Regression trees

Trees involve **stratifying or segmenting** the predictor space into a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs.

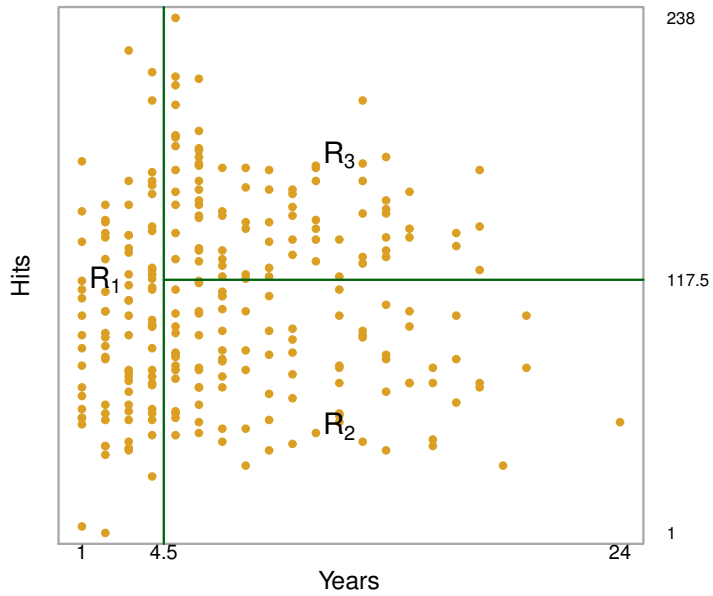

Figure: For the Hitters data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. The three-region partition for the Hitters data set from the regression tree.

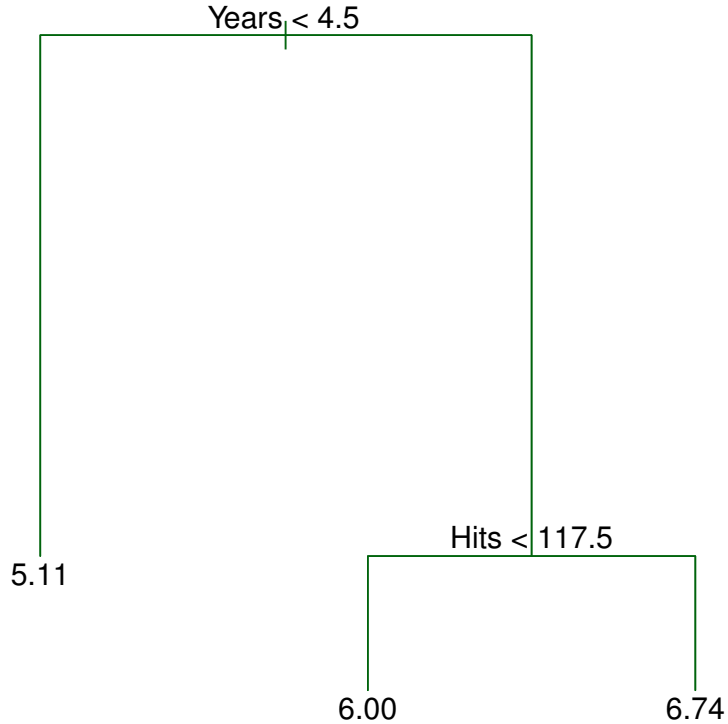

Figure: At a given internal node, the label (of the form $X_j < t_k$) indicates the left-hand branch emanating from that split, and the right-hand branch corresponds to $X_j \geq t_k$. For instance, the split at the top of the tree results in two large branches. The left-hand branch corresponds to Years<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. We refer to the segments of the trees that connect the nodes as **branches**.

