7

Tree-structured Classifiers

The use of tree-based methods for classification is relatively unfamiliar in both statistics and pattern recognition, yet they are widely used in some applications such as botany (Figure 7.1) and medical diagnosis as being extremely easy to comprehend (and hence have confidence in).

The automatic construction of decision trees dates from work in the social sciences by Morgan & Sonquist (1963) and Morgan & Messenger (1973). (Later work such as Doyle, 1973, and Doyle & Fenwick, 1975, commented on the pitfalls of such automated procedures.) In statistics Breiman *et al.* (1984) had a seminal influence both in bringing the work to the attention of statisticians and in proposing new algorithms for constructing trees. At around the same time decision tree induction was beginning to be used in the field of *machine learning*, which we review in Section 7.4, and in engineering (for example, Sethi & Sarvarayudu, 1982).

The terminology of trees is graphic, although conventionally trees such as Figure 7.2 are shown growing down the page. The *root* is the top node, and examples are passed down the tree, with decisions being made at each *node* until a terminal node or *leaf* is reached. Each non-terminal node contains a question on which a split is based. Each leaf contains the label of a classification. A *subtree* of T is a tree with root a node of T; it is a *rooted subtree* if its root is the root of T.

A classification tree partitions the space $\mathscr{X}$ of possible observations into sub-regions corresponding to the leaves, since each example will be classified by the label of the leaf it reaches. Thus decision trees can be seen as a hierarchical way to describe a partition of $\mathscr{X}$. We could give the botanist a description of each species and ask for the description which matches the current specimen. Even in small domains this can be too difficult, and a decision tree provides a structured description of the knowledge base. Often the same information can be structured in

1.	Leaves subterete to slightly flattened, plant with bulb	*2.*
	Leaves flat, plant with rhizome	*4.*
2.	Perianth-tube >10mm	**I. × hollandica**
	Perianth-tube <10mm	*3.*
3.	Leaves evergreen	**I. xiphium**
	Leaves dying in winter	**I. latifolia**
4.	Outer tepals bearded	**I. germanica**
	Outer tepals not bearded	*5.*
5.	Tepals predominately yellow	*6.*
	Tepals blue, purple, mauve or violet	*8.*
6.	Leaves evergreen	**I. foetidissima**
	Leaves dying in winter	*7.*
7.	Inner tepals white	**I. orientalis**
	Tepals yellow all over	**I. pseudocorus**
8.	Leaves evergreen	**I. foetidissima**
	Leaves dying in winter	*9.*
9.	Stems hollow, perianth-tube 4–7mm	**I. sibirica**
	Stems solid, perianth-tube 7–20mm	*10.*
10.	Upper part of ovary sterile	*11.*
	Ovary without sterile apical part	*12.*
11.	Capsule beak 5–8mm, 1 rib	**I. enstata**
	Capsule beak 8–16mm, 2 ridges	**I. spuria**
12.	Outer tepals glabrous, many seeds	**I. versicolor**
	Outer tepals pubescent, 0–few seeds	**I. × robusta**

Figure 7.1: Key to British species of the genus *Iris*. Simplified from Stace (1991) p. 1140, by omitting parts of his descriptions.

other ways. Many botanical trees amount to a set of rules describing one class, so each class is eliminated in turn. Another research area in machine learning has been to induce sets of rules from a training set, either directly or via an induced tree (e.g. Michalski, 1980; Quinlan, 1987a, b, 1993).

The idea of tree induction is to construct a decision tree from a set of examples, which is how humans construct trees. It is usual to do so by growing the tree, that is by successively splitting leaves. Tree construction is easiest when there is an exact partition of $\mathcal{X}$, that is one which classifies every example correctly. The alternative, in which the distributions of observations from the classes overlap, is often called a *noisy* classification problem. For the exact case, we need to continue to grow the tree until every example is classified correctly. In a noisy problem to do so would over-fit the examples at hand, and the two possible strategies are to stop growing the tree early, or to prune the tree after constructing, closely analogous to forwards and backwards selection in regression.

Confusingly, these strategies are sometimes called pre- and post-pruning.

Table 7.1: Example decisions for the space shuttle autolander problem, from Michie (1989).

stability	error	sign	wind	magnitude	visibility	decision
any	any	any	any	any	no	auto
xstab	any	any	any	any	yes	noauto
stab	LX	any	any	any	yes	noauto
stab	XL	any	any	any	yes	noauto
stab	MM	nn	tail	any	yes	noauto
any	any	any	any	Out of range	yes	noauto
stab	SS	any	any	Light	yes	auto
stab	SS	any	any	Medium	yes	auto
stab	SS	any	any	Strong	yes	auto
stab	MM	pp	head	Light	yes	auto
stab	MM	pp	head	Medium	yes	auto
stab	MM	pp	tail	Light	yes	auto
stab	MM	pp	tail	Medium	yes	auto
stab	MM	pp	head	Strong	yes	noauto
stab	MM	pp	tail	Strong	yes	auto

