9.2 Tree-Based Methods

9.2.1 Background

Tree-based methods partition the feature space into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful. We first describe a popular method for tree-based regression and classification called CART, and later contrast it with C4.5, a major competitor.

Let's consider a regression problem with continuous response Y and inputs X_1 and X_2, each taking values in the unit interval. The top left panel of Figure 9.2 shows a partition of the feature space by lines that are parallel to the coordinate axes. In each partition element we can model Y with a different constant. However, there is a problem: although each partitioning line has a simple description like $X_1 = c$, some of the resulting regions are complicated to describe.

To simplify matters, we restrict attention to recursive binary partitions like that in the top right panel of Figure 9.2. We first split the space into two regions, and model the response by the mean of Y in each region. We choose the variable and split-point to achieve the best fit. Then one or both of these regions are split into two more regions, and this process is continued, until some stopping rule is applied. For example, in the top right panel of Figure 9.2, we first split at $X_1 = t_1$. Then the region $X_1 \leq t_1$ is split at $X_2 = t_2$ and the region $X_1 > t_1$ is split at $X_1 = t_3$. Finally, the region $X_1 > t_3$ is split at $X_2 = t_4$. The result of this process is a partition into the five regions $R_1, R_2, \ldots, R_5$ shown in the figure. The corresponding regression model predicts Y with a constant c_m in region R_m, that is,

$$\hat{f}(X) = \sum_{m=1}^{5} c_m I\{(X_1, X_2) \in R_m\}. \tag{9.9}$$

This same model can be represented by the binary tree in the bottom left panel of Figure 9.2. The full dataset sits at the top of the tree. Observations satisfying the condition at each junction are assigned to the left branch, and the others to the right branch. The terminal nodes or leaves of the tree correspond to the regions $R_1, R_2, \ldots, R_5$. The bottom right panel of Figure 9.2 is a perspective plot of the regression surface from this model. For illustration, we chose the node means $c_1 = -5, c_2 = -7, c_3 = 0, c_4 = 2, c_5 = 4$ to make this plot.

A key advantage of the recursive binary tree is its interpretability. The feature space partition is fully described by a single tree. With more than two inputs, partitions like that in the top right panel of Figure 9.2 are difficult to draw, but the binary tree representation works in the same way. This representation is also popular among medical scientists, perhaps because it mimics the way that a doctor thinks. The tree stratifies the

