Tree-based Methods

Jasmine. Hao[1]

[1]University of Hong Kong

ECON 6083: Machine Learning

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 labelled 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.

Hitter's Dataset Overview

Context:

- Part of the R-package ISLR.

Content:

- Originally from StatLib library, Carnegie Mellon University.
- Salary data from Sports Illustrated (April 20, 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).

Tree

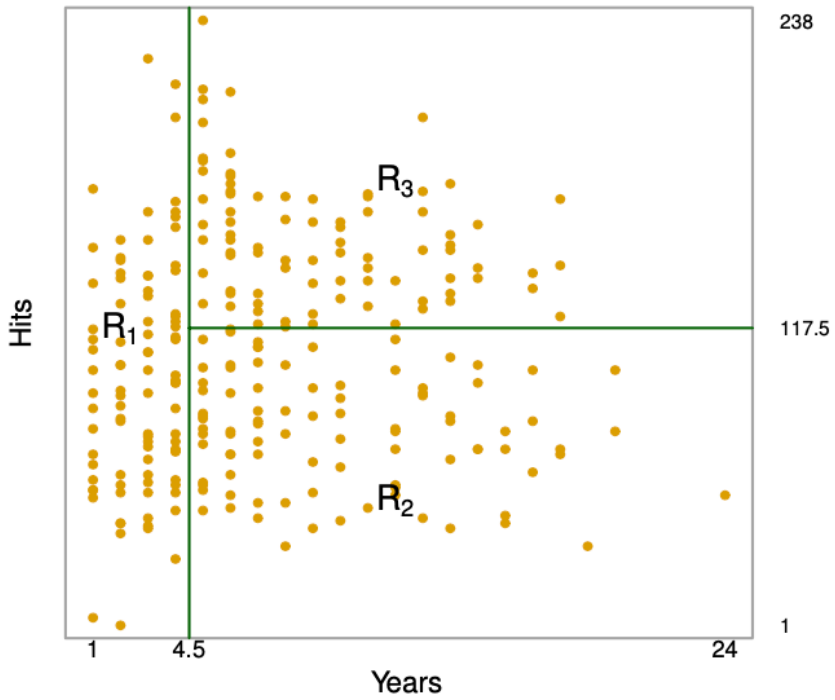
Tree Construction Example

- **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 branch emanating from that split, and the right branch corresponds to $X_j \geq t_k$.
- **Tree Split:** For example, 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:
  - $R1 = \{X | Years < 4.5\}$,
  - $R2 = \{X | Years >= 4.5, Hits < 117.5\}$,
  - $R3 = \{X | Years >= 4.5, Hits >= 117.5\}$.

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

- We use a top-down, greedy approach known as recursive binary splitting.
- **Top-down approach:**
  - Begins at the top of the tree: all observations belong to a single region.
  - Successively splits the predictor space, creating 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
  - **Not looking ahead** for a split that will lead to a better tree in some future step.

