# 30. Tree-Based Methods and Their Applications

The first part of this chapter introduces the basic structure of tree-based methods using two examples. First, a classification tree is presented that uses e-mail text characteristics to identify spam. The second example uses a regression tree to estimate structural costs for seismic rehabilitation of various types of buildings. Our main focus in this section is the interpretive value of the resulting models.

This brief introduction is followed by a more detailed look at how these tree models are constructed. In the second section, we describe the algorithm employed by classification and regression tree (CART), a popular commercial software program for constructing trees for both classification and regression problems. In each case, we outline the processes of growing and pruning trees and discuss available options. The section concludes with a discussion of practical issues, including estimating a tree's predictive ability, handling missing data, assessing variable importance, and considering the effects of changes to the learning sample.

The third section presents several alternatives to the algorithms used by CART. We begin with a look at one class of algorithms – including QUEST, CRUISE, and GUIDE– which is designed to reduce potential bias toward variables with large numbers of available splitting values. Next, we explore C4.5, another program popular in the artificial-intelligence and machine-learning communities. C4.5 offers the added functionality of converting any tree to a series of *decision rules*, providing an alternative means of viewing and interpreting its results. Finally, we discuss chi-square automatic interaction detection (CHAID), an early classification-tree construction algorithm used with categorical predictors. The section concludes with a brief comparison of the characteristics of CART and each of these alternative algorithms.

In the fourth section, we discuss the use of ensemble methods for improving predictive ability. Ensemble methods generate collections of trees using different subsets of the training data. Final predictions are obtained by aggregating over the predictions of individual members of these collections. The first ensemble method we consider is boosting, a recursive method of generating small trees that each specialize in predicting cases for which its predecessors perform poorly. Next, we explore the use of random forests, which generate collections of trees based on bootstrap sampling procedures. We also comment on the tradeoff between the predictive power of ensemble methods and the interpretive value of their single-tree counterparts.

The chapter concludes with a discussion of tree-based methods in the broader context of supervised learning techniques. In particular, we compare classification and regression trees to multivariate adaptive regression splines, neural networks, and support vector machines.

## 30.1 Overview

Given a data set for a particular application, a researcher will typically build a statistical model with one (or both) of the following objectives in mind: (1) to use information from this data to make useful predictions about future observations, and (2) to gain some insights into the underlying structure of the data. Tree-based models are attractive because of their potential to blend both of these characteristics quite effectively.

Tree-based models comprise one set of tools useful for supervised learning tasks. In supervised learning problems, a researcher is trying to use a set of *inputs*, or *independent variables*, to predict an *output*, or *dependent variable*. If the output is a categorical variable, we call this a problem of classification. On the other hand, if the output is a continuous variable, we call this a problem of regression.

Tree-based models approach these problems by recursively partitioning a learning sample over its input variable space and fitting a simple function to each resulting subgroup of cases. In classification, this function is assignment to a single category; in regression, the function could be a constant. We shall discuss several tree-fitting procedures in detail throughout this chapter.

To see the tree-based models at work, we present two applications in this section.

### 30.1.1 Classification Example: Spam Filtering

First, we consider the task of designing an automatic spam (junk e-mail) filter. The data for this task are publicly available from the University of California, Irvine (UCI) machine learning repository [30.1], and were donated by George Forman from Hewlett–Packard laboratories in Palo Alto, California.

The data consist of 58 variables describing 4601 messages. The dependent variable indicates whether or not each message is spam. The 57 predictor variables are all continuous, and describe the relative frequencies of various keywords, characters, and strings of consecutive uppercase letters. A resulting tree model is shown in Fig. 30.1, and the variables present in the tree are summarized in Table 30.1.

In Fig. 30.1, we see that the messages are first partitioned based on the frequency of the "$" character. Messages with few dollar signs are sent down the left branch, and messages with many dollar signs are sent down the right branch. Following the right branch, we find that those messages with many dollar signs are further partitioned based on the frequency of the word "hp". If "hp" appears only infrequently, the message is classified as spam, otherwise, it is classified as a legitimate message. This splitting makes sense in the context of this data set, because messages containing "hp" most likely address company business.

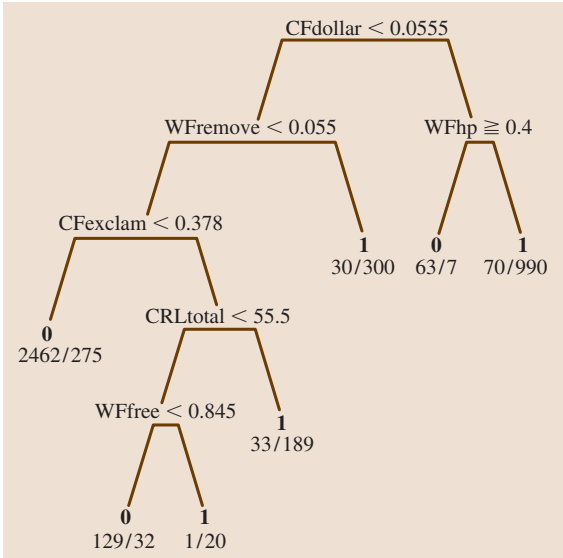



**Fig. 30.1** A classification tree for the spam filtering data. Terminal nodes labeled "1" are classified as spam, and those labeled "0" are classified as non-spam

Following the left branch from the root (top) node, we find that a message will be classified as spam if it has a high frequency of the word "remove", or a combination of exclamation points and either the word "free" or many strings of uppercase letters. The structure of the tree is consistent with our intuition about the message characteristics that separate spam from legitimate e-mail.

**Table 30.1** Electronic mail characteristics

| Variable | Definition |
|---|---|
| spam | 1 if spam, 0 if not |
| CFdollar | percent of "$" characters in message |
| CFexclam | percent of "!" characters in message |
| CRLtotal | sum of lengths of uppercase letter strings |
| WFfree | percent of "free" words in message |
| WFhp | percent of "hp" words in message |
| WFremove | percent of "remove" words in message |

**Table 30.2** Seismic rehabilitation cost-estimator variables

| Variable | Definition |
|---|---|
| AGE | Year of construction |
| AREA | Building area in square feet |
| MODELC1 | Building has concrete moment frame (yes = 1, no = 0) |
| MODELC3 | Building has concrete frame w/ infill walls (yes = 1, no = 0) |
| MODELS5 | Building has steel frame w/ infill walls (yes = 1, no = 0) |
| MODELURM | Building has unreinforced masonry (yes = 1, no = 0) |
| MODELW1 | Building has light wood frame (yes = 1, no = 0) |
| POBJ_DC | Performance objective is damage control (yes = 1, no = 0) |
| POBJ_IO | Performance objective is immediate occupancy (yes = 1, no = 0) |
| POBJ_RR | Performance objective is risk reduction (yes = 1, no = 0) |
| SEISMIC | Location seismicity on a scale from 1 (low) to 7 (very high) |

We have chosen a small tree for the sake of illustration. For this particular application, one may consider competing methods such as logistic regression or logistic regression trees described in Chapt. 29 by Loh; also see *Chan* and *Loh* [30.2]. Having seen a successful classification example, we now examine a regression tree application.

### 30.1.2 Regression Example: Seismic Rehabilitation Cost Estimator

The seismic rehabilitation cost estimator is an online program developed by the Federal Emergency Management Agency (FEMA) [30.3, 4] that enables calculation of structural cost estimates for seismic rehabilitation of buildings. A group of structural engineers collaborated with us to develop two tree models based on data from over 1900 seismic rehabilitation projects.

The first model is designed for use early in developing budget estimates when specific building details are not yet available. This model requires information about a building's original year of construction, its size, its structural system, the seismic zone in which it resides, and the rehabilitation performance objective. We summarize the 11 relevant predictor variables in Table 30.2.

The regression tree is presented in Fig. 30.2. We see that the first split is based on the building's original date of construction. As one might expect, rehabilitation tends to be more costly for older buildings. Regardless of the age of the building, the cost estimate is refined based on the purpose of the rehabilitation effort. Far more expense is required to prepare a building for immediate occupancy than for other purposes. Further down the tree, these cost estimates may be adjusted based on the building's structural characteristics, size, and location.

The second model is used to refine these estimates as more comprehensive data become available. In addition to the basic information included in the smaller model, this larger model uses information about occupancy, number of floors, diaphragm type, the rehabilitation project scope, and other details. More detailed information about the data set used to build these models can be found in FEMA [30.3, 4].

### 30.1.3 Outline

In the rest of this chapter, we will review various tree-based methods for classification and prediction. Section 30.2 details the classification and regression trees (CART) method [30.5] and discusses issues common to all tree-building algorithms. Section 30.3 outlines competing methods, including QUEST [30.6], CRUISE [30.7], GUIDE [30.8], C4.5 [30.9], and chi-square automatic interaction detection (CHAID) [30.10]. Section 30.4 introduces ensemble methods. Finally, Sect. 30.5 discusses briefly how tree methods compare to a broader spectrum of classification and prediction methods.