Advantages of regression trees

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- It's non-model based and fully nonparametric

Build a regression tree

1. We divide the predictor space-that is, the set of possible values for $X_1, X_2, ..., X_p$-into J distinct and non-overlapping regions, $R_1, R_2, ..., R_J$.
2. For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

How do we construct the regions $R_1, ..., R_J$?

we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, ..., R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

Recursive binary splitting: a top-down, greedy approach

- The approach is **top-down** because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is **greedy** because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Recursive binary splitting

- first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $X|X_j < s$ and $X|X_j \geq s$ leads to the greatest possible reduction in RSS. (The notation $X|X_j < s$ means the region of predictor space in which X_j takes on a value less than s.)
- we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions.
- The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

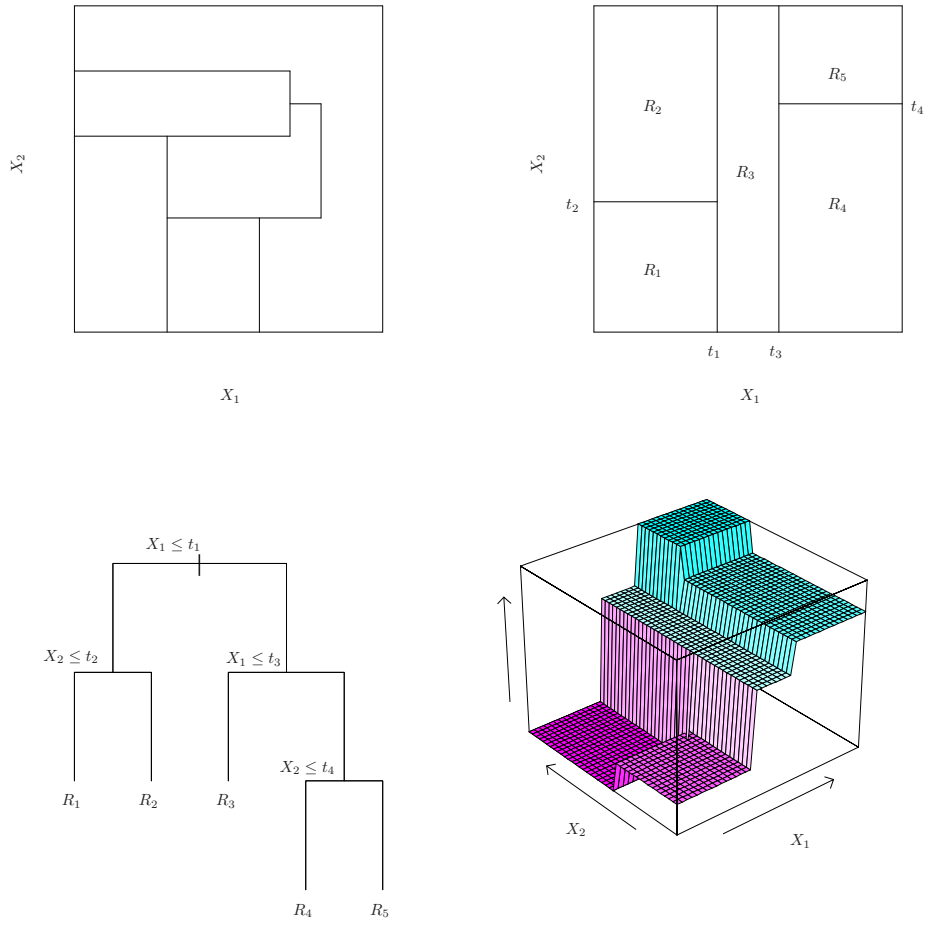

Figure: Top Left: A partition of two-dimensional feature space that could not result from recursive binary splitting. Top Right: The output of recursive binary splitting on a two-dimensional example. Bottom Left: A tree corresponding to the partition in the top right panel. Bottom Right: A perspective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

Tree pruning

When should the recursive binary splitting stop? Goal: a smaller tree that doesn't overfit.

- One possible approach is to build the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold. This strategy will result in smaller trees, but is too short-sighted since a seemingly worthless split early on in the tree might be followed by a very NA
- A **better strategy** is to grow a very large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached. And then prune it back in order to obtain a subtree, preferably with the lowest test error rate.

Cost complexity pruning (Weakest link pruning)

Rather than considering every possible subtree, the weakest link pruning procedure:

- successively collapses the internal node that produces the smallest per-node increase in RSS.
- continues until we produce the single-node (root) tree

It turns out this procedure generates a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α, there corresponds a subtree $T \in T_0$ such that

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

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T.