Recursive Binary Splitting

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

A five-region example

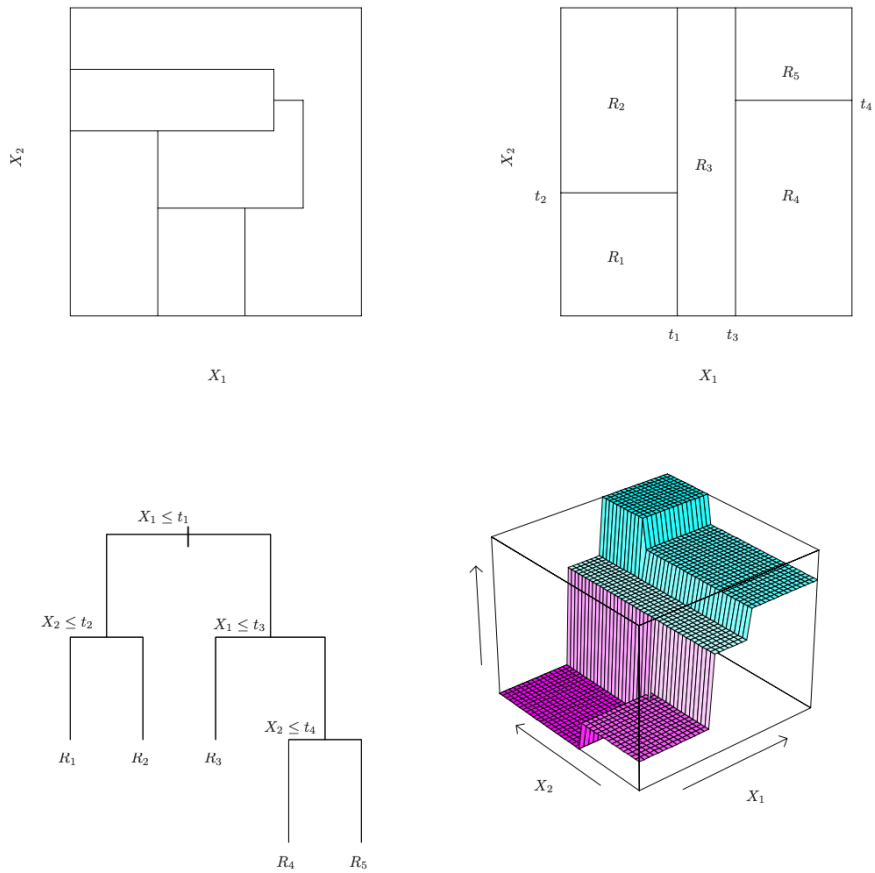


FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A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

Tree Pruning

- **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

- **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,
- whole sequence of subtrees as a function of α is easy.
  - We can select a value of α using a validation set or by 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.