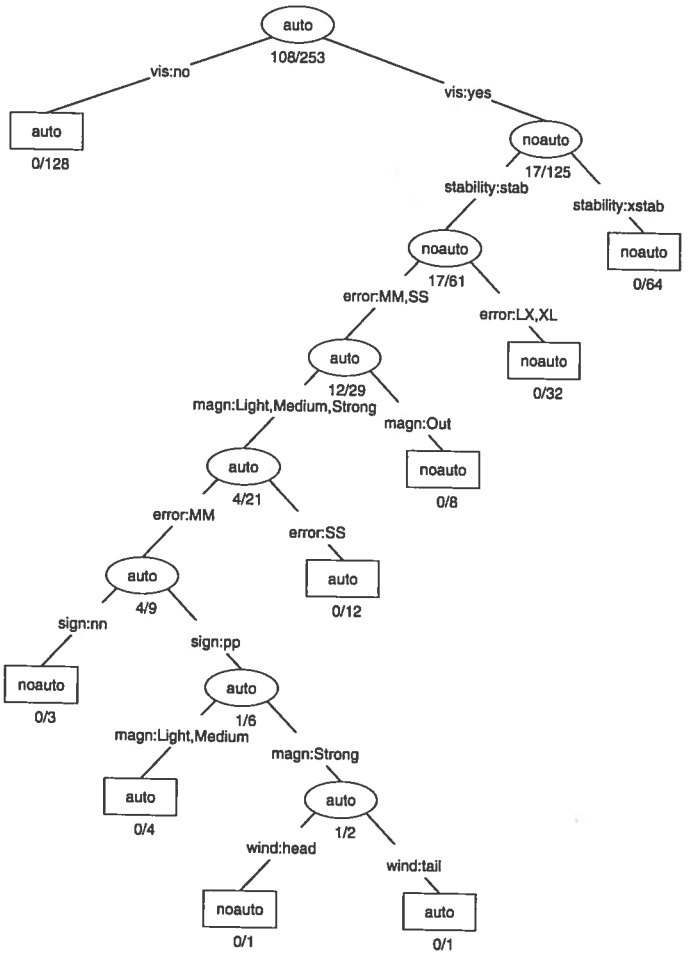

Figure 7.2: Decision tree for shuttle autolander problem. The numbers m/n denote the proportion of training examples reaching that node which are misclassified.

The main differences between algorithms for tree construction are the pruning strategy used and the exact rule for splitting nodes. Many algorithms only allow binary splits, that is to divide a node into two; a few allow multi-way splits (for example by flower colour). Note that these are just algorithms; there are only very simple models and no deep theorems in this field.

There are two types of optimality to be considered. One is optimality of the partition of $\mathcal{X}$, which can be judged by the error rate achieved. In principle we could seek an optimal partition amongst all prescribed partitions of $\mathcal{X}$, for example those representable by a set of decision rules splitting on a single feature. This is a computationally infeasible procedure for all but the smallest problems, but the stepwise construction of a partition by a decision tree can be seen as an approximation to finding the optimal partition.

The other sense of optimality is to represent a partition by a tree in the best possible way. The most obvious criterion is to use the minimal expected number of tests. Hyafil & Rivest (1976) showed this particular problem to be NP-complete; Payne & Meisel (1977) give an algorithm to construct optimal trees with respect to fairly general cost functions.

There are a number of partial surveys of the literature. Dietterich (1990) covers 'recent developments in practical learning algorithms'. Safavian & Landgrebe (1991) is wide-ranging but shallow. Quinlan (1986, 1990, 1993) surveys the machine-learning approaches within his own school.

7.1 Splitting rules

In this section and the next we consider the component pieces of currently favoured tree-construction algorithms. Some historical alternatives are mentioned in Section 7.4. Note that the number of possible trees is vast, so there is no question of an exhaustive search over trees.

Consider first splitting a leaf. There is a set of features from which to construct splitting attributes. For binary features we will clearly consider the binary split on that feature. For categorical features with $L > 2$ levels we can either consider an L-way split, or consider binary splits dividing the levels into two groups. (There will be $2^{L-1} - 1$ non-empty pairs of groups, so this generates many attributes for large L.) For ordered features the natural splits are binary of the form $x \leqslant x_c$; this applies both to continuous measurements and to ordered categories. Some systems also consider linear combinations of continuous features and Boolean combinations of logical ones. (See Section 7.5.)

Each leaf will have a set of *attributes* A on which it might be split. How should we consider the value of the split? There have been many suggestions from several different viewpoints. Consider first a population viewpoint. That is, there is a known probability distribution over $\mathscr{X} \times \mathscr{C}$ of examples which would reach that leaf. This gives a marginal probability distribution p_k over $\mathscr{C}$. Consider splitting on attribute A which has levels $a_1, \ldots, a_m$. There is then a probability distribution p_{ik} over attributes and classes, and the child leaf corresponding to $A = a_i$ would have probability distribution $p(k \mid a_i) = p_{ik}/p_{i\cdot}$ over classes k.

$\mathscr{C}$ is the set of classes.

A $\cdot$ denotes summation over that index.

We can then ask if the child nodes are on average 'purer' than their parent. A measure of impurity should according to Breiman *et al.* (1984, p. 24) be zero if p_j is concentrated on one class, and maximal if p_j is uniform. Two commonly used measures of impurity are the *entropy*

$$i(p) = -\sum_j p_j \log p_j$$

(where $0 \log 0 = 0$) and the *Gini index*

$$i(p) = \sum_{i \neq j} p_i p_j = 1 - \sum_j p_j^2.$$

One interpretation of the Gini index is the expected error rate if the label is chosen randomly from the class distribution at the node. (It may be better to use this than the error rate from the Bayes rule at the node since it gives an element of 'look ahead'. Quite often no feasible split reduces the error rate, yet after two or three splits large reductions in error rate emerge; see the right-hand branch of Figure 7.2.)

The decrease in average impurity on splitting by attribute A is then

$$i(p_c) - \sum_{i=1}^{m} p_{i\cdot} \times i(p(c \mid a_i)).$$

A common approach is to choose the split that maximizes this. Since this will in general favour many-valued attributes, Breiman *et al.* and many others confine attention to binary attributes. (See Section 7.4 for adjustments for multi-way splits.)

Breiman *et al.* preferred the Gini index. The entropy index has been used widely, for example by Sethi & Sarvarayudu (1982) and Quinlan (1983) in the engineering and machine learning literature respectively.

The premise of the following proposition holds for both the entropy and Gini measures of impurity. Part (ii) reduces the number of attributes which need consideration for two classes from $2^{L-1} - 1$ to $L - 1$, but

it has no simple extension to three or more classes. (The result is due to Breiman *et al.*, but the very much shorter proof is original.)

Proposition 7.1 *Suppose $i(p)$ is strictly concave.*

(i) *The decrease in impurity is non-negative, and zero if and only if the the distributions are the same in all children.*

(ii) *Suppose there are two classes. For a categorical feature, order the levels in increasing $p(1 \mid x = x_i)$. Then a split of the form $\{x_1, \ldots x_\ell\}, \{x_{\ell+1}, \ldots, x_L\}$ maximizes the reduction in average impurity.*

Proof: (i) We have by Jensen's inequality

See the glossary.

$$\sum p_i . i(p(c \mid a_i)) \leqslant i(\sum_i p_i . p(c \mid a_i)) = i(p_c)$$

with equality if and only if $p(c \mid a_i) = p(c)$ for all i and c.

