Tree-Based Methods (Chapter 8)

Setup: Regression or classification of a response Y based on p predictors $X_1, \ldots, X_p$ denoted by X

Data: (Y_i, X_i), $i = 1, \ldots, n$ from n independent subjects

We will learn some **decision-tree** methods. In particular:

- **Classification and Regression Trees (CART)** algorithm for growing a *tree* — dividing the predictor space into a set of rectangles (or boxes) and fitting a constant in each box.
- **Bagging, random forests, and boosting** procedures that grow multiple trees and combine them to get a single consensus tree. These methods are competitive with other methods we have seen.

We will begin with **regression trees**.

Some tree terminology: The tree is drawn *upside down* so that the *leaves* are at the bottom and the *root* is at the top.

- **Internal nodes**: points along the tree where the predictor space is split
- **Branches**: segments of the trees that connect the nodes
- **Terminal nodes**: leaves — each terminal node is associated with a region (subset) of the predictor space and together they form a partition of the predictor space
- **Binary tree**: each internal node is split into two branches
- **Size of a tree**: # terminal nodes
- **Size of a terminal node**: # of training observations falling in the corresponding region of the predictor space

Ex: `Hitters data`: These consist of data on `Salary` of $n = 263$ baseball players (after omitting those with missing observations) and a total of 19 predictors but we will use only 9 features, including `Years` and `Hits`. The goal is to predict `Salary`. Since `Salary` is right-skewed, we will work with log(`Salary`) as the response.

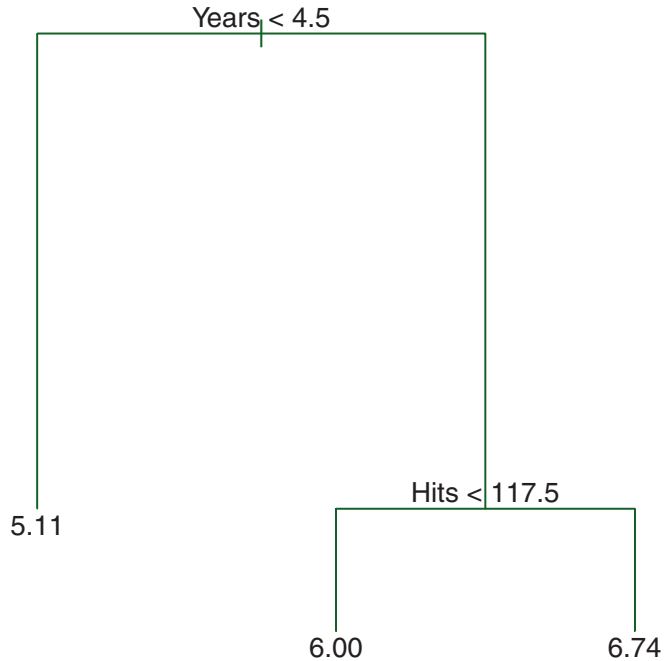

FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form* $X_j < t_k$*) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to* $X_j \geq t_k$*. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

Source: ISL

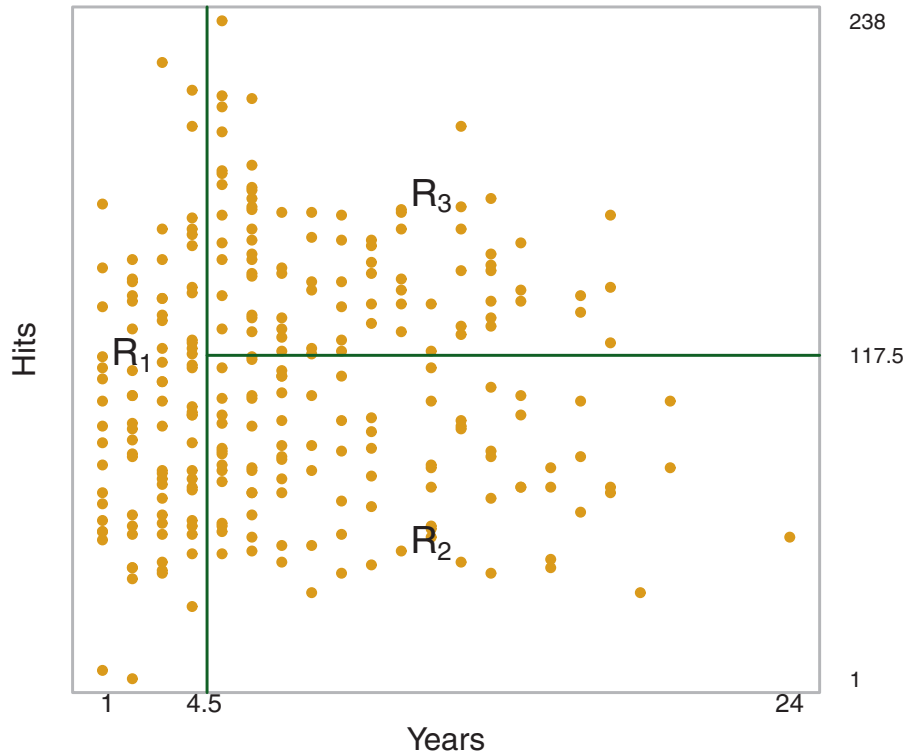

FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

- Sides of the rectangles will be parallel to the axes.