The tuning parameter α controls a trade-off between the subtree complexity and its fit to the training data. It turns out that 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.

Breiman (1984) and Ripley (1996) have shown that

- For each α, there is a unique smallest subtree T_α that minimizes the cost complexity.
- The sequence of subtrees obtained by pruning under the weakest link, must contain T_α.

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.

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.

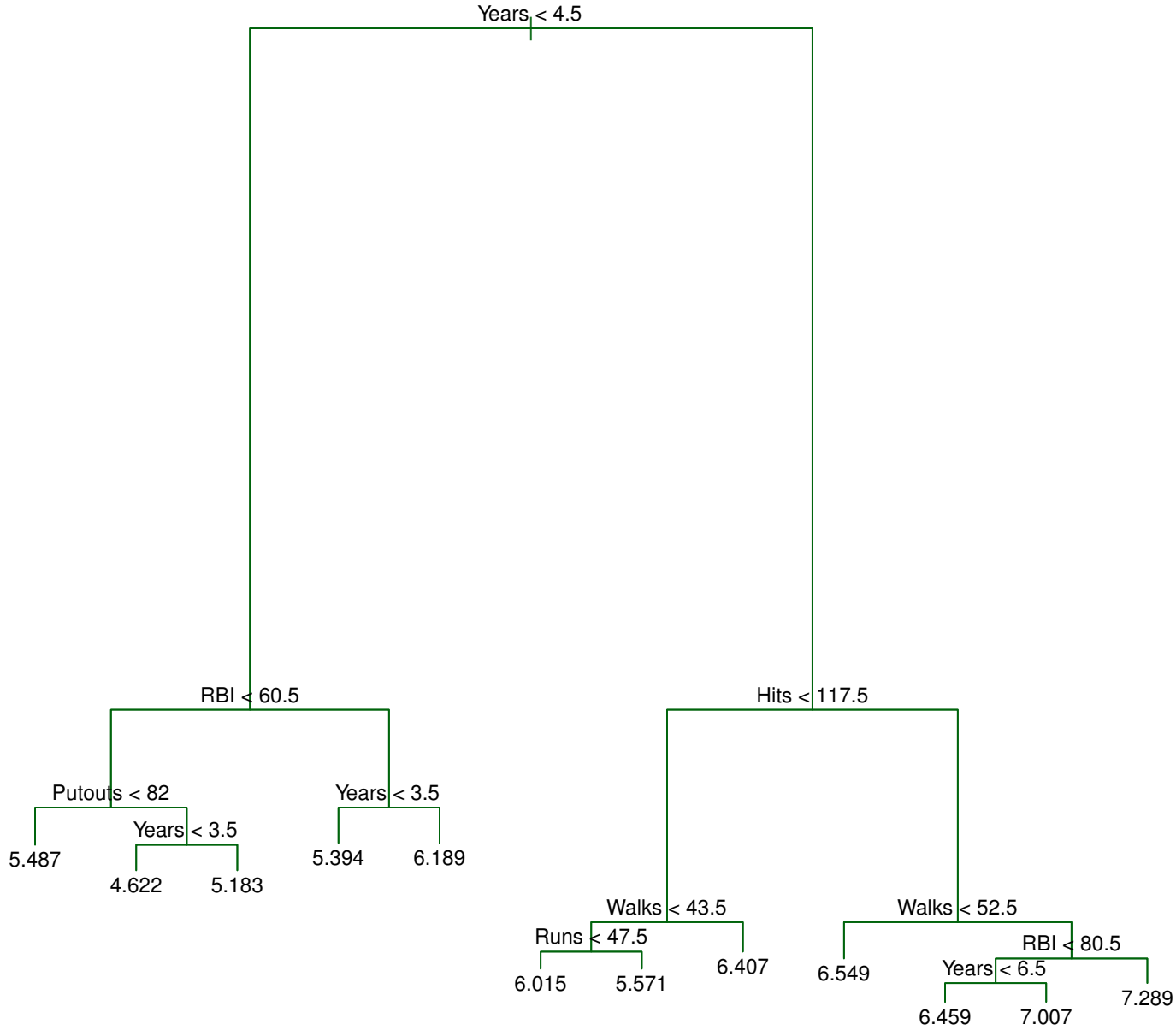

Figure: Regression tree analysis for the Hitters data. The unpruned tree that results from top-down greedy splitting on the training data is shown.