(ii) With just two classes we can regard $i(p)$ as a function of p_1 only; it remains strictly concave. Consider dividing into two groups by allocating to group 1 with probability a_i when $x = x_i$. Then

(a) the average impurity of the two groups is minimized by taking $a_i = 0$ or 1 by concavity, and

(b) the partial right derivative of the average impurity with respect to a_i (which exists by concavity) at $a_i = 0$ is of the form

$$p(X = x_i)[Ap(1 \mid x = x_i) + B]$$

for constants A and B, and so is positive (when the optimal solution is to allocate x_i to group 1) for all $i \leqslant \ell$ or all $i > \ell$ for some ℓ.

and both examples lead to the postulated form of split since which group is labelled 1 is arbitrary. □

Another way to look at this approach is to define the average impurity of the tree as

$$I(T) = \sum_{\text{leaves } t} q_t i(p(c \mid t))$$

where q_t is the probability an example reaches node t. The decrease in I on splitting the node is then q_t times the decrease in node impurity we considered before, so that strategy is equivalent to splitting the node to minimize the average tree impurity.

Of course, to use this population approach we estimate all the probabilities by frequencies in the training-set examples reaching the node. Ciampi *et al.* (1987) and Clark & Pregibon (1992) take another approach, viewing the tree as a probability model for the training set. For each node t there is a probability π_{tc} that an example reaches that node and is of class c, which can in principle be computed from the distribution over $\mathscr{X} \times \mathscr{C}$ by partitioning. Suppose we condition on the features for all the examples in the training set. We then know the number n_t of examples which will reach leaf t, and the numbers n_{tc} of each class at that node will have a multinomial distribution with probabilities $\pi_{c|t} = \pi_{tc}/\pi_{t\cdot}$. The conditional likelihood is then proportional to

$$\prod_{\text{leaves } t} \prod_{\text{classes } c} \pi_{c|t}^{n_{tc}}$$

and this allows us to write a deviance for the tree probability model as

$$D(T) = \sum_{\text{leaves } t} D_t\,, \qquad D_t = -2 \sum_{\text{classes } c} n_{tc} \log \pi_{c|t}.$$

(This *is* a deviance, since in the perfect model $\pi_{c|t} = 1$ whenever $n_{tc} > 0$ at a leaf t.) If we estimate π_{tc} by the maximum likelihood estimate $\hat{\pi}_{c|t} = n_{tc}/n_t$, the maximized deviance is

$$D(T) = 2\Big[\sum_t n_t \log n_t - \sum_{t,c} n_{tc} \log n_{tc}\Big].$$

The splitting strategy is to choose the attribute which maximizes the deviance.

Now consider the average tree impurity for the entropy measure. When the probabilities are estimated this is

$$\begin{aligned} I(T) &= \sum_t \frac{n_t}{n} i(n_{tc}/n_t) = -\sum_{t,c} \frac{n_t}{n} \frac{n_{ct}}{n_t} \log \frac{n_{ct}}{n_t} \\ &= -\sum_{t,c} \frac{n_{ct}}{n} \log \hat{\pi}_{c|t} = D(T)/2n \end{aligned}$$

so the splitting strategies for deviances and for entropy-based impurity are identical.

This paragraph is technical and not needed elsewhere.

We can take this duality of approaches further. Many impurity measures can be written as a sum over examples: for example the Gini index is the sum of $(1 - p_c)/n$ where c is the class of the example. Suppose there are n_c examples of class c. Then

$$I = \sum_c n_c(1 - p_c)/n$$

and this is minimized over (p_c) by taking $\hat{p}_c = n_c/n$. Thus the use of the Gini index can be considered as a probability model with a different measure of goodness-of-fit. Chou (1991) considers a larger class of measures of the form

$$I(T) = \sum_{\text{leaves } t} i(t) = \sum_t \frac{n_t}{n} \mathsf{E}\left[\ell(Y, \hat{\mathbf{p}}(t)) \mid t\right]$$

where Y is the class of an example, ℓ is a loss function and $\hat{\mathbf{p}}(t)$ is chosen to minimize the conditional expectation. The Gini index then arises from $\ell(Y, \mathbf{p}) = \|\text{ind}(Y) - \mathbf{p}\|^2 = 1 + \|\mathbf{p}\|^2 - 2p_Y$ (and $\text{ind}(Y)$ is the K-tuple of indicators $Y = k$) and the entropy from $\ell(Y, \mathbf{p}) = -\log p_Y$. Clearly $I(T)$ is always reduced by a split (since there is a $\mathbf{p}(t)$ for each child to minimize over). Further, let

$$d(t, \mathbf{p}(t)) = \mathsf{E}\left[\ell(Y, \mathbf{p}(t)) \mid t\right] - i(t).$$

Then if we consider a binary split over values of a categorical split, it is optimal only if for each category x assigned to the left child t_L $d(x, \hat{\mathbf{p}}(t_L)) \leqslant d(x, \hat{\mathbf{p}}(t_R))$, and conversely for the right child t_R (Chou, 1991). This extends Proposition 7.1, at least for impurity indices of Chou's form.

Priors, weights and costs

There are several assumptions made so far which we may wish to relax. Quite often the training set is not a random sample from the whole population, but chosen to disproportionally represent the classes, especially to over-represent rare classes. Suppose that the classes are known to have probabilities (π_k) in the population, but have n_k representatives out of n in the training set. The population approach works with (p_k), the population distribution of classes within the node t. Clearly n_{tk}/n_t is no longer an appropriate estimate of p_k, and we would use the probability vector proportional to $n_{tk}/n_t \times n\pi_k/n_k$.

Another extension is to allow *weights* to be attached to the examples. The most obvious reason is that we have an integer number w_i of examples like this one, and wish to avoid the overhead of computing with many copies. This suggests interpreting n_{tk} and n_t as the sum of weights, not merely counts of examples. Note that we can incorporate priors for the classes via weights, by giving all examples in class k a weight $n\pi_k/n_k$ (or multiplying the current weight by this factor).