Source: ISL

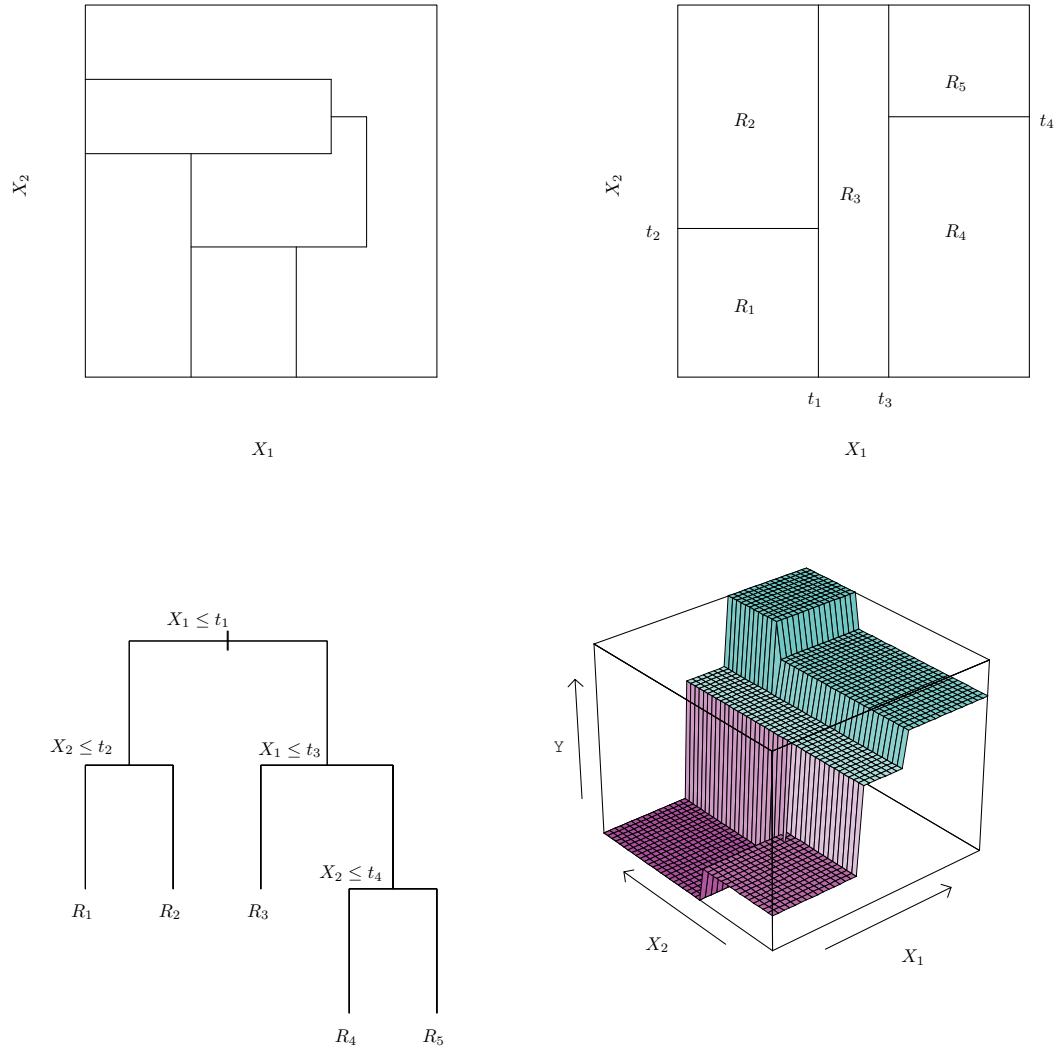

FIGURE 8.3. Top Left: *A partition of two-dimensional feature space that could not result from recursive binary splitting.* Top Right: *The output of recursive binary splitting on a two-dimensional example.* Bottom Left: *A tree corresponding to the partition in the top right panel.* Bottom Right: *A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

Source: ISL

Growing a Regression Tree

Partition the predictor space into J distinct and non-overlapping regions, $R_1, \ldots, R_J$, each corresponding to a terminal node $j = 1, \ldots, J$. Model the response as a constant c_j in each R_j. This gives the mean function

$$f(X) = \sum_{j=1}^{J} c_j I(X \in R_j).$$

Criterion: Minimize wrt R_j and c_j, $j = 1, \ldots, J$,

$$\text{RSS} = \sum_{i=1}^{n} (Y_i - f(X_j))^2 = \sum_{j=1}^{J} \sum_{i:X_i \in R_j} (Y_i - c_j)^2 = \sum_{j=1}^{J} n_j Q_j,$$

where $n_j = \#$ observations in R_j, and

$$Q_j = \frac{1}{n_j} \sum_{i:X_i \in R_j} (Y_i - c_j)^2$$

is a measure of *impurity* of node j.

For given R_j, the minimum wrt c_j occurs when

$$\hat{c}_j = \text{ave}\{Y_i | X_i \in R_j\},$$

implying that Q_j is simply the MSE of node j. Thus, the problem reduces to that of finding the best regions. This is not computationally feasible since it involves partitioning the predictor space into all possible regions which can have any shape. So we restrict attention to the regions that are **rectangles (or boxes)**. However, even this is not computationally feasible. Therefore, we take a **recursive binary splitting** approach that:

- is **top-down** — begins at the top of the tree where all observations belong to a single region,
- successively splits the predictor space into two regions — each split makes two new branches down the tree, and
- is **greedy** — makes the split that is the best at that particular step rather than looking ahead and picking a split that may lead to a better tree in some future step.

Fitting algorithm: Start with all the data. Consider a splitting variable m and split point s, and define the pair of half-planes

$$R_1(m,s) = \{X | X_m < s\} \text{ and } R_2(m,s) = \{X | X_m \geq s\}.$$

Let the constants c_1 and c_2 be the predictions in the regions R_1 and R_2, respectively. Find the values of m, s, c_1, and c_2 that minimize the RSS, i.e., solve

$$\min_{m,s} \left\{ \min_{c_1,c_2} \left(\sum_{i:x_i \in R_1(m,s)} (Y_i - c_1)^2 + \sum_{i:x_i \in R_2(m,s)} (Y_i - c_2)^2 \right) \right\}.$$

For any choice of m and s, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}\{Y_i | X_i \in R_1(m,s)\} \text{ and } \hat{c}_2 = \text{ave}\{Y_i | X_i \in R_2(m,s)\}.$$

The outer minimization can also be done rather quickly.

The resulting m and s provide the best split at the top of the tree. Next, we repeat the process, looking for the best split in each of the two regions identified in the previous step. This process continues until a stopping criterion is reached, e.g., size of terminal nodes fall below 5, e.g. Once the regions $R_1, \ldots, R_J$ have been created, the predicted response $\hat{Y}$ for a given observation X is simply the average of the training observations in the region in which X falls, i.e.,

$$\hat{Y} = \text{ave}\{Y_i | X \in R_j\}.$$

Issue: The resulting tree may be too complex — too many splits (i.e., too many regions), overfitting the data.

Alternative 1: Grow the tree only so long as the decrease in RSS due to each split exceeds some (high) threshold. Gives smaller trees, but is short-sighted since a seemingly worthless split early on might be followed by a good split later on.