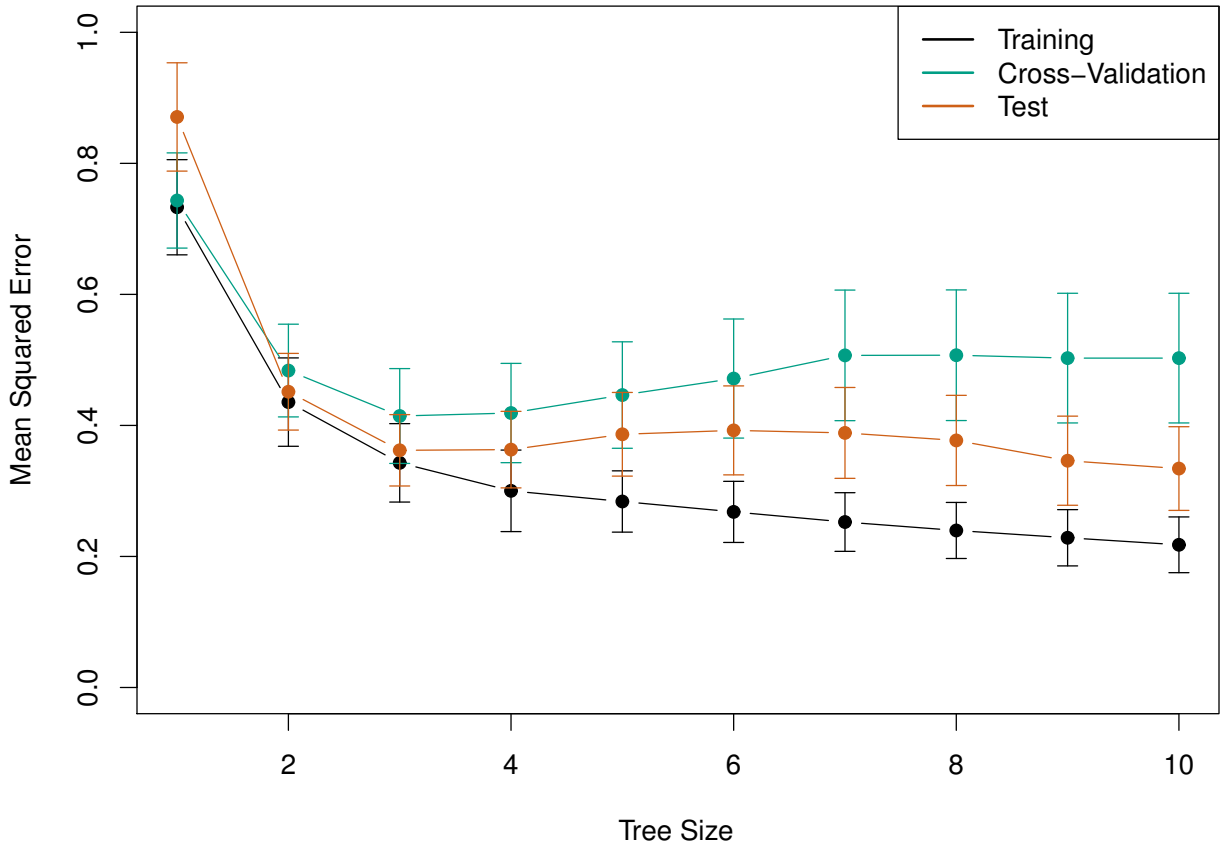

Figure: Regression tree analysis for the Hitters data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three. The dataset is randomly split in half, ielding 132 observations in the training set and 131 observations in the test set. The training error and CV error are based on the 132 training data and the test error is based on the 131 test data.