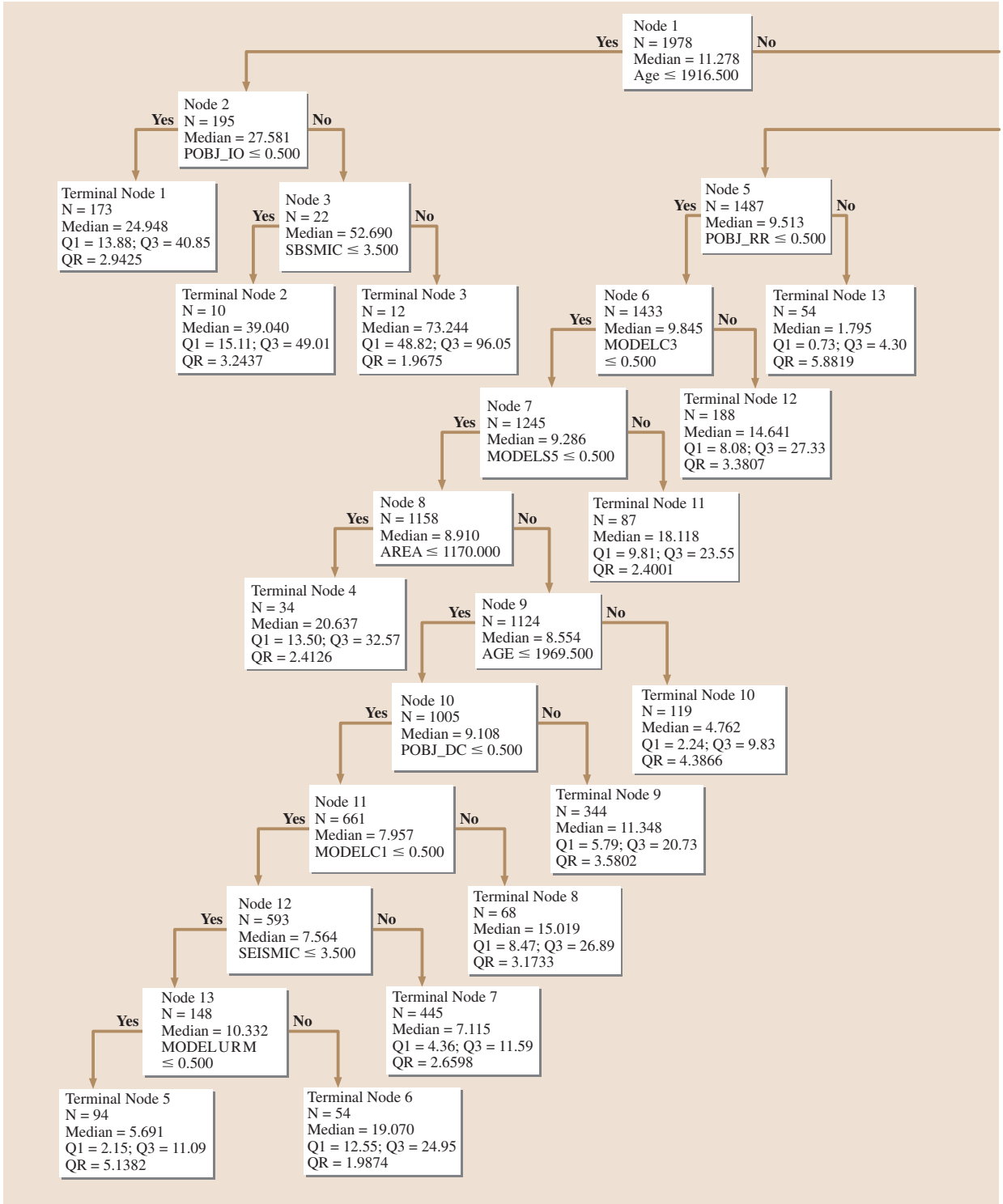
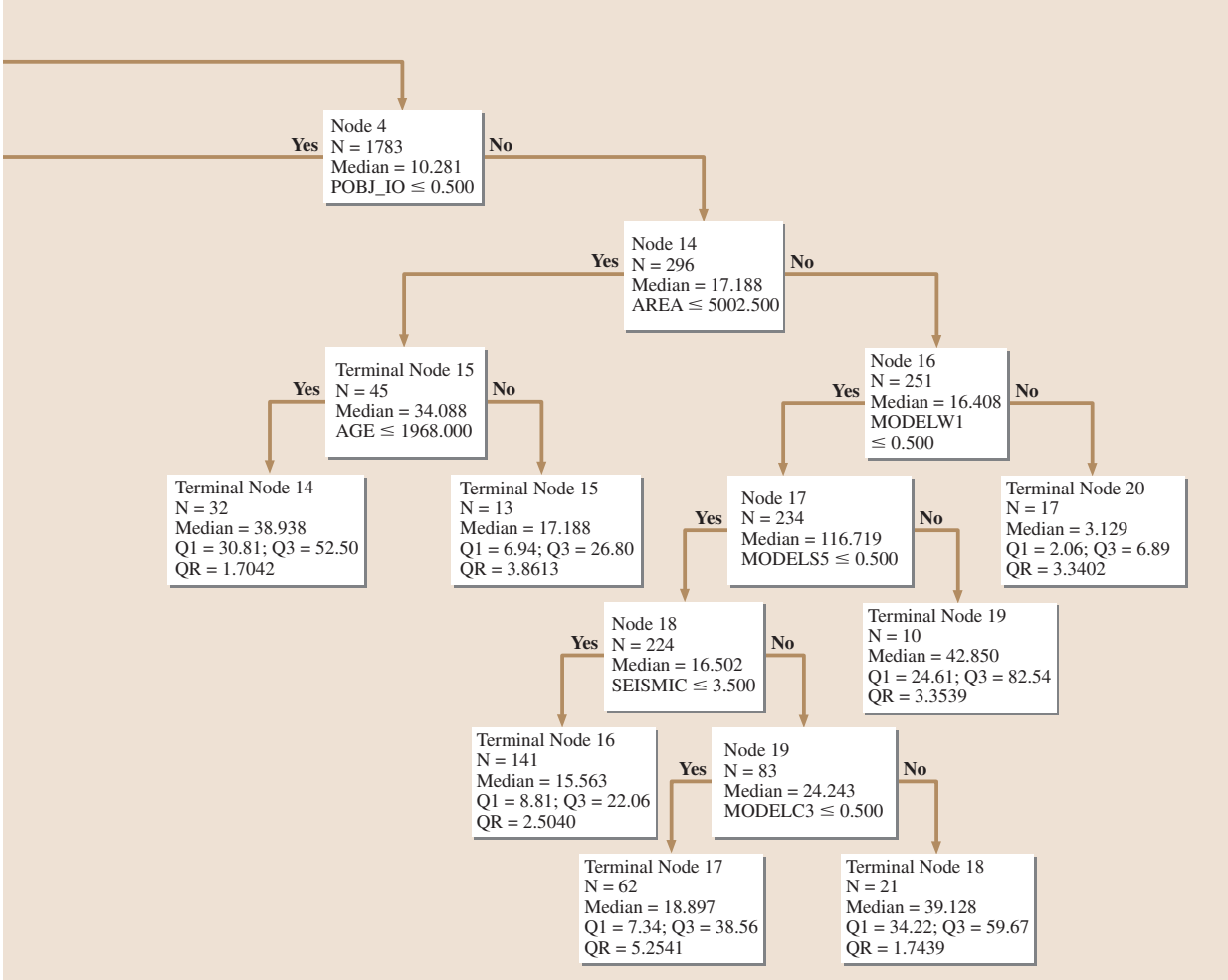



**Fig. 30.2** FEMA seismic rehabilitation cost estimator






## 30.2 Classification and Regression Tree (CART)

### 30.2.1 Introduction

A widely used tree-based method and software is called CART, which stands for classification and regression tree [30.5]. CART is based on statistical methodology developed for classification with categorial outcomes or regression with continuous outcomes. We shall start with classification trees in Sect. 30.2.2 and 30.2.3 and then discuss the regression tree in Sect. 30.2.4.

Take the iris data classification problem [30.11] as an example. The iris data set contains the lengths and widths of sepals and petals of three types of irises: Setosa, Versicolor, and Virginica. The purpose of the analysis is to learn how one can discriminate among the three types of flowers, $Y$, based on four measures of width and length of petals and sepals, denoted by $X_1$, $X_2$, $X_3$, and $X_4$, respectively. Figure 30.3 presents the classification tree constructed by CART. The whole sample sits at the top of the tree. The tree first splits the entire sample into two groups at $X_2 = 2.45$. Observations satisfying the condition $X_2 < 2.45$ are assigned to the left branch and classified as Setosa, while the other observations ($X_2 \geq 2.45$) are assigned to the right branch and split further into two groups at $X_1 = 1.75$.

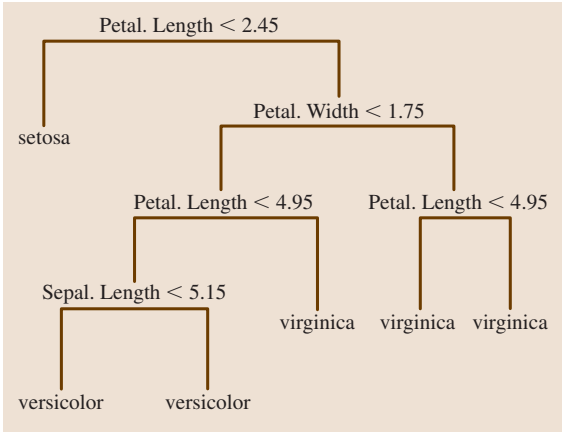



**Fig. 30.3** A classification tree for the iris data

At the end, the tree partitions the whole sample into six exclusive subgroups (terminal nodes in the tree). This tree indicates that a good classification rule can be constructed based on the width and length of the petal, and the length of the sepal. The binary tree structure also makes the classification rule easily understood. For example, if the sepal length of an iris with unknown type is 3 cm, its petal length is 4 cm and width is 1.3 cm, then this iris will be classified as a Versicolor iris.