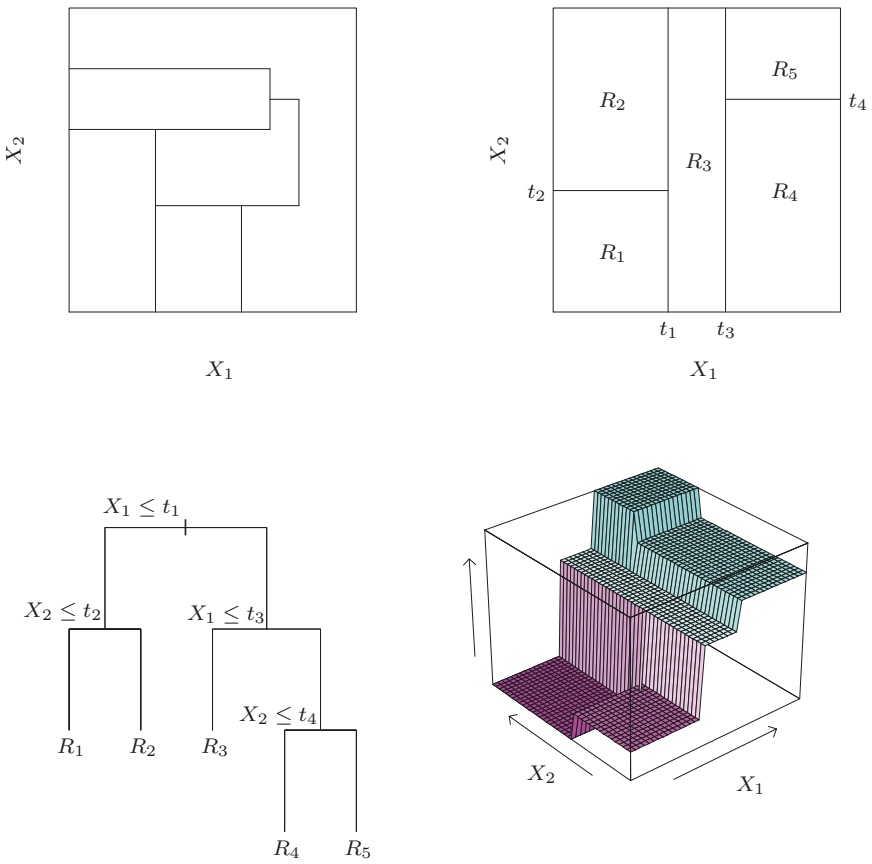

FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

population into strata of high and low outcome, on the basis of patient characteristics.

9.2.2 Regression Trees

We now turn to the question of how to grow a regression tree. Our data consists of p inputs and a response, for each of N observations: that is, (x_i, y_i) for $i = 1, 2, \ldots, N$, with $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m). \tag{9.10}$$

If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m:

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m). \tag{9.11}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j, s) = \{X | X_j \leq s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j > s\}. \tag{9.12}$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\, s} \Big[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \Big]. \tag{9.13}$$

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

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \quad \text{and} \quad \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s)). \tag{9.14}$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible.

Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions.

How large should we grow the tree? Clearly a very large tree might overfit the data, while a small tree might not capture the important structure.

Tree size is a tuning parameter governing the model's complexity, and the optimal tree size should be adaptively chosen from the data. One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold. This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it.

The preferred strategy is to grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using *cost-complexity pruning*, which we now describe.

We define a subtree $T \subset T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes. We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{9.15}$$

we define the cost complexity criterion

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{9.16}$$

The idea is to find, for each α, the subtree $T_\alpha \subseteq T_0$ to minimize $C_\alpha(T)$. The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data. Large values of α result in smaller trees T_α, and conversely for smaller values of α. As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0. We discuss how to adaptively choose α below.

For each α one can show that there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. To find T_α we use *weakest link pruning*: we successively collapse the internal node that produces the smallest per-node increase in $\sum_m N_m Q_m(T)$, and continue until we produce the single-node (root) tree. This gives a (finite) sequence of subtrees, and one can show this sequence must contain T_α. See Breiman et al. (1984) or Ripley (1996) for details. Estimation of α is achieved by five- or tenfold cross-validation: we choose the value $\hat{\alpha}$ to minimize the cross-validated sum of squares. Our final tree is $T_{\hat{\alpha}}$.

9.2.3 Classification Trees

If the target is a classification outcome taking values $1, 2, \ldots, K$, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes and pruning the tree. For regression we used the squared-error node

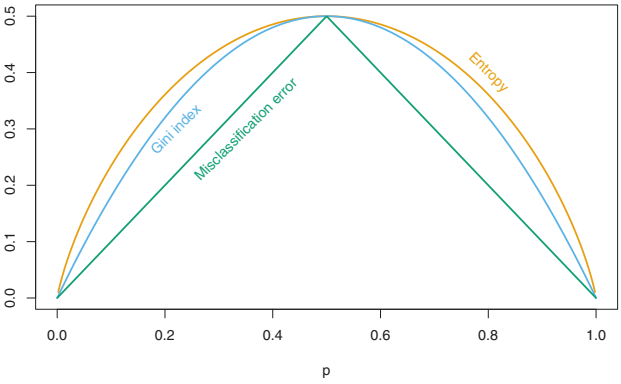

FIGURE 9.3. *Node impurity measures for two-class classification, as a function of the proportion p in class* 2. *Cross-entropy has been scaled to pass through* $(0.5, 0.5)$.