Alternative 2 (Cost-complexity pruning): Grow a very large tree T_0 and prune it to obtain a subtree $T \subseteq T_0$. Pruning means collapsing some internal nodes. Let $|T|$ denote the size of the subtree T — # of terminal nodes (or leaves) in T, and $\alpha\,(\geq 0)$ be a **tuning parameter**. For a given α, obtain the subtree that minimizes the objective function:

$$\text{RSS} + \alpha|T|.$$

- Similar to penalized RSS criterion used for regularization
- $\alpha = 0 \implies T = T_0$ (i.e., no pruning)
- $\alpha = \infty \implies |T| = 1$ (i.e., only one region with all data)
- Larger $\alpha \implies$ smaller subtree
- Have one subtree for each value of α — T_α
- Use the **weakest link pruning** algorithm to obtain T_α
- Choose α by cross-validation

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.

- **Splitting a qualitative predictor:** Assign one or more categories to one branch and the remaining to the other.

Hitters data

- Use nine features
- Take a validation set approach — equal split in training and test sets
- Grow a large unpruned true on the training set
- Apply cost-complexity pruning — vary α to create subtrees with different sizes (i.e., the number of terminal nodes)
- Perform six-fold CV (six is a factor of 132) to estimate the CV error of the subtrees as a function of α
- Compute the test error as a function of α
- Plot results as a function of size of the subtree instead of α