The basic idea of CART is to first grow a very large and complicated tree classifier that explains the sample very accurately but may have poor generalization characteristics, and then prune this tree using cost-complexity pruning to avoid overfitting but still with good accuracy. The CART algorithm grows the classification tree by recursive binary partitioning the sample into subsets. It first splits the entire sample into two subsets, and classifies the observations in each subset using the majority rule. Other class assignment rules can be derived based on preassigned classification costs for different classes. Then one or both of these subsets are split further into more subsets, and this process is continued until no further splits are possible or some stopping rule is triggered. A convenient way to represent this recursive binary partition of the feature space is to use a binary tree like the one in Fig. 30.3, in which subsets are represented by nodes.

## 30.2.2 Growing the Tree

Let us first look at how CART grows the tree, i. e., how to determine the splitting variable and the split point at each partition. The fundamental idea is to select each split of a node so that the observations in each of the descendant nodes are *purer* than those in the parent node.

Consider a classification problem with a categorical response $Y$ taking values $1, 2, \ldots, K$, and $p$ predictors $X_1, \ldots, X_p$ based on a sample of size $N$. At node $m$, which contains a subset of $N_m$ observations, define the node proportion of class $k$ by

$$\hat{p}_m(k) = \frac{1}{N_m} \sum_{i=1}^{N_m} I(y_i = k), \quad k = 1, \ldots, K ,$$

where $I(A) = 1$ when condition $A$ is satisfied and 0 otherwise.

Before discussing how CART splits at a node, we first describe how it classifies a node. In its basic form, CART classifies observations in node $m$ to the majority class $k(m) = \arg\max_k \left[\hat{p}_m(k)\right]$. A more general rule is to assign node $m$ to class $k(m) = \arg\min_k [r_k(m)]$, where $r_k(m)$ is the expected misclassification cost for class $k$. Letting $\pi_m(k)$ be the prior probability of node $m$ as class $k$, and $c(i|j)$ be the cost of classifying a class $j$ case as a class $i$ case that satisfies $c(i|j) \geq 0$ if $i \neq j$ and $c(i|j) = 0$ if $i = j$, we have

$$r_k(m) = \sum_j c(k|j)\pi_m(j) .$$

The application of this rule takes into account the severity of misclassifying cases to certain class. If the misclassification cost is constant and the priors $\pi_m(j)$ are estimated by the node proportions, it converts back to the basic form.

CART has two types of splitting criteria: the Gini criterion and the twoing criterion. In general, for a nominal outcome variable, either criterion can be used; for an ordinal outcome variable, the twoing criterion is used.

### Gini Criterion

By the Gini criterion, we seek the splitting variable and the split point for node $m$ by maximizing the decrease in the Gini index. The Gini index is an impurity measure defined as a nonnegative function of node proportions $\hat{p}_m(k), k = 1, \ldots, K$,

$$i(m) = \sum_{k=1}^{K} \hat{p}_m(k)\left[1 - \hat{p}_m(k)\right] = 1 - \sum_{k=1}^{K} \left[\hat{p}_m(k)\right]^2 . \tag{30.1}$$

This impurity measure attains its minimum when all cases at a node belong to only one class, so $i(m) = 0$ defines node $m$ as a pure node. Let $m_L$ and $m_R$ be the left and right branches resulting from splitting node $m$

on predictor $x_j$, and $q_L$ and $q_R$ be the proportion of cases in node $m$ classified into $m_L$ and $m_R$, respectively. For each predictor $x_j$, the algorithm finds the split by maximizing the decrease in the impurity measure

$$\Delta i_j(t,m) = i(m) - [q_L i(m_L) + q_R i(m_R)] \,. \qquad (30.2)$$

This is equivalent to minimizing the weighted average of the two child nodes' impurity measures, $q_L i(m_L) + q_R i(m_R)$. When $x_j$ is continuous or ordinal, $m_L$ and $m_R$ are given by $x_j < t$ and $x_j \geq t$ for a splitting point $t$, and the solution of $t$ can be obtained quickly; if $x_j$ is nominal with a large number of levels, finding the split point $t$ by exhaustive subset search can be computationally prohibitive. The computer program CART only searches over all possible subsets of a categorical predictor for a limited number of levels. The CART algorithm proceeds with a greedy approach that scans through all predictor variables to find the best pair $(j,t)$ with the largest decrease in $\Delta i_j(t,m)$.

Possible choices of $i(m)$ include

- Cross-entropy or deviance:

$$\sum_{k=1}^{K} \hat{p}_m(k) \log \hat{p}_m(k) \,. \qquad (30.3)$$

- Misclassification error:

$$\frac{1}{N_m} \sum_{i=1}^{N_m} I\left[y_i \neq k(m)\right] \,. \qquad (30.4)$$

The cross-entropy measure (30.3) was used in the early development of CART but the Gini index was adopted in later work. The misclassification error measure (30.4) is typically used during the pruning stage (Sect. 30.2.3). For further discussion of the impurity measures, we refer to *Hastie* et al. [30.12].

#### Twoing Rule

Under the second splitting criterion, the split at a node $m$ is determined by minimizing the twoing rule

$$q_L q_R \left[\sum_{k=1}^{K} |\hat{p}_{m_L}(k) - \hat{p}_{m_R}(k)|\right]^2 \,.$$

When $K$ is large, twoing is a more desirable splitting criterion.

Comparisons between the Gini and twoing splitting criteria have shown only slight differences, but the Gini criterion is preferred by the inventors of CART and implemented as the default option in the commercial CART software by Salford Systems.

A tree continues to grow until either (1) there is only one observation in each of the terminal nodes, or (2) all observations within each terminal node have an identical distribution of independent variables or dependent variable, making splitting impossible, or (3) it reaches an external limit on the number of observations in each terminal node set by the user.

### 30.2.3 Pruning the Tree

Growing a very large tree can result in overfitting, that is, the tree classifier has small classification errors on the training sample, but may perform poorly on a new test data set. To avoid overfitting but still capture the important structures of the data, CART reduces the tree to an optimal size by *cost-complexity pruning*. Suppose the tree-growing algorithm stops at a large tree $T_{\max}$. The size of $T_{\max}$ is not critical as long as it is large enough. Define a subtree $T \subset T_{\max}$ to be any tree that can be obtained by pruning $T_{\max}$, that is, collapsing any number of its nodes. The idea is first to find subtrees $T_\alpha \subset T_{\max}$ for a given tuning parameter $\alpha \geq 0$ that minimize the cost-complexity criterion

$$R_\alpha(T) = R(T) + \alpha|T| = \sum_{m=1}^{|T|} N_m i(m) + \alpha|T| \,, \qquad (30.5)$$

where $m$ indexes the terminal nodes, $|T|$ is the number of terminal nodes in tree $T$, and $N_m$ and $i(m)$ are the number of observations and the impurity measure of node $m$, respectively. Then the optimal tree is selected from this sequence of $T_\alpha$s. The cost-complexity criterion is a combination of the misclassification cost of the tree, $R(T)$, and its complexity $|T|$. The constant $\alpha$ can be interpreted as the complexity cost per terminal node. If $\alpha$ is small, the penalty for having a larger tree is small and hence $T_\alpha$ is large. As $\alpha$ increases, $|T_\alpha|$ also increases. Typically, the misclassification error impurity measure (30.4) is used in pruning the tree. Equation (30.5) presents a special form of the misclassification cost $R(T)$ when the cost of misclassifying an observation of class $j$ to class $i$ is the same for all $i \neq j$. Other misclassification cost functions $R(T)$ can be applied; see *Breiman* et al. [30.5], but our description of the algorithm will be based on (30.5).

CART uses *weakest-link pruning* to find the $T_\alpha$s. The algorithm successively collapses the branch that produces the smallest per-node increase in $R(T)$ from the bottom up and continues until it produces the single-node (root) tree. This gives us a sequence of nested subtrees $\{T_0, T_1, T_2, \ldots, T_I\}$ with decreasing

complexity and increasing cost. It is shown in *Breiman* et al. [30.5] that this sequence of subtrees is characterized by distinct and increasing $\alpha_i$s and the $\alpha$ corresponding to the optimal size tree can be found from $\{\alpha_i | i = 0, \ldots, I\}$.

The weakest-link pruning works as follows. Define $T_m$ as a branch of $T_{i+1}$ containing a node $m$ and its descendants. When $T_i$ is pruned at node $m$, its misclassification cost increases by $R(m) - R(T_m)$, while its complexity decreases by $|T_m| - 1$. Hence the ratio

$$g_i(m) = \frac{R(m) - R(T_m)}{|T_m| - 1}$$

measures the increase in misclassification cost per pruned terminal node, and $T_{i+1}$ is obtained by pruning all nodes in $T_i$ with the lowest value of $g_i(m)$, i. e., the weakest link. The $\alpha$ associated with tree $T_i$ is given by $\alpha_i = \min_m g_i(m)$ and it is easily seen that $\alpha_i < \alpha_{i+1}$. The first tree $T_0$ is obtained by pruning $T_{\max}$ of those branches whose $g_0(m)$ value is 0. Starting with $T_0$, the cost-complexity pruning algorithm initially tends to prune off large branches with many terminal nodes. As the trees get smaller, it tends to cut off fewer at a time. The pruning stops when the last subtree $T_I$ is the root tree. These recursive pruning steps are computationally rapid and require only a small fraction of the total tree construction time.

