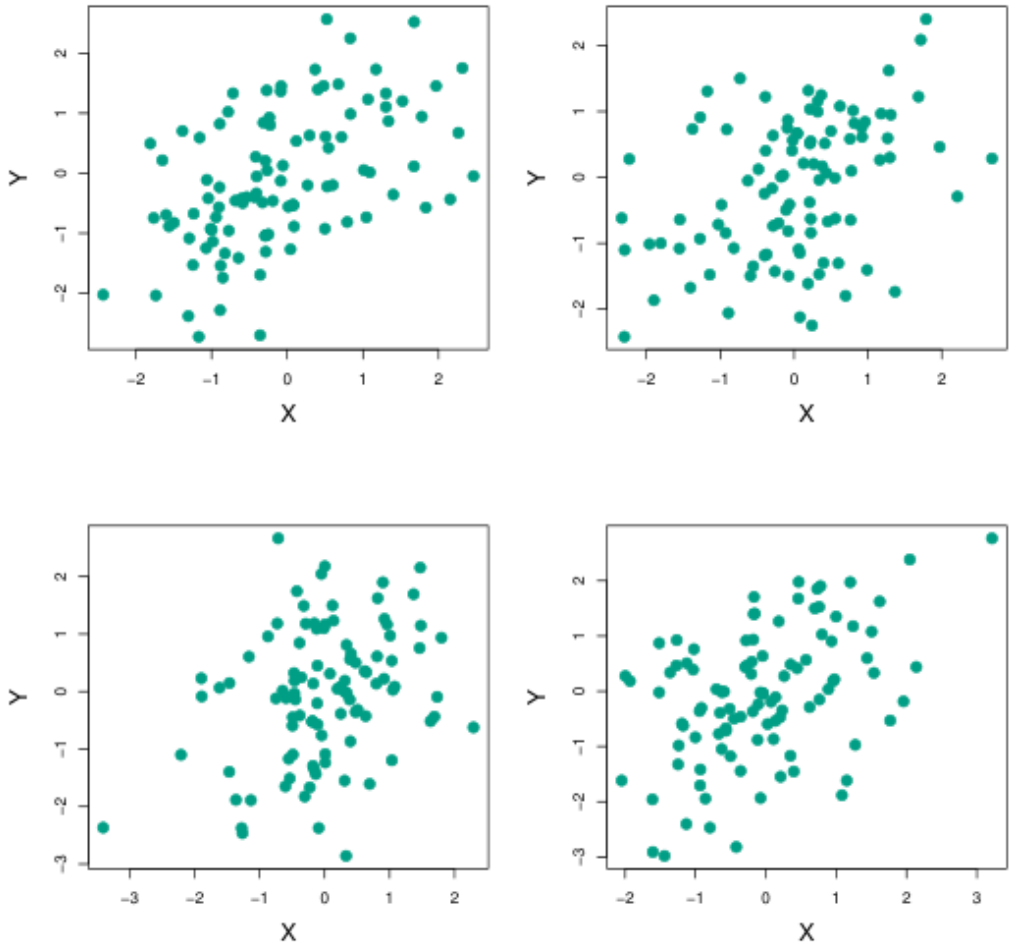


FIGURE 5.9. *Each panel displays* 100 *simulated returns for investments* X *and* Y. *From left to right and top to bottom, the resulting estimates for* α *are* 0.576, 0.532, 0.657, *and* 0.651.

Simulation Approach

- Want to characterize the **accuracy** of $\hat{\alpha}$
- **Simulation Approach**:
  1. To estimate the standard deviation of α, we repeat the process of simulating 100 paired observations of X and Y
  2. Estimate $\hat{\alpha}$ for 1000 times.
  3. Have 1000 observations of $\hat{\alpha}$, $\hat{\alpha}_1$,... , $\hat{\alpha}_{1000}$
  4. The mean over 1000 estimates of α is $\bar{\alpha} = \frac{1}{1000}\sum_{r=1}^{1000}\hat{\alpha}_r - 0.5996$, close to the true value of 0.6
  5. The standard deviation of the estimates is $\sqrt{\frac{1}{1000-1}\sum_{r=1}^{1000}(\hat{\alpha}_r - \bar{\alpha})^2} = 0.083$
  6. $SE(\hat{\alpha}) \approx 0.083$

Outline

Understanding the Bootstrap Approach

The Challenge with Real Data:

- Traditional methods for estimating the standard error of an estimator, such as $SE(\hat{\alpha})$, cannot be directly applied to real data because generating additional samples from the original population is not feasible.

Bootstrap Solution:

- The bootstrap uses computational power to simulate the process of sampling from the population by resampling the original dataset.
- This method creates distinct datasets by repeatedly sampling *with replacement* from the original dataset, providing an avenue to estimate the variability of $\hat{\alpha}$.