Suppose we wish to attach costs to different misclassifications, say the cost C_{ij} of misclassifying examples of class i as class j. One

approach is to say that the tree construction is merely modelling the posterior probabilities $p(k \mid \mathbf{x})$, and the costs should be used to choose the classification at each node, but not otherwise. However, differential costs suggest that we would like a more accurate model for some classes than for others. Breiman *et al.* (1984, §4.4) suggest that this can sometimes be incorporated into the impurity index. For example, the interpretation given for the Gini index suggests the modified form

$$i(p) = \sum_{i \neq j} C_{ij}\, p_i p_j.$$

Unfortunately, this effectively symmetrizes the costs (since the coefficient of $p_i p_j$ is $C_{ij} + C_{ji}$) and so is completely ineffective in two-class problems. It can also fail to be concave, and so give rise to splits with negative 'decreases' in impurity.

For two classes there is a simple approach to misclassification costs. Each example in class 2 costs a factor C_{21}/C_{12} more to misclassify than an example in class 1, which suggests weighting the examples in class i by C_{ij} for $j \neq i$. This will also be appropriate for more classes if the misclassification costs depends only on i and not on $j \neq i$.

How about the deviance approach? We use the weights for each example to weight the log-likelihood; the deviance becomes

$$D(T) = \sum_{\text{leaves } t} D_t, \qquad D_t = -2 \sum_{\text{classes } c} n_{tc} \log \pi_{c|t}.$$

where n_{tc} now represents the sum of the weights of examples reaching leaf t of class c. (This is certainly appropriate if weights represent multiple examples.) We will once again estimate $\pi_{c|t}$ by n_{tc}/n_t, but this is an estimate of the biased posteriors, and will be adjusted to be proportional to $n_{tc}/n_t \times n\pi_k/n_k$ to estimate the posteriors in the population. Note that the latter is what we get if we weight examples in class k by $n\pi_k/n_k$.

7.2 Pruning rules

There are $\lfloor 1.5028369^{\ell} \rfloor$ rooted subtrees of a binary tree with ℓ leaves; Breiman *et al.* (1984, p. 284).

The number of rooted subtrees of a binary tree is very large so we need a way to navigate this family efficiently.

Cost-complexity pruning

The best-known procedure for tree pruning is that proposed by Breiman *et al.* (1984). Let $R(T)$ be a measure of a tree formed by adding the

contributions from the leaves. One obvious candidate is the number of misclassifications on the training set or a test set; another is the entropy or deviance of the partition. Let the *size* of a tree be the number of leaves. (For a binary tree the total number of nodes is twice the size minus one.) Then Breiman *et al.* (1984) proposed choosing a rooted subtree T of the full tree T_0 which minimizes

$$R_\alpha(T) = R(T) + \alpha \,\text{size}\,(T).$$

If we have weights, we would use these in calculating $R(T)$. To handle missing values we will need a modest extension, in which $R(T)$ also contains contributions from all nodes. Precisely, $R(T)$ is assumed to be a sum over leaves plus a sum over non-leaves, the summands being different in the two cases.

We can also consider $R_\alpha(T)$ as the sum of $R(t) + \alpha$ over the leaves of T. This can be seen as using a Lagrange multiplier for size, so finding the minimizing trees for all α is equivalent to finding the trees with minimum $R(T)$ for each size. (Our results are equally valid for other measures of size such as the total costs of the tests at the nodes.)

When using the apparent error rate on the training set we will want to choose a positive α to penalize size, but our results also apply to $\alpha = 0$ which would be appropriate with the error rate on a test set. Ciampi *et al.* (1987) consider pruning with the Akaike Information Criterion which corresponds to taking $R(T)$ as the deviance and $\alpha = 2(K-1)$. (The AIC penalizes minus the log-likelihood by the number of parameters. Estimating the probability distribution within a leaf takes $K-1$ parameters. This count ignores parameters in the splitting attribute and the selection of the attribute itself; it is unclear how these should be counted.)

Breiman *et al.* showed that there is a nested family of subtrees T_k of $T_0 = T$ such that each is optimal for a range of α, and so there are values

$$-\infty = \alpha_0 < \alpha_1 < \cdots \infty$$

such that T_i is an optimal tree for $\alpha \in [\alpha_i, \alpha_{i+1})$. Further, they gave an algorithm to construct the tree sequence (T_k). Often $\alpha_1 \geqslant 0$, for example if $R(T)$ is the measure used to grow the tree (such as deviance or Gini) or the error rate on the training set (from Proposition 7.5). However, $\alpha_1 = 0$ is quite common.

We will now prove these results. There can be a number of trees with the same value of $R_\alpha(T)$; we will consider only one which is a subtree of all to be optimal, and if this exists we call it $T(\alpha)$. Consider a non-trivial tree T, and for any non-terminal node t let T_t be the subtree rooted at that node. Let

$$g(t, T) = \frac{R(t) - R(T_t)}{\text{size}\,(T_t) - \text{size}\,(t)}$$

which compares the reduction in R by including the subtree with the increase in size. Note that $g(t, T) > \alpha$ if and only if $R_\alpha(t) > R_\alpha(T_t)$. The effect of pruning at node t is to replace T_t by t.

Proposition 7.2 *Suppose we number the nodes of a tree T so that each node precedes its parent. If we visit the nodes in this order (bottom-up) and prune at node t if $R_\alpha(t) \leqslant R_\alpha(T'_t)$ for the current tree T', the result is $T(\alpha)$.*

This was proposed as a new algorithm by Gelfand & Delp (1991), Gelfand *et al.* (1991) and Guo & Gelfand (1992), the latter including a more complex proof. However, the algorithm is implicit in earlier work, and explicit, without proof, in Quinlan (1987a), under the name of *reduced error pruning*. It follows immediately from Theorems 10.7 and 10.10 of Breiman *et al.* (1984).

