CKW

Statistics 8330: Data Analysis III
Tree Based Models: Classification and Regression Trees

Suggested Reading: James, Witten, Hastie, and Tibshirani (JWHT), 2013, Chapter 8.1

Supplemental Reading: Hastie, Tibshirani, and Friedman (HTF), 2009, Chapter 9.2

Christopher K. Wikle

University of Missouri
Department of Statistics

Classification and Regression Trees (CART)

We now consider an alternative to classical regression and classification that is based on segmenting the predictor space into a number of simple regions. Basically, we develop a collection of splitting rules that make these segments. The splitting rules can conveniently be thought of as a "tree" and so we call these methods **decision trees**.

Decision trees can be used for both classification and regression, and hence often go by the name *classification and regression trees* (CART).

In the case of regression, as we partition the predictor space into segments, the prediction is then just the average for that region.

In the case of classification, we classify according to the most prevalent class in each segment. We will consider the regression case first.

Regression Trees

CKW

Assume we have p inputs and a response for each of n observations, $(x_i, y_i), i = 1, \ldots, n$ with $x_i = (x_{i1}, \ldots, x_{ip})^T$. Suppose that we have a partition of the predictor space into J regions $R_1, R_2, \ldots, R_J$. Then, our prediction in each region is some constant, c_j. Thus, we could write our response function as:

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

where $I(x \in R_j)$ is an indicator function that takes the value of 1 if x is in the jth region and 0 otherwise. Given the partitions, the estimate $\hat{c}_j$ is just the average of y_i in R_j,

$$\hat{c}_j = \hat{y}_{R_j} \equiv \text{ave}(y_i | x_i \in R_j).$$

The challenge is finding the partitions!

Regression Trees (cont.)

In principle, any segmentation of the predictor space could work, but we seek partitions that facilitate interpretation. Thus, we seek to divide the predictor space into high-dimensional rectangles (boxes). That is, we want to find the boxes $R_1, \ldots, R_J$ that minimize the residual sum of squares (RSS):

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

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box (e.g., $\hat{c}_j$ from above).

Not surprisingly, we can't consider every possible partition of the feature space as it would be computationally prohibitive. Instead we seek *binary partitions* and specifically, a *top down, greedy* approach known as *recursive binary splitting*. Before we define this, consider the simple example from JWHT (2013).

Regression Trees (cont.)

Consider the baseball hitters example from JWHT (2013):

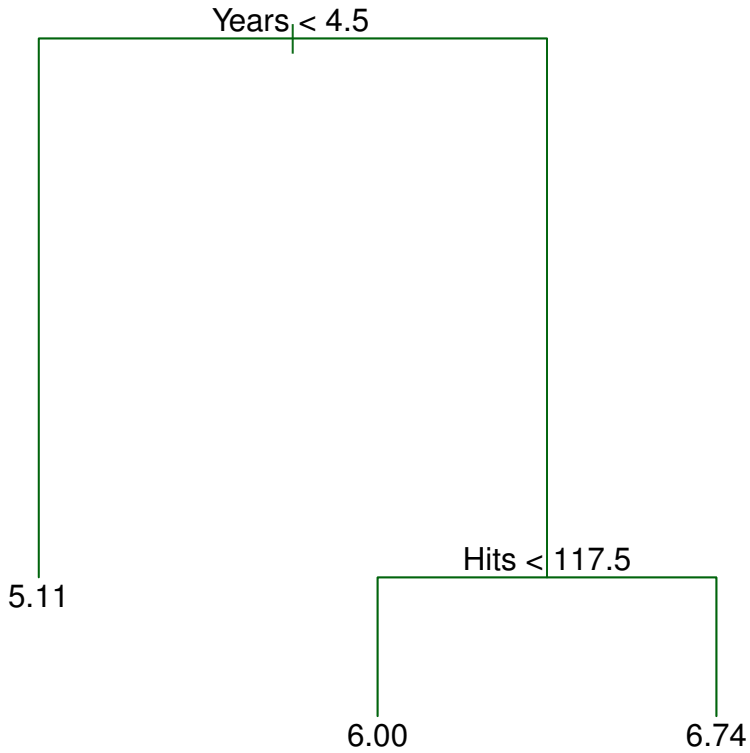


FIGURE 8.1. *For the* `Hitters` *data, a regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year. At a given internal node, the label (of the form $X_j < t_k$) indicates 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.*