Implementing the Bootstrap Approach

Step-by-Step Procedure:

1. Generate a *bootstrap dataset* Z^{*1} by randomly selecting n observations from the original dataset, allowing for repetitive selection of the same observation.
   - Include both X and Y values for any selected observations.
2. Calculate a new estimator $\hat{\alpha}^{*1}$ from the bootstrap dataset Z^{*1}.
3. Repeat steps 1 and 2 for B iterations to form a series of bootstrap datasets $Z^{*1}, Z^{*2}, \ldots, Z^{*B}$ and corresponding estimates $\hat{\alpha}^{*1}, \hat{\alpha}^{*2}, \ldots, \hat{\alpha}^{*B}$.
4. Estimate the standard error of $\hat{\alpha}$ using the bootstrap estimates with the formula:

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1}\sum_{r=1}^{B}\left(\hat{\alpha}^{*r} - \frac{1}{B}\sum_{r'=1}^{B}\hat{\alpha}^{*r'}\right)^2}$$

Conclusion:

- The bootstrap standard error $SE_B(\hat{\alpha})$ provides a robust estimate of uncertainty for $\hat{\alpha}$, even when the population data cannot be

[Figure 5.10] Simulation v.s. Bootstrap

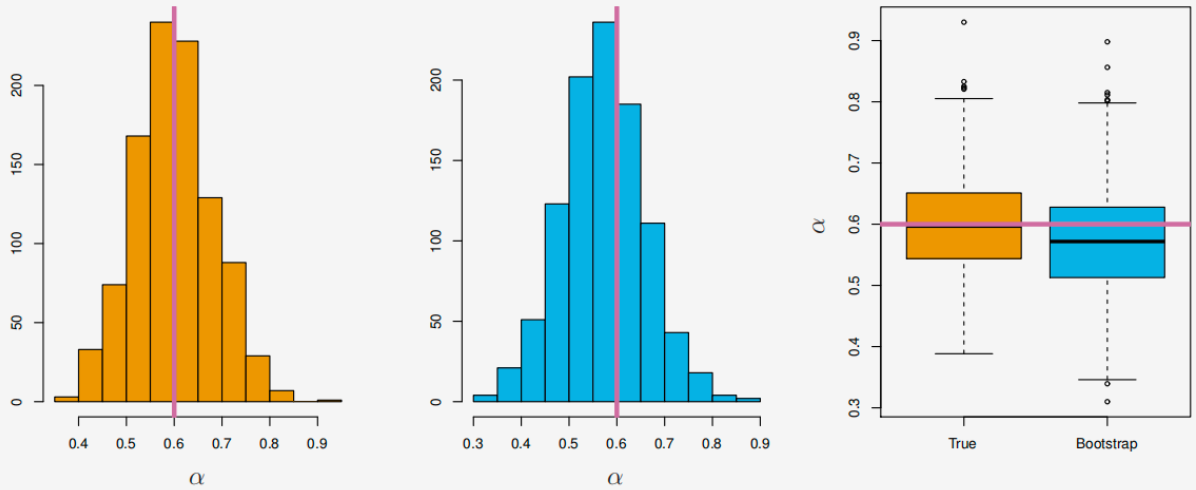


FIGURE 5.10. Left: *A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population.* Center: *A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set.* Right: *The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.*

[Figure 5.11] Bootstrap Sample

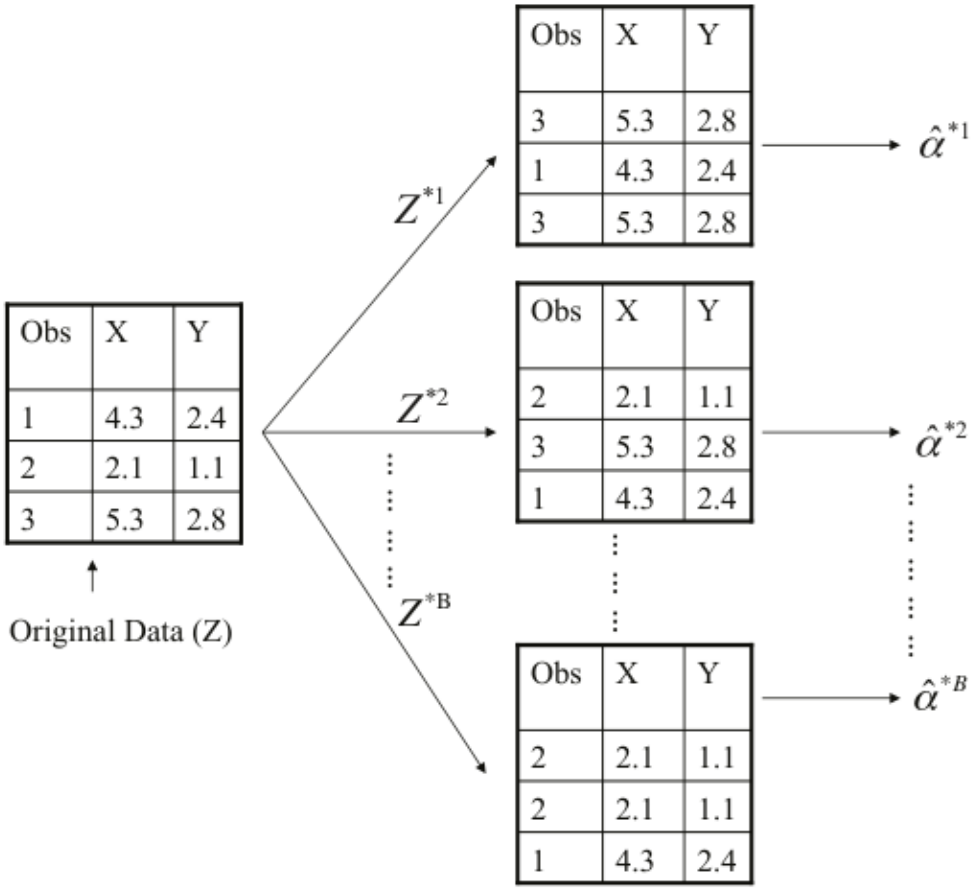


FIGURE 5.11. *A graphical illustration of the bootstrap approach on a small sample containing $n = 3$ observations. Each bootstrap data set contains n observations, sampled with replacement from the original data set. Each bootstrap data set is used to obtain an estimate of α.*