CART then identifies from $\{T_i | i = 0, 1, \ldots, I\}$ the optimal subtree as the one with the minimal classification error (0-SE rule) or the smallest tree within one standard error of the minimum error rate (1-SE rule). The classification error of each subtree $T_i$ can be estimated using test samples when data are sufficient or $V$-fold cross-validation. The reason for using the 1-SE rule is to favor smaller trees with estimated misclassification errors close to that of the minimum error tree. The 1-SE rule is good for small data sets, whereas the 0-SE rule works better on large data sets. With sufficient data, one can simply divide the sample into learning and test sub-samples. The learning sample is used to grow $T_{\max}$ and to obtain the subsequence $\{T_i | i = 0, 1, \ldots, I\}$. The test sample is then used to estimate the misclassification error rate for the $T_i$s.

When the data are insufficient to allow a good-sized test sample, CART employs cross-validation to estimate the misclassification rate. Cross-validation is a computationally intensive method for validating a procedure for model building, which avoids the requirement for a new or independent validation data set. For $V$-fold cross-validation, CART proceeds by dividing the learning sample into $V$ parts, stratified by the dependent variable, to assure that a similar outcome distribution is present in each of the $V$ subsets of data. CART takes the first $V - 1$ parts of the data, constructs the auxiliary trees for $\{T_i | i = 0, 1, \ldots, I\}$ characterized by the $\alpha_i$s, and uses the remaining data to obtain initial estimates of the classification error of selected subtrees. The same process is then repeated on other $V - 1$ parts of the data. The process repeats $V$ times until each part of the data has been held in reserve one time as a test sample. The estimates of the classification errors for $\{T_i | i = 0, 1, \ldots, I\}$ are then given by averaging their initial estimates over $V$ artificial test samples.

Many other pruning methods are also available for decision trees, such as reduced error pruning (REP), pessimistic error pruning (PEP), minimum error pruning (MEP), critical value pruning (CVP) and error-based pruning (EBP). *Esposito* et al. [30.13] provides a comprehensive empirical comparison of these methods.

### 30.2.4 Regression Tree

CART constructs a regression tree when the outcome variable is continuous. The process of constructing a regression tree is similar to that for a classification tree, but differs in the criteria for splitting and pruning. CART constructs the regression tree by detecting the heterogeneity that exists in the data set and then purifying the data set. At each node, the predicted value of the dependent variable is a constant, usually as the average value of the dependent variable within the node. An example of a regression tree is given in Fig. 30.4. The analysis tried to construct a predictive model of cen-

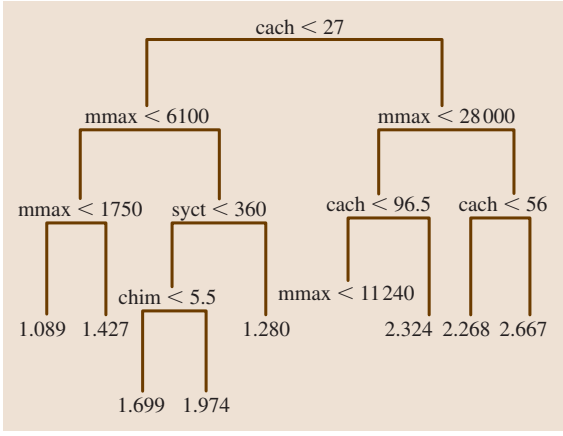



**Fig. 30.4** A regression tree for predicting CPU performance

tral processing unit (CPU) performance using nine CPU characteristics (Table 30.3) based on a learning sample of 209 CPUs [30.14].

When CART grows a regression tree, it determines the splitting variable and split point by minimizing the mean square error (MSE) or the mean absolute deviation from the median. Since the mechanisms for the two rules are similar, we only describe the former. Under this circumstance, the node impurity is measured by

$$i(m) = \frac{1}{N_m} \sum_i \left[ y_i - \bar{y}(m) \right]^2 , \tag{30.6}$$

where $\bar{y}(m)$ is the average value of the dependent variable at node $m$. The best split $(j, t)$ is hence determined by solving

$$\min_{j,t} \left\{ \sum_{i \in m_L} \left[ y_i - \bar{y}(m_L) \right]^2 + \sum_{i \in m_R} \left[ y_i - \bar{y}(m_R) \right]^2 \right\} , \tag{30.7}$$

where $m_L$ is the left descendent node given by $x_j < t$, and $m_R$ is for the right branch. An alternative criterion to (30.7) is the weighted variance

$$p_L i(m_L) + p_R i(m_R) ,$$

where $p_L$ and $p_R$ are the proportions of cases in node $m$ that go left and right, respectively. Correspondingly, the cost-complexity criterion in the pruning process also adopts (30.6).

### 30.2.5 Some Algorithmic Issues

In this section, we discuss several algorithmic issues of CART that are important in practice.

#### Estimating Within-Node Classification Error

In practice, the users desire to know not only the class assignment from a CART tree for any future case, but also about a classification error associated with this prediction. This classification error can be represented by the probability of misclassification given that the case falls into a particular terminal node. We denote this value by $r(m)$ if a case falls into terminal node $m$. A naive estimate of $r(m)$ is the proportion of cases that are misclassified by the tree constructed from the entire sample, as shown in (30.4). This estimate however can be misleading since it is computed from the same data used in constructing the tree. It is also unreliable if the terminal node $m$ is tracked down through many splits from the root and has a relatively small number of observations.

**Table 30.3** Characteristics of CPUs

| Variable | Definition |
|---|---|
| name | Manufacturer and model |
| syct | Cycle time in nanoseconds |
| mmin | Minimum main memory in kilobytes |
| mmax | Maximum main memory in kilobytes |
| cach | Cache size in kilobytes |
| chmin | Minimum number of channels |
| chmax | Maximum number of channels |
| perf | Published performance on a benchmark mix |
| | relative to an IBM 370/158-3 |
| estperf | Estimated performance by the authors |

*Breiman* et al. [30.5] proposed an ad hoc estimate that is significantly better than the naive one,

$$\hat{r}(m) = \hat{r}_o(m) + \frac{\epsilon}{N_m + \lambda} , \tag{30.8}$$

where $\hat{r}_o(m)$ is defined in (30.4), $N_m$ is the size of node $m$, and $\epsilon$ and $\lambda$ are constants to be determined below. Define the resubstitution classification error of the tree $T$ by aggregating the classification errors across all terminal nodes as follows,

$$\hat{R}_o(T) = \sum_{m=1}^{|T|} \hat{r}_o(m) N_m .$$

Denote the cross-validated classification error of tree $T$ by $\hat{R}_{CV}(T)$. Then the constants $\lambda$ and $\epsilon$ are obtained from the following equations

$$\lambda \sum_{m=1}^{|T|} \frac{N_m}{N_m + \lambda} = \frac{\hat{R}_{CV}(T) - \hat{R}_o(T)}{2 \min_V \hat{R}_{CV}(T_V)} ,$$

$$\epsilon = 2\lambda \min_V \hat{R}_{CV}(T_V) ,$$

where $\min_V \hat{R}_{CV}(T_V)$ is the minimum obtained during $V$-fold cross-validation. If $\hat{R}_{CV}(T) \le \hat{R}_o(T)$, the naive estimate (30.4) is used.

#### Splitting on a Linear Combination of Variables

Sometimes, the data are intrinsically classified by some hyperplanes. This can possibly challenge the tree-based method in its original form using binary partitions, which tends to produce a large tree in trying to approximate the hyperplanes by many hyperrectangular regions. It is also very hard for the analysts to recognize the neat linear structure from the output. The CART algorithm deals with this problem by allowing splits over

linear combination of predictor variables $\sum_j a_j X_j$. The weights $a_j$ and split point $t$ are optimized to minimize the relevant criterion (such as the Gini index). While this can improve the predictive power of the tree, the results are no longer invariant under monotone transformations of individual independent variables. The introduction of linear combinations also causes a loss in interpretability that is viewed as an important advantage of tree-based methods.

#### Missing Data on Predictors

We often have incomplete data with missing values on some independent variables. We might exclude these incomplete observations from analysis, but this could lead to serious depletion of the learning sample. A common alternative is to impute the missing values [30.15]. CART however uses two different approaches. A simple treatment for categorical predictors is to put the missing values into a new "missing" category. This however puts all observations with missing values into the same branch of the tree, which could be misleading in practice. A more refined approach is to use surrogate splits. This approach makes full use of the data to construct the tree, and results in a tree that can classify cases with missing information. Surrogate variables are constructed as follows. When we consider a split on a predictor $x_j$ with missing values, only the cases containing values of $x_j$ are used, and we find the best split as discussed in Sect. 30.2.2. The first surrogate split is the split on a predictor that most accurately predicts the action of the best split in terms of a predictive measure of association. The second surrogate is the split on another predictor that does second best, and so on; for details see [30.5]. The surrogate splits can cope with missing observations during both the training phase of CART and prediction. If a case has missing values so that the best split is not useable, the next best surrogate split would be used.

#### Variable Importance

Another nice feature of CART is that it automatically produces a variable ranking. The ranking considers the fact that an important variable might not appear in any split in the final tree when the tree includes another masking variable. If we remove the masking variable, this variable could show up in a prominent split in a new tree that is almost as good as the original. The importance score of a particular variable is the sum of the improvement of impurity measures across all nodes in the tree when it acts as a primary or surrogate splitter.