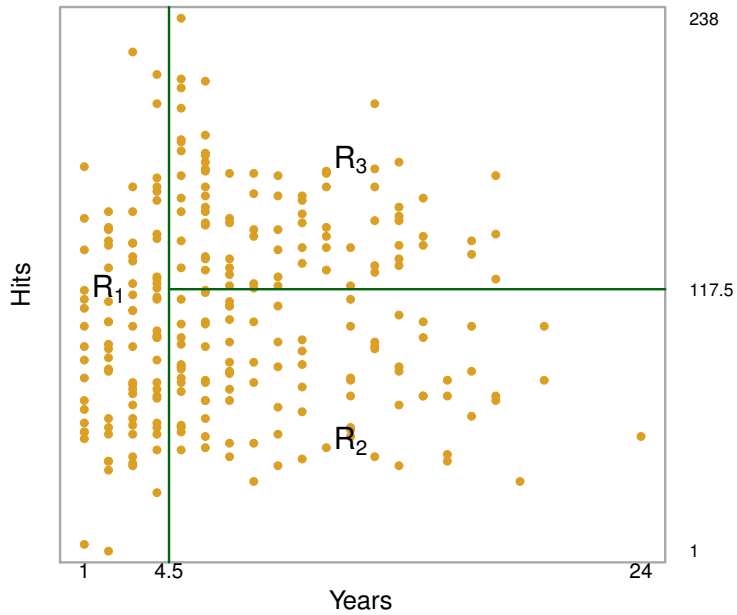


FIGURE 8.2. *The three-region partition for the* `Hitters` *data set from the regression tree illustrated in Figure 8.1.*

Regression Trees (cont.)

CKW

With recursive binary splitting, we start at the top of the tree (i.e., with one node) and start making binary splits successively. We say it is a *greedy* algorithm because at each step the *best* split is made at that particular step, rather than looking ahead and picking a split that leads to a better tree at a later step.

To start, we first select the predictor X_j and the *cut point* s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the maximum reduction in RSS. That is, for *any* j and s, we define the regions (half-planes):

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

and seek the value of j and s that minimizes

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

where $\hat{y}_{R_1} = \hat{c}_1$ is the mean response in $R_1(j,s)$ (and similarly for $\hat{y}_{R_2}$).

Regression Trees (cont.)

When p is not too large, it is computationally efficient to find the values of j and s that accomplishes this minimization.

After this partition is selected, we then repeat the process, looking for the best predictor and best cut point to split one of the two new regions so as to minimize the RSS. This now gives us three regions. We then seek to split one of these regions to minimize the RSS.

This splitting procedure continues until a stopping criterion is reached – usually, this is when a region contains fewer than some specified number of observations (remember, we will need to take averages of these regions for our mean response).