impurity measure $Q_m(T)$ defined in (9.15), but this is not suitable for classification. In a node m, representing a region R_m with N_m observations, let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k),$$

the proportion of class k observations in node m. We classify the observations in node m to class $k(m) = \arg\max_k \hat{p}_{mk}$, the majority class in node m. Different measures $Q_m(T)$ of node impurity include the following:

$$\begin{aligned} &\text{Misclassification error:} && \tfrac{1}{N_m} \textstyle\sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}. \\ &\text{Gini index:} && \textstyle\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}). \\ &\text{Cross-entropy or deviance:} && -\textstyle\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}. \end{aligned} \tag{9.17}$$

For two classes, if p is the proportion in the second class, these three measures are $1 - \max(p, 1-p)$, $2p(1-p)$ and $-p \log p - (1-p) \log (1-p)$, respectively. They are shown in Figure 9.3. All three are similar, but cross-entropy and the Gini index are differentiable, and hence more amenable to numerical optimization. Comparing (9.13) and (9.15), we see that we need to weight the node impurity measures by the number N_{m_L} and N_{m_R} of observations in the two child nodes created by splitting node m.

In addition, 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 misclassification 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. For this reason, either the Gini index or cross-entropy should be used when growing the tree. To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate.

The Gini index can be interpreted in two interesting ways. Rather than classify observations to the majority class in the node, we could classify them to class k with probability $\hat{p}_{mk}$. Then the expected training error rate of this rule in the node is $\sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'}$—the Gini index. Similarly, if we code each observation as 1 for class k and zero otherwise, the variance over the node of this 0-1 response is $\hat{p}_{mk}(1 - \hat{p}_{mk})$. Summing over classes k again gives the Gini index.

9.2.4 Other Issues

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. However, with a $0 - 1$ outcome, this computation simplifies. We order the predictor classes according to the proportion falling in outcome class 1. Then we split this predictor as if it were an ordered predictor. One can show this gives the optimal split, in terms of cross-entropy or Gini index, among all possible $2^{q-1}-1$ splits. This result also holds for a quantitative outcome and square error loss—the categories are ordered by increasing mean of the outcome. Although intuitive, the proofs of these assertions are not trivial. The proof for binary outcomes is given in Breiman et al. (1984) and Ripley (1996); the proof for quantitative outcomes can be found in Fisher (1958). For multicategory outcomes, no such simplifications are possible, although various approximations have been proposed (Loh and Vanichsetakul, 1988).

The partitioning algorithm tends to favor categorical predictors with many levels q; the number of partitions grows exponentially in q, and the more choices we have, the more likely we can find a good one for the data at hand. This can lead to severe overfitting if q is large, and such variables should be avoided.

The Loss Matrix

In classification problems, the consequences of misclassifying observations are more serious in some classes than others. For example, it is probably worse to predict that a person will not have a heart attack when he/she actually will, than vice versa. To account for this, we define a $K \times K$ loss matrix $\mathbf{L}$, with $L_{kk'}$ being the loss incurred for classifying a class k observation as class k'. Typically no loss is incurred for correct classifications,

that is, $L_{kk} = 0\ \forall k$. To incorporate the losses into the modeling process, we could modify the Gini index to $\sum_{k \neq k'} L_{kk'} \hat{p}_{mk} \hat{p}_{mk'}$; this would be the expected loss incurred by the randomized rule. This works for the multiclass case, but in the two-class case has no effect, since the coefficient of $\hat{p}_{mk} \hat{p}_{mk'}$ is $L_{kk'} + L_{k'k}$. For two classes a better approach is to weight the observations in class k by $L_{kk'}$. This can be used in the multiclass case only if, as a function of k, $L_{kk'}$ doesn't depend on k'. Observation weighting can be used with the deviance as well. The effect of observation weighting is to alter the prior probability on the classes. In a terminal node, the empirical Bayes rule implies that we classify to class $k(m) = \arg\min_k \sum_\ell L_{\ell k} \hat{p}_{m\ell}$.