Tree Pruning

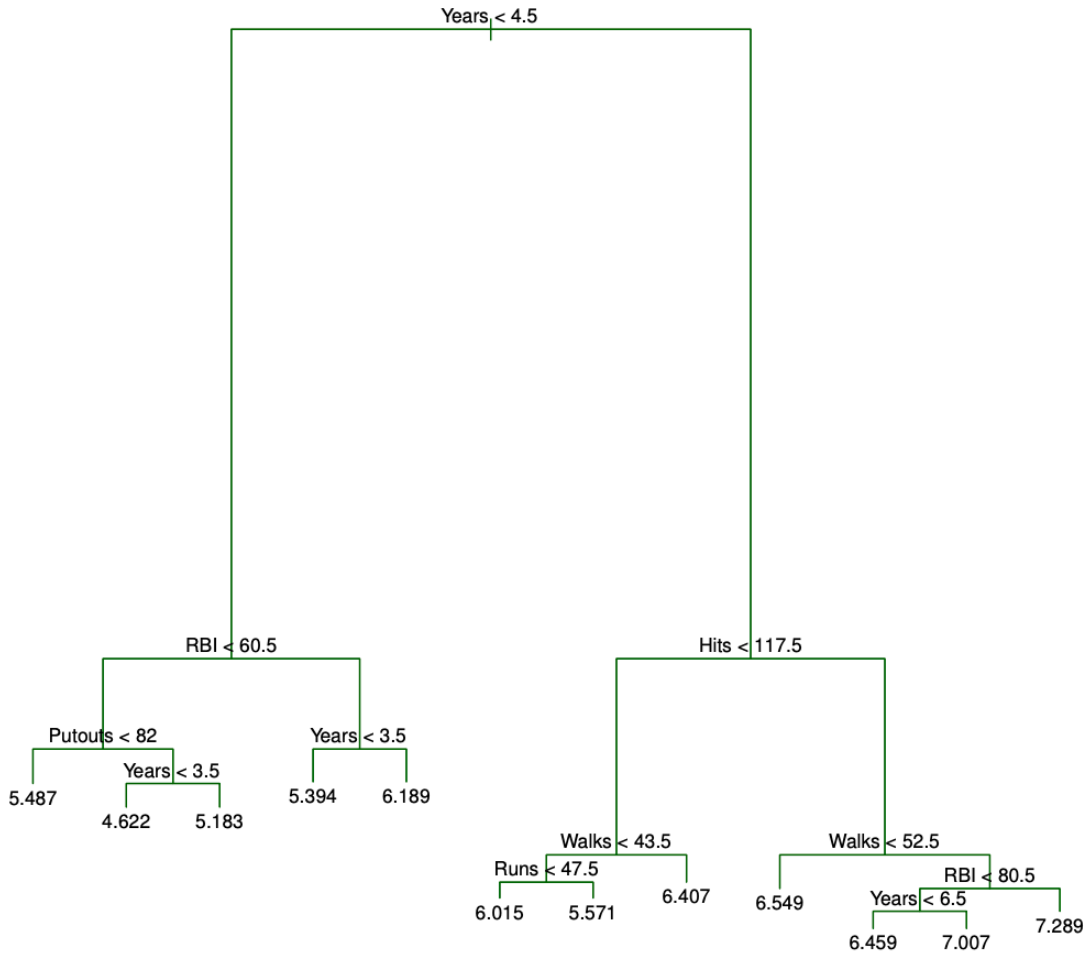


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

Regression Tree

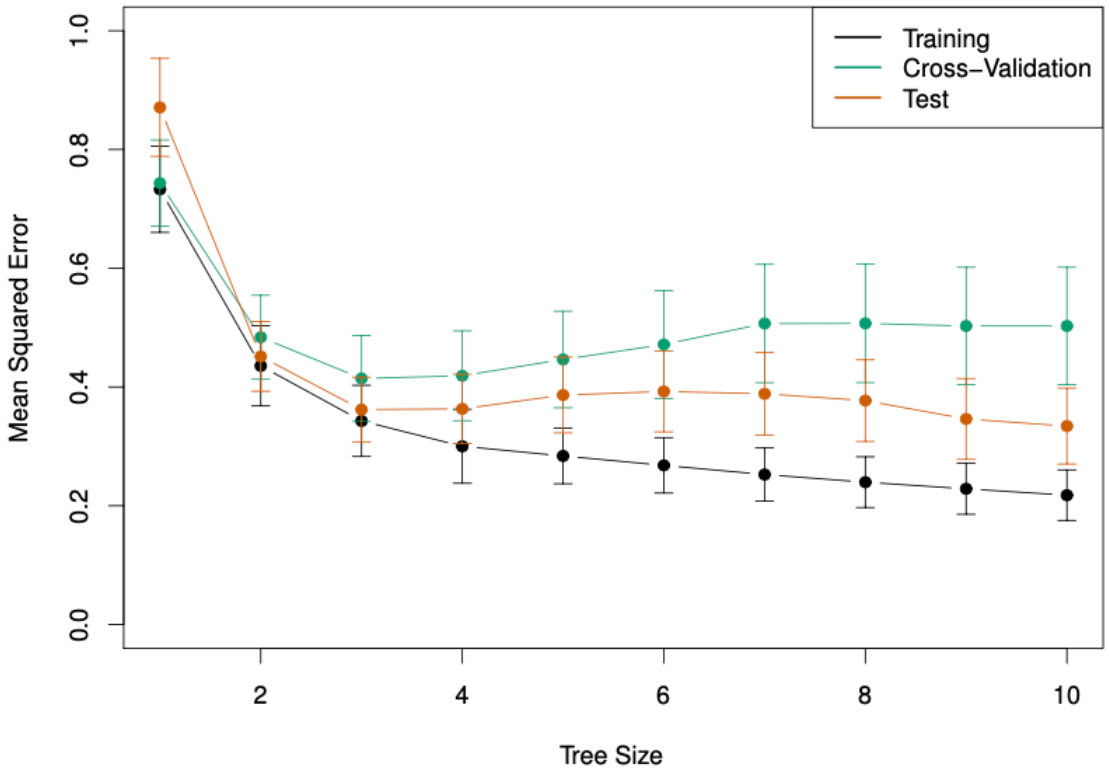


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

Regression Tree Fitting and Pruning I

- **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.

Regression Tree Fitting and Pruning II

- **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.
- **Minimum Error:** The CV error and the test error both show a dip for a three-node tree, with the test error achieving its minimum for a ten-node tree.
- **Pruned Tree:** The pruned tree with three terminal nodes is displayed in Figure 8.1.

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.

Measures for Classification Trees

- **Growing a Classification Tree:** Similar to regression trees, using recursive binary splitting.
- **Classification Error Rate:** $E = 1 - \max_k(\hat{p}_{km})$. Used when RSS is not applicable.
- **Gini Index:** $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$. Measures total variance across classes, used to evaluate splits.
- **Entropy:** $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk})$. Another measure for node purity and evaluating splits.

Outline

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 the response is linear.
- The 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.