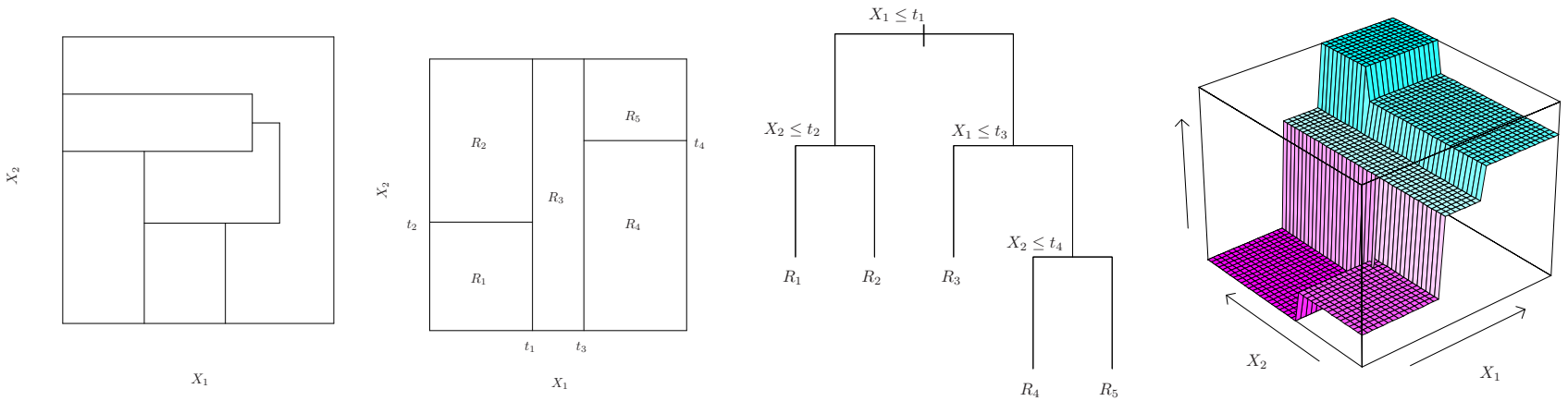
Terminology: the end regions of the tree $(R_1, \ldots, R_J)$ are known as *terminal nodes* or *leaves* of the tree. The points along the tree where the predictor space is split are referred to as *internal nodes*.

Regression Trees (cont.)

CKW

Consider the partition example for a two-dimensional feature space from JWHT (2013):

(a) (b) (c) (d)



- (a) A partition of two-dimensional feature space that could not result from recursive binary splitting.
- (b) The output of recursive binary splitting on a two-dimensional example.
- (c) A tree corresponding to the partition in (b)
- (d) A perspective plot of the prediction surface corresponding to the tree in (c)

Regression Trees (cont.)

CKW

Although this procedure for building a tree can produce good predictions on the training data, it typically leads to over-fitting on the test set. It typically helps to make the tree smaller by having fewer regions. One way to improve the tree is to *prune* it (i.e., remove leaves) to obtain a *subtree*. Not surprisingly, we could probably benefit from using cross-validation.

We consider using so-called *cost complexity pruning* (also called *weakest link pruning*). In this case, we first grow a very large tree, T_0. Rather than consider every possible subtree, we consider a sequence of trees indexed by a non-negative tuning parameter, α. That is, for each value of α there is a subtree, T such that $T \subset T_0$, that minimizes

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

where $|T|$ indicates the number of terminal nodes.

Regression Trees (cont.)

$\mathcal{CKW}$

The parameter α controls a trade-off between the subtree's complexity and how well it fits the training data (note: $\alpha = 0$ implies $T = T_0$). As α gets larger, the penalty for a larger tree is higher, so it tends to choose a smaller tree. Note the similarity of this penalization to the lasso!

It turns out that as α increases, the branches get pruned in a nested and predictable fashion, and so it is possible to get the whole sequence of subtrees as a function of α easily. Thus, it is easy to select α based on cross-validation. After selecting α using cross-validation, we then use that value of α with the full data set to obtain the fitted subtree.

Regression Trees (cont.)

CKW

The algorithm for building a regression tree as given in JWHT (2013):

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.

Regression Trees (cont.)