[Illustration] Bootstrapped $SE(\hat{\alpha})$

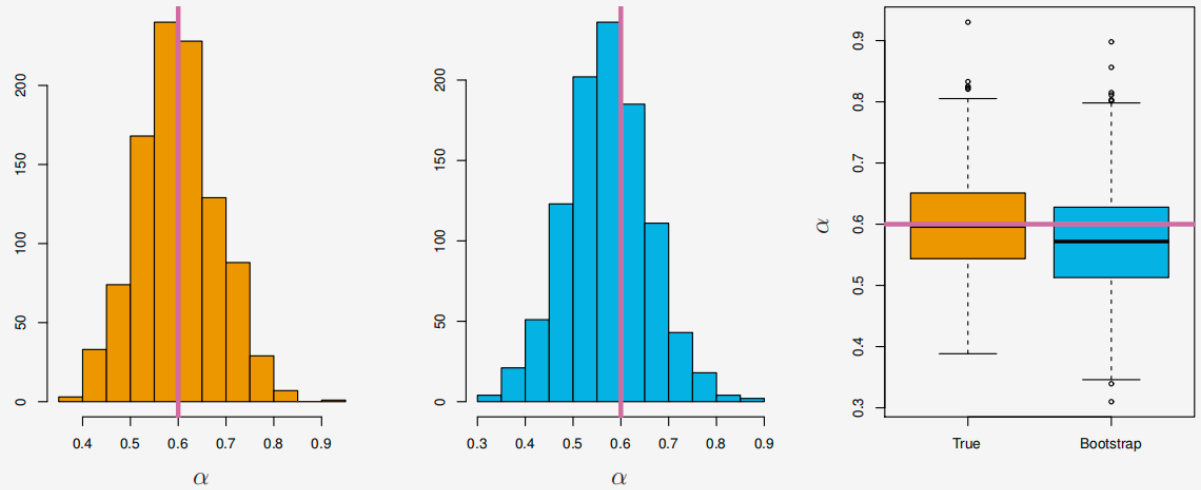


FIGURE 5.10. Left: *A histogram of the estimates of α obtained by generating 1,000 simulated data sets from the true population.* Center: *A histogram of the estimates of α obtained from 1,000 bootstrap samples from a single data set.* Right: *The estimates of α displayed in the left and center panels are shown as boxplots. In each panel, the pink line indicates the true value of α.*

Comparison of Estimates for α using Simulation and Bootstrap

- **Histogram Analysis:** The histogram for estimates of α obtained from 1,000 simulated data sets (from the true population and bootstrapped data sets) are very similar (left-hand side).
- **Bootstrap Estimate:** The bootstrap estimate $SE(\hat{\alpha})$ is 0.087, closely approximating the estimate of 0.083 obtained from 1,000 simulated data sets.
- **Boxplot Presentation:** The right-hand panel presents the same data through boxplots of estimates for α obtained from 1,000 simulated data sets and the bootstrap approach.
- **Boxplot Analysis:** The similar spread of the boxplots indicates that the bootstrap approach effectively estimates the variability associated with $\hat{\alpha}$.

Outline

Introduction to Bagging

- **Variance in Decision Trees:**
  - Decision trees are prone to high variance, which can cause the model to fit differently to various subsets of the training data (overfitting).
- **Contrast with Low Variance Models:**
  - In contrast, models with low variance, like linear regression, provide more consistent results across different datasets, especially when the number of observations n is much larger than the number of predictors p.
- **Reducing Variance with Bagging:**
  - Bagging, short for Bootstrap Aggregation, is a technique designed to decrease the variance of prediction models by generating multiple versions of a predictor and using these to get an aggregated predictor.
  - The methodology is particularly beneficial for high variance models such as decision trees, leading to improved model stability and accuracy.

Bagging Mechanism

- **Variance Reduction:** Averaging a set of observations reduces variance. Given a set of n independent observations $Z_1, ..., Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- **Increasing Test Set Accuracy:** One way to reduce variance and increase test set accuracy is to:
  - Take many training sets from the population,
  - Build a separate prediction model using each training set,
  - Average the resulting predictions.
- This leads to a single low-variance statistical learning model:

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

Practical Implementation of Bagging

- **Challenge:** We typically do not have access to multiple training sets.
- **Solution:** We use bootstrap to take repeated samples from the single training data set.
- **Procedure:** Generate B different bootstrapped training data sets, train our method on each one to get $\hat{f}^{*b}(x)$, and average all the predictions to obtain

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

- This procedure is known as **bagging**.

Bagging for Decision Trees and Majority Vote

- **Bagging and Decision Trees:** While bagging can improve predictions for many regression methods, it is particularly useful for decision trees.
- **Applying Bagging to Regression Trees:** To implement bagging, construct B regression trees using B bootstrapped training sets, and average the resulting predictions.
  - These trees are grown deep and are not pruned, each tree having high variance but low bias.
  - Averaging these B trees reduces the variance. Bagging, by combining hundreds or even thousands of trees, has shown impressive improvements in accuracy.
- **Bagging in Classification Context:** In a classification problem where Y is qualitative, bagging can be extended by:
  - Recording the class predicted by each of the B trees for a given test observation,
  - Taking a majority vote. The overall prediction is the most commonly occurring class among the B predictions.