Missing Predictor Values

Suppose our data has some missing predictor values in some or all of the variables. We might discard any observation with some missing values, but this could lead to serious depletion of the training set. Alternatively we might try to fill in (impute) the missing values, with say the mean of that predictor over the nonmissing observations. For tree-based models, there are two better approaches. The first is applicable to categorical predictors: we simply make a new category for "missing." From this we might discover that observations with missing values for some measurement behave differently than those with nonmissing values. The second more general approach is the construction of surrogate variables. When considering a predictor for a split, we use only the observations for which that predictor is not missing. Having chosen the best (primary) predictor and split point, we form a list of surrogate predictors and split points. The first surrogate is the predictor and corresponding split point that best mimics the split of the training data achieved by the primary split. The second surrogate is the predictor and corresponding split point that does second best, and so on. When sending observations down the tree either in the training phase or during prediction, we use the surrogate splits in order, if the primary splitting predictor is missing. Surrogate splits exploit correlations between predictors to try and alleviate the effect of missing data. The higher the correlation between the missing predictor and the other predictors, the smaller the loss of information due to the missing value. The general problem of missing data is discussed in Section 9.6.

Why Binary Splits?

Rather than splitting each node into just two groups at each stage (as above), we might consider multiway splits into more than two groups. While this can sometimes be useful, it is not a good general strategy. The problem is that multiway splits fragment the data too quickly, leaving insufficient data at the next level down. Hence we would want to use such splits only when needed. Since multiway splits can be achieved by a series of binary splits, the latter are preferred.

Other Tree-Building Procedures

The discussion above focuses on the CART (classification and regression tree) implementation of trees. The other popular methodology is ID3 and its later versions, C4.5 and C5.0 (Quinlan, 1993). Early versions of the program were limited to categorical predictors, and used a top-down rule with no pruning. With more recent developments, C5.0 has become quite similar to CART. The most significant feature unique to C5.0 is a scheme for deriving rule sets. After a tree is grown, the splitting rules that define the terminal nodes can sometimes be simplified: that is, one or more condition can be dropped without changing the subset of observations that fall in the node. We end up with a simplified set of rules defining each terminal node; these no longer follow a tree structure, but their simplicity might make them more attractive to the user.

Linear Combination Splits

Rather than restricting splits to be of the form $X_j \leq s$, one can allow splits along linear combinations 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 (such as the Gini index). While this can improve the predictive power of the tree, it can hurt interpretability. Computationally, the discreteness of the split point search precludes the use of a smooth optimization for the weights. A better way to incorporate linear combination splits is in the hierarchical mixtures of experts (HME) model, the topic of Section 9.5.

Instability of Trees

One major problem with trees is their high variance. Often a small change in the data can result in a very different series of splits, making interpretation somewhat precarious. 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. One can alleviate this to some degree by trying to use a more stable split criterion, but the inherent instability is not removed. It is the price to be paid for estimating a simple, tree-based structure from the data. *Bagging* (Section 8.7) averages many trees to reduce this variance.

Lack of Smoothness

Another limitation of trees is the lack of smoothness of the prediction surface, as can be seen in the bottom right panel of Figure 9.2. In classification with 0/1 loss, this doesn't hurt much, since bias in estimation of the class probabilities has a limited effect. However, this can degrade performance in the regression setting, where we would normally expect the underlying function to be smooth. The MARS procedure, described in Section 9.4,

TABLE 9.3. *Spam data: confusion rates for the* 17*-node tree (chosen by cross–validation) on the test data. Overall error rate is* 9.3%*.*

	Predicted	
True	`email`	`spam`
`email`	57.3%	4.0%
`spam`	5.3%	33.4%

can be viewed as a modification of CART designed to alleviate this lack of smoothness.

Difficulty in Capturing Additive Structure

Another problem with trees is their difficulty in modeling additive structure. In regression, suppose, for example, that $Y = c_1 I(X_1 < t_1) + c_2 I(X_2 < t_2) + \varepsilon$ where ε is zero-mean noise. Then a binary tree might make its first split on X_1 near t_1. At the next level down it would have to split both nodes on X_2 at t_2 in order to capture the additive structure. This might happen with sufficient data, but the model is given no special encouragement to find such structure. If there were ten rather than two additive effects, it would take many fortuitous splits to recreate the structure, and the data analyst would be hard pressed to recognize it in the estimated tree. The "blame" here can again be attributed to the binary tree structure, which has both advantages and drawbacks. Again the MARS method (Section 9.4) gives up this tree structure in order to capture additive structure.