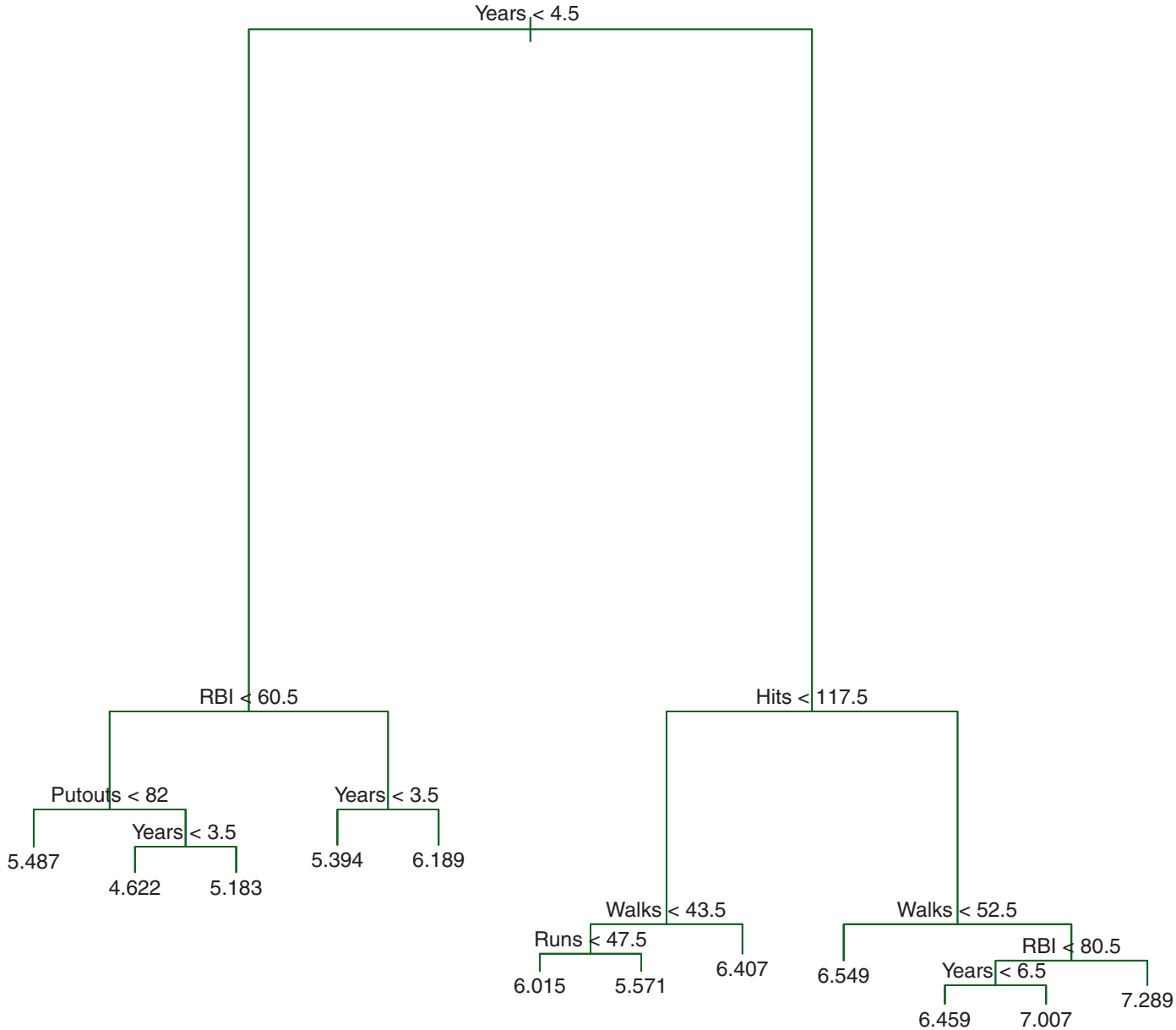

FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

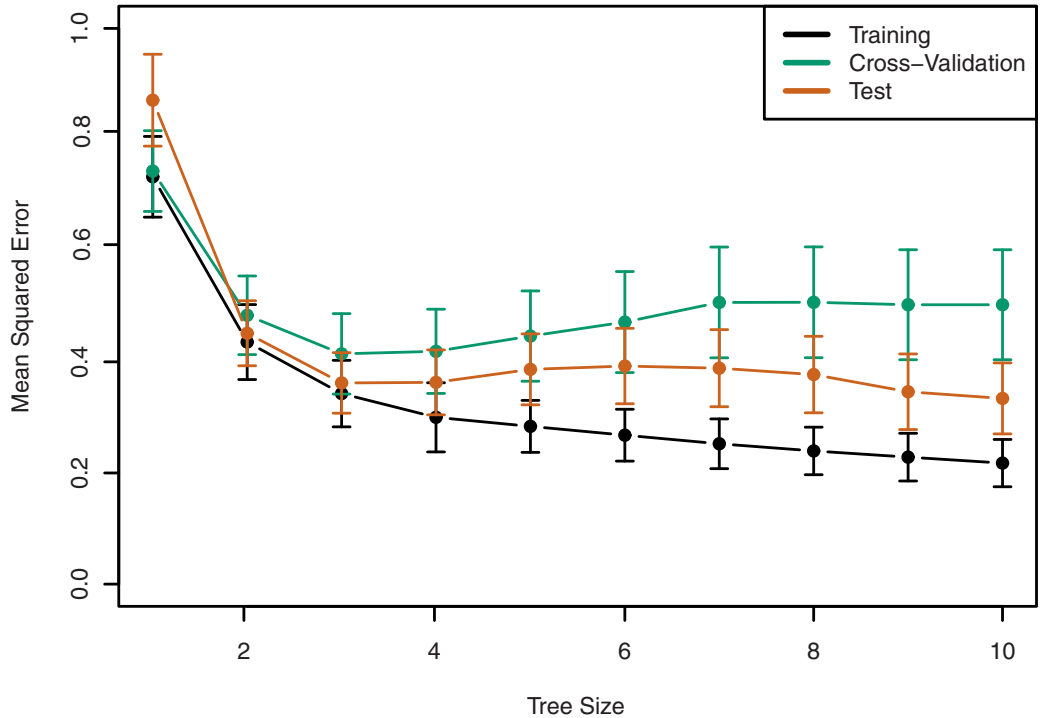

FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

- CV error is minimized for a 3-node tree
- Test error is minimized for a 10-node tree

Source: ISL

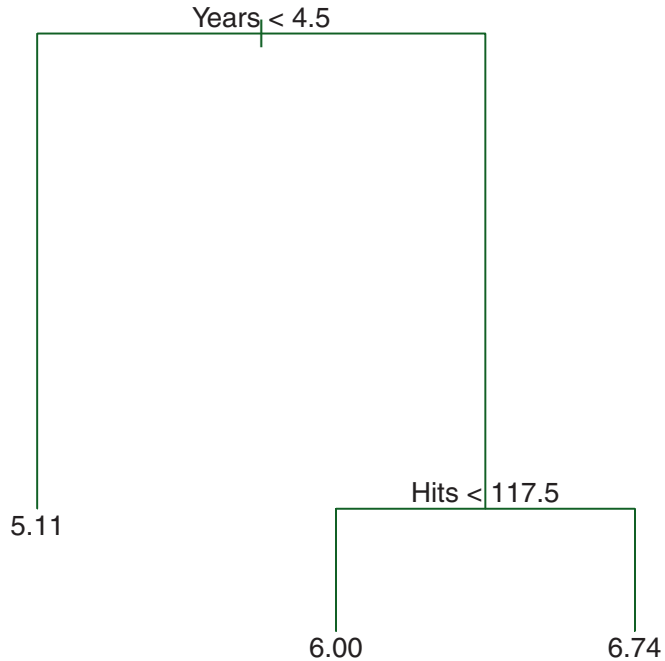

FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form* $X_j < t_k$*) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to* $X_j \geq t_k$*. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to* `Years<4.5`*, and the right-hand branch corresponds to* `Years>=4.5`*. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.*

Source: ISL

Growing a Classification Tree

Setup: Same as before but Y is a **qualitative response** with K classes, indexed by $k = 1, \ldots, K$.

Recall for a regression setup: $\text{RSS} = \sum_{j=1}^{J} n_j Q_j$, where the node impurity measure Q_j is the MSE in node j. This Q_j was used for both growing a tree and pruning it.

Class probability: For a terminal node j, representing the region R_j with n_j observations, let

$$\hat{p}_{jk} = \frac{1}{n_j} \sum_{i:x_i \in R_j} I(Y_i = k)$$

be the proportion of class k observations in region R_j.

Class prediction: The observations in region R_j are classified to the **majority class**, i.e., the class k for which $\hat{p}_{jk}$ is maximum.

Node impurity measures for classification: For the jth terminal node, $j = 1, \ldots, J$, we have:

Misclassification error: $1 - \max_k \hat{p}_{jk}$ — the proportion of observations in R_j that do not belong to the majority class

Gini index: $\sum_{k \neq l} \hat{p}_{jk}\hat{p}_{jl} = \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$

Cross-entropy (or deviance): $-\sum_{k=1}^{K} \hat{p}_{jk} \log(\hat{p}_{jk})$

- The measures are ≥ 0 and take values near zero when one of the $\hat{p}_{jk}$ is close to one and the others are close to zero
- A near-zero value indicates that the node predominantly consists of observations from a single class
- The last two tend to be similar and are more sensitive than the first one to changes in tree structures.
- Use either Gini index or deviance for tree growing and misclassification error for pruning (if the final tree will be used for prediction)