#### Instability of Trees

Small changes in the learning sample may cause dramatic changes in the output tree. Thus two similar samples could generate very different classification rules, which is against human intuition and complicates interpretation of the trees. The major reason for this instability is the hierarchical nature of the recursive partitioning. For example, if at some partition, there are surrogate splits that are almost as good as the primary split, the tree could be very sensitive to small changes, because a minor change in the learning sample could cause the surrogate split to become slightly superior to the primary split. This effect in the top nodes can cascade to all their descendant nodes. Aggregating methods, such as bagging [30.16] and boosting [30.17] have been incorporated into the algorithm to mitigate the instability problem, but the improvement comes at the price of sacrificing the simple interpretability of a single tree.

### 30.2.6 Summary

CART makes no distributional assumptions on any dependent or independent variable, and allows both categorical and continuous variables. The algorithm can effectively deal with large data sets with many independent variables, missing values, outliers and collinearity. Its simple binary tree structure offers excellent interpretability. Besides, CART ranks the independent variables in terms of their importance to prediction power and therefore serves as a powerful exploratory tool for understanding the underlying structure in the data.

However, CART does have limitations. While it takes advantages of the simple binary tree structure, it suffers from instability and has difficulty capturing additive structures. In general, if a parametric statistical model fits the data well and its assumptions appear to be satisfied, it would be preferable to a CART tree.

## 30.3 Other Single-Tree-Based Methods

### 30.3.1 Loh's Methods

One drawback of exhaustive-search tree-growth algorithms such as that used in CART is the potential for variable selection bias. In particular, such algorithms tend to choose variables that provide greater numbers of potential splitting points. Hence, continuous variables tend to be favored over categorical variables, and polychotomous variables are selected more frequently than dichotomous ones. These characteristics complicate interpretation of resulting trees, because any insights gained from the tree structure could potentially be clouded by systematic biases toward certain variables. The methods developed by Loh and his coauthors attempt to address this bias issue.

#### QUEST

*Loh* and *Shih* [30.6] developed the quick, unbiased and efficient statistical tree (QUEST) algorithm to address this variable selection bias issue. The algorithm is an enhancement of the much earlier fast algorithm for classification trees (FACT) of *Loh* and *Vanichsetaukul* [30.18], which was primarily designed as a computationally efficient alternative to exhaustive-search methods, but still suffered from variable-selection bias in the presence of categorical predictors.

The basic strategy employed by QUEST is to select each splitting variable and its associated split value sequentially rather than simultaneously. To determine the splitting variable at a particular node, a series of statistical tests is performed:

1. Specify an overall level of significance, $\alpha \in (0, 1)$. Let $K$ be the number of variables, and $K_1$ be the number of continuous and ordinal variables.
2. Identify the variable with the smallest $p$-value resulting from the appropriate analysis of variance test (for continuous or ordinal variables) or Pearson's $\chi^2$ test (for categorical variables). If this $p$-value is less than $\alpha/K$, split on this variable.
3. If the lowest $p$-value exceeds this threshold, perform Levene's $F$-test for unequal variances on each continuous/ordinal variable. If the smallest of these $p$-values from the $F$-tests is less than $\alpha/(K + K_1)$, split on its associated variable. Otherwise, split on the variable with the smallest $p$-value in step 2.

The Bonferroni-adjusted thresholds used above is meant to render the potential variable-selection bias negligible.

Once the splitting variable is selected, the split point is needed. If more than two classes are present at the node, they are first combined into two *superclasses* using two-means clustering [30.19]. Then, a modified quadratic discriminant analysis is employed to select the split point. Categorical variables must be transformed into ordered variables before this split can be performed. This is accomplished by recoding the represented categories as 0–1 dummy vectors and projecting them onto their largest discriminate coordinate.

The algorithm described above focuses on univariate splits. However, as with CART, QUEST can also be used to build trees with linear-combination splits. Generally, QUEST trees based on linear-combination splits tend to be shorter and more accurate than those based on univariate splits.

The QUEST package may be obtained from http://www.stat.wisc.edu/~loh/loh.html. The full package includes an exhaustive-search algorithm to mimic basic CART, and offers options for pruning or stopping rules.

#### CRUISE

*Kim* and *Loh* [30.7] extended the unbiased variable selection idea beyond the capabilities of QUEST. First, while QUEST forces binary splits at each node, classification rule with unbiased interaction selection and estimation (CRUISE) allows multiway splitting. Moreover, CRUISE includes look-ahead methods for detecting two-variable interactions during variable selection.

Multiway splitting offers two key advantages over binary splitting. First, although any multiway split can be represented by a series of binary splits, trees that allow multiway splits are often shorter and thus more easily interpreted. Second, Kim and Loh demonstrate that, with binary trees, some dependent variable categories can be completely dropped after pruning. For example, a tree intended to classify cases into two categories might ultimately include paths to only two of the classes of interest. Trees employing multiway splits are less apt to *losing* categories in this manner.

Another key benefit of CRUISE is the inclusion of look-ahead methods for detecting two-variable interactions during variable selection. CRUISE contains two methods for splitting-variable selection: 1D and 2D. The

1D method is similar to what is used in the QUEST algorithm. $p$-values are obtained from $F$-tests for continuous and ordinal variables and from Pearson's $\chi^2$ tests for categorical variables. If the smallest $p$-value is significant, its associated variable is selected for the split. Otherwise, Levene's test for unequal variances is carried out for the continuous and ordinal variables to select the splitting variable. A major drawback of the 1D method is that, because analysis of variation (ANOVA) and Levene's tests do not look ahead, strong interactions are often completely overlooked. In addition, because these tests restrict their attention to differences in means and variances, other distributional differences may remain unnoticed.

The 2D method uses contingency tables to remedy these problems. First, consider interaction detection. Given a pair of categorical variables, category pairs are tabulated against classifications. Then, Pearson's $\chi^2$ test for independence is performed. If a strong interaction is present between the two categorical variables, the test is likely to result in a low $p$-value. Interactions involving continuous variables are detected similarly. Prior to testing, each continuous variable is transformed into a dichotomous variable by partitioning its domain at the median.

The same idea is applied to identify marginal distributional effects. For each categorical variable, Pearson's $\chi^2$ test for independence is performed. Continuous variables are handled similarly, first transforming them into four-category variables by partitioning at their quartiles. The basic idea is that the one- or two-variable table with the smallest $p$-value should determine the splitting variable. However, this simple procedure would be somewhat biased toward categorical variables, so Kim and Loh employ a bootstrap adjustment prior to variable selection.

Once the splitting variable has been selected, CRUISE uses linear discriminant analysis (LDA) to determine the splitting points. Since LDA is best applied to normally distributed data, Kim and Loh apply a Box–Cox transformation to the selected variable prior to running the discriminant analysis. Categorical variables must be converted to their discriminant coordinate values before this process is carried out. A shift transformation may be needed to produce the positive-valued inputs required for the Box–Cox procedure. Split points are converted back to the original scale when constructing the tree.

Our description thus far assumes the availability of complete data, but an important advantage of CRUISE is the elimination of the variable-selection bias that often results from the treatment of missing data. Kim and Loh note that, because CART uses proportions rather than sample sizes to determine variable selections, the procedure is biased toward variables with more missing data. CRUISE on the other hand, through its use of statistical tests that take account of sample size, does not encounter this type of bias. This bias may not be critical if it does not affect the overall predictive quality of the tree, but it may have a large impact on the interpretation of CART's variable-importance measures.

#### GUIDE

With generalized, unbiased interaction detection and estimation (GUIDE), *Loh* [30.8] expanded unbiased variable selection to regression tree applications. GUIDE includes procedures for weighted least squares, Poisson regression and quantile regression. In addition, categorical variables may be used for prediction through dummy coding, or they may be restricted to node-splitting.

### 30.3.2 Quinlan's C4.5

*Quinlan* [30.9, 20] wrote his first decision tree program in 1978 while visiting Stanford University. His iterative dichotomizer 3rd (ID3) and its replacement, C4.5, programs have served as the primary decision tree programs in the artificial-intelligence and machine-learning communities. He attributes the original ideas to the concept learning systems of *Hunt* et al. [30.21].

#### Splitting Rules

Suppose a node $T$ contains $|T|$ observations that fall into $K$ classes. Letting $p_k(T)$ represent the proportion of these cases belonging to class $k$, we define the *information* contained within $T$ (also known as the *entropy* of $T$) by:

$$\text{Info}(T) = -\sum_{k=1}^{K} p_k(T) \times log_2 \left[ p_k(T) \right] .$$

Now suppose that a candidate variable $X$ partitions $T$ into $n$ smaller nodes, $T_1, T_2, \ldots, T_n$. The information of $T$ given the value of $X$ is given by the weighted average of the information contained within each subnode:

$$\text{Info}(X, T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \text{Info}(T_i) .$$

Therefore, the *information gain* provided by the split is simply

$$\text{Gain}(X, T) = \text{Info}(T) - \text{Info}(X, T) . \qquad (30.9)$$

ID3 selects attributes and splits to maximize the information gain at each node. However, this procedure tends to heavily favor variables with many categories. To compensate for this effect at least partially, C4.5 instead uses the *gain ratio* criterion. The gain ratio of a split is defined as the ratio of the information gain to the information contained in the resulting split:

$$\text{GainRatio}(X, T) = \frac{\text{Gain}(X, T)}{\text{SplitInfo}(X, T)} , \qquad (30.10)$$

where

$$\text{SplitInfo}(X, T) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right) .$$

The C4.5 algorithm creates binary splits on continuous variables and multiway splits on categorical variables. To determine the best splits on categorical variables, each category is first assigned to a unique branch. Then, pairs of branches are iteratively merged until only two branches exist. The split with the maximum gain ratio among those observed becomes the candidate split for that variable. This search method is, of course, heuristic and might not actually find the categorical split with the largest gain ratio. On the other hand, searches on continuous variables always find the best possible binary split. To determine the ultimate splitting variable, the algorithm first restricts its choices to those variables achieving at least average information gain (30.9). The split is then selected to maximize the gain ratio (30.10) among these choices.