Majority Vote in Bagging

- **Bagging in Regression Context:** Bagging has been discussed in the context of regression to predict a quantitative outcome Y. The question arises: How can bagging be extended to a classification problem where Y is qualitative?
- **Possible Approaches:** There are several possible ways to extend bagging to classification problems, but the simplest approach is as follows:
- **Majority Voting:** For a given test observation, record the class predicted by each of the B trees. Then, take a majority vote. The overall prediction is the most common class among the B predictions.

Out-of-Bag Error Estimation

- **Usage of Observations:** On average, each bagged tree makes use of approximately two-thirds of the observations. [Exercise 5.2]
- **Out-of-Bag 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.
- **Prediction for ith Observation:** Predict the response for the ith observation using each tree where that observation was OOB. This yields approximately $B/3$ predictions for the ith observation.
  - To obtain a single prediction for the ith observation, *average these predicted responses* (for regression) or take a *majority vote* (for classification).
  - An OOB prediction can be obtained for each of the n observations. From these, the overall OOB Mean Squared Error (for regression) or classification error (for classification) can be computed.
  - The resulting OOB error is a valid estimate of the test error for the bagged model, as the response for each observation is predicted using only the trees not fit using that observation.

[Illustration] Out-of-Bag Error Estimation

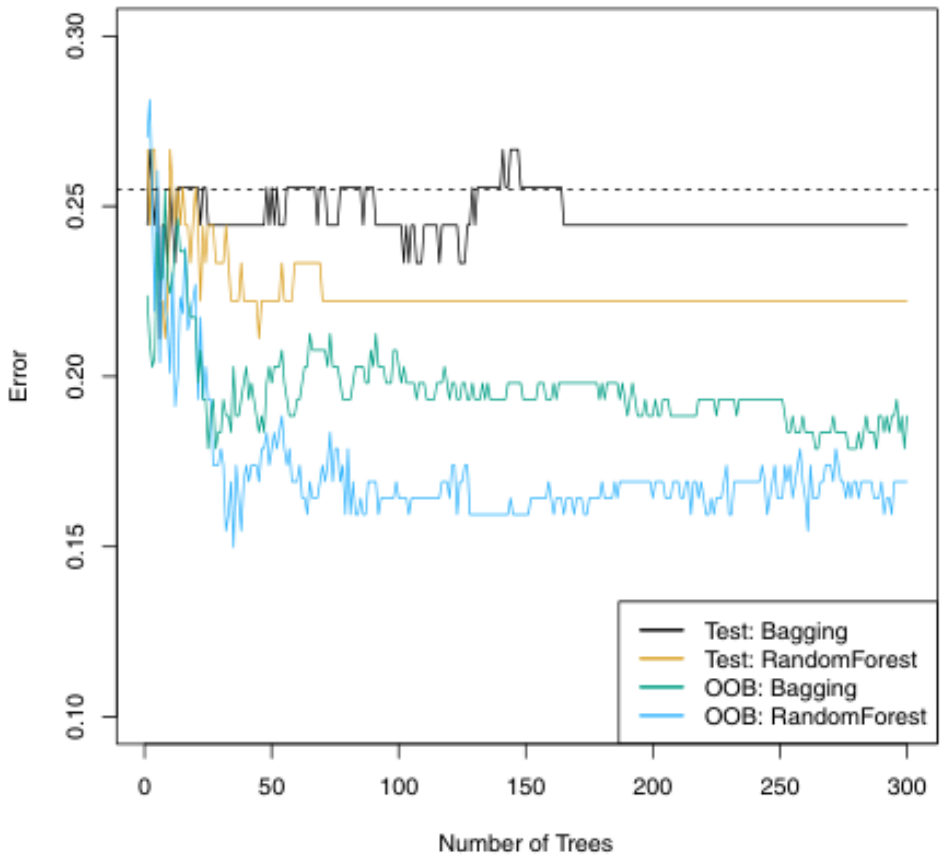


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification tree. The green and blue traces show the OOB error, which in this case is — by chance — considerably lower.*

Out-of-Bag Error and Cross-Validation

- **OOB and Cross-Validation:** When B is sufficiently large, Out-of-Bag (OOB) error is virtually equivalent to leave-one-out cross-validation error.
- **Convenience of OOB:** The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets, for which cross-validation would be computationally taxing.

Variable Importance in Bagging

- **Enhanced Accuracy with a Trade-Off:**
  - While bagging generally boosts the accuracy of prediction models compared to a single decision tree, it tends to obscure interpretability due to the ensemble nature of the model. The intricacies of multiple trees cannot be distilled into a single, simple representation, making it difficult to pinpoint the most influential predictors.
- **Tackling the Interpretability Challenge:**
  - Despite the complexity of bagged models, we can still gauge the significance of each predictor by assessing their influence on the model's accuracy.
  - For **regression trees**, this involves tracking the reduction in Residual Sum of Squares (RSS) attributable to each predictor across all trees in the ensemble. A substantial decrease in RSS signifies a high-importance predictor.
  - In the case of **classification trees**, we measure the decrease in the Gini index resulting from each predictor's splits. The greater the reduction, the more critical the predictor is deemed.