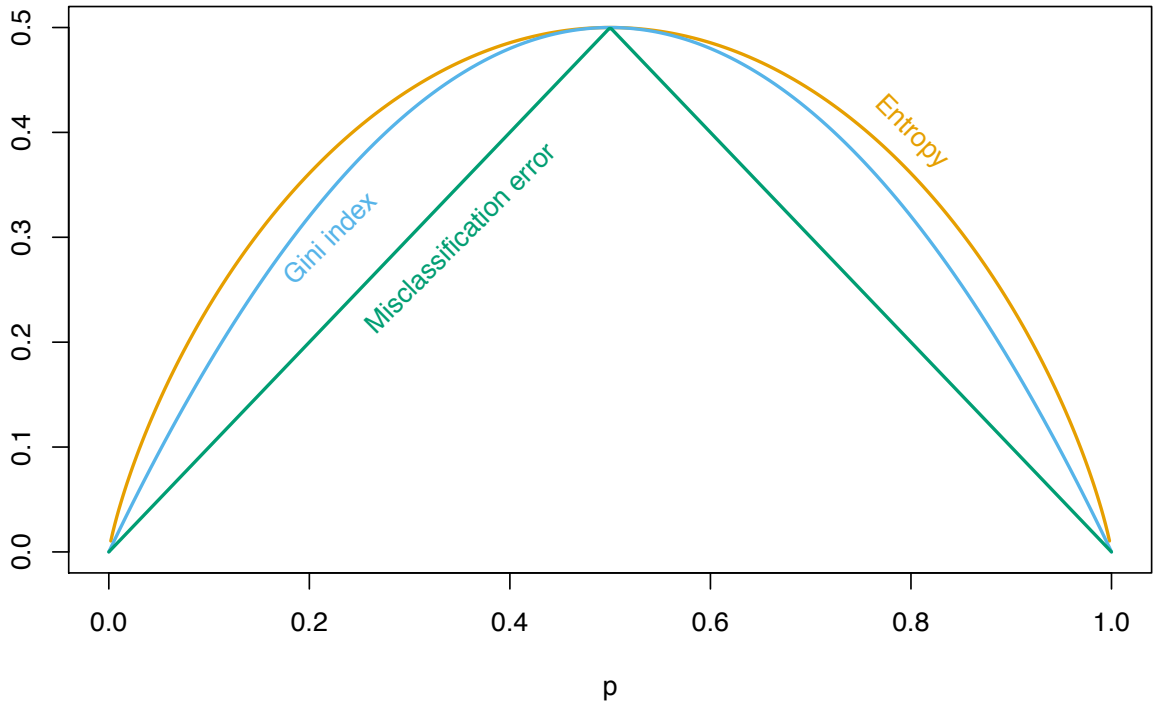

FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

Source: ESLII

Ex: `Heart data:` The data contain a binary outcome `HD` — indicator of heart disease (`Yes` or `No`) for 303 patients who presented with chest pain. There are 13 predictors, including `Age`, `Sex`, and `Chol`. Cross-validation results in a tree with six terminal nodes.

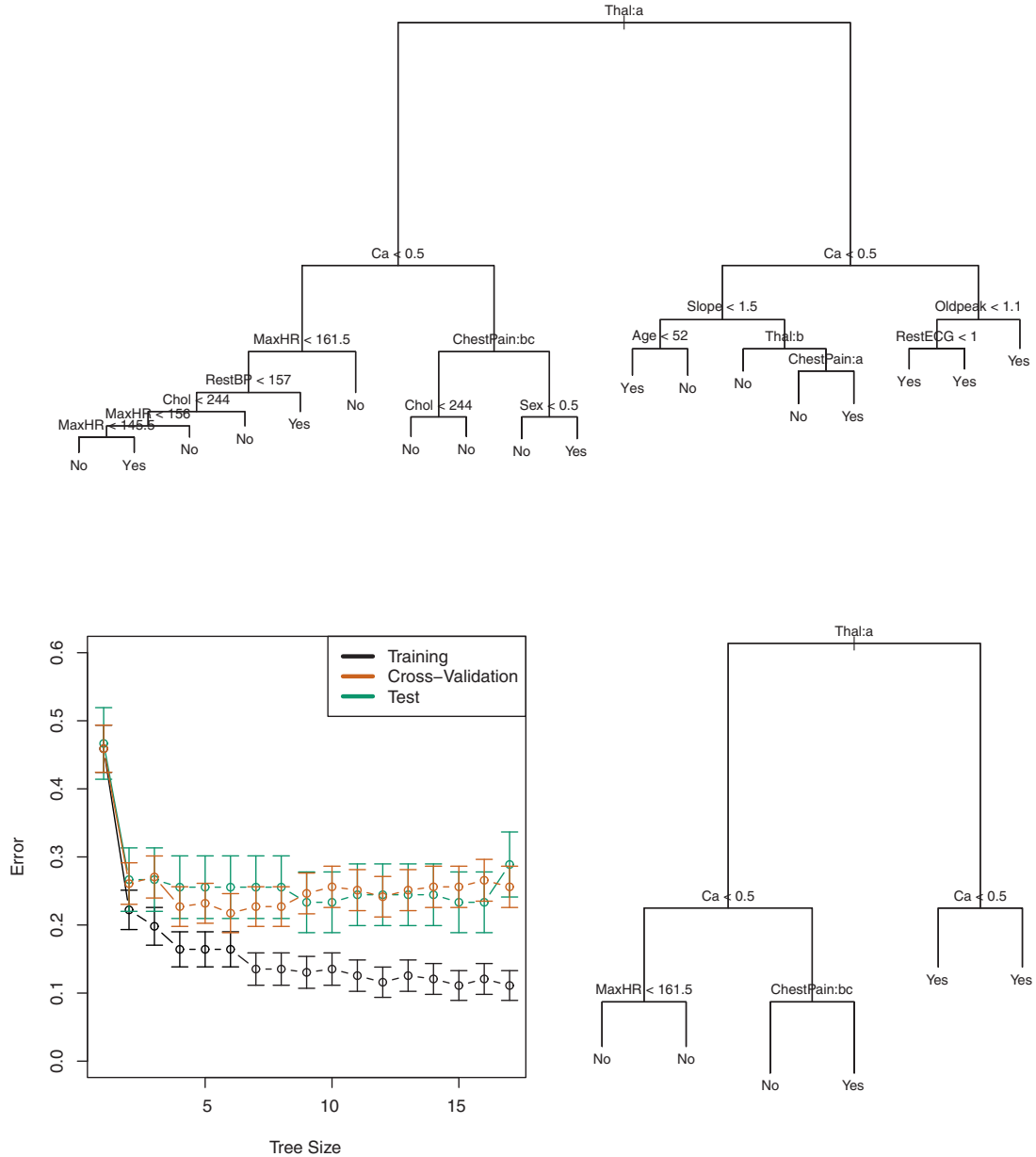

FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross -validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*

Issue: Some terminal nodes in the tree have the **same** predicted value, see, e.g., the split `RestECG < 1` on the bottom right of the unpruned tree. Thus, regardless of whether or not `RestECG` < 1, the predicted outcome is the same — `Yes`.

Q: If this split is not performed, will it change the misclassification error rate? **A:** No

Q: Then, why is the split performed at all? **A:** It leads to greater node purity. In particular, we can see that:

- all 9 of the observations corresponding to the right-hand leaf have the response `Yes`. If a test observation falls here, we can be pretty certain of the outcome.
- 7/11 of the observations corresponding to the left-hand leaf have the response `Yes`. If a test observation falls here, the response is probably `Yes`, but we are less certain.

The split is performed as it improves Gini index or deviance, which is what we generally use for growing a tree.