Consider the baseball hitters example from JWHT (2013, Fig 8.4, 8.5):

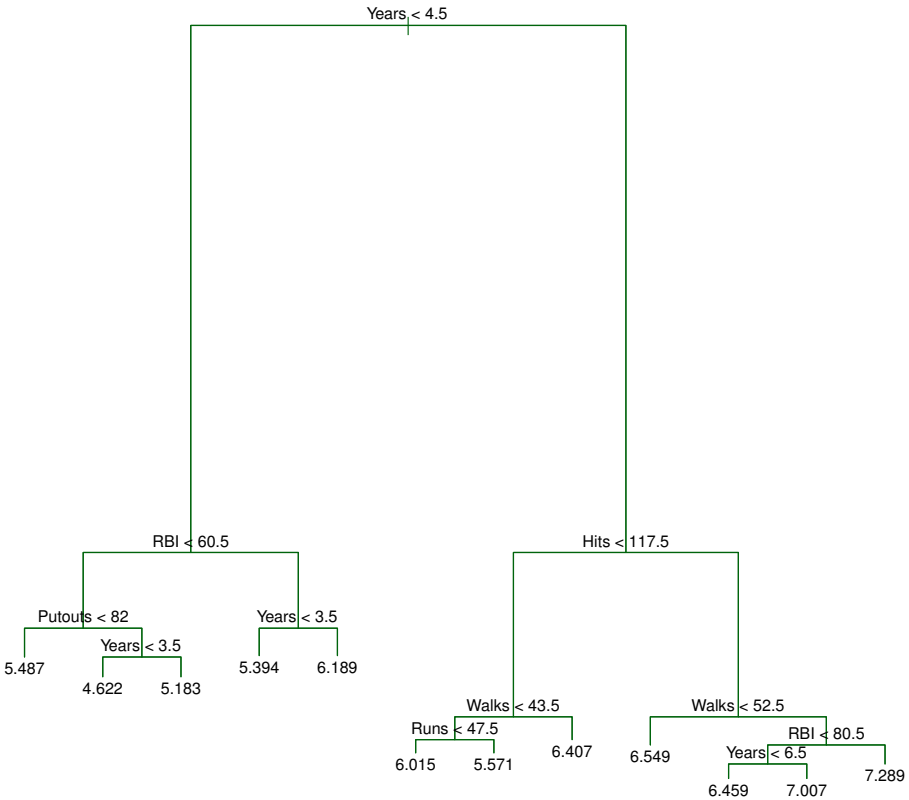


FIGURE 8.4. *Regression tree analysis for the* `Hitters` *data. The unpruned tree that results from top-down greedy splitting on the training data is shown.*