### Variable-Selection Bias

Even if the gain ratio is used as an alternative to straight information gain to alleviate the algorithm's bias toward continuous variables, this original remedy is far from perfect. *Dougherty* et al. [30.22] demonstrated that, for many data sets, the predictive performance of C4.5 was improved by first discretizing all continuous variables. This result suggested that the existing selection method was biased toward continuous variables.

In C4.5 release 8, *Quinlan* [30.23] introduces a complexity-cost parameter into the information gain expression for continuous variables. For a continuous variable with $n$ distinct values, the information gain is redefined as

$$\text{Gain}(X, T) = \text{Info}(T) - \text{Info}(X, T) - \log_2(n-1)/|T| .$$

Effectively, each continuous variable is penalized for the information required to search among its numerous potential splitting points.

### Missing Values

The description of C4.5 has thus far assumed complete data. Cases with missing values for a particular variable are excluded from the split search on that variable, and also from the numerator of the gain ratio. The entropy of the split is computed as if missing values constitute an additional branch.

When a missing value prevents the application of a splitting rule to a new case, the case is replaced by weighted replicates, each being assigned to a different branch. The weights are equal to the proportion of non-missing training cases assigned to that branch. Class probabilities for the original case are based on the weighted sum of the probabilities of the generated observations.

### Pruning

*Quinlan* [30.9] advocates retrospective pruning instead of stopping rules. If enough data were available, the pruning process would use data withheld from training the tree to compare error rates of candidate sub-trees. The software does not assume that data are withheld from training, so it implements *pessimistic* pruning. In each node, an upper confidence limit of the number of misclassified cases is estimated assuming a binomial distribution around the observed number of misclassified cases. The confidence limit serves as an estimate of the error rate on future data. The pruned tree minimizes the sum over leaves of upper confidence limits.

### Decision Rules

C4.5 includes the capability to convert its decision trees to an equivalent simplified set of decision rules. Decision rules are often preferred to the tree structure because their interpretation is very straightforward. Given a decision tree, rule-set generation proceeds as follows:

1. Convert every decision tree path to a decision rule. (Each decision encountered along a path becomes a *test* in the resulting decision rule.)
2. Prune each decision rule by removing as many tests as possible without reducing its accuracy.
3. Track the estimated accuracy of each resulting rule, and classify new items based on high-accuracy rules first.

### Improvements in C5.0/See5

C5.0 and See5 are the current commercial implementations of Quinlan's methods. These programs offer several enhancements to C4.5, including the ability to specify unequal misclassification costs, the application of fuzzy splits on continuous variables, and boosting trees.

### 30.3.3 CHAID

Chi-square automatic interaction detection (CHAID) is a parametric recursive partitioning technique that builds non-binary classification trees. It was originally developed by *Kass* [30.10] to handle categorical predictors only. Continuous predictors need to be discretized into a number of categories with approximately equal number of observations. In dealing with missing values on predictors, CHAID simply places them in an additional category. The algorithm employs a sequential merge-and-split procedure based on significance tests on predictor variables to generate node splits and determine the size of a tree. It is worth noting that CHAID differs from CART in that it determines where to stop in tree growth rather than using retrospective pruning after growing an oversized tree.

CHAID produces non-binary trees that are sometimes more succinct representations than equivalent binary trees. For example, it may yield a split on an income variable that divides people into four income groups according to some important consumer-behavior-related variable (e.g., types of cars most likely to be purchased). In this case, a binary tree is not an efficient representation and can be hard to interpret. On the other hand, CHAID is primarily a step-forward model-fitting method. Known problems with forward stepwise regression fitting models are probably applicable for this type of analysis.

### 30.3.4 Comparisons of Single-Tree-Based Methods

We have discussed six single-tree methods, viz. CART, C4.5, CHAID, CRUISE, GUIDE and QUEST. Table 30.4 lists the features offered by these six methods. Among these methods, GUIDE is a regression tree method, CHAID, CRUISE and QUEST are classification tree methods, and CART and C4.5 deal with both classification and regression problems.

Empirical comparisons on real data sets [30.24] showed that, among all these methods, there is none

**Table 30.4** Comparison of tree-based algorithms

| Feature | CART | C4.5 | CHAID | CRUISE | QUEST | GUIDE |
|---|---|---|---|---|---|---|
| *Dependent variable* | | | | | | |
| Discrete | x | x | x | x | x | |
| Continuous | x | x | | | | x |
| *Split at each node* | | | | | | |
| Binary | x | | | | x | x |
| Multiple | | x | x | x | | |
| Split on linear combinations | x | | | x | x | x |
| *Searching splitting variable* | | | | | | |
| Exhaustive | x | | x | x | x | x |
| Heuristic | | x | x | | | |
| *Splitting criterion* | | | | | | |
| Impurity measure | x | x | | | | |
| Twoing rule | x | | | | | |
| Statistical test | | | x | x | x | x |
| *Split variable selection* | | | | | | |
| Unbiased selection | | | | x | x | x |
| Pairwise interaction detection | | | | x | | |
| *Tree size control* | | | | | | |
| Cost-complexity pruning | x | | | x | x | x |
| Pessimistic error pruning | | x | | | | |
| Stopping rules | | | x | | | x |
| *Missing data* | | | | | | |
| Surrogate | x | | | x | | |
| Imputation | | | | x | x | x |
| An additional level | | x | x | | | |

**Table 30.5** Data-mining software for tree-based methods

| Software | CART | C4.5 | CHAID | Software Provider |
|---|---|---|---|---|
| AnswerTree | x | x | x | SPSS Inc. |
| Clementine | | x | | Integral Solutions, Ltd. |
| Darwin | x | | | Thinking Machines, Corp. |
| Enterprise Miner | x | x | x | SAS Institute |
| Gain Smarts | x | | x | Urban Science |
| MineSet | x | | x | Silicon Graphics, Inc. |
| Model 1 | x | | x | Group 1/Unica Technologies |
| Model Quest | x | | | AbTech Corp. |
| CART | x | | | Salford Systems |
| R | x | | | R Foundation for Statistical Computing |
| S-Plus | x | | | MathSoft |
| See5 | | x | | RuleQuest Research |

that is absolutely superior to the others in terms of accuracy, complexity, interpretability and computation time. There is no significant difference in terms of prediction accuracy among these methods. Therefore, users may choose algorithms based on desired features for their applications, e.g., binary-split, multi-split, split on combination of variables. C4.5 tends to produce trees with many more leaves than other algorithms possibly due to under-pruning. In general, the multi-split tree methods (C4.5, CHAID, CRUISE) take more computation time than the binary-split methods. In problems with mixtures of continuous variables and categorical variables having different numbers of levels, methods such as QUEST, CRUISE, and GUIDE may be preferable because they are likely to protect against variable-selection bias.

CART, CHAID and C4.5 have been implemented in several commercial software platforms; see the list of software providers in Table 30.5. Free software for CRUISE, GUIDE, and QUEST can be obtained from the website http://www.stat.wisc.edu/~loh/. An earlier version of C4.5 is available free of charge http://www.cse.unsw.edu.au/~quinlan/.

## 30.4 Ensemble Trees

Instability of single trees provides room for improvement by ensemble methods. Ensemble methods create a collection of prediction/classification models by applying the same algorithm on different samples generated from the original training sample, then make final predictions by aggregating (voting) over the ensembles. It has been shown to improve the prediction/classification accuracy of a single model with significant effectiveness; see *Bauer* and *Kohavi* [30.25], *Breiman* [30.16, 26, 27], *Dietterich* [30.28], and *Freund* and *Schapire* [30.29]. The mechanism used by the ensemble methods to reduce prediction errors for unstable prediction models, such as trees, is well understood in terms of variance reduction due to averaging [30.12]. In this section, we will discuss two ensemble tree methods: boosting decision trees [30.17] and random forests [30.26, 27] that are motivated by boosting [30.29] and bagging [30.16], the two most widely used ensemble techniques today. However, it should be realized that better performance of ensemble trees comes at the price of sacrificing the explicit structure of a single tree and hence becoming less interpretable.

### 30.4.1 Boosting Decision Trees

Boosting was originally developed to improve the performance of binary classifiers. In his original boosting algorithm, *Schapire* [30.30] enhances a *weak learner* (i. e., a binary classifier with slightly better performance than random guessing) by using it to train two additional classifiers on specially filtered versions of the training data. The first new classifier is trained on cases for which the original weak learner performs no better than random guessing. The second new classifier is trained on cases where the first two learners disagree. In this way, each successive learner is trained on cases which are increasingly difficult to classify. The final boosted classifier is obtained by taking the majority vote of the orig-

inal weak learner and its two subsequent derivatives. Schapire's *strength of weak learnability* theorem proves that this simple boosted classifier always improves on the performance of the original weak learner.