Decision Trees vs Linear Models

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

Trees: $f(X) = \sum_{j=1}^{J} c_j I(X \in R_j)$

Which is better?

- If the relationship between response and predictors is approximately linear, a linear model will likely be better.
- If the relationship is highly non-linear and complex, a tree will likely be better.
- May examine their test error rates and pick the one that is more accurate.
- Other considerations, such as interpretability and data visualization, may also come into play.

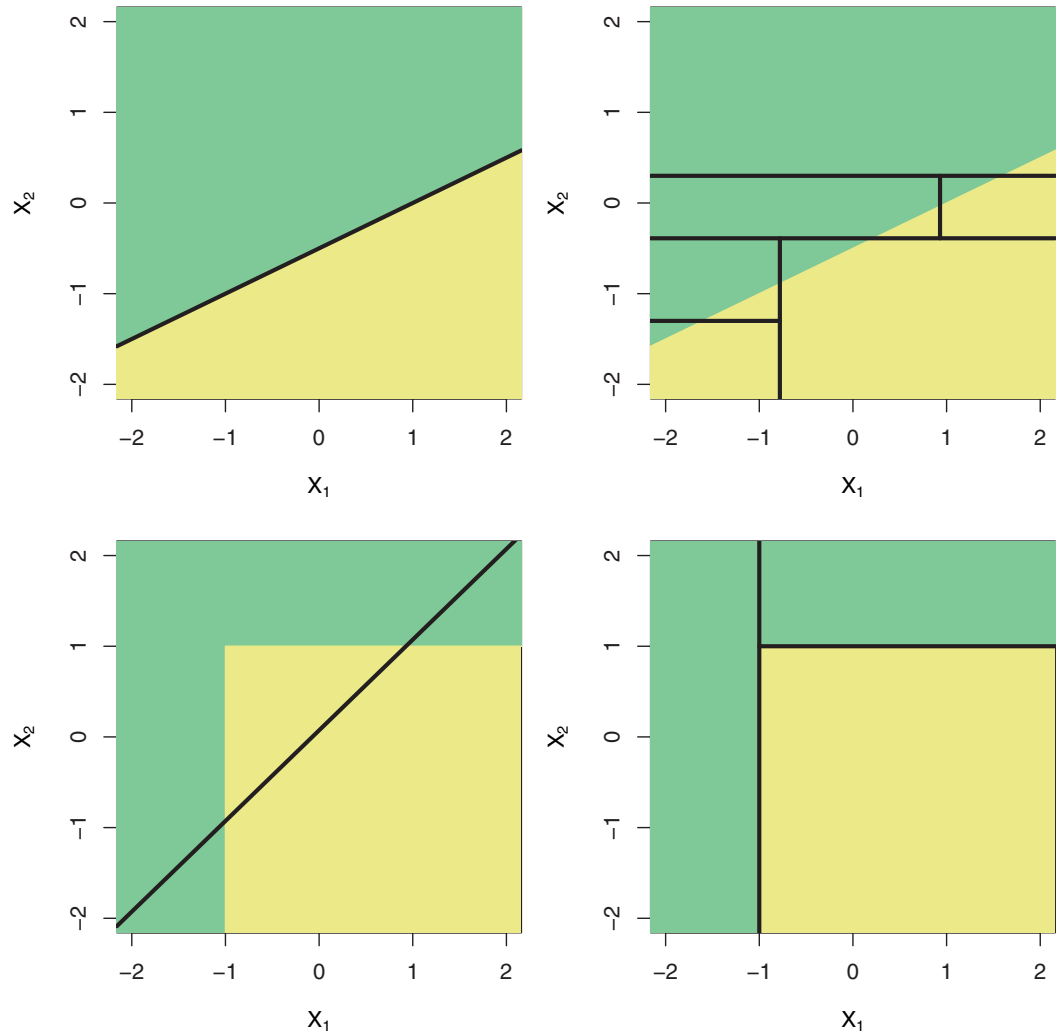

FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: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

- **Why binary splits?** We can consider multiway splits and also splits into more than two groups but this will fragment the data too quickly, leaving insufficient data for the next level down.
- Trees are easy to explain and they provide a simple display to visualize the data (even easier than linear regression)
- Trees can handle qualitative predictors without the need to create dummy variables
- Trees don't provide a smooth estimate of the mean function — an issue in regression, not in classification
- Trees tend to be unstable — a small change in the data can result in a very different tree
- Generally, trees have **less** predictive accuracy than the other regression and classification approaches we have seen.
- The techniques of bagging, random forest, and boosting alleviate the problems of stability and predictive accuracy.

TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲ *= good,* ◆ *=fair, and* ▼ *=poor.*

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲

Source: ESLII

Bagging Decision Trees

Issue: Decision trees have high variance. Need a way to average trees to reduce variance, thereby improving prediction accuracy.

Bagging: (**Bootstrap aggregating**) A general method to reduce variance of a learning method. It has the following steps:

- generate a large # B of training sets using bootstrap,
- repeat training the method on the bth set to get $\hat{f}^{*b}(x)$, $b = 1, \ldots, B$, and
- aggregate the results to get one consensus estimate $\hat{f}_{\text{bag}}(x)$.

Bagging trees: Grow B trees that are deep and **not** pruned.

Bagging regression trees: Average the B predictions to get the overall prediction:

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

Bagging classification trees: Record the class predicted by each of the B trees and take the *majority vote*. Thus, the overall prediction is the most commonly occurring class among the B predictions.

- Bagging trees greatly improves their prediction accuracy
- **How to choose B?** Take B large enough so that the test error rate stabilizes. Often, $B = 500$ is enough. May also choose it by plotting the test error rate against B.
- Increasing B will **not** lead to overfitting.
- Results are difficult to visualize and interpret as they can no longer be displayed as a tree.

Bagging also provides a simply way to estimate the test error rates. This is an alternative to cross-validation or a validation set approach — computationally intensive for large datasets.

Out-of-Bag (OOB) Error Estimation

OOB observations:For any given bootstrap training set, there will be observations in the original dataset that do not fall in it. These observations are OOB for the given bootstrap set and they can be predicted by the method trained on this set.

OOB error estimation: For $i = 1, \ldots, n$, repeat:

- predict observation i using each of the trees in which this observation is OOB,
- combine the multiple predictions that result to get one overall prediction for observation i — using average for regression and majority vote for classification.