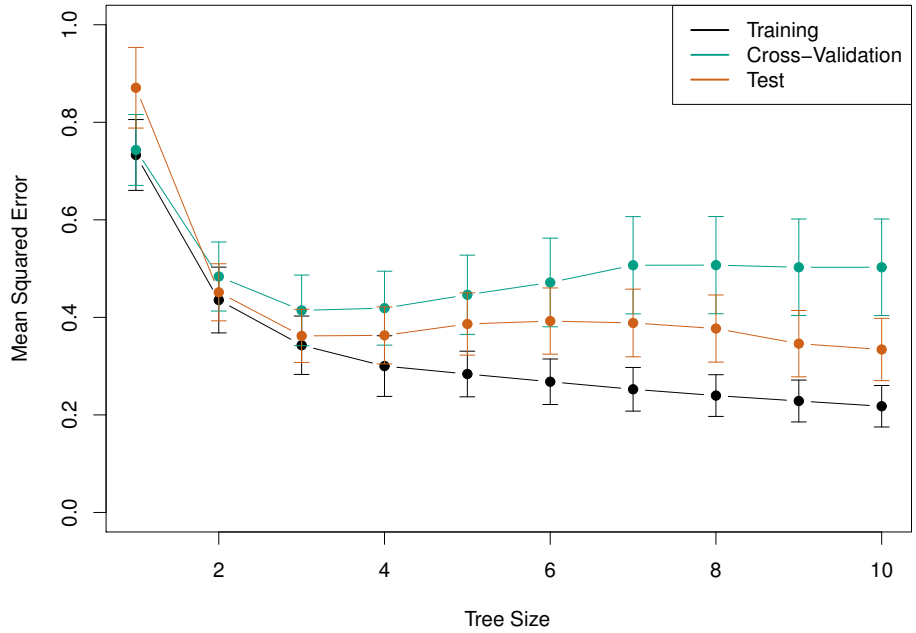


FIGURE 8.5. *Regression tree analysis for the* `Hitters` *data. The training, cross-validation, and test MSE are shown as a function of the number of terminal nodes in the pruned tree. Standard error bands are displayed. The minimum cross-validation error occurs at a tree size of three.*

Classification Trees

When we have a qualitative response we can also use a tree, known as a *classification tree*. The main difference is that we predict the response based on the most commonly occurring class in the region associated with that terminal node. We also may be interested in the class proportions for each node when interpreting a classification tree.

We grow a classification tree in the same manner as the regression tree, with recursive binary splitting. However, we don't use RSS as the basis for making the binary splits. Rather, we have to use some other measure related to the classification summary. There are three commonly used choices. First, we define the proportion of class k observations in the jth node as:

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

Then, as mentioned, we classify an observation in node j to class k with the maximum value of $\hat{p}_{jk}$.

Classification Trees (cont.)

CKW

Consider the following three possible measures for growing the tree:

$$\text{Classification Error Rate:} \quad 1 - \max_k(\hat{p}_{jk})$$

$$\text{Gini Index:} \quad \sum_{k=1}^{K} \hat{p}_{jk}(1 - \hat{p}_{jk})$$

$$\text{Cross-Entropy or Deviance:} \quad -\sum_{k=1}^{K} \hat{p}_{jk} \log(\hat{p}_{jk})$$

The classification error rate measures the fraction of the training observations in that region that do not belong to the most common class and the Gini Index is a measure of the total variance across the K classes. Cross-Entropy is related to an information measure and also the deviance.

Classification Trees (cont.)

CKW

Note that the Gini index and Cross-Entropy index both take small values when all of the $\hat{p}_{jk}$ are close to zero or one. In that sense, they are said to be a measure of *node purity*, and so are more sensitive to node probabilities than the classification error rate. Thus, these measures should be used to grow the tree.

Typically, Gini or Cross-Entropy are used to build a tree and can be used to prune it. But, classification error rate is often preferred for pruning if prediction accuracy of the pruned tree is the goal.

Classification Trees (cont.)

Consider the classification tree example from JWHT (2013):