Classification trees

Different from regression tree, classification tree

- predicts that each observation belongs to the **most commonly occurring class** of training observations in the region to which it belongs.
- replace the goodness of fit measure at node m, RSS_m, with $N_m Q_m$ where N_m is the number of observations node m contains and Q_m is an impurity measure of node m.

Three commonly used impurity measures

- Classification error rate: $E = 1 - \max_k(\hat{p}_{mk})$ where $\hat{p}_{mk}$ represents the proportion of training observations in the mth region that are from the kth class.
- Gini index: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
- Cross-entropy (deviance): $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

In two-class problems, let p be the proportion of one class

- Classification error rate $E = 1 - max(p, 1 - p)$
- Gini index $G = 2p(1 - p)$
- Cross-entropy $D = -p \log(p) - (1 - p) \log(1 - p)$

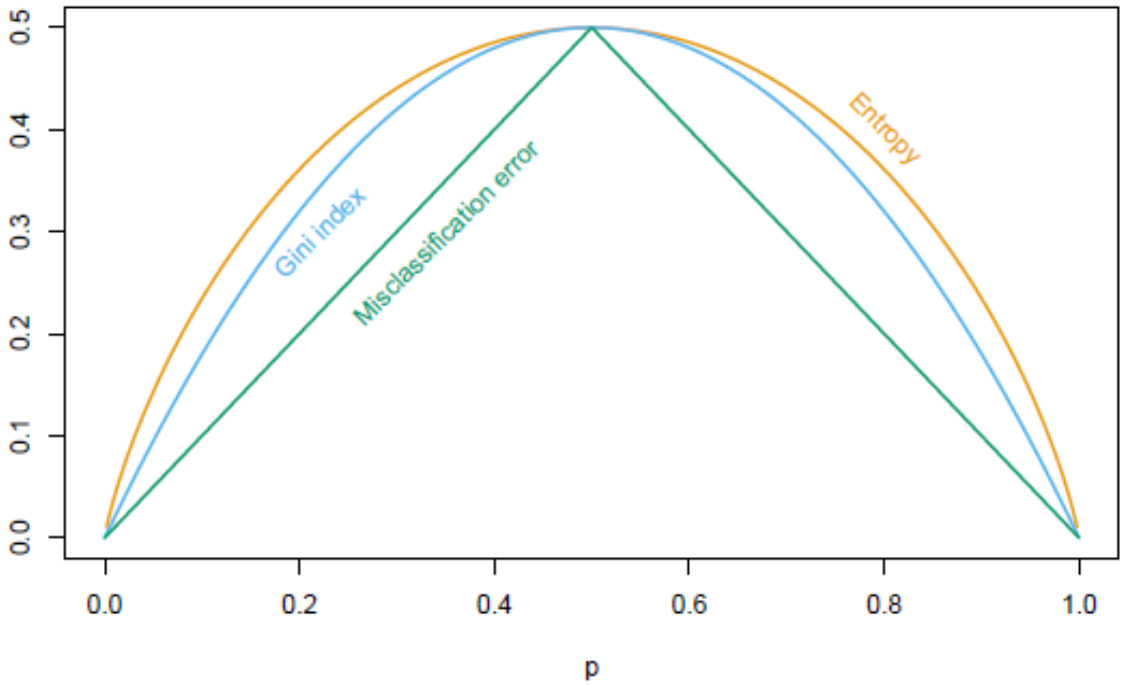

FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

- The Gini-index and cross-entropy take on a small value if all of the $\hat{p}_{mk}$'s are close to zero or one, i.e., if the mth node is pure. In fact, it turns out that the Gini index and the cross-entropy are quite similar numerically.

- Cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. For example, in a two-class problem with 400 observations in each class (denote this by (400,400)), suppose one split created nodes (300, 100) and (100, 300), while the other created nodes (200 , 400) and (200 , 0). Both splits produce a mis- classification rate of 0.25, but the second split produces a pure node and is probably preferable. Both the Gini index and cross-entropy are lower for the second split.