Proof: We will establish by induction that when node t is considered all the branches at t are optimally pruned. This is clearly true for the leaves. At node t we either prune with value $R_\alpha(t)$ or not with value $R_\alpha(T'_t) = \sum_{\text{branches } B} R_\alpha(T'_B)$ if this is strictly smaller. If there is a subtree T'' rooted at t with a smaller value of R_α it must be non-trivial, and there must be a branch B with $R_\alpha(T''_B) < R_\alpha(T'_B)$ and so T'_B is not optimally pruned, a contradiction. Now suppose there is another subtree with the same value of R_α. Then each of its branches (it must have some) will have the same value of R_α as the corresponding branch of T'_t and so include that branch. Thus after node t is considered, the current T'_t is optimally pruned. When the root is reached the current tree is optimally pruned, so is $T(\alpha)$. □

This gives an algorithm to find $T(\alpha)$ for a single α. We now show how to find (α_k) and the tree sequence T_k. From now on we assume that size is increasing, that is adding nodes increases (weakly) the size.

Proposition 7.3 *Let α_1 be the smallest value of $g(t, T)$ for any non-terminal node t of T. The optimally pruned tree is T for $\alpha < \alpha_1$, and $T_1 = T(\alpha_1)$ is obtained by pruning at all nodes t with $g(t, T) = \alpha_1$. Further, $g(t, T_1) > \alpha_1$ for all non-terminal nodes of T_1.*

Proof: The optimality of T for $\alpha < \alpha_1$ is immediate from $R_\alpha(t) > R_\alpha(T_t)$ and Proposition 7.2. Consider $\alpha = \alpha_1$, and pruning by Proposition 7.2. Whenever the tree is pruned, $R_\alpha(T_s)$ is unchanged for all nodes s of the new tree. Thus $R_\alpha(t) \leqslant R_\alpha(T'_t)$ for the current tree T' if and only if $R_\alpha(t) \leqslant R_\alpha(T_t)$ if and only if $g(t, T) \leqslant \alpha_1$. Then for a retained node t,

$$\begin{aligned} R_{\alpha_1}(t) - R_{\alpha_1}(T_{1t}) &= R_{\alpha_1}(t) - R_{\alpha_1}(T_t) + [R_{\alpha_1}(T_t) - R_{\alpha_1}((T_1)_t)] \\ &= R_{\alpha_1}(t) - R_{\alpha_1}(T_t) = g(t, T)\big[\text{size}(T_t) - \text{size}(t)\big] \\ &> \alpha_1\big[\text{size}(T_t) - \text{size}(t)\big] \geqslant \alpha_1\big[\text{size}((T_1)_t) - \text{size}(t)\big] \end{aligned}$$

so $g(t, T_1) > \alpha_1$. □

Proposition 7.4 *For $\beta > \alpha$ $T(\beta)$ is a subtree of $T(\alpha)$ and is the result of β-pruning of $T(\alpha)$.*

Proof: We will show by induction that $T_t(\beta)$ is a subtree of $T_t(\alpha)$ and conclude that $T(\beta)$ is a subtree of $T(\alpha)$. This is true at the leaves. At node t we compare $R_\alpha(t)$ to $R_\alpha(T_t(\alpha))$ and $R_\beta(t)$ to $R_\beta(T_t(\beta))$ and in each example prune if the first is (weakly) smaller. We must show that if $R_\alpha(t) \leqslant R_\alpha(T_t(\alpha))$ then $R_\beta(t) \leqslant R_\beta(T_t(\beta))$. Now since $T_t(\beta)$ is a candidate for α-pruning of the tree rooted at t, we have

$$\begin{aligned} R_\beta(t) &= R_\alpha(t) + (\beta - \alpha)\text{size}(t) \leqslant R_\alpha(T_t(\alpha)) + (\beta - \alpha)\text{size}(t) \\ &\leqslant R_\alpha(T_t(\beta)) + (\beta - \alpha)\text{size}(t) \\ &= R_\beta(T_t(\beta)) - (\beta - \alpha)[\text{size}(T_t(\beta)) - \text{size}(t)] \\ &\leqslant R_\beta(T_t(\beta)). \end{aligned}$$

Since $T(\beta)$ minimizes $R_\beta(T')$ over all rooted subtrees T' of T and is a subtree of $T(\alpha)$, it also minimizes $R_\beta(T')$ over rooted subtrees of $T(\alpha)$. □

The algorithm of Proposition 7.3 can be applied to the new tree $T_1 = T(\alpha_1)$ to find $\alpha_2 > \alpha_1$ (since $g(t, T_1) > \alpha_1$ for all non-terminal nodes of T_1) and $T_2 = T(\alpha_2)$ and so on until T_k is the trivial tree, the root of $T_0 = T$. From Propositions 7.3 and 7.4, $T(\alpha) = T_1$ for $\alpha_1 \leqslant \alpha < \alpha_2$ and $T(\alpha_2) = T_2$. Repeating the process gives $T(\alpha)$ for all $\alpha \leqslant \alpha_k$, and Proposition 7.4 shows that the trivial tree is optimal for $\alpha \geqslant \alpha_k$. This completes the algorithm to find the tree sequence:

1 Set $k = 0$ and write out $T_0 = T$.

2 Set $\alpha = \infty$.

3 Visit the non-terminal nodes t in bottom-up order and calculate $R(T_t)$ and $\text{size}(T_t)$ by summing over the descendants (and including any contribution at t). Set

$$g(t) = \frac{R(t) - R(T_t)}{\text{size}(T_t) - \text{size}(t)}$$

and $\alpha = \min(\alpha, g(t))$.

4 Visit the nodes in top-down order and prune whenever $g(t) = \alpha$.

5 Set $k = k + 1$ and write out $\alpha_k = \alpha$ and $T_k = T$.

6 If T is non-trivial go to 2.

We could visit the nodes in any order at step 4, but a top-down order avoids considering nodes which themselves will be pruned away.

In fact Breiman averaged only a values ($\sqrt{\alpha_k \alpha_{k+}}$ the sequence α_k original tree, bu saves but little

It remains to choose the particular tree within this sequence. If we have a *validation* set we can use its error rate with $\alpha = 0$. Otherwise Breiman *et al.* propose selecting the value of α using *cross-validation*. The training set is split into V parts; Breiman *et al.* (1984, pp. 11–12) seem to prefer 3 but later users (e.g. Clark & Pregibon, 1992) recommend 10. For each of the parts, a tree sequence is constructed from the remaining $V-1$ parts of the training set, and its measures $R(T_k)$ calculated, to give a piecewise constant function for $R(T(\alpha))$. This is averaged over all V parts, and α chosen to minimize the function.