[Figure 8.7] Tree v.s. Linear

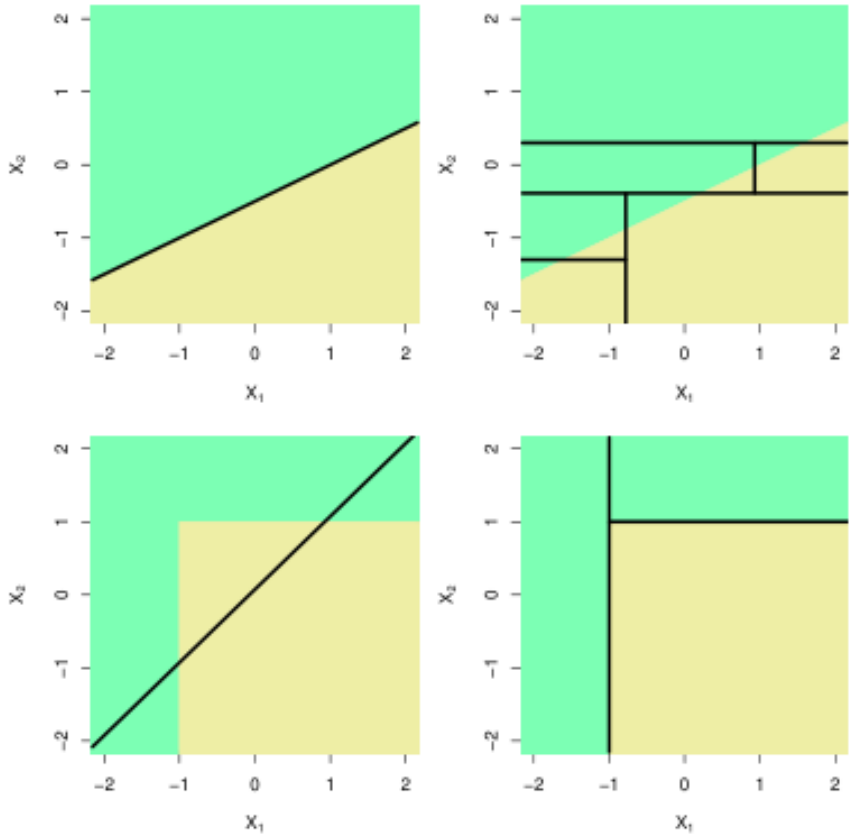


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: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

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.

Majority Vote in Bagging

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.
- **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.
- **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 observations n. 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, since the response for each observation is predicted using only the trees that did not fit using that observation.

Out-of-Bag Error Estimation

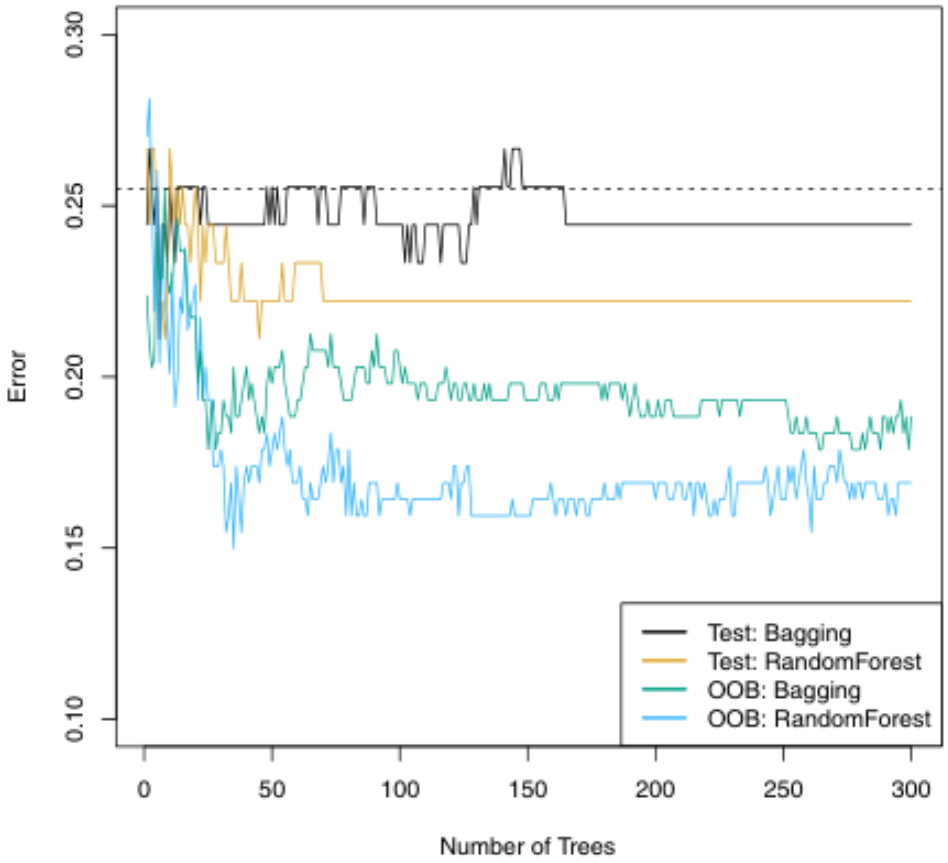


FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification 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.