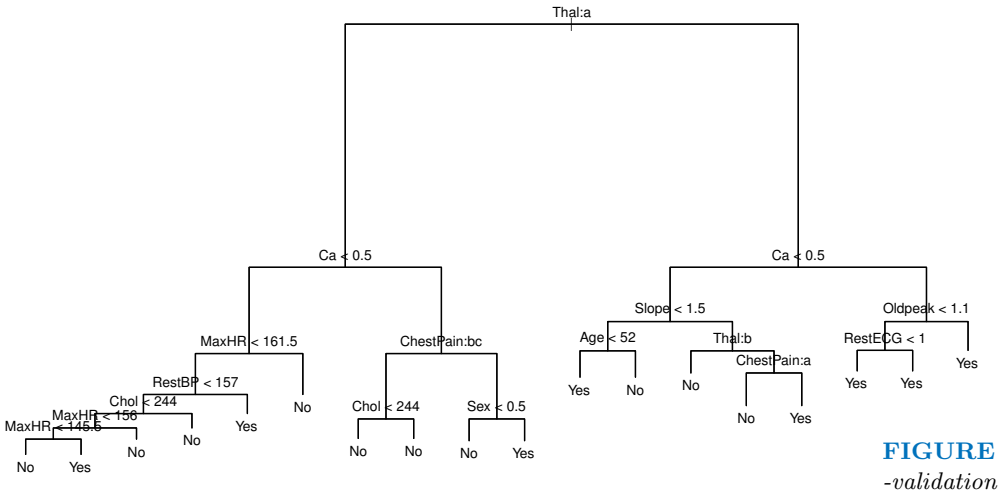
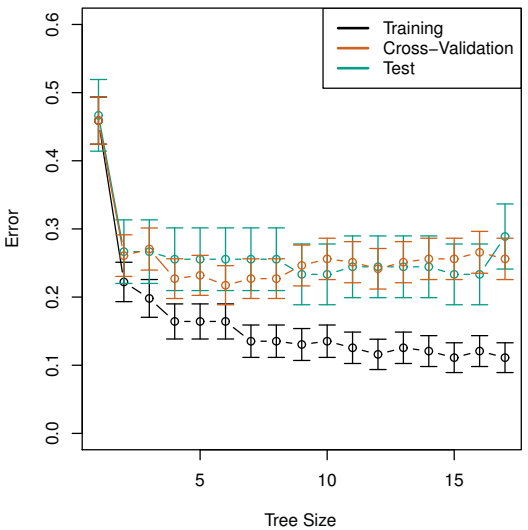
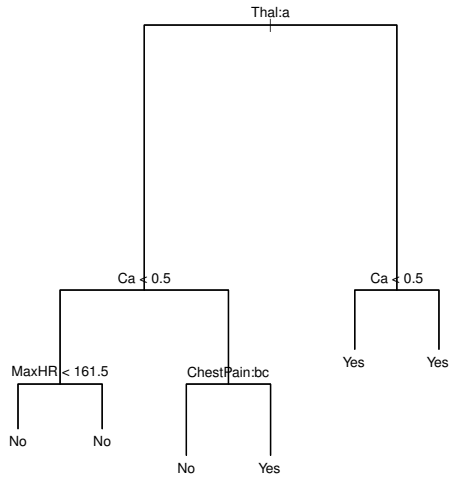


FIGURE 8.6. `Heart` *data.* Top: *The unpruned tree.* Bottom Left: *Cross -validation error, training, and test error, for different sizes of the pruned tree.* Bottom Right: *The pruned tree corresponding to the minimal cross-validation error.*





Classification Trees (cont.)

Consider the email spam example from HTF (2009):

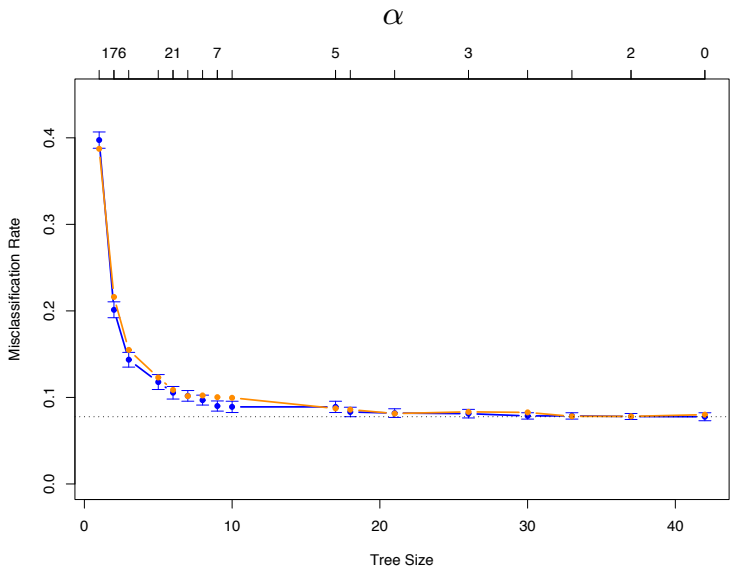


FIGURE 9.4. *Results for* `spam` *example. The blue curve is the* 10*-fold cross-validation estimate of misclassification rate as a function of tree size, with standard error bars. The minimum occurs at a tree size with about* 17 *terminal nodes (using the "one-standard--error" rule). The orange curve is the test error, which tracks the CV error quite closely. The cross-validation is indexed by values of* α*, shown above. The tree sizes shown below refer to* $|T_\alpha|$*, the size of the original tree indexed by* α*.*