In fact Breiman *et al.* averaged only at the values $(\sqrt{\alpha_k \alpha_{k+1}})$ for the sequence α_k for the original tree, but this saves but little effort.

Because the training set is disjoint from the test set in each of the V cross-validation experiments, we can expect to form a reasonably unbiased estimate of $R(T(\alpha))$. If V is small we have used a considerably smaller training set, and so might expect to overestimate the error rates, but this does not necessarily mean that the relative values of $R(T(\alpha))$ for different α are seriously biased. In practice the estimates of $R(T(\alpha))$ are highly variable over the choice of parts of the training set, and the estimated function may have no minimum within the range of α considered, or a very broad minimum. Breiman *et al.* suggest choosing the largest value of α with the cross-validated $R(T(\alpha))$ just above the minimum (the 'one SE' rule). The standard error can be estimated from a binomial distribution for error-count pruning, or a chi-squared distribution for deviance pruning.

There is a difficulty with cross-validating deviance measures $R(T)$ not found with error rates nor the Gini measure. Suppose that at some leaf t a class c occurs in the test set but not in the training set. Then the fitted probability $\hat{\pi}_{tc} = 0$ and so the deviance at that leaf is infinite. (The other measures give a unit penalty.) This might be thought appropriate, and will certainly lead to that leaf being pruned, but makes it difficult to average $R(T(\alpha))$. There are several *ad hoc* solutions, all of which involve altering the fitted probabilities. One we have used successfully is to give a prior of one example per class at each node so $\hat{\pi}_{tc} = (n_{tc} + 1)/(n_t + K)$ for K classes, which is never zero, but can approach zero if a class does not occur in a large number of examples.

This is one of a family of *shrinking* approaches. Bahl *et al.* (1989), Chou (1991) and Buntine (1992) each smooth at all splits, not just the leaves, taking the fitted probabilities to be a convex combination of those of the parent and the frequencies in the child node. (Clark & Pregibon, 1992, also propose this.) It remains to choose the convex combination, and indeed to decide if it should be the same at each

node. Chou uses leave-one-out cross-validation at each node; Buntine uses a combination which depends on the sample size (see Section 7.7). Clark & Pregibon use a constant factor over the tree, chosen by cross-validation, and see this as an alternative to pruning.

Another approach to pruning

Gelfand *et al.* (1991) and Gelfand & Delp (1991) point out that the optimally pruned tree with respect to the true misclassification rate $R(T)$, were this available, need not be within the family $T(\alpha)$ pruned with respect to the apparent error rate. We have already seen how to prune with respect to an honest measure of error rate. Gelfand *et al.* propose a pruning algorithm based on dividing the training set into two and alternating the role of the halves. Initially the tree is grown (and nodes labelled) using one half and pruned using the error rate on the other half. The tree is then re-grown from the pruned tree using the previous test half and pruned using the previous training half to estimate the error rate. This is repeated until the tree size is unchanged. The pruned subtrees are nested and increasing, and if a node is terminal at two successive steps growth from that node can be stopped.

Long formal proofs are given in Gelfand *et al.* (1991) for the Gini measure of impurity. We can give a short and general argument. It is important here that ties are broken consistently when labelling leaves; we need to choose the class of the parent node if this is a contender.

Proposition 7.5 *Suppose a training set is partitioned, and the whole set and each cell of the partition are labelled by a class with the highest frequency within it. Then the apparent error rate is decreased by partitioning, strictly so unless the whole partition is given the same class.*

Proof: Let the frequency of class c within cell i be n_{ci}. Then the success count before division is $\max_c n_{c\cdot}$, maximized by k, say, and after division is $\sum_i \max_c n_{ci} \geqslant \max_c \sum_i n_{ci} = n_{k\cdot}$ with equality only if k maximizes n_{ci} for each class. If the error rate is the same, the class of the parent is a contender for the class of each cell. □

Suppose the two halves of the training set are $\mathcal{T}_1$ and $\mathcal{T}_2$, and let $R^{(i)}(T)$ denote the number of errors for test set $\mathcal{T}_i$, using the labels assigned when the tree was grown. Let T^* denote the tree grown using $\mathcal{T}_1$ and optimally pruned using $R^{(2)}$.

Proposition 7.6 *The tree T^* is unchanged under optimal pruning using $R^{(1)}$.*

Proof: From Proposition 7.2 it suffices to show that $R^{(1)}(T_t) < R^{(1)}(t)$ for each interior node. Now T_t corresponds to a partition of the examples of $\mathscr{T}_1$ reaching node t, so by Proposition 7.5 $R^{(1)}(T_t) \leqslant R^{(1)}(t)$ with equality only if T_t gives the same partition as t and hence $R^{(2)}(T_t) = R^{(2)}(t)$ which would contradict the optimality of the pruning of T^*. □

Now suppose a tree S is grown starting from T^* using $\mathscr{T}_2$ and optimally pruned to S^* using $R^{(1)}$. Since S contains T^*, Proposition 7.6 shows that S^* contains T^* (since pruning is monotone on trees). If a leaf in T^* remains a leaf in S^* the growing and pruning process to form T^* will be repeated at the next step, and so that node will always remain a leaf. As there are only a finite number of examples and empty leaves will never be generated, the process must stop.

Gelfand *et al.* (1991) propose reporting an error rate based on classifying at each leaf the examples from the half of the training set other than the one on which that leaf was labelled (when it was grown). They recommend this procedure only for large datasets (since it works with half the data at a time), and we have found it unsatisfactory for moderately sized datasets, in which T^* is often just the root subtree.