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.

Random Forest

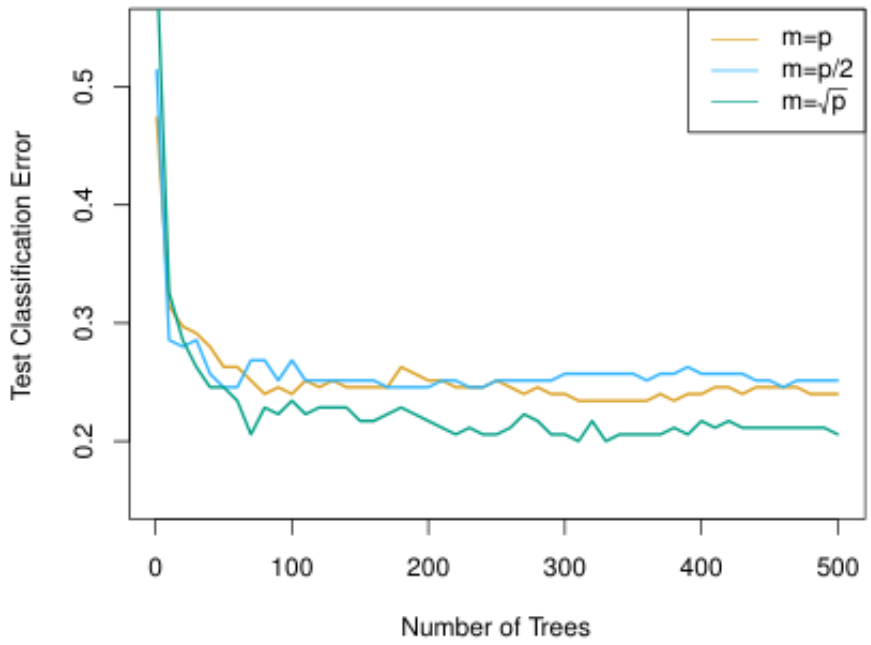


FIGURE 8.10. *Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree 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 enhances models, like decision trees, for regression and classification tasks.
- **Comparison with Bagging:**
  - Bagging creates multiple independent trees from bootstrapped datasets and aggregates them.
  - **Boosting** builds trees **sequentially**, learning from the errors of predecessors, for iterative refinement.
- **Sequential Tree Growth:**
  - Boosting modifies the dataset based on earlier tree performance, **without bootstrap sampling**.
  - Trees are combined using a **weighted vote**, where weights reflect tree accuracy.
- **Boosting in Practice:**
  - In regression, a series of trees $\hat{f}_1, ..., \hat{f}_B$ are developed, each correcting the residual errors 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 incrementally refines its model, aiming for gradual learning to **avoid overfitting**, unlike single large decision trees.
- **Residual Corrections:**
  - Each step fits a small tree to the **current residuals**, updating the model with corrected predictions.
- **Size of the Trees:**
  - The trees are small (few nodes), with size controlled by parameter d, focusing on correcting specific errors.
- **Shrinkage for Better Generalization:**
  - The **shrinkage parameter** λ moderates the learning rate, introducing diverse tree shapes for improved generalization.

Tuning Parameters

- Boosting is designed to learn slowly, which is a characteristic of statistical learning methods that tend to perform well.
- In boosting, the construction of each tree is significantly influenced by the previously grown trees, unlike in bagging where trees are grown independently.
- The three primary tuning parameters in boosting are:
  1. **Number of Trees (B):** The increase can be too large if B is too large, but this happens gradually. Cross-validation is used to find the optimal B.
  2. **Shrinkage Parameter (λ):**
     1. A small positive number that regulates learning speed.
     2. The common values are 0.01 or 0.001, and the optimal choice may be specific to the problem.
     3. A smaller λ often requires a larger B for effectiveness.
  3. **Number of Splits (d) in Each Tree:** Determines the complexity of the boosted ensemble.
     1. A value of $d = 1$ is often effective.
     2. This results in an additive model where each term involves only one variable.