9.2.5 Spam Example (Continued)

We applied the classification tree methodology to the `spam` example introduced earlier. We used the deviance measure to grow the tree and misclassification rate to prune it. Figure 9.4 shows the 10-fold cross-validation error rate as a function of the size of the pruned tree, along with ± 2 standard errors of the mean, from the ten replications. The test error curve is shown in orange. Note that the cross-validation error rates are indexed by a sequence of values of α and *not* tree size; for trees grown in different folds, a value of α might imply different sizes. The sizes shown at the base of the plot refer to $|T_\alpha|$, the sizes of the pruned *original* tree.

The error flattens out at around 17 terminal nodes, giving the pruned tree in Figure 9.5. Of the 13 distinct features chosen by the tree, 11 overlap with the 16 significant features in the additive model (Table 9.2). The overall error rate shown in Table 9.3 is about 50% higher than for the additive model in Table 9.1.

Consider the rightmost branches of the tree. We branch to the right with a `spam` warning if more than 5.5% of the characters are the \$ sign.

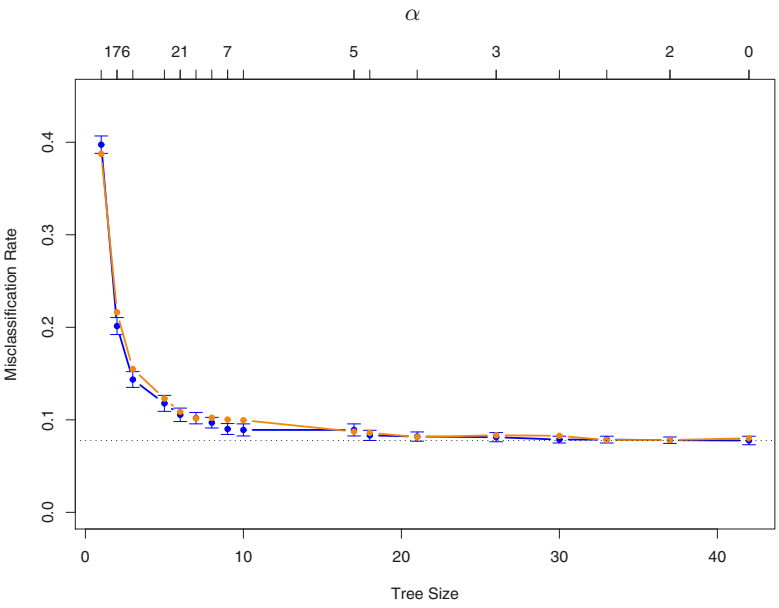

FIGURE 9.4. *Results for* `spam` *example. The blue curve is the* 10*-fold cross-validation estimate of misclassification rate as a function of tree size, with standard error bars. The minimum occurs at a tree size with about* 17 *terminal nodes (using the "one-standard-error" rule). The orange curve 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*.*

However, if in addition the phrase `hp` occurs frequently, then this is likely to be company business and we classify as `email`. All of the 22 cases in the test set satisfying these criteria were correctly classified. If the second condition is not met, and in addition the average length of repeated capital letters `CAPAVE` is larger than 2.9, then we classify as `spam`. Of the 227 test cases, only seven were misclassified.

In medical classification problems, the terms *sensitivity* and *specificity* are used to characterize a rule. They are defined as follows:

Sensitivity: probability of predicting disease given true state is disease.

Specificity: probability of predicting non-disease given true state is non-disease.