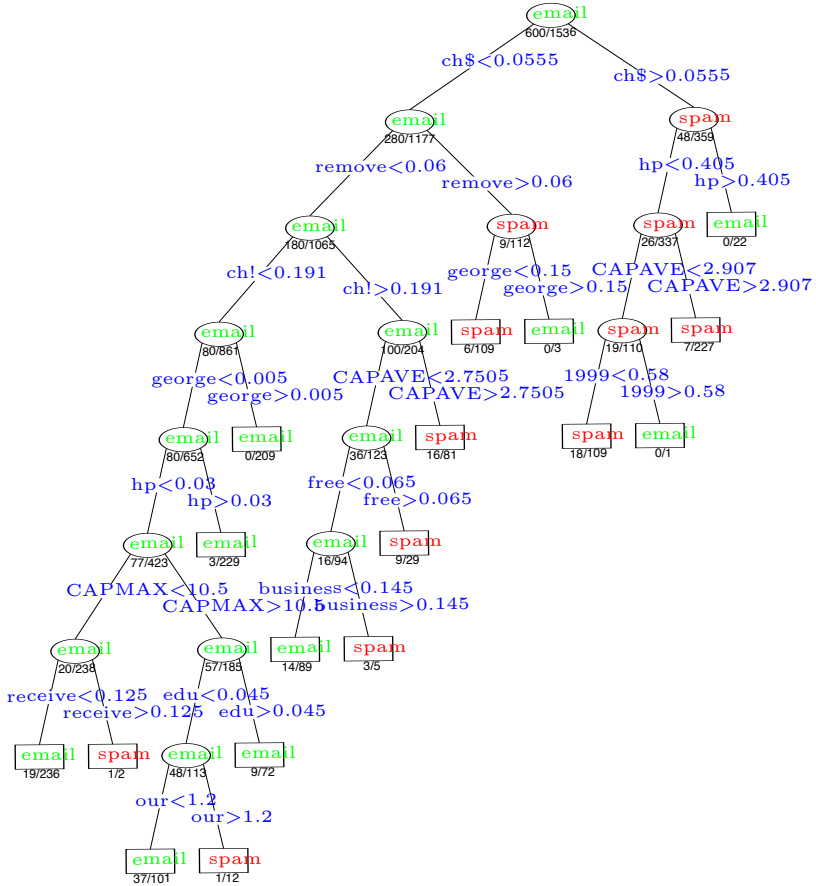


FIGURE 9.5. *The pruned tree for the* `spam` *example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.*

CART vs Linear Models

Not surprisingly, when the underlying relationship is linear, the linear regression/classification setting works better than CART. But, if there is a highly nonlinear and complex relationship between the features and the response, then the CART model is typically better. Consider the illustration from JWHT (2013):