[Figure 8.9]Variable Importance Measure

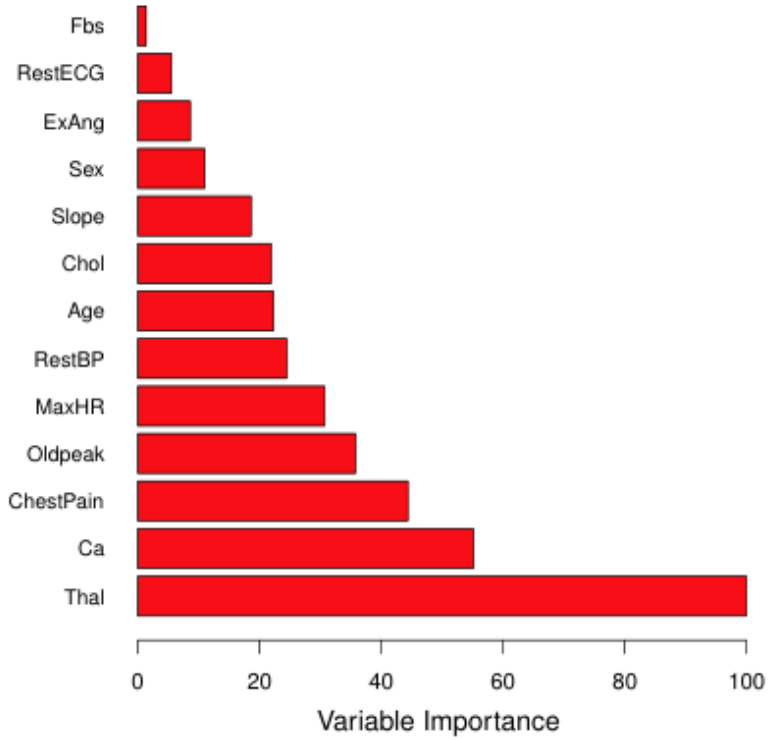


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Random Forest: An Improvement over Bagged Trees

- **Decorrelation of Trees:** Random forests provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.
  - 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...
- **Random Sample of Predictors:** A random sample 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 sample of m predictors is taken at each split, and typically we choose $m \approx \sqrt{p}$, that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).
- **Limited Predictor Consideration:** At each split in the tree, the algorithm is not even allowed to consider a majority of the available predictors.

Rationale for Random Forest: Part 1 - Bagging Limitations

- **Strong Predictor Scenario:** Suppose there is one very strong predictor in the data set, along with a number of other moderately strong predictors.
  - **Bagged Trees:** In the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other.
  - **Correlation Problem:** The predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities.
  - **Variance Reduction:** This means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

Rationale for Random Forest: Part 2 - Random Forest Solution

- **Random Forests Solution:** Random forests overcome the correlation problem:
  - **Increased Chance for Other Predictors:** On average $(p - m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance.
  - **Decorrelation:** Decorrelating the trees, thereby making the average of the resulting trees less variable and hence more reliable.
- **Difference between Bagging and Random Forests:** The main difference between bagging and random forests is the choice of predictor subset size m. If a random forest is built using $m = p$, then this amounts simply to bagging.

[Figure 8.10] Random Forest

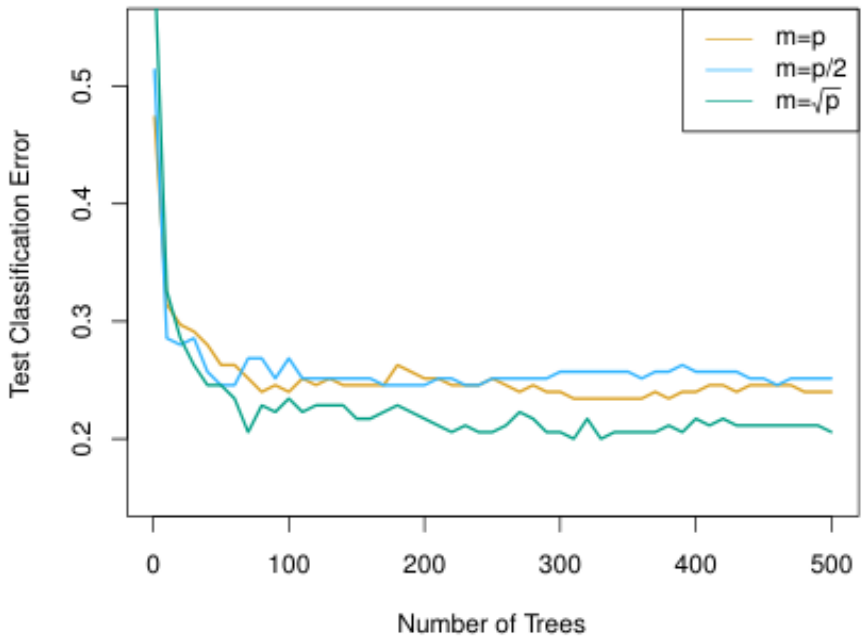


FIGURE 8.10. *Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree 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: An Overview I

- **Boosting and Decision Trees:**
  - Boosting, like bagging, is a versatile technique that improves the prediction power of models, including decision trees, for both regression and classification tasks.
- **Comparison with Bagging:**
  - Bagging generates multiple decision trees on bootstrapped datasets and aggregates them for the final model, with each tree trained independently.
  - In contrast, boosting builds trees sequentially, with each tree learning from the errors of its predecessors, refining the model iteratively.
- **Sequential Tree Growth:**
  - Unlike bagging, boosting does not use bootstrap sampling; it strategically modifies the original dataset based on the performance of earlier trees.
  - Trees are combined using a weighted vote to produce the final prediction, where weights reflect the accuracy of individual trees.