In later work, *Freund* [30.31] improved on the performance of Schapire's method by expanding to a much larger ensemble of combined weak learners and again employing the majority vote principle. Subsequent theoretical improvements led to the more flexible AdaBoost algorithm [30.29] and various derivatives.

Our presentation of boosting algorithms and their application to classification and regression trees is based on the example of *Hastie* et al. [30.12].

### AdaBoost

We begin by presenting the most popular of the AdaBoost algorithms, AdaBoost.M1 [30.32], which is used for binary classification problems.

Consider a binary classification problem with categories coded as $Y \in \{-1, 1\}$. Given a predictor vector $\boldsymbol{X}$, the classifier $G(\boldsymbol{X})$ takes on values in $\{-1, 1\}$. The error rate on the training sample is given by:

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} I\left[y_i \neq G(x_i)\right]$$

and the expected future prediction error is

$$E_{XY} I\left[Y \neq G(X)\right] .$$

The AdaBoost.M1 algorithm proceeds as follows:

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$.
2. For $m = 1$ to $M$:
   a) Fit a classifier $G_m(x)$ to the training data using the weights $w_i$.
   b) Compute
   $$err_m = \frac{\sum_{i=1}^{N} w_i I\left[y_i \neq G_m(x_i)\right]}{\sum_{i=1}^{N} w_i} ;$$
   c) Compute
   $$\alpha_m = \log\left(\frac{1 - err_m}{err_m}\right) ;$$
   d) Set $w_i \leftarrow w_i \exp\left\{\alpha_m I\left[y_i \neq G_m(x_i)\right]\right\}$, $i = 1, 2, \ldots, N$ .
3. Define the boosted classifier as $G(x)$
   $= \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$ .

This boosting process begins with a weak learner, $G_1$, which is developed using an unweighted training set. The data are then weighted to deemphasize correctly classified observations and focus on incorrectly classified observations. A new weak classifier, $G_2$, is then trained from this weighted data. Next, these two classifiers are weighted according to their individual error rates (with the more accurate classifier given greater influence). Based on the weighted performance of the two classifiers, the training data is again reweighted for emphasis on difficult-to-classify observations, and the process iterates. Each new learner, $G_m$, is thus designed to address increasingly difficult aspects of the classification problem. The final boosted classifier, $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$, is derived from the weighted votes of the $M$ individual weak classifiers. The error rates of the individual weak classifiers $G_m$ tend to increase with each iteration, but prediction from the overall ensemble, $G$, tends to improve.

Our discussion of boosting thus far applies to classifiers in general. We now narrow our discussion to the particular application of boosting techniques to tree-based models.

### Boosting Trees

Boosting procedures in general fit additive expansions of weak classifiers or regressors. In the case of tree models, such expansions have the form

$$f(x) = \sum_{m=1}^{M} \beta_m T(x; \Theta_m) ,$$

where the parameter $\Theta$ includes information about the structure of each tree.

Such models are fit by minimizing a loss function, $L$, averaged over the training data, that is, $y$ solving

$$\min_{\{\beta, \Theta\}} \sum_{i=1}^{N} L\left[y_i, \sum_{m=1}^{M} \beta_m T(x_i; \Theta_m)\right] .$$

The solution to this problem is approximated using a forward stagewise additive algorithm. The idea is to build the expansion one term at a time. At a given iteration $m$, the optimal basis tree and scaling coefficient are sought to append to the old expansion $f_{m-1}$, producing $f_m$. The algorithm goes as follows:

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to $M$:
   a) Compute
   $$(\beta_m, \Theta_m) = \arg\min_{\beta, \Theta} \sum_{i=1}^{N} L\left[y_i, f_{m-1}(x_i) + \beta T(x_i; \Theta)\right] .$$

b) Set $f_m(x) = f_{m-1}(x) + \beta_m T(x; \Theta_m)$.

*Friedman* et al. [30.33] discovered that, under the exponential loss, $L[y, T(x; \Theta)] = \exp[-yf(x)]$, for binary classification problems, the forward stagewise algorithm is equivalent to the AdaBoost.M1 procedure discussed earlier. The expansion $\sum_{m=1}^{M} \alpha_m G_m(x)$ produced by the AdaBoost procedure estimates half the log-odds of $P(Y = 1|x)$. Therefore, taking the sign of this expression provides a reasonable classification rule.

For $K$-class classification and regression problems, the multiple additive regression trees (MART) procedure is used. MART is based on the gradient tree-boosting algorithm for regression, and can be implemented using a variety of available loss functions; see *Hastie* et al. [30.12] for details.

#### Selecting Component Tree Sizes

One consideration when applying boosting to tree models is the appropriate size of each weak learner tree. Early tree boosting applications, of which *Drucker* and *Cortes's* [30.17] optical character-recognition problem was the first, applied standard pruning methods to each weak learner in sequence. However, as *Hastie* et al. [30.12] note, this method implicitly prunes each weak learner as if it were the last in the sequence. This can result in poor predictive performance of the ensemble, as well as some unnecessary computations.

A common strategy to avoid this problem is to restrict each tree in the ensemble to a fixed number, $J$, of terminal nodes. The choice of this parameter is dependent on the problem at hand. Of course, the degree of variable interaction will be affected by the tree size. For example, boosting tree stumps (i. e., trees with only one split) considers no interaction effects, whereas boosting three-node trees can capture two-variable interactions. If each weak learner consists of $J$ terminal nodes, interactions of up to $J-1$ variables may be estimated.

In practice, $J$ is typically determined through trial and error to maximize performance. *Hastie* et al. [30.12] indicate that $4 \le J \le 8$ terminal nodes per tree typically work well, with little sensitivity to choices within that range. For some applications, boosting stumps ($J = 2$) may be sufficient, and very rarely is $J > 10$ needed.

#### Interpretation

Although boosting trees provides significant improvements in classification and predictive accuracy, these benefits do come at a cost. Because the final model is comprised of the weighted average of many weaker models, we lose the attractive structural interpretability of a single tree.

However, additional useful information can still be gleaned from the data. As we discussed in Sect. 30.2.5, *Breiman* et al. [30.5] provide a measure of the relative importance of predictor variables in a single tree. This measure is easily generalized to the context of boosting. Single-tree importance measures are calculated for each weak learner and averaged over the group. In $K$-class classification problems, importance measures are generated in this manner for each class. These values can be averaged across classes to obtain overall importance measures for each variable, or across subsets of variables to determine the relevance of each subset in predicting each class.

Once the most relevant variables are identified, certain visualization tools can aid interpretation. *Hastie* et al. [30.12] suggest the use of partial dependence plots to look for interactions between variables.

### 30.4.2 Random Forest

*Breiman* [30.26, 27] developed random forests (RF) based on bagging and random feature selection [30.28, 34]. Bagging is a resampling procedure that produces bootstrap samples by randomly sampling with replacement from the original training sample. A random forest is essentially an ensemble of CART trees in which each tree is grown in accordance with a different bootstrap sample. Suppose $M$ bootstrap samples are generated, viewing them as realizations of independent identically distributed (iid) random vectors $\Theta_1, \ldots, \Theta_M$, we denote the random forest by $h(\boldsymbol{x}; \Theta)$ as an ensemble of individual CART trees $h(\boldsymbol{x}; \Theta_j)$, $j = 1, \ldots, M$. For classification problems, the final prediction of the forest is made by majority vote,

$$h(\boldsymbol{x}; \Theta) = \arg\max_k \sum_{j=1}^{M} I\left[h(\boldsymbol{x}; \Theta_j) = k\right] ;$$

for regression problems, the final prediction is obtained by aggregating over $M$ trees, typically using equal weights,

$$h(\boldsymbol{x}; \Theta) = \frac{1}{M} \sum_{j=1}^{M} h(\boldsymbol{x}; \Theta_j) .$$

In accordance with the basic principle of bagging to reduce prediction errors from averaging over the ensemble, better accuracy of the random forest can be obtained by keeping errors of individual trees low, and

minimization of the correlation between multiple trees. Therefore, individual trees are not pruned but grown to maximum depth. Recently, *Segal* [30.35] suggested that this strategy can overfit the data and it is beneficial to regulate tree size by limiting the number of splits and/or the size of nodes for which splitting is allowed. In addition, the correlation of multiple trees can be reduced by random feature selection. Instead of determining the split at a given node in an individual tree using all the predictors, only $m < p$ randomly selected predictors are considered for the split. This also enables the algorithm to build models for high-dimensional data very quickly. Alternatives to this random feature selection include: (1) picking the best out of several random feature subsets by comparing how well the subsets perform on the samples left out of the bootstrap training sample (out-of-bag samples), and (2) using random linear combinations of features in the selected feature subset, i. e., selected features are added together with coefficients that are uniform random numbers on $[-1, 1]$. Due to the large number of simple trees and the minimized correlations among the individual trees, the prediction error of the forest converges toward the error rates comparable to AdaBoost [30.29].

Usually, about one third of the observations are left out of each bootstrap sample. These out-of-bag (oob) observations are used to internally estimate prediction error for future data, the strength of each tree, and correlation between trees; see details in *Breiman* [30.27]. This avoids the cross-validation needed for construction of a single tree and greatly enhances the computational efficiency of random forests.