Boosting

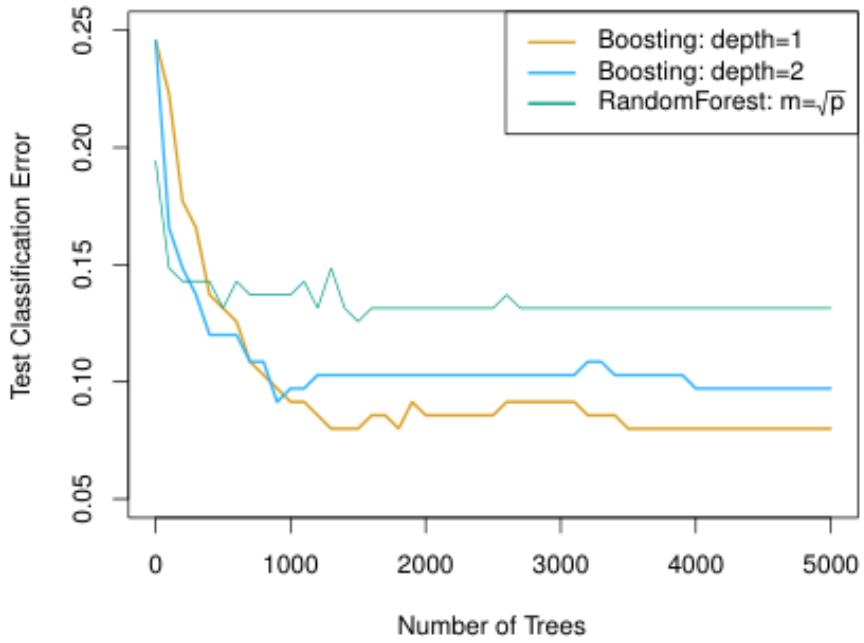


FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform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

Bayesian Additive Regression Trees - Overview

- **Sum of 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 $b = 1, \ldots, B$.
- **Initialization:** In the first iteration of the BART algorithm, all trees are initialized to have a single terminal node, with $\hat{f}_k^1(x) = \frac{1}{nK} \sum_{i=1}^{n} y_i$.
- **Subsequent Iterations:** BART updates each of the K trees, one at a time.

Bayesian Additive Regression Trees - Iterative Updates

- **Updating Trees:** In the b-th iteration, to update the k-th tree:
  - Compute the partial residual for each observation $i = 1, \ldots, n$:

$$r_i = y_i - \sum_{k' < k} \hat{f}_{k'}^{b}(x_i) - \sum_{k' > k} \hat{f}_{k'}^{b-1}(x_i)$$

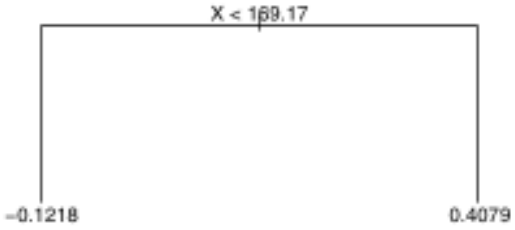
  - Rather than fitting a new tree, BART randomly chooses a modification for the tree from the previous iteration, preferring ones that improve the fit to the partial residual:
    - ★ Alter the structure of the tree by adding or pruning branches.
    - ★ Adjust the prediction in each terminal node of the tree.

BART algorithm

(a): $\hat{f}_k^{b-1}(X)$

(b): Possibility #1 for $\hat{f}_k^b(X)$

(c): Possibility #2 for $\hat{f}_k^b(X)$



(d): Possibility #3 for $\hat{f}_k^b(X)$

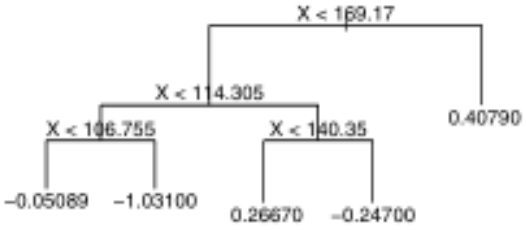


FIGURE 8.12. *A schematic of perturbed trees from the BART algorithm.* (a): *The kth tree at the* $(b-1)$*st iteration,* $\hat{f}_k^{b-1}(X)$*, is displayed. Panels (b)–(d) display three of many possibilities for* $\hat{f}_k^b(X)$*, given the form of* $\hat{f}_k^{b-1}(X)$*.* (b): *One possibility is 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 in BART