Boosting: An Overview II

- **Boosting in Practice:**
  - In the regression context, a series of trees $\hat{f}_1, ..., \hat{f}_B$ are developed, with each tree built to correct the residual errors of the sum of the preceding trees.

[Algorithm] Boosting

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Rationale for Boosting

- **Gradual Learning Process:**
  - Boosting refines its model incrementally, unlike a single large decision tree that may fit the data too closely, risking overfitting. This gradual approach aims to learn progressively and avoid overfitting.
- **Residual Corrections:**
  - At each step, boosting fits a small tree, not to the outcome Y itself, but to the current residualsthe differences between the observed and predicted values by the model so far.
  - This new tree is then incorporated into the existing model, updating the residuals and, consequently, the predictions.
- **Size of the Trees:**
  - The trees used in boosting are generally small, often just a few nodes, controlled by a parameter d, which keeps the learning steps modest and focused on correcting specific prediction errors.
- **Shrinkage for Better Generalization:**
  - The shrinkage parameter λ further moderates the learning rate, allowing the model to take smaller, more careful steps in learning. This parameter helps in introducing more diverse tree shapes to

Tuning Parameters I

- Statistical learning methods that learn gradually tend to perform well, and boosting is designed to learn slowly.
- In boosting, each tree's construction is significantly influenced by the previously grown trees, unlike in bagging where trees are grown independently.
- Classification with boosting is similar, albeit more complex; here, the specifics are not discussed. There are three primary tuning parameters in boosting:

Tuning Parameters II

1. **The number of trees** B: Boosting can potentially overfit if B is too large, although this tends to happen more gradually compared to other methods. Cross-validation is employed to determine the optimal value of B.
2. **The shrinkage parameter** λ: This small positive number regulates the learning speed of boosting. Common values are 0.01 or 0.001, and the optimal choice may be problem-specific. A very small λ often necessitates a larger B for effective performance.
3. **The number** d **of splits in each tree**: This determines the complexity of the boosted ensemble.
    - A value of $d = 1$ is frequently effective, where each tree is a "stump" with only a single split. Here, the boosted ensemble fits an additive model, since each term involves only one variable.

[Figure 8.11] Boosting

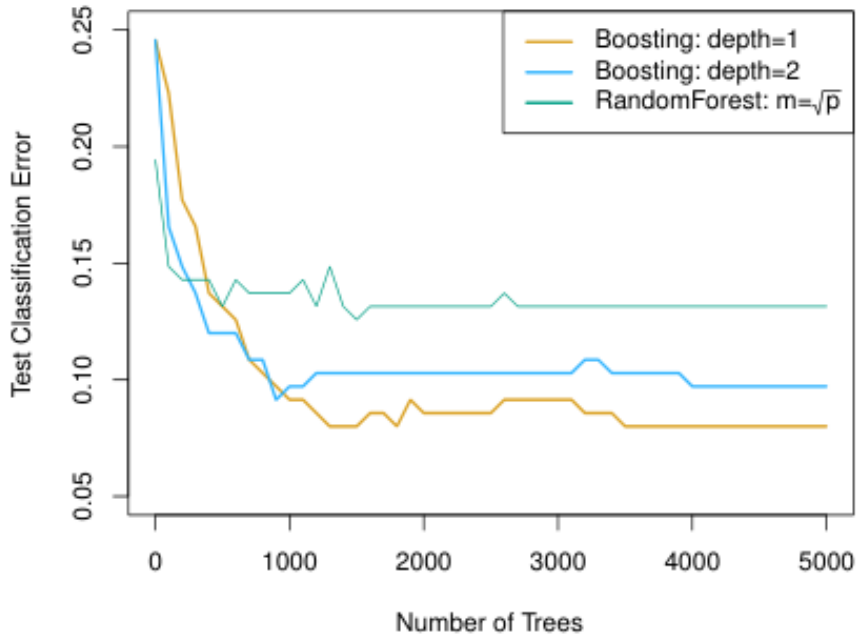


FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform 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 %.*

Bayesian Additive Regression Trees

- For simplicity, we focus on BART for regression (as opposed to classification).
  - BART is related to both bagging(random forests) and boosting approaches: each tree is constructed in a random manner as in bagging(random forests), and each tree tries to capture signal not yet accounted for by the current model, as in boosting. The main novelty in BART is the way in which new trees are generated.
- Notations:
  - let K denote the number of regression trees,
  - and B the number of iterations for which the *BART* algorithm will be run.
  - $\hat{f}_k^b(x)$ represents the prediction at x for the kth regression tree used in the bth iteration.