Then, combine the prediction errors in the usual way to get an **OOB error rate** — MSE in case of regression and misclassification rate in case of classification.

- OOB error rate is a valid estimate of test error rate since each obs is predicted only using the trees that are not fit using this obs. For large B, OOB $\approx$ LOOCV.

Variable Importance Measures

Issue: How to measure importance of predictors (because the results can no longer be displayed as a tree)?

While bagging trees, we can record the total decrease in the fitting criterion due to splits over a given predictor, and average it over the B trees. This gives an **importance measure** for each predictor.

- Larger value = higher importance
- Consider decrease in RSS for regression and Gini index (or deviance) for classification.

A Random Forest of Decision Trees

Bagging: Average the predictions from the B trees to get an overall prediction. The B predictions are generally *positively correlated* (dependent). On the other hand, we know that averaging uncorrelated observations lead to a greater reduction in variance than averaging positively correlated observations.

Random forest: Works exactly like bagging by growing B trees but employs a small tweak — each time a split in a tree is considered, a random sample of $m\,(\leq p)$ predictors is chosen from the full set of p predictors as split candidates.

- A fresh sample of m predictors is taken at each split
- Often, we take $m \approx \sqrt{p}$
- Bagging is a special case of random forest if $m = p$
- When $m < p$, random forest forces the B trees to be dissimilar, which *decorrelates* the trees in the sense of reducing the correlation between their predictions.

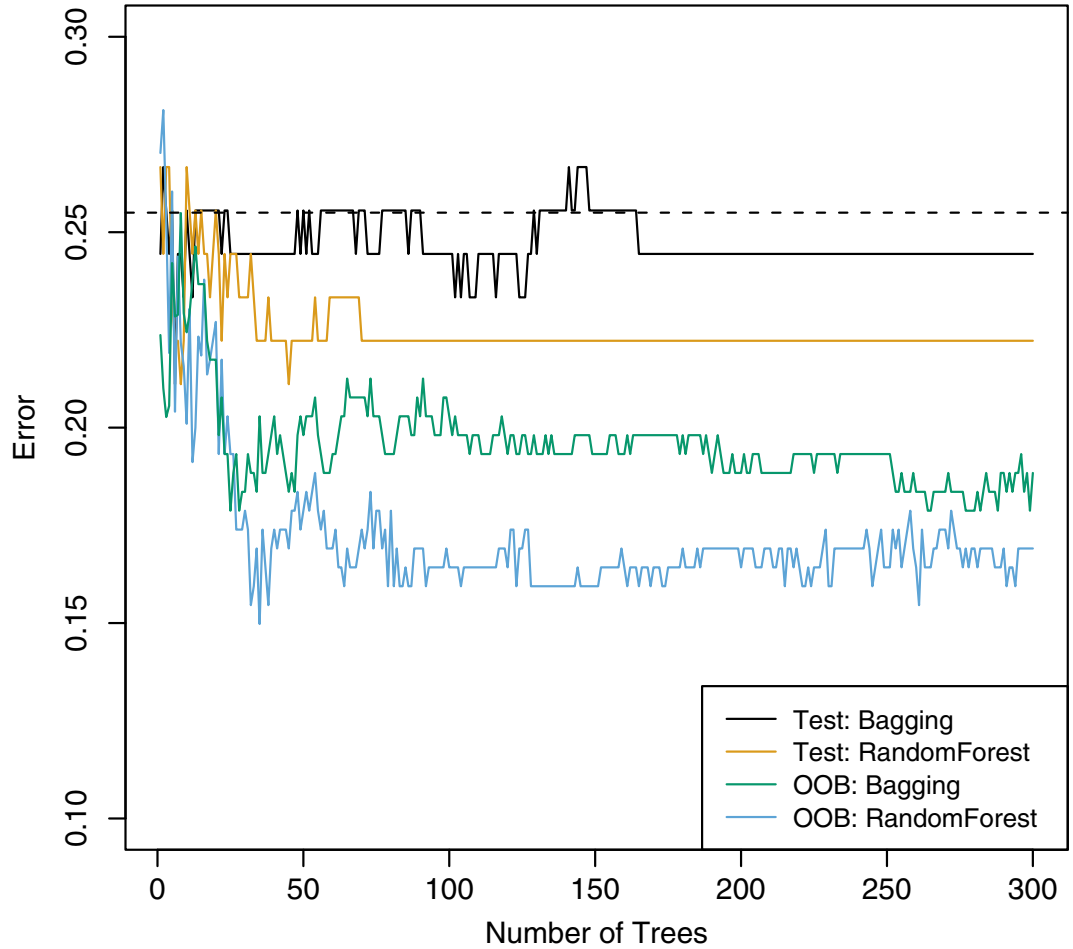

FIGURE 8.8. *Bagging and random forest results for the* `Heart` *data. The test error (black and orange) is shown as a function of* B*, the number of bootstrapped training sets used. Random forests were applied with* $m = \sqrt{p}$*. The dashed line indicates the test error resulting from a single classification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

Source: ISL

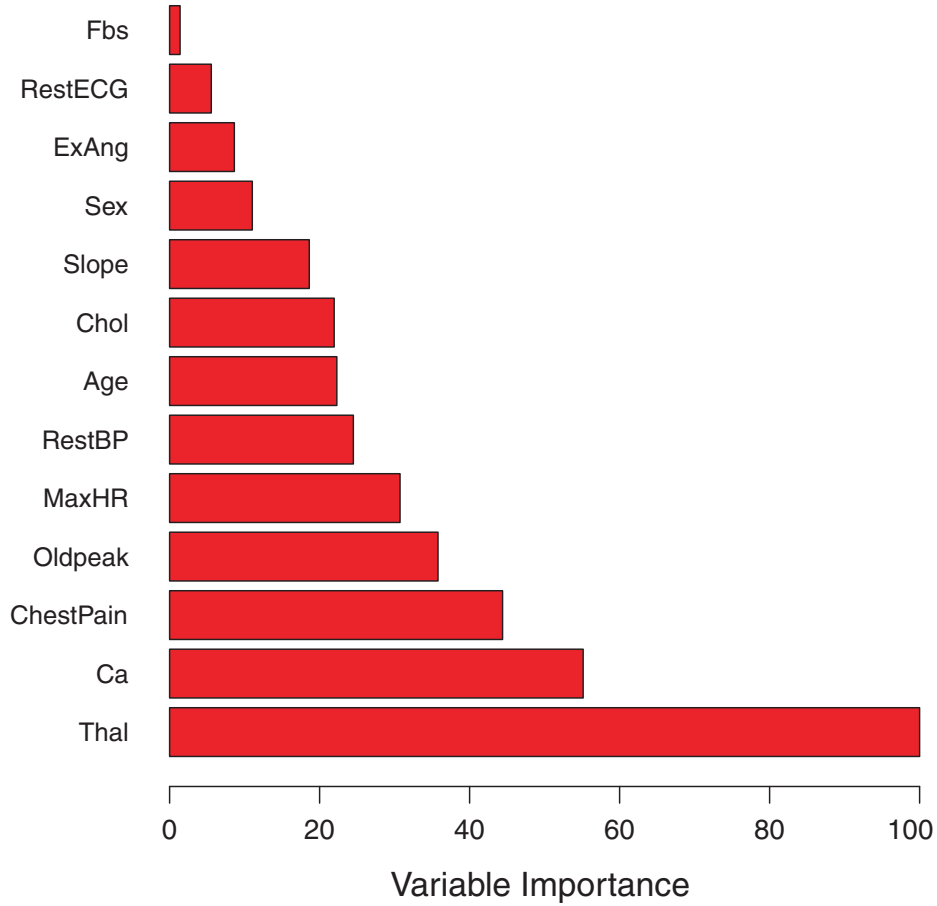

FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Source: ISL

Boosting Decision Trees

Bagging: Create multiple copies of the original dataset using bootstrap, fit a separate tree to each dataset, and combine to create a single predictive model.

Boosting: Grow the trees *sequentially* — grow each tree using information from previously grown trees. Each tree in the sequence is "weak" by itself but together they produce a "powerful committee" — a general idea that works in other contexts as well.

- Does not involve bootstrap sampling
- Learns slowly

We will consider boosting only in the context of regression trees. Similar ideas apply for classification trees (see Chapter 10 of ESLII for details).

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,

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

Source: ISL

- Given the current model, we fit a tree to the *residuals* from the model rather than the outcome Y. Then, add the new tree into the fitted function to update the residuals.
- Each of these trees can be rather small — controlled by d.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well.
- The shrinkage parameter λ slows down this process further, allowing more and different shaped trees.
- A *slow learning approach* generally performs well.
- See ESLII for the objective function being minimized using a *gradient boosting algorithm*. Allows loss functions other than the squared error and misclassification error, and performs *subsampling* — at each iteration we sample a fraction η of the training observations (without replacement) and grow the next tree using that subsample. In practice, $\eta = 0.5$ is typical.

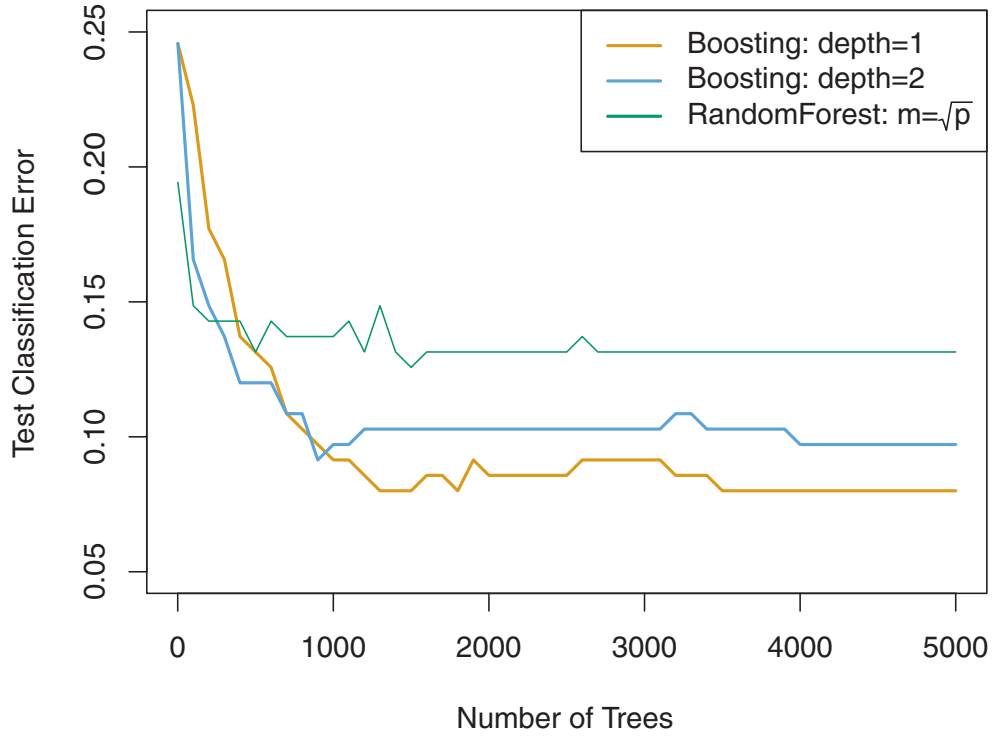

FIGURE 8.11. *Results from performing boosting and random forests on the 15-class gene expression data set in order to predict* cancer *versus* normal. *The test error is displayed as a function of the number of trees. For the two boosted models,* $\lambda = 0.01$. *Depth-1 trees slightly outperform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

Source: ISL

Three Tuning Parameters in Boosting

 # trees B: Boosting can overfit if B is too large — unlike bagging and random forest. We can use cross-validation to select B.

Shrinkage parameter λ: A small positive # that controls the rate at which boosting learns. A very small value can require a very large B to achieve good performance. Typical values are 0.01 or 0.001, but the right choice depends on the problem.

# of splits d (or terminal nodes $d+1$): It controls the complexity of the trees. Often, $d = 1$ works well — each tree is a *stump*, consisting of a single split. More generally, d is called the *interaction depth* as it controls the interaction order of the boosted model, since d splits can involve at most d variables. In practice, $d \approx 5$ is often good enough.

Tree-Based Methods

Python coding exercises

- Classification trees
- Regression trees
- Bagging & Random Forests
- Boosting