- either the Gini index or the cross- entropy are typically used to evaluate the quality of a particular split, since these two approaches are more sensitive to node purity than is the classification error rate. Any of these three approaches might be used when pruning the tree, but the classification error rate is preferable if prediction accuracy of the final pruned tree is the goal.

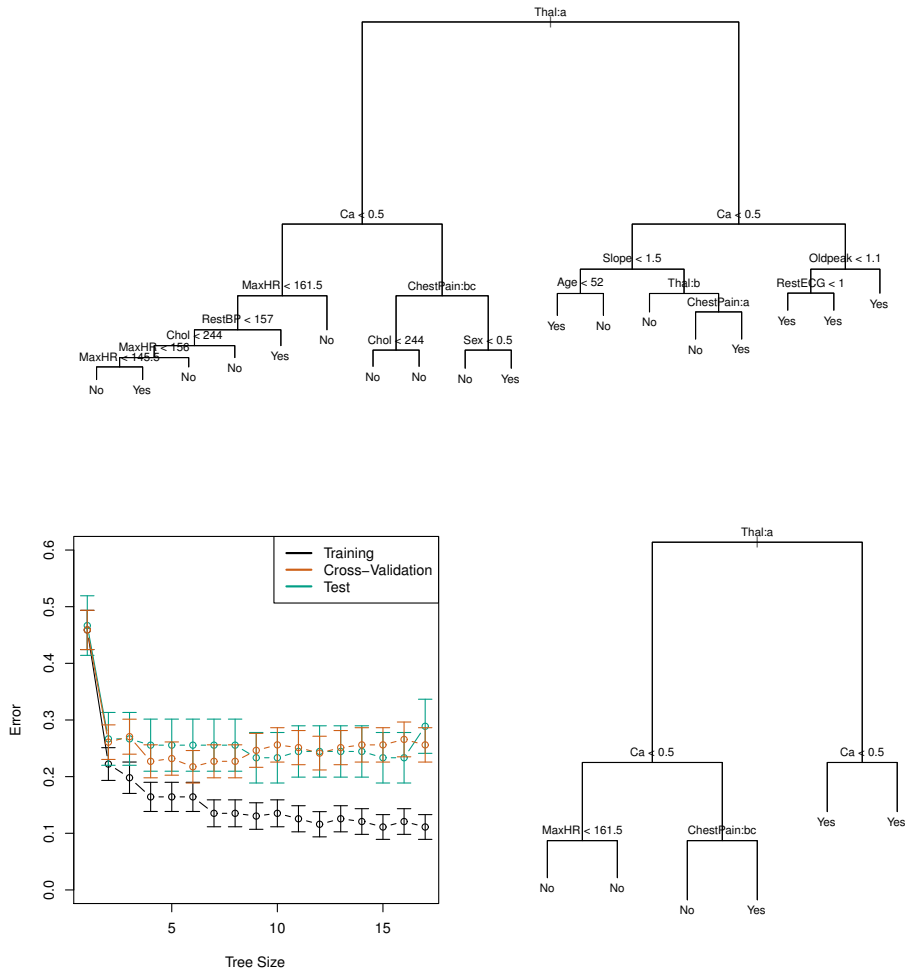

Figure: Heart data. Top: The unpruned tree. Bottom Left: Cross -validation error, training, and test error, for different sizes of the pruned tree. Bottom Right: The pruned tree corresponding to the minimal cross-validation error.

Trees Versus Linear Models

Linear regression

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

Regression tree:

$$f(X) = \sum_{m=1}^{M} I_{\{X \in R_m\}}$$

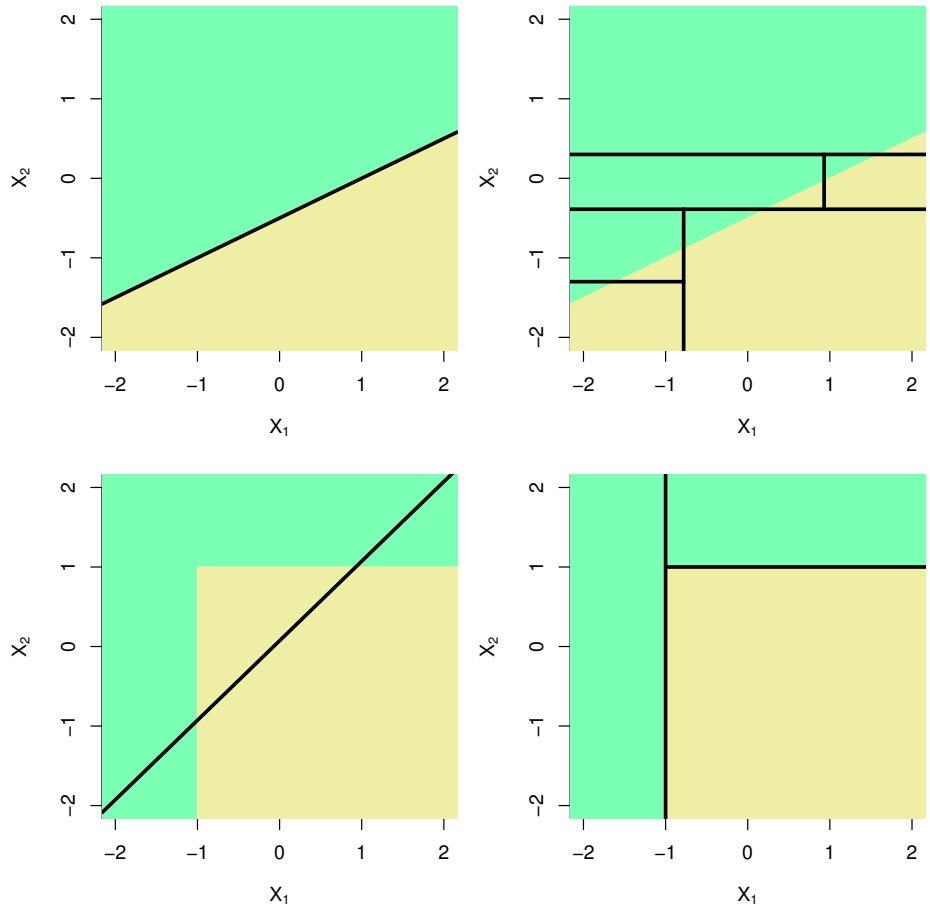

Figure: Top Row: A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a de- cision tree that performs splits parallel to the axes (right). Bottom Row: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

Issues with categorical predictors

- When splitting a predictor having q possible unordered values, there are $2^{q-1} - 1$ possible partitions of the q values into two groups, and the computations become prohibitive for large q.
- With a 0-1 outcome, and continuous outcome(with square error loss), the computation simplifies. The categories of the categorical predictor are ordered by the proportion falling in outcome class 1 or by the mean of the outcome, then the predictor is split as if it were an ordered one. One can show this gives the optimal split, in terms of cross-entropy or Gini index, among all possible $2^{q-1} - 1$ splits.
- The partitioning algorithm tends to favor categorical predictors with many levels q. This can lead to severe overfitting if q is large, and such variables should be avoided.

Issues with missing values

Possible approaches:

- For categorical variables, **create the category "missing"**. This also helps to discover that observations with missing values behave differently than those complete observations.
- Construct **surrogate variables** besides the best splitting variable. If the primary splitting variable is missing, use surrogate splits in order.

- Multiway splits
  - Multiway splits fragment the data too quickly, leaving insuffcient data at the next level down
  - Multiway splits can be achieved by a series of binary splits
- Linear combination splits
  - Use the splits of the form $\sum a_j X_j \leq s$. The weights a_j and split point s are optimized to minimize the relevant criterion.
  - It can improve the predictive power of the tree, but hurts interpretability.
  - The amount of computation is increases signifficantly. Model becomes more complex.

Limitations of Trees

- Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.
- Trees have **high variance** - Small change in data may result in a very different series of splits, making interpretations somewhat precautious. The major reason for this instability is the **hierarchical nature of the process**: the effect of an error in the top split is propagated down to all of the splits below it.
- **Remedy**: by aggregating many decision trees, using methods like **bagging , random forests ,and boosting** , the predictive perfor mance and stability of trees can be substantially improved.