'Pessimistic' and 'error-based' pruning

Quinlan (1987a, 1993) introduced two much cruder ideas for pruning. In the first approach, he proposes a continuity correction for cost-complexity pruning, so that the number of errors on the training set at each node is increased by one half. The idea was to better estimate the true rather than apparent error rate. (This idea is exactly equivalent to taking $\alpha = 0.5$, since $R(t)$ is increased by one half.) He compares the error rate of the tree T_t with the error rate at node t (after a 'continuity correction' adding one half to each error count). Rather than prune only if the adjusted error rate for node t is smaller, he proposes to prune unless it is somewhat larger, specifically when

$$\text{error rate for } t < \text{error rate for } T_t + \text{std. dev.(error rate for } T_t).$$

(Quinlan is vague about how to calculate the last term; his example appears to use a binomial formula with a common probability, but it would be better to calculate the standard deviation within each leaf.) This looks like an approximation to a significance test, except that the variability of the left-hand side is not taken into account. Again the details are vague, but Quinlan states that all subtrees are considered as

candidates for pruning (unlike Proposition 7.2). As there are very many such subtrees, this seems unlikely.

A much larger adjustment of the apparent error rate is proposed in his 1993 book. Suppose a leaf t covers N examples, J of which are misclassified. Then $R(t)$ is taken to be the 87.5% point of a binomial$(N, J/N)$ distribution. This could be calculated exactly (and is in the C4.5 program) but given the approximate nature of the justification, using a normal approximation to the binomial to give

$$R(t) = J + 1.15 \times \sqrt{J(1 - J/N)}$$

seems perfectly adequate. Our understanding is that the algorithm of Proposition 7.2 is used.

Examples

The data on diabetes amongst Pima Indians have provided a difficult example for many methods, and is typical of the difficulty of using tree-based methods. It is easy to grow a tree (using the deviance/entropy approach) with many nodes: our initial tree has 22 nodes, shown in Figure 7.3. (One of the splits has another attribute with exactly the same split.) Four nodes can be pruned without changing the classifications at all; the error rate on the test set is 81/332. This is just about significantly worse than the logistic regression with 66/332, as the McNemar statistic is 1.96.

Figure 7.4 shows the difficulty in choosing the size by 10-fold cross-validation of error-rate pruning. There is little variation with the size of tree down to size 3, which suggests the latter should be adopted. This gives the rule that diabetes should be predicted if the plasma glucose level exceeds 123.5 and the diabetes pedigree function exceeds 0.31. This rule has a test-set error rate of 90/332, worse than the unpruned tree. However, the difference is not statistically significant, for McNemar's test statistic is $[|29 - 20| - 1]/\sqrt{29 + 20} \approx 1.14$. In this circumstance we should not use risk averaging, as most examples are predicted with $\max \widehat{p}(c \mid x) = 1$, and so the true test-set error rate is dramatically underestimated (11% instead of 24.4%).

Quinlan's 'pessimistic' pruning removes just 3 nodes, and AIC removes just one. Indeed, if we grow an even larger tree by allowing smaller populations within the leaves, both Quinlan and AIC select a tree with about 30 nodes. Such large trees have a test-set error rate of about 99/332.

Using the Gini index rather than entropy to grow the tree produced a similar but not identical tree, with splits occurring in a slightly different

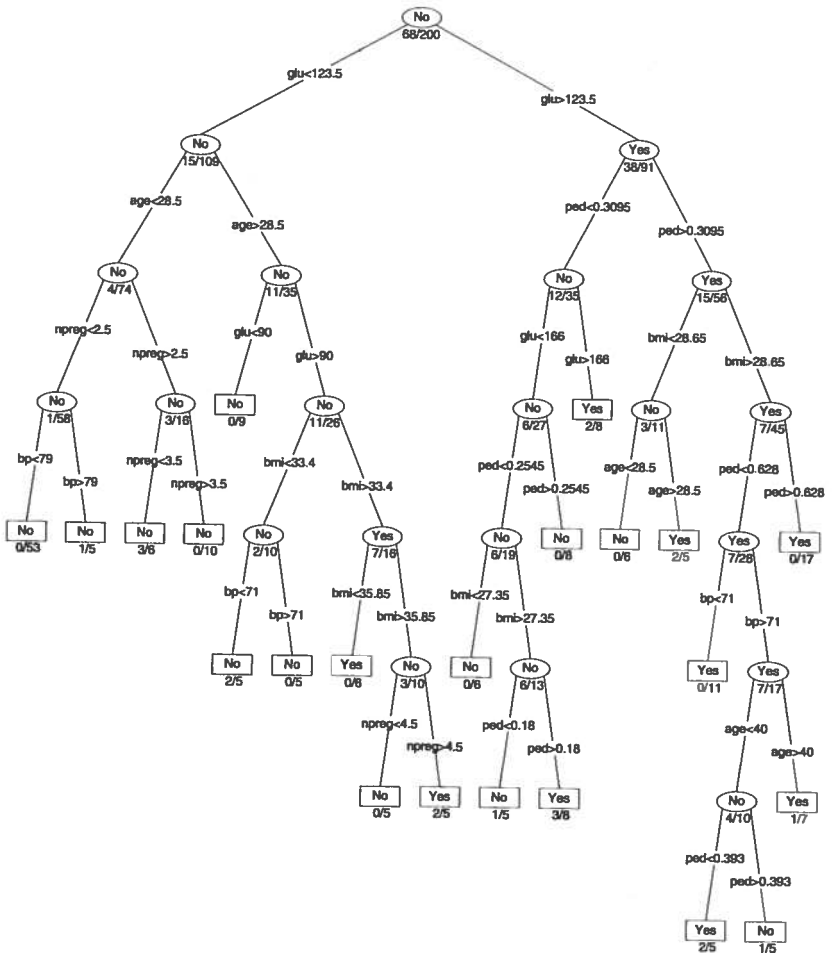

Figure 7.3: The classification tree grown for the Pima Indians diabetes data based on a training set of size 200. Growth was stopped at leaves with 10 or fewer examples.