With random forests, an intuitive measure of variable importance can be computed as follows. In every tree grown in the forest, put down the oob cases and count the number of votes cast for the correct class. Now randomly permute the values of variable $m$ in the oob cases and put these cases down the tree. Subtract the number of votes for the correct class in the variable-$m$-permuted oob data from the number of votes for the correct class in the untouched oob data. The average of this number over all trees in the forest is the raw importance score for the variable $m$.

For each case, consider all the trees for which it is oob. Subtract the percentage of votes for the correct class in the variable-$m$-permuted oob data from the percentage of votes for the correct class in the untouched oob data. This is the local importance score for variable $m$ for this case, and is used in the graphics program RAFT (Random Forest Tool).

For further details on random forests, please refer to the random forests website http://www.math.usu.edu/~adele/forests/cc_home.htm maintained by Professor Adele Cutler at Utah State University.

In summary, random forests do not overfit and and enjoy prediction accuracy that is as good as AdaBoost and sometimes better. The algorithm runs fast on large high-dimensional data and is somewhat robust to outliers. It also has an effective mechanism for handling missing data. In the forest-building process, it internally estimates the classification error, the strength of each tree and the correlation between trees. It also distinguishes itself from some black-box methods (e.g. neural networks) by providing the importance score for each predictor, and hence makes the output more interpretable to users. Furthermore, random forests can serve as an exploratory tool to find interactions among predictors, locate outliers and provide interesting views of the data. Its application can also be extended to unsupervised clustering.



## 30.5 Conclusion

In this chapter, we have discussed several tree-based methods for classification and regression. Of course, many more supervised learning methods are available, including multivariant adaptive regression splines (MARS), neural networks, and support vector machines (SVM). In this last section, we discuss the relative merits of tree-based methods among this much larger set of well-known supervised learning tools.

*Hastie* et al. [30.12] note that typical characteristics found in real-world data sets make direct application of most supervised-learning tools difficult. First, data-mining applications tend to involve very large data sets in terms of both the number of observations and the number of variables (the majority of which are often irrelevant). Moreover, these data sets generally contain both quantitative and qualitative variables. The quantitative variables are typically measured on different scales, and the qualitative variables may have different numbers of categories. Missing data are abundant, and outliers are also very common.

Tree-based methods are particularly well-suited to deal with these difficulties. Trees grow quickly, so the

size of a data set is not a big concern. Tree algorithms readily admit mixed variable types, and feature selection is a part of the building process, so irrelevant variables have little impact on the resulting model. Tree-building methods account for missing data in an effective way, and the results for classification or prediction are often robust against outliers.

Many other supervised learning methods fall short in some of these areas. MARS has difficulty with outliers in predictor variables, and transformations on variables can dramatically impact its results. Neural networks and SVM both require dummy coding of categorical variables, they are not adept at handling missing values, and they are sensitive to outliers and transformations.

Tree-based methods have one other important advantage over black-box techniques such as neural networks; tree models are much more readily interpretable. This characteristic is vital to those applications for which predictive accuracy is secondary to the main goal of obtaining qualitative insight into the structure of the data.

In spite of these advantages, tree-based methods do suffer one key drawback: a relative lack of predictive power. Neural networks and support vector machines commonly outperform classification and regression trees, particularly when the underlying structure of the data depends on linear combinations of variables. As we discussed in Sect. 30.4, ensemble methods such as boosting and random forests can be quite effective at improving their accuracy. However, this predictive improvement comes with some cost. Ensemble methods lose the interpretive value in a single tree, and they are much more computationally expensive. The tree-based methods do not always yield the best possible results for classification and prediction, but they are worth a try in a wide variety of applications. In any scientific application, we certainly encourage you to see the forest – not just a few trees.

## References


30.1 C. L. Blake, C. J. Merz: *UCI repository of machine learning databases http://www.ics.uci.edu/mlearn/MLRepository.html* (Department of Information and Computer Science (Univ. California), Irvine 1998)

30.2 K.-Y. Chan, W.-Y. Loh: LOTUS: An algorithm for building accurate, comprehensible logistic regression trees, J. Comput. Graph. Stat. **13**(4), 826–852 (2004)

30.3 Federal Emergency Management Agency: *Typical Costs of Seismic Rehabilitation of Existing Buildings*, FEMA 156, Vol. 1–Summary, 2nd edn. (FEMA, Washington 1993)

30.4 Federal Emergency Management Agency: *Typical Costs of Seismic Rehabilitation of Existing Buildings*, FEMA 157, Vol. 2–Supporting Documentation, 2nd edn. (FEMA, Washington 1993)

30.5 L. Breiman, J. Friedman, R. Olshen, C. Stone: *Classification and Regression Trees* (Chapman Hall, New York 1984)

30.6 W.-Y. Loh, Y.-S. Shih: Split selection methods for classificaiton trees, Stat. Sin. **7**, 815–840 (1997)

30.7 H. Kim, W.-Y. Loh: Classification trees with unbiased multiway splits, J. Am. Stat. Assoc. **96**, 589–604 (2001)

30.8 W.-Y. Loh: Regression trees with unbiased variable selection, interaction detection, Stat. Sin. **12**, 361–386 (2002)

30.9 J. R. Quinlan: *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Mateo 1993)

30.10 G. V. Kass: An exploratory technique for investigating large quantities of categorical data, Appl. Stat. **29**, 119–127 (1980)

30.11 R. A. Fisher: The use of multiple measurements in taxonomic problems, Ann. Eugenic. **7**, 179–188 (1936)

30.12 T. Hastie, R. Tibshirani, J. Friedman: *The Elements of Statistical Learning: Data Mining, Inference, Prediction* (Springer, Berlin Heidelberg New York 2001)

30.13 F. Esposito, D. Malerba, G. Semeraro: A comparative analysis of methods for pruning decision trees, IEEE Trans. Pattern Anal. **19**, 476–491 (1997)

30.14 P. Ein-Dor, J. Feldmesser: Attributes of the performance of central processing units: a relative performance prediction model, Commun. ACM **30**, 308–317 (1987)

30.15 R. J. Little, D. B. Rubin: *Statistical Analysis with Missing Data*, 2nd edn. (Wiley, Boboken 2002)

30.16 L. Breiman: Bagging predictors, Mach. Learn. **24**, 123–140 (1996)

30.17 H. Drucker, C. Cortes: Boosting decision trees. In: *Adv. Neur. Inf. Proc. Syst.*, Proc. NIPS'95, Vol. 8, ed. by M. C. Mozer D. S. Touretzky, E. Hasselmo (Ed.) M. (MIT Press, Cambridge 1996) pp. 479–485

30.18 W.-Y. Loh, N. Vanichsetakul: Tree-structured classification via generalized discriminant analysis (with discussion), J. Am. Stat. Assoc. **83**, 715–728 (1988)

30.19 J. A. Hartigan, M. A. Wong: Algorithm 136, A *k*-means clustering algorithm, Appl. Stat. **28**, 100 (1979)
30.20 J. R. Quinlan: Discovering rules by induction from large collections of examples. In: *Expert Systems in the Micro-Electronic Age*, ed. by D. Michie (Edinburgh Univ. Press, Edinburgh 1979) pp. 168–201
30.21 E. B. Hunt, J. Marin, P. J. Stone: *Experiments in Induction* (Academic, New York 1966)
30.22 J. Dougherty, R. Kohavi, M. Sahami: Supervised, unsupervised discretization of continuous features. In: *Proceedings of the Twelfth International Conference on Machine Learning*, ed. by A. Prieditis, S. J. Russel (Morgan Kaufmann, San Mateo 1995) pp. 194–202
30.23 J. R. Quinlan: Improved use of continuous attributes in C4.5, J. Artif. Intell. Res. **4**, 77–90 (1996)
30.24 T.-S. Lim, W.-Y. Loh, Y.-S. Shih: A comparison of prediction accuracy, complexity, training time of thirty-three old and new classification algorithms, Mach. Learn. J. **40**, 203–228 (2000)
30.25 E. Bauer, R. Kohavi: An empirical comparison of voting classification algorithms: bagging, boosting, variants, Mach. Learn. **36**, 105–139 (1999)
30.26 L. Breiman: Statistical modeling: the two cultures, Stat. Sci. **16**, 199–215 (2001)
30.27 L. Breiman: Random forests, Mach. Learn. **45**, 5–32 (2001)
30.28 T. G. Dietterich: An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, randomization, Mach. Learn. **40**, 139–157 (2000)
30.29 Y. Freund, R. E. Schapire: Experiments with a new boosting algorithm. In: *Machine Learning: Proceedings of the Thirteenth International Conference*, ed. by L. Saitta (Morgan Kaufmann, San Mateo 1996) pp. 148–156
30.30 R. Schapire: The strength of weak learnability, Mach. Learn. **5**(2), 197–227 (1990)
30.31 Y. Freund: Boosting aweak learning algorithm by majority, Inform. Comput. **121**(2), 256–285 (1995)
30.32 Y. Freund, R. E. Schapire: A decision-theoretic generalization of on-line learning, an application to boosting, J. Comput. Syst. Sci. **55**, 119–139 (1997)
30.33 J. Friedman, T. Hastie, R. Tibshirani: Additive logistic regression: astatistical view of boosting (with discussion), Ann. Stat. **28**, 337–374 (2000)
30.34 T. K. Ho: The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. **20**, 832–844 (1998)
30.35 M. R. Segal: *Machine learning benchmarks, random forest regression*, Technical Report, Center for Bioinformatics and Molecular Biostatistics (Univ. California, San Francisco 2004)