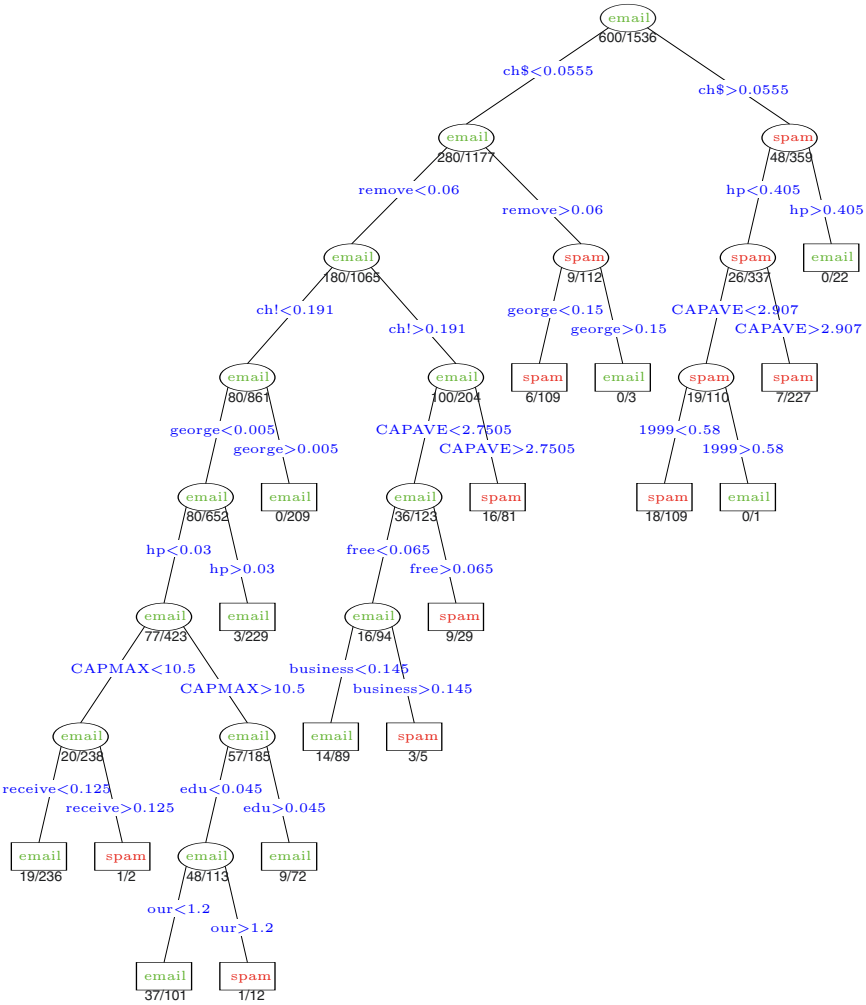

FIGURE 9.5. *The pruned tree for the* `spam` *example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.*