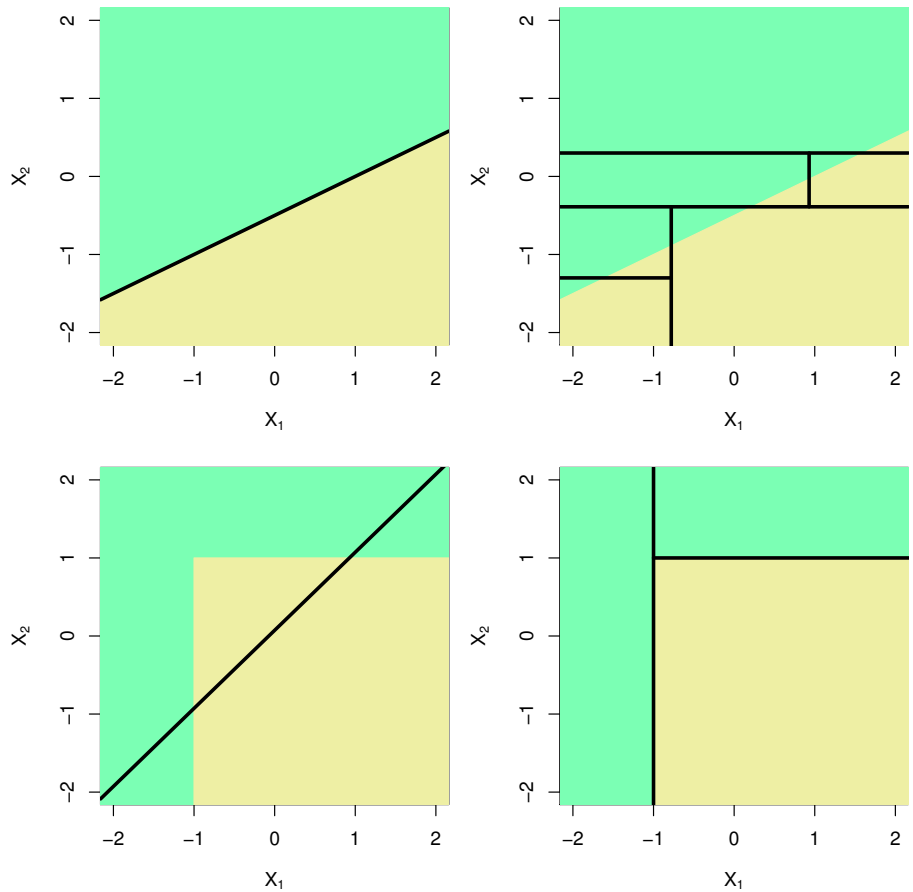


FIGURE 8.7. Top Row: *A two-dimensional classification example in which the true decision boundary is linear, and is indicated by the shaded regions. A classical approach that assumes a linear boundary (left) will outperform a decision tree that performs splits parallel to the axes (right).* Bottom Row: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

CART Issues

CKW

One can consider alternatives and more complicated CART models. In addition, there are some issues that come up when using CART in practice (see HTF, 2009, Section 9.2.4).

- *Categorical Predictors:* It can be difficult to implement CART when one has $q > 2$ possible unordered values for the predictor as there are $2^{q-1} - 1$ possible partitions of the q values into two groups. With a binary outcome this can be simplified. In general, this situation can lead to serious over-fitting.
- *Missing Predictor Values:* There are two options – for categorical variables one sometimes makes a separate category for "missing"; more generally, we might construct surrogate variables that mimic the split of the training data achieved by using only the non-missing variables.
- *Non-Binary Splits:* Not generally recommended as it splits the data too quickly and doesn't leave enough data for the next level of the tree.

CART Issues (cont.)

- *Linear Combination Splits:* One can make linear combinations of the variables to choose the splits (sort of like PC-regression), which can help prediction. This inhibits interpretation of the tree.
- *Instability of Trees:* CART has high variance (a small change in the data can lead to different splits). One way to handle this is through *"bagging,"* which is a bootstrapping procedure. We will talk about this later.
- *Lack of Smoothness:* The CART procedure necessarily leads to a non-smooth prediction surface. This is particularly problematic for regression applications. The MARS procedure alleviates this problem; we will look at that later.
- *Additive Structure:* The binary nature of the CART procedure prevents one from being able to accommodate additive structure. The MARS procedure can help with this as well.

CART Advantages and Disadvantages

- The CART procedure is easy to explain to non-statisticians
- Decision trees may more closely mirror human decision making than traditional regression and classification
- Trees are easily interpreted (and, the graphical depiction helps)
- Trees can handle qualitative predictors without needing dummy variables
- Trees do not generally have as good of predictive accuracy as other regression and classification approaches.