Algorithm 8.3 *Bayesian Additive Regression Trees*

1. Let $\hat{f}_1^1(x) = \hat{f}_2^1(x) = \cdots = \hat{f}_K^1(x) = \frac{1}{nK}\sum_{i=1}^n y_i$.
2. Compute $\hat{f}^1(x) = \sum_{k=1}^K \hat{f}_k^1(x) = \frac{1}{n}\sum_{i=1}^n y_i$.
3. For $b = 2, \ldots, B$:
   (a) For $k = 1, 2, \ldots, K$:
      i. For $i = 1, \ldots, n$, compute the current partial residual
      $$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i).$$
      ii. Fit a new tree, $\hat{f}_k^b(x)$, to r_i, by randomly perturbing the kth tree from the previous iteration, $\hat{f}_k^{b-1}(x)$. Perturbations that improve the fit are favored.
   (b) Compute $\hat{f}^b(x) = \sum_{k=1}^K \hat{f}_k^b(x)$.
4. Compute the mean after L burn-in samples,
$$\hat{f}(x) = \frac{1}{B-L}\sum_{b=L+1}^{B} \hat{f}^b(x).$$

Bayesian Additive Regression Trees

- At the end of each iteration, the K trees from the iteration are summed, $\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x)$, for $b = 1, \ldots, B$.
- In the first iteration of the BART algorithm, all trees are initialized to have a single terminal node, with $\hat{f}_k^1(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$.
- In subsequent iterations, BART updates each of the K trees, one at a time.
- In the b-th iteration, to update the k-th tree, we subtract from each response value the predictions from all but the k-th tree, to obtain a partial residual:
  - $$r_i = y_i - \sum_{k'<k} \hat{f}_{k'}^b(x_i) - \sum_{k'>k} \hat{f}_{k'}^{b-1}(x_i)$$
  - for the i-th observation, $i = 1, \ldots, n$.
  - Rather than fitting a new tree to this partial residual, BART randomly chooses a modification for the tree from the previous iteration $\hat{f}_{k'}^{b-1}(x_i)$ from a set of possible modifications, preferring ones that improve the fit to the partial residual:
    - We may alter the structure of the tree by adding or pruning branches.
    - We may adjust the prediction in each terminal node of the tree.

[Figure 8.12] BART algorithm

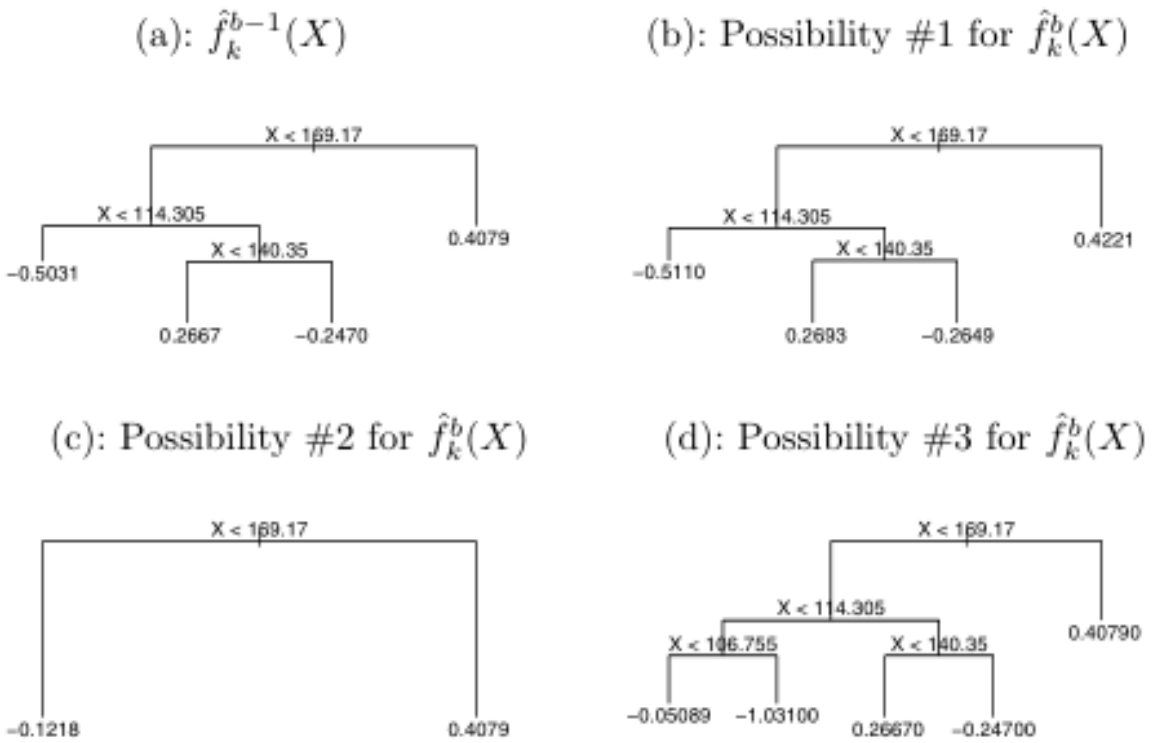


FIGURE 8.12. *A schematic of perturbed trees from the BART algorithm.* (a): *The kth tree at the* $(b-1)$*st iteration,* $\hat{f}_k^{b-1}(X)$*, is displayed. Panels (b)–(d) display three of many possibilities for* $\hat{f}_k^b(X)$*, given the form of* $\hat{f}_k^{b-1}(X)$*.* (b): *One possibility is that* $\hat{f}_k^b(X)$ *has the same structure as* $\hat{f}_k^{b-1}(X)$*, but with different predictions at the terminal nodes.* (c): *Another possibility is that* $\hat{f}_k^b(X)$ *results from pruning* $\hat{f}_k^{b-1}(X)$*.* (d): *Alternatively,* $\hat{f}_k^b(X)$ *may have more terminal nodes than* $\hat{f}_k^{b-1}(X)$*.*