- **Model Output:** The output is a collection of prediction models:

$$\hat{f}^b(x) = \sum_{k=1}^{K} \hat{f}_k^b(x), \quad \text{for } b = 1, 2, \ldots, B$$

- **Burn-in Period:** Discard initial models (L iterations) to avoid poor early results.
- **Final Prediction:** Average models post burn-in for a single prediction:

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

- **Uncertainty Measure:** Use percentiles of $\hat{f}^{L+1}(x), ..., \hat{f}^B(x)$ to gauge prediction uncertainty.

[Illustration] BART

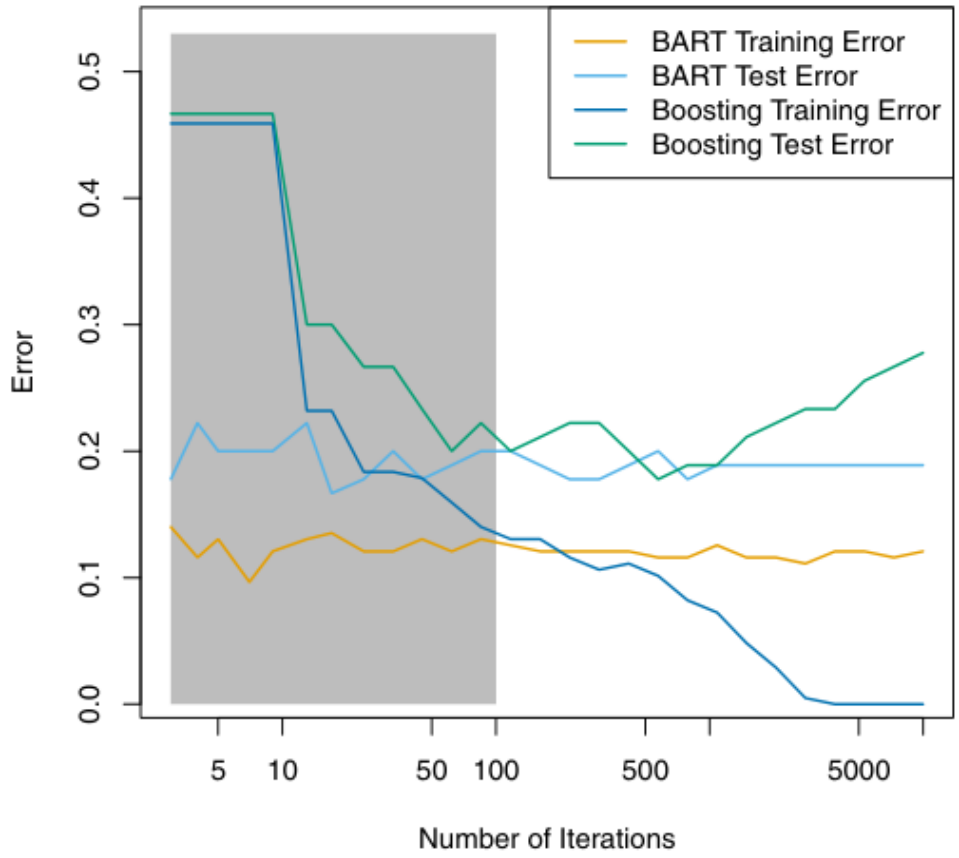


FIGURE 8.13. *BART and boosting results for the* `Heart` *data. Both training and test errors are displayed. After a burn-in period of* 100 *iterations (shown in gray), the error rates for BART settle down. Boosting begins to overfit after a few hundred iterations.*

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.

For Further Reading I

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

Susan Athey and Guido Imbens.
Recursive partitioning for heterogeneous causal effects.
Proceedings of the National Academy of Sciences, 113(27):7353-7360, 2016.

Justin Grimmer, Solomon Messing, and Sean J. Westwood.
Estimating heterogeneous treatment effects and the effects of heterogeneous treatments with ensemble methods.
Political Analysis, 25(4):413–434, 2017.

Sridhar Narayanan, Kirthi Kalyanam, and others.
Behavioral Targeting, Machine Learning and Regression Discontinuity Designs.
Tech Report, 2020.