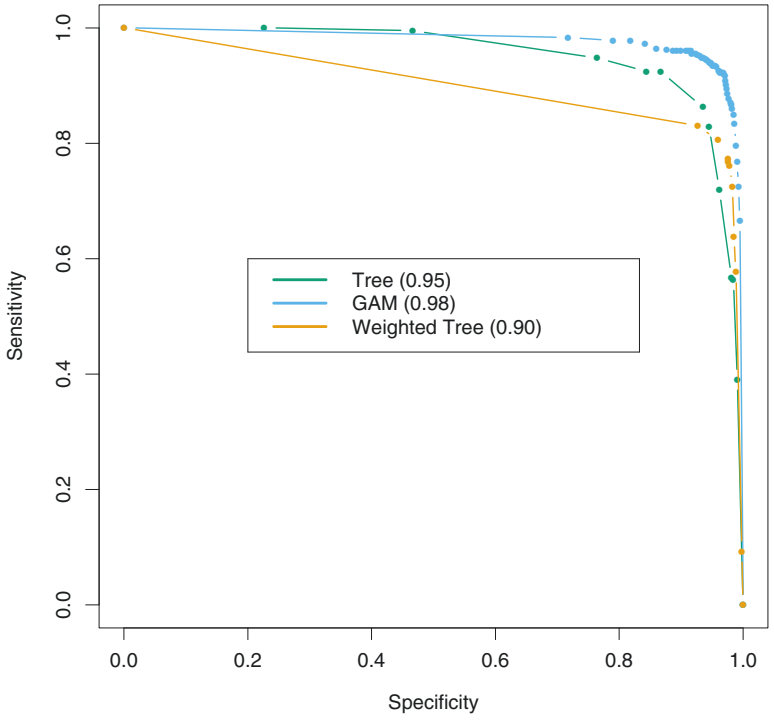

FIGURE 9.6. *ROC curves for the classification rules fit to the* `spam` *data. Curves that are closer to the northeast corner represent better classifiers. In this case the GAM classifier dominates the trees. The weighted tree achieves better sensitivity for higher specificity than the unweighted tree. The numbers in the legend represent the area under the curve.*

If we think of `spam` and `email` as the presence and absence of disease, respectively, then from Table 9.3 we have

$$\begin{aligned} \textit{Sensitivity} &= 100 \times \frac{33.4}{33.4 + 5.3} = 86.3\%, \\ \textit{Specificity} &= 100 \times \frac{57.3}{57.3 + 4.0} = 93.4\%. \end{aligned}$$

In this analysis we have used equal losses. As before let $L_{kk'}$ be the loss associated with predicting a class k object as class k'. By varying the relative sizes of the losses L_{01} and L_{10}, we increase the sensitivity and decrease the specificity of the rule, or vice versa. In this example, we want to avoid marking good `email` as `spam`, and thus we want the specificity to be very high. We can achieve this by setting $L_{01} > 1$ say, with $L_{10} = 1$. The Bayes' rule in each terminal node classifies to class 1 (`spam`) if the proportion of `spam` is $\geq L_{01}/(L_{10} + L_{01})$, and class zero otherwise. The

receiver operating characteristic curve (ROC) is a commonly used summary for assessing the tradeoff between sensitivity and specificity. It is a plot of the sensitivity versus specificity as we vary the parameters of a classification rule. Varying the loss L_{01} between 0.1 and 10, and applying Bayes' rule to the 17-node tree selected in Figure 9.4, produced the ROC curve shown in Figure 9.6. The standard error of each curve near 0.9 is approximately $\sqrt{0.9(1-0.9)/1536} = 0.008$, and hence the standard error of the difference is about 0.01. We see that in order to achieve a specificity of close to 100%, the sensitivity has to drop to about 50%. The area under the curve is a commonly used quantitative summary; extending the curve linearly in each direction so that it is defined over $[0, 100]$, the area is approximately 0.95. For comparison, we have included the ROC curve for the GAM model fit to these data in Section 9.2; it gives a better classification rule for any loss, with an area of 0.98.

Rather than just modifying the Bayes rule in the nodes, it is better to take full account of the unequal losses in growing the tree, as was done in Section 9.2. With just two classes 0 and 1, losses may be incorporated into the tree-growing process by using weight $L_{k,1-k}$ for an observation in class k. Here we chose $L_{01} = 5, L_{10} = 1$ and fit the same size tree as before ($|T_\alpha| = 17$). This tree has higher sensitivity at high values of the specificity than the original tree, but does more poorly at the other extreme. Its top few splits are the same as the original tree, and then it departs from it. For this application the tree grown using $L_{01} = 5$ is clearly better than the original tree.

The area under the ROC curve, used above, is sometimes called the *c-statistic*. Interestingly, it can be shown that the area under the ROC curve is equivalent to the Mann-Whitney U statistic (or Wilcoxon rank-sum test), for the median difference between the prediction scores in the two groups (Hanley and McNeil, 1982). For evaluating the contribution of an additional predictor when added to a standard model, the c-statistic may not be an informative measure. The new predictor can be very significant in terms of the change in model deviance, but show only a small increase in the c-statistic. For example, removal of the highly significant term `george` from the model of Table 9.2 results in a decrease in the c-statistic of less than 0.01. Instead, it is useful to examine how the additional predictor changes the classification on an individual sample basis. A good discussion of this point appears in Cook (2007).

9.3 PRIM: Bump Hunting

Tree-based methods (for regression) partition the feature space into box-shaped regions, to try to make the response averages in each box as differ-