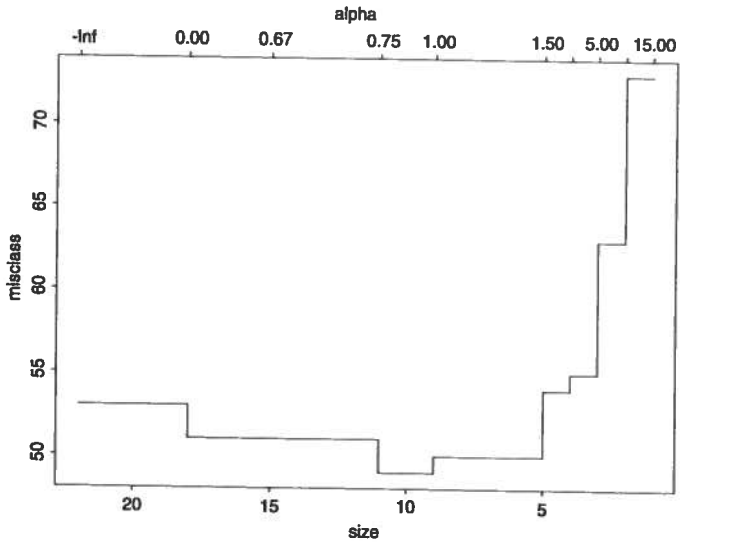

Figure 7.4: Number of errors *vs* size for error-rate pruning of the tree of Figure 7.3.

order. The Gelfand *et al.* (1991) procedure depends on the (random) division into two sets, but normally produced a tree with around 15 nodes, and a test-set error rate of around 85/332.

For the forensic glass data, growing an initial tree using the entropy/deviance measure and using cost-complexity pruning on the cross-validated error rate gives the plot shown in Figure 7.5. This suggests choosing a tree of size 12, or 9 using the '1 SE' rule as these counts are approximately Poisson and so have a standard error of about 8.

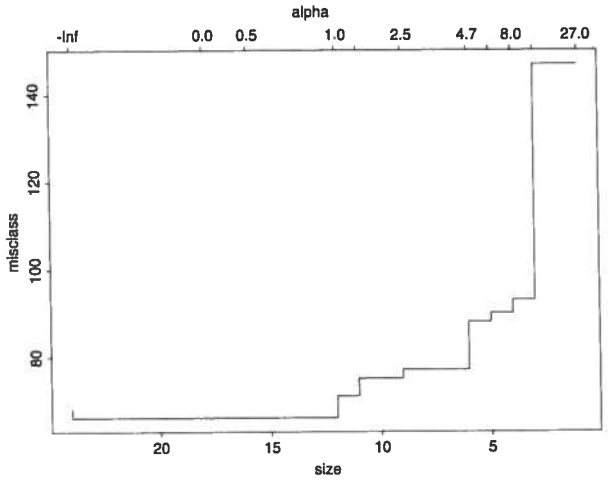

Figure 7.5: Cross-validated error count *vs* tree size for the forensic glass data.

It is tedious to cross-validate this choice of tree size within a cross-validatory assessment of performance, and we introduce a slight bias by not doing so here. (The cross-validation partition chosen for cost-complexity pruning was not the same as that used for assessment.) The performance of the trees pruned to size 9 and 12 were almost identical; size 9 gave the estimated confusion matrix

	WinF	WinNF	Veh	Con	Tabl	Head
WinF	55	13	2	0	0	0
WinNF	14	50	6	3	2	1
Veh	5	8	4	0	0	0
Con	0	3	0	9	0	1
Tabl	0	1	0	1	5	2
Head	2	2	0	2	1	22

and an error rate of 32.2%.

The whole Gelfand *et al.* (1991) procedure was cross-validated. On the whole dataset it grew a tree with 6 nodes that did not classify as container or tableware at all, just using the refractive index and magnesium and calcium oxides. Under cross-validation the tree size

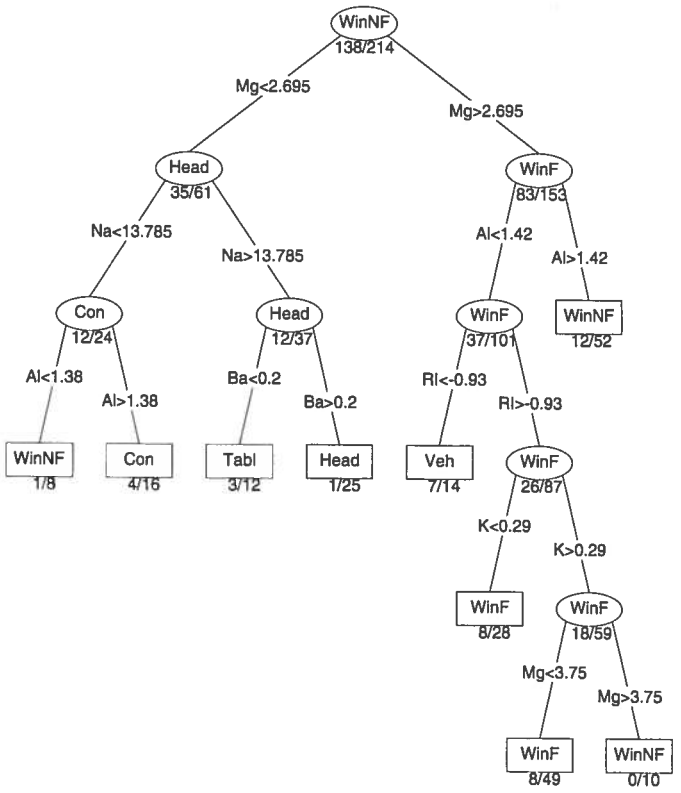

Figure 7.6: Pruned classification tree for the forensic glass data.

ıted error
e size for
glass data.

varied widely between subsets; the assessment of the error rate was 42%.

The Quinlan pruning procedure tended to prune lightly; its cross-validated error rate was 31%.

7.3 Missing values

One attraction of tree-based methods is the ease with which missing values can be handled. Consider the botanical key of Figure 7.1. We only need to know about a small subset of the 10 observations to classify any example, and part of the art of constructing such trees is to avoid observations which will be difficult or missing in some of the species (or, as in the case of capsules, for some of the examples). However, missing values may be unavoidable, and there are several approaches to handling them.

1 A general strategy is to 'drop' an example down the tree as far as it will go. If it reaches a leaf we can predict y for it. Otherwise we use the distribution at the node reached to predict y, as shown in