Prediction

- Figure 8.12 illustrates examples of possible perturbations to a tree. The output of BART is a collection of prediction models:
- $$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \quad \text{for } b = 1, 2, \ldots, B$$
- We typically throw away the first few of these prediction models, since models obtained in the earlier iterations — known as the burn-in period— tend not to provide very good results.
- Let L denote the number of burn-in iterations; for instance, we might take $L = 200$. Then, to obtain a single prediction, we simply take the average after the burn-in iterations,

$$\hat{f}(x) = \frac{1}{B-L} \sum_{b=L+1}^{B} \hat{f}^b(x)$$

- the percentiles of $\hat{f}^{L+1}(x), \ldots, \hat{f}^B(x)$ provide a measure of uncertainty in the final prediction.

[Illustration] BART

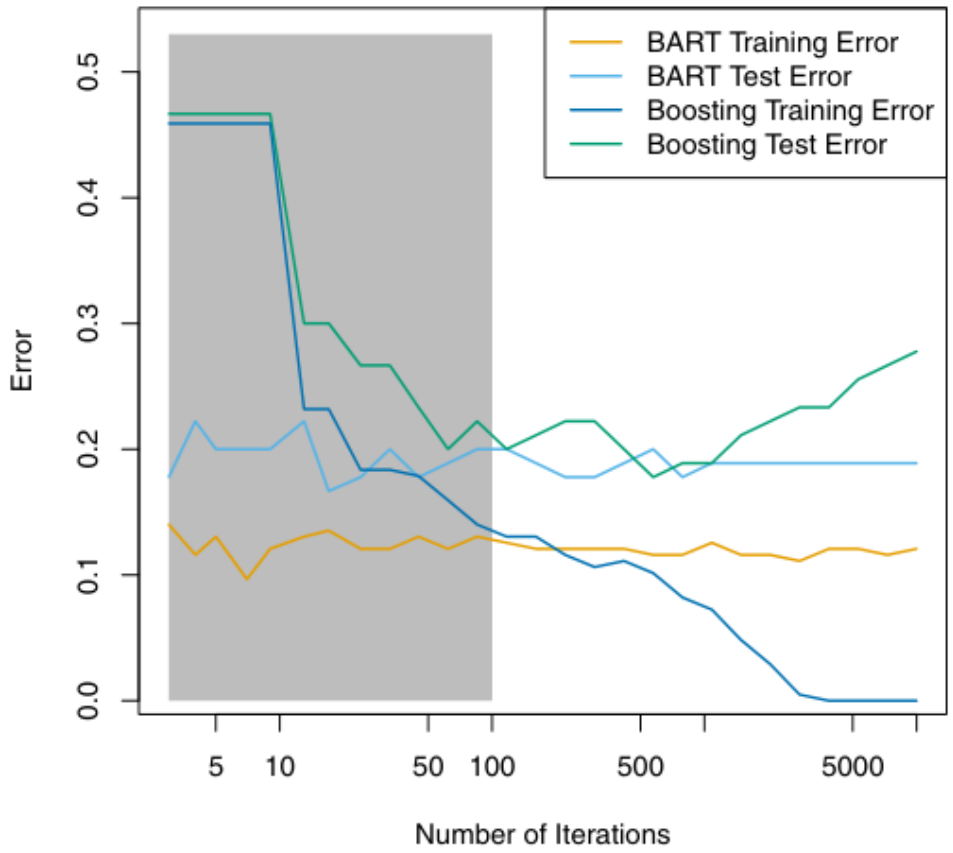


FIGURE 8.13. *BART and boosting results for the* `Heart` *data. Both training and test errors are displayed. After a burn-in period of* 100 *iterations (shown in gray), the error rates for BART settle down. Boosting begins to overfit after a few hundred iterations.*

Comparison of BART and Boosting

BART Methodology

- Randomly modifies a tree to fit residuals.
- Draws a new tree from a posterior distribution.
- Uses MCMC strategy for implementation.

BART (Heart Dataset)

- Utilizes $K = 200$ trees.
- Errors stabilize after burn-in phase.
- Slight discrepancy between training and test errors.
- Effectively mitigates overfitting.

Boost Algorithm

- Test error approximates BART initially.
- Test error rises with more iterations.
- Training error diminishes, signaling overfitting.

Summary of Tree Ensemble Methods

- **Bagging**: Trees grown independently on random samples. *Trees are similar*, may lead to local optima.
- **Random Forests**: Trees grown independently with random feature subsets for splits. *Decorrelates trees*, enhancing model exploration.
- **Boosting**: Uses original data, grows trees successively. *Slow learning*, fitting new trees to remaining signal, with shrinkage.
- **BART**: Uses original data, grows trees successively. *Trees are perturbed* to avoid local minima, ensuring thorough exploration.

Lab: Bootstrap(ISLR section 5.3.4, pp. 216)

- Data: The Auto Data

Lab: Decision Trees

- Dataset: The Carseats Data, The Boston Data
  - Fitting Classificaiton Trees
  - Fitting Regression Trees
  - Bagging and Random Forests
  - Boosting
  - Bayesian Additive Regression Trees

For Further Reading I

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: