Chapter 2
Decision-Tree Induction

Abstract Decision-tree induction algorithms are highly used in a variety of domains for knowledge discovery and pattern recognition. They have the advantage of producing a comprehensible classification/regression model and satisfactory accuracy levels in several application domains, such as medical diagnosis and credit risk assessment. In this chapter, we present in detail the most common approach for decision-tree induction: top-down induction (Sect. 2.3). Furthermore, we briefly comment on some alternative strategies for induction of decision trees (Sect. 2.4). Our goal is to summarize the main design options one has to face when building decision-tree induction algorithms. These design choices will be specially interesting when designing an evolutionary algorithm for evolving decision-tree induction algorithms.

Keywords Decision trees · Hunt's algorithm · Top-down induction · Design components

2.1 Origins

Automatically generating rules in the form of decision trees has been object of study of most research fields in which data exploration techniques have been developed [78]. Disciplines like engineering (pattern recognition), statistics, decision theory, and more recently artificial intelligence (machine learning) have a large number of studies dedicated to the generation and application of decision trees.

In statistics, we can trace the origins of decision trees to research that proposed building binary segmentation trees for understanding the relationship between target and input attributes. Some examples are AID [107], MAID [40], THAID [76], and CHAID [55]. The application that motivated these studies is survey data analysis. In engineering (pattern recognition), research on decision trees was motivated by the need to interpret images from remote sensing satellites in the 70s [46]. Decision trees, and induction methods in general, arose in machine learning to avoid the knowledge acquisition bottleneck for expert systems [78].

Specifically regarding top-down induction of decision trees (by far the most popular approach of decision-tree induction), Hunt's Concept Learning System (CLS)

[49] can be regarded as the pioneering work for inducing decision trees. Systems that directly descend from Hunt's CLS are ID3 [91], ACLS [87], and Assistant [57].

2.2 Basic Concepts

Decision trees are an efficient nonparametric method that can be applied either to classification or to regression tasks. They are hierarchical data structures for supervised learning whereby the input space is split into local regions in order to predict the dependent variable [2].

A decision tree can be seen as a graph $G = (V, E)$ consisting of a finite, nonempty set of nodes (vertices) V and a set of edges E. Such a graph has to satisfy the following properties [101]:

- The edges must be ordered pairs (v, w) of vertices, i.e., the graph must be directed;
- There can be no cycles within the graph, i.e., the graph must be acyclic;
- There is exactly one node, called the root, which no edges enter;
- Every node, except for the root, has exactly one entering edge;
- There is a unique path—a sequence of edges of the form $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$—from the root to each node;
- When there is a path from node v to w, $v \neq w$, v is a proper *ancestor* of w and w is a proper *descendant* of v. A node with no proper descendant is called a *leaf* (or a *terminal*). All others are called *internal* nodes (except for the root).

Root and internal nodes hold a test over a given data set attribute (or a set of attributes), and the edges correspond to the possible outcomes of the test. Leaf nodes can either hold class labels (classification), continuous values (regression), (non-) linear models (regression), or even models produced by other machine learning algorithms. For predicting the dependent variable value of a certain instance, one has to navigate through the decision tree. Starting from the root, one has to follow the edges according to the results of the tests over the attributes. When reaching a leaf node, the information it contains is responsible for the prediction outcome. For instance, a traditional decision tree for classification holds class labels in its leaves.

Decision trees can be regarded as a disjunction of conjunctions of constraints on the attribute values of instances [74]. Each path from the root to a leaf is actually a conjunction of attribute tests, and the tree itself allows the choice of different paths, that is, a disjunction of these conjunctions.

Other important definitions regarding decision trees are the concepts of *depth* and *breadth*. The average number of layers (levels) from the root node to the terminal nodes is referred to as the *average depth* of the tree. The average number of internal nodes in each level of the tree is referred to as the *average breadth* of the tree. Both depth and breadth are indicators of tree complexity, that is, the higher their values are, the more complex the corresponding decision tree is.

In Fig. 2.1, an example of a general decision tree for classification is presented. Circles denote the root and internal nodes whilst squares denote the leaf nodes. In

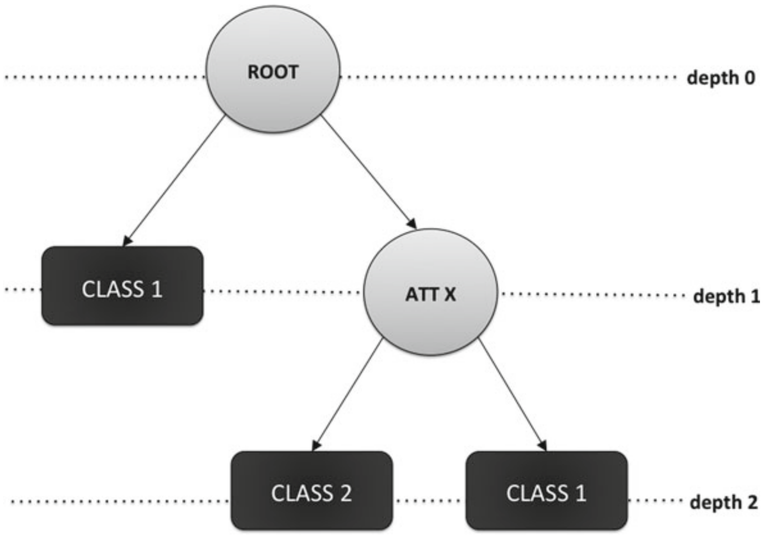

Fig. 2.1 Example of a general decision tree for classification

this particular example, the decision tree is designed for classification and thus the leaf nodes hold class labels.

There are many decision trees that can be grown from the same data. Induction of an optimal decision tree from data is considered to be a hard task. For instance, Hyafil and Rivest [50] have shown that constructing a minimal binary tree with regard to the expected number of tests required for classifying an unseen object is an NP-complete problem. Hancock et al. [43] have proved that finding a minimal decision tree consistent with the training set is NP-Hard, which is also the case of finding the minimal equivalent decision tree for a given decision tree [129], and building the optimal decision tree from decision tables [81]. These papers indicate that growing optimal decision trees (a brute-force approach) is only feasible in very small problems.

Hence, it was necessary the development of heuristics for solving the problem of growing decision trees. In that sense, several approaches which were developed in the last three decades are capable of providing reasonably accurate, if suboptimal, decision trees in a reduced amount of time. Among these approaches, there is a clear preference in the literature for algorithms that rely on a greedy, top-down, recursive partitioning strategy for the growth of the tree (top-down induction).

2.3 Top-Down Induction

Hunt's Concept Learning System framework (CLS) [49] is said to be the pioneer work in top-down induction of decision trees. CLS attempts to minimize the cost of classifying an object. Cost, in this context, is referred to two different concepts: the measurement cost of determining the value of a certain property (attribute) exhibited by the object, and the cost of classifying the object as belonging to class j when it actually belongs to class k. At each stage, CLS exploits the space of possible decision trees to a fixed depth, chooses an action to minimize cost in this limited space, then moves one level down in the tree.

In a higher level of abstraction, Hunt's algorithm can be recursively defined in only two steps. Let $\mathbf{X}_t$ be the set of training instances associated with node t and $y = \{y_1, y_2, \ldots, y_k\}$ be the class labels in a k-class problem [110]:

1. If all the instances in $\mathbf{X}_t$ belong to the same class y_t then t is a leaf node labeled as y_t
2. If $\mathbf{X}_t$ contains instances that belong to more than one class, an attribute test condition is selected to partition the instances into smaller subsets. A child node is created for each outcome of the test condition and the instances in $\mathbf{X}_t$ are distributed to the children based on the outcomes. Recursively apply the algorithm to each child node.

Hunt's simplified algorithm is the basis for all current top-down decision-tree induction algorithms. Nevertheless, its assumptions are too stringent for practical use. For instance, it would only work if every combination of attribute values is present in the training data, and if the training data is inconsistency-free (each combination has a unique class label).

Hunt's algorithm was improved in many ways. Its stopping criterion, for example, as expressed in step 1, requires all leaf nodes to be pure (i.e., belonging to the same class). In most practical cases, this constraint leads to enormous decision trees, which tend to suffer from *overfitting* (an issue discussed later in this chapter). Possible solutions to overcome this problem include prematurely stopping the tree growth when a minimum level of impurity is reached, or performing a *pruning* step after the tree has been fully grown (more details on other stopping criteria and on pruning in Sects. 2.3.2 and 2.3.3). Another design issue is how to select the attribute test condition to partition the instances into smaller subsets. In Hunt's original approach, a cost-driven function was responsible for partitioning the tree. Subsequent algorithms such as ID3 [91, 92] and C4.5 [89] make use of information theory based functions for partitioning nodes in purer subsets (more details on Sect. 2.3.1).

An up-to-date algorithmic framework for top-down induction of decision trees is presented in [98], and we reproduce it in Algorithm 1. It contains three procedures: one for growing the tree (*treeGrowing*), one for pruning the tree (*treePruning*) and one to combine those two procedures (*inducer*). The first issue to be discussed is how to select the test condition $f(A)$, i.e., how to select the best combination of attribute(s) and value(s) for splitting nodes.

Algorithm 1 Generic algorithmic framework for top-down induction of decision trees. Inputs are the training set $\mathbf{X}$, the predictive attribute set A and the target attribute y.

```
1: procedure inducer($\mathbf{X}$, $A$, $y$)
2:     $T$ = treeGrowing($\mathbf{X}$, $A$, $y$)
3:     return treePruning($\mathbf{X}$, $T$)
4: end procedure
5: procedure treeGrowing($\mathbf{X}$, $A$, $y$)
6:     Create a tree $T$
7:     if one of the stopping criteria is fulfilled then
8:         Mark the root node in $T$ as a leaf with the most common value of $y$ in $\mathbf{X}$
9:     else
10:        Find an attribute test condition $f(A)$ such that splitting $\mathbf{X}$ according to $f(A)$'s outcomes $(v_1, \ldots, v_l)$ yields
             the best splitting measure value
11:        if best splitting measure value > threshold then
12:            Label the root node in $T$ as $f(A)$
13:            for each outcome $v_i$ of $f(A)$ do
14:                $\mathbf{X_{f(A)=v_i}} = \{x \in \mathbf{X} \mid f(A) = v_i\}$
15:                $Subtree_i$ = treeGrowing($\mathbf{X_{f(A=v_i)}}$, $A$, $y$)
16:                Connect the root node of $T$ to $Subtree_i$ and label the corresponding edge as $v_i$
17:            end for
18:        else
19:            Mark the root node of $T$ as a leaf and label it as the most common value of $y$ in $\mathbf{X}$
20:        end if
21:    end ifreturn $T$
22: end procedure
23: procedure treePruning($\mathbf{X}$, $T$)
24:    repeat
25:        Select a node $t$ in $T$ such that pruning it maximally improves some evaluation criterion
26:        if $T \neq \emptyset$ then
27:            $T$ = pruned($T$, $t$)
28:        end if
29:    until $T = \emptyset$ return $T$
30: end procedure
```

2.3.1 Selecting Splits

A major issue in top-down induction of decision trees is which attribute(s) to choose for splitting a node in subsets. For the case of *axis-parallel* decision trees (also known as *univariate*), the problem is to choose the attribute that better discriminates the input data. A decision rule based on such an attribute is thus generated, and the input data is filtered according to the outcomes of this rule. For *oblique* decision trees (also known as *multivariate*), the goal is to find a combination of attributes with good discriminatory power. Either way, both strategies are concerned with ranking attributes quantitatively.

We have divided the work in univariate criteria in the following categories: (i) information theory-based criteria; (ii) distance-based criteria; (iii) other classification criteria; and (iv) regression criteria. These categories are sometimes fuzzy and do not constitute a taxonomy by any means. Many of the criteria presented in a given category can be shown to be approximations of criteria in other categories.

2.3.1.1 Information Theory-Based Criteria

Examples of this category are criteria based, directly or indirectly, on Shannon's entropy [104]. Entropy is known to be a unique function which satisfies the four axioms of uncertainty. It represents the average amount of information when coding each class into a codeword with ideal length according to its probability. Some interesting facts regarding entropy are:

- For a fixed number of classes, entropy increases as the probability distribution of classes becomes more uniform;
- If the probability distribution of classes is uniform, entropy increases logarithmically as the number of classes in a sample increases;
- If a partition induced on a set $\mathbf{X}$ by an attribute a_j is a refinement of a partition induced by a_i, then the entropy of the partition induced by a_j is never higher than the entropy of the partition induced by a_i (and it is only equal if the class distribution is kept identical after partitioning). This means that progressively refining a set in sub-partitions will continuously decrease the entropy value, regardless of the class distribution achieved after partitioning a set.

The first splitting criterion that arose based on entropy is the *global mutual information* (GMI) [41, 102, 108], given by:

$$GMI(a_i, \mathbf{X}, y) = \frac{1}{N_x} \sum_{l=1}^{k} \sum_{j=1}^{|a_i|} N_{v_j \cap y_l} \log_e \frac{N_{v_j \cap y_l} N_x}{N_{v_j, \bullet} N_{\bullet, y_l}} \tag{2.1}$$

Ching et al. [22] propose the use of GMI as a tool for supervised discretization. They name it *class-attribute mutual information*, though the criterion is exactly the same. GMI is bounded by zero (when a_i and y are completely independent) and its maximum value is $max(\log_2 |a_i|, \log_2 k)$ (when there is a maximum correlation between a_i and y). Ching et al. [22] reckon this measure is biased towards attributes with many distinct values, and thus propose the following normalization called *class-attribute interdependence redundancy* (CAIR):

$$CAIR(a_i, \mathbf{X}, y) = \frac{GMI}{-\sum_{j=1}^{|a_i|} \sum_{l=1}^{k} p_{v_j \cap y_l} \log_2 p_{v_j \cap y_l}} \tag{2.2}$$

which is actually dividing GMI by the joint entropy of a_i and y. Clearly $CAIR(a_i, \mathbf{X}, y) \geq 0$, since both GMI and the joint entropy are greater (or equal) than zero. In fact, $0 \leq CAIR(a_i, \mathbf{X}, y) \leq 1$, with $CAIR(a_i, \mathbf{X}, y) = 0$ when a_i and y are totally independent and $CAIR(a_i, \mathbf{X}, y) = 1$ when they are totally dependent. The term *redundancy* in CAIR comes from the fact that one may discretize a continuous attribute in intervals in such a way that the class-attribute interdependence is kept intact (i.e., *redundant* values are combined in an interval). In the decision tree partitioning context, we must look for an attribute that maximizes CAIR (or similarly, that maximizes GMI).

Information gain [18, 44, 92, 122] is another example of measure based on Shannon's entropy. It belongs to the class of the so-called *impurity-based criteria*. The term *impurity* refers to the level of class separability among the subsets derived from a split. A *pure* subset is the one whose instances belong all to the same class. Impurity-based criteria are usually measures with values in [0, 1] where 0 refers to the purest subset possible and 1 the impurest (class values are equally distributed among the subset instances). More formally, an impurity-based criterion $\phi(.)$ presents the following properties:

- $\phi(.)$ is minimum if $\exists i$ such that $p_{\bullet,y_i} = 1$;
- $\phi(.)$ is maximum if $\forall i$, $1 \leq i \leq k$, $p_{\bullet,y_i} = 1/k$;
- $\phi(.)$ is symmetric with respect to components of p_y;
- $\phi(.)$ is smooth (differentiable everywhere) in its range.

Note that impurity-based criteria tend to favor a particular split for which, on average, the class distribution in each subset is most uneven. The impurity is measured before and after splitting a node according to each possible attribute. The attribute which presents the greater gain in purity, i.e., that maximizes the difference of impurity taken before and after splitting the node, is chosen. The gain in purity ($\Delta\Phi$) can be defined as:

$$\Delta\Phi(a_i, \mathbf{X}, y) = \phi(y, \mathbf{X}) - \sum_{j=1}^{|a_i|} p_{v_j,\bullet} \times \phi(y, \mathbf{X}_{\mathbf{a_i=v_j}}) \tag{2.3}$$

The goal of information gain is to maximize the reduction in entropy due to splitting each individual node. Entropy can be defined as:

$$\phi^{entropy}(\mathbf{X}, y) = -\sum_{l=1}^{k} p_{\bullet,y_l} \times \log_2 p_{\bullet,y_l}. \tag{2.4}$$

If entropy is calculated in (2.3), then $\Delta\Phi(a_i, \mathbf{X})$ is the information gain measure, which calculates the goodness of splitting the instance space $\mathbf{X}$ according to the values of attribute a_i.

Wilks [126] has proved that as $N \to \infty$, $2 \times N_x \times GMI(a_i, \mathbf{X}, y)$ (or similarly replacing GMI by information gain) approximate the χ^2 distribution. This measure is often regarded as the *G statistics* [72, 73]. White and Liu [125] point out that the G statistics should be adjusted since the work of Mingers [72] uses logarithms to base e, instead of logarithms to base 2. The adjusted G statistics is given by $2 \times N_x \times \Delta\Phi^{IG} \times \log_e 2$. Instead of using the value of this measure as calculated, we can compute the probability of such a value occurring from the χ^2 distribution on the assumption that there is no association between the attribute and the classes. The higher the calculated value, the less likely it is to have occurred given the assumption. The advantage of using such a measure is making use of the levels of significance it provides for deciding whether to include an attribute at all.

Quinlan [92] acknowledges the fact that the information gain is biased towards attributes with many values. This is a consequence of the previously mentioned particularity regarding entropy, in which further refinement leads to a decrease in its value. Quinlan proposes a solution for this matter called *gain ratio* [89]. It basically consists of normalizing the information gain by the entropy of the attribute being tested, that is,

$$\Delta\Phi^{gainRatio}(a_i, \mathbf{X}, y) = \frac{\Delta\Phi^{IG}}{\phi^{entropy}(a_i, \mathbf{X})}. \tag{2.5}$$

The gain ratio compensates the decrease in entropy in multiple partitions by dividing the information gain by the attribute self-entropy $\phi^{entropy}(a_i, \mathbf{X})$. The value of $\phi^{entropy}(a_i, \mathbf{X})$ increases logarithmically as the number of partitions over a_i increases, decreasing the value of gain ratio. Nevertheless, the gain ratio has two deficiencies: (i) it may be undefined (i.e., the value of self-entropy may be zero); and (ii) it may choose attributes with very low self-entropy but not with high gain. For solving these issues, Quinlan suggests first calculating the information gain for all attributes, and then calculating the gain ratio only for those cases in which the information gain value is above the average value of all attributes.

Several variations of the gain ratio have been proposed. For instance, the *normalized gain* [52] replaces the denominator of gain ratio by $\log_2 |a_i|$. The authors demonstrate two theorems with cases in which the normalized gain works better than or at least equally as either information gain or gain ratio does. In the first theorem, they prove that if two attributes a_i and a_j partition the instance space in pure sub-partitions, and that if $|a_i| > |a_j|$, normalized gain will always prefer a_j over a_i, whereas gain ratio is dependent of the self-entropy values of a_i and a_j (which means gain ratio may choose the attribute that partitions the space in more values). The second theorem states that given two attributes a_i and a_j, $|a_i| = |a_j|$, $|a_i| \geq 2$, if a_i partitions the instance space in pure subsets and a_j has at least one subset with more than one class, normalized gain will always prefer a_i over a_j, whereas gain ratio will prefer a_j if the following condition is met:

$$\frac{E(a_j, \mathbf{X}, y)}{\phi^{entropy}(y, \mathbf{X})} \leq 1 - \frac{\phi^{entropy}(a_j, \mathbf{X})}{\phi^{entropy}(a_i, \mathbf{X})}$$

where:

$$E(a_j, \mathbf{X}, y) = -\sum_{l=1}^{|a_j|} p_{v_l,\bullet} \times \phi^{entropy}(y, \mathbf{X}_{\mathbf{a_j=v_l}}) \tag{2.6}$$

For details on the proof of each theorem, please refer to Jun et al. [52].

Other variation is the *average gain* [123], that replaces the denominator of gain ratio by $|dom(a_i)|$ (it only works for nominal attributes). The authors do not demonstrate theoretically any situations in which this measure is a better option than gain ratio. Their work is supported by empirical experiments in which the average gain outperforms gain ratio in terms of runtime and tree size, though with no significant differences regarding accuracy. Note that most decision-tree induction algorithms provide one branch for each nominal value an attribute can take. Hence, the average gain [123] is practically identical to the normalized gain [52], though without scaling the number of values with $\log_2$.

Sá et al. [100] propose a somewhat different splitting measure based on the *minimum entropy of error* principle (MEE) [106]. It does not directly depend on the class distribution of a node p_{v_j,y_l} and the prevalences $p_{v_j,\bullet}$, but instead it depends on the errors produced by the decision rule on the form of a Stoller split [28]: if $a_i(x) \leq \Delta, y(x) = y_\omega; \hat{y}$ otherwise. In a Stoller split, each node split is binary and has an associated class y_ω for the case $a_i(x) \leq \Delta$, while the remaining classes are denoted by $\hat{y}$ and associated to the complementary branch. Each class is assigned a code $t \in \{-1, 1\}$, in such a way that for $y(x) = y_\omega, t = 1$ and for $y(x) = \hat{y}, t = -1$. The splitting measure is thus given by:

$$MEE(a_i, \mathbf{X}, y) = -(P_{-1} log_e P_{-1} + P_0 log_e P_0 + P_1 log_e P_1)$$

$$\text{where:}$$

$$\begin{aligned} P_{-1} &= \frac{N_{\bullet,y_l}}{n} \times \frac{e_{1,-1}}{N_x} \\ P_1 &= \left(1 - \frac{N_{\bullet,y_l}}{n}\right) \times \frac{e_{-1,1}}{N_x} \\ P_0 &= 1 - P_{-1} - P_1 \end{aligned} \tag{2.7}$$

where $e_{t,t'}$ is the number of instances t classified as t'. Note that unlike other measures such as information gain and gain ratio, there is no need of computing the impurity of sub-partitions and their subsequent average, as MEE does all the calculation needed at the current node to be split. MEE is bounded by the interval $[0, \log_e 3]$, and needs to be minimized. The meaning of minimizing MEE is constraining the probability mass function of the errors to be as narrow as possible (around zero). The authors argue that by using MEE, there is no need of applying the pruning operation, saving execution time of decision-tree induction algorithms.

2.3.1.2 Distance-Based Criteria

Criteria in this category evaluate separability, divergency or discrimination between classes. They measure the distance between class probability distributions.

A popular distance criterion which is also from the class of impurity-based criteria is the *Gini index* [12, 39, 88]. It is given by:

$$\phi^{Gini}(y, \mathbf{X}) = 1 - \sum_{l=1}^{k} {p_{\bullet,y_l}}^2 \tag{2.8}$$

Breiman et al. [12] also acknowledge Gini's bias towards attributes with many values. They propose the *twoing* binary criterion for solving this matter. It belongs to the class of binary criteria, which requires attributes to have their domain split into two mutually exclusive subdomains, allowing binary splits only. For every binary criteria, the process of dividing attribute a_i values into two subdomains, d_1 and d_2, is exhaustive[1] and the division that maximizes its value is selected for attribute a_i. In other words, a binary criterion β is tested over all possible subdomains in order to provide the optimal binary split, β^*:

$$\begin{aligned} &\beta^* = \max_{d_1,d_2} \beta(a_i, d_1, d_2, \mathbf{X}, y) \\ &s.t. \\ &d_1 \cup d_2 = dom(a_i) \\ &d_1 \cap d_2 = \emptyset \end{aligned} \tag{2.9}$$

Now that we have defined binary criteria, the twoing binary criterion is given by:

$$\beta^{twoing}(a_i, d_1, d_2, \mathbf{X}, y) = 0.25 \times p_{d_1,\bullet} \times p_{d_2,\bullet} \times \left(\sum_{l=1}^{k} abs(p_{y_l|d_1} - p_{y_l|d_2}) \right)^2 \tag{2.10}$$

where $abs(.)$ returns the absolute value.

Friedman [38] and Rounds [99] propose a binary criterion based on the Kolmogorov-Smirnoff (KS) distance for handling binary-class problems:

$$\beta^{KS}(a_i, d_1, d_2, \mathbf{X}, y) = abs(p_{d_1|y_1} - p_{d_1|y_2}) \tag{2.11}$$

Haskell and Noui-Mehidi [45] propose extending β^{KS} for handling multi-class problems. Utgoff and Clouse [120] also propose a multi-class extension to β^{KS}, as well as missing data treatment, and they present empirical results which show their criterion is similar in accuracy to Quinlan's gain ratio, but produces smaller-sized trees.

The χ^2 statistic [72, 125, 130] has been employed as a splitting criterion in decision trees. It compares the observed values with those that one would expect if there were no association between attribute and class. The resulting statistic is distributed approximately as the chi-square distribution, with larger values indicating greater association. Since we are looking for the predictive attribute with the highest degree of association to the class attribute, this measure must be maximized. It can be calculated as:

[1] Coppersmith et al. [25] present an interesting heuristic procedure for finding subdomains d_1 and d_2 when the partition is based on a nominal attribute. Shih [105] also investigates the problem of efficiency on finding the best binary split for nominal attributes.

$$\chi^2(a_i, \mathbf{X}, y) = \sum_{j=1}^{|a_i|} \sum_{l=1}^{k} \frac{\left(N_{v_j \cap y_l} - \frac{N_{v_j,\bullet} \times N_{\bullet,y_l}}{N_x}\right)^2}{\frac{N_{v_j,\bullet} \times N_{\bullet,y_l}}{N_x}} \tag{2.12}$$

It should be noticed that the χ^2 statistic (and similarly the G statistic) become poor approximations with small expected frequencies. Small frequencies make χ^2 over-optimistic in detecting informative attributes, i.e., the probability derived from the distribution will be smaller than the true probability of getting a value of χ^2 as large as that obtained.

De Mántaras [26] proposes a distance criterion that "provides a clearer and more formal framework for attribute selection and solves the problem of bias in favor of multivalued attributes without having the limitations of Quinlan's Gain Ratio". It is actually the same normalization to information gain as CAIR is to GMI, i.e.,

$$1 - \Delta\Phi^{distance}(a_i, \mathbf{X}, y) = \frac{\Delta\Phi^{IG}}{-\sum_{j=1}^{|a_i|} \sum_{l=1}^{k} p_{v_j \cap y_l} \log_2 p_{v_j \cap y_l}}. \tag{2.13}$$

Notice that in (2.13), we are actually presenting the complement of Mántaras distance measure, i.e., $1 - \Delta\Phi^{distance}(a_i, \mathbf{X}, y)$, but with no implications in the final result (apart from the fact that (2.13) needs to be maximized, whereas the original distance measure should be minimized).

Fayyad and Irani [35] propose a new family of measures called C-SEP (from Class SEParation). They claim that splitting measures such as information gain (and similar impurity-based criteria) suffer from a series of deficiencies (e.g., they are insensitive to within-class fragmentation), and they present new requirements a "good" splitting measure $\Gamma(.)$ (in particular, binary criteria) should fulfill:

- $\Gamma(.)$ is maximum when classes in d_1 and d_2 are disjoint (inter-class separability);
- $\Gamma(.)$ is minimum when the class distributions in d_1 and d_2 are identical;
- $\Gamma(.)$ favors partitions which keep instances from the same class in the same sub-domain d_i (intra-class cohesiveness);
- $\Gamma(.)$ is sensitive to permutations in the class distribution;
- $\Gamma(.)$ is non-negative, smooth (differentiable), and symmetric with respect to the classes.

Binary criteria that fulfill the above requirements are based on the premise that a good split is the one that separates as many different classes from each other as possible, while keeping examples of the same class together. Γ must be maximized, unlike the previously presented impurity-based criteria.

Fayyad and Irani [35] propose a new binary criterion from this family of C-SEP measures called *ORT*, defined as:

$$\begin{aligned} \Gamma^{ORT}(a_i, d_1, d_2, \mathbf{X}, y) &= 1 - \theta(v_{d_1}, v_{d_2}) \\ \theta(v_{d_1}, v_{d_2}) &= \frac{v_{d_1} \cdot v_{d_2}}{||v_{d_1}|| \times ||v_{d_2}||} \end{aligned} \tag{2.14}$$

where v_{d_i} is the class vector of the set of instances $\mathbf{X_i} = \{x \in \mathbf{X} \mid \mathbf{X_{a_i \in d_i}}\}$, "$\cdot$" represents the inner product between two vectors and $||.||$ the magnitude (norm) of a vector. Note that *ORT* is basically the complement of the well-known *cosine* distance, which measures the orthogonality between two vectors. When the angle between two vectors is 90, it means the non-zero components of each vector do not overlap. The *ORT* criterion is maximum when the cosine distance is minimum, i.e., the vectors are orthogonal, and it is minimum when they are parallel. The higher the values of *ORT*, the greater the distance between components of the class vectors (maximum *ORT* means disjoint classes).

Taylor and Silverman [111] propose a splitting criterion called *mean posterior improvement* (MPI), which is given by:

$$\beta^{MPI}(a_i, d_1, d_2, \mathbf{X}, y) = p_{d_1,\bullet} p_{d_2,\bullet} - \sum_{l=1}^{k} [p_{\bullet,y_l} p_{d_1 \cap y_l} p_{d_2 \cap y_l}] \tag{2.15}$$

The MPI criterion provides maximum value when individuals of the same class are all placed in the same partition, and thus, (2.15) should be maximized. Classes over-represented in the father node will have a greater emphasis in the MPI calculation (such an emphasis is given by the $p_{\bullet,y_l}$ in the summation). The term $p_{d_1 \cap y_l} p_{d_2 \cap y_l}$ is desired to be small since the goal of MPI is to keep instances of the same class together and to separate them from those of other classes. Hence, $p_{d_1,\bullet} p_{d_2,\bullet} - p_{d_1 \cap y_l} p_{d_2 \cap y_l}$ is the improvement that the split is making for class y_l, and therefore the MPI criterion is the mean improvement over all the classes.

Mola and Siciliano [75] propose using the predictability index τ originally proposed in [42] as a splitting measure. The τ index can be used first to evaluate each attribute individually (2.16), and then to evaluate each possible binary split provided by grouping the values of a given attribute in d_1 and d_2 (2.17).

$$\beta^{\tau}(a_i, \mathbf{X}, y) = \frac{\sum_{j=1}^{|a_i|} \sum_{l=1}^{k} (p_{v_j \cap y_l})^2 \times p_{v_j,\bullet} - \sum_{l=1}^{k} p_{\bullet,y_l}^2}{1 - \sum_{l=1}^{k} p_{\bullet,y_l}^2} \tag{2.16}$$

$$\beta^{\tau}(a_i, d_1, d_2, \mathbf{X}, y) = \frac{\sum_{l=1}^{k} (p_{d_1 \cap y_l})^2 p_{d_1,\bullet} + \sum_{l=1}^{k} (p_{d_2 \cap y_l})^2 p_{d_2,\bullet} - \sum_{l=1}^{k} p_{\bullet,y_l}^2}{1 - \sum_{j=1}^{k} p_{y_j}^2} \tag{2.17}$$

Now, consider that $\beta^{\tau *}(a_i) = \max_{d_1,d_2} \beta^{\tau}(a_i, d_1, d_2, \mathbf{X}, y)$. Mola and Siciliano [75] prove a theorem saying that

$$\beta^{\tau}(a_i, \mathbf{X}, y) \geq \beta^{\tau}(a_i, d_1, d_2, \mathbf{X}, y) \tag{2.18}$$

and also that

$$\beta^{\tau}(a_i, \mathbf{X}, y) \geq \beta^{\tau *}(a_i). \tag{2.19}$$

This theoretical evidence is of great importance for providing a means to select the best attribute and its corresponding binary partitions without the need of exhaustively trying all possibilities. More specifically, one has to calculate (2.16) for all attributes, and to sort them according to the highest values of $\beta^{\tau}(*, \mathbf{X}, y)$, in such a way that a_1 is the attribute that yields the highest value of $\beta^{\tau}(*, \mathbf{X}, y)$, a_2 is the second highest value, and so on. Then, one has to test all possible splitting options in (2.17) in order to find $\beta^{\tau *}(a_1)$. If the value of $\beta^{\tau *}(a_1)$ is greater than the value of $\beta^{\tau}(a_2, \mathbf{X}, y)$, we do not need to try any other split possibilities, since we know that $\beta^{\tau *}(a_2)$ is necessarily lesser than $\beta^{\tau}(a_2, \mathbf{X}, y)$. For a simple but efficient algorithm implementing this idea, please refer to the appendix in [75].

2.3.1.3 Other Classification Criteria

In this category, we include all criteria that did not fit in the previously-mentioned categories.

Li and Dubes [62] propose a binary criterion for binary-class problems called *permutation statistic*. It evaluates the degree of similarity between two vectors, V_{a_i} and y, and the larger this statistic, the more alike the vectors. Vector V_{a_i} is calculated as follows. Let a_i be a given numeric attribute with the values $[8.20, 7.3, 9.35, 4.8, 7.65, 4.33]$ and $N_x = 6$. Vector $y = [0, 0, 1, 1, 0, 1]$ holds the corresponding class labels. Now consider a given threshold $\Delta = 5.0$. Vector V_{a_i} is calculated in two steps: first, attribute a_i values are sorted, i.e., $a_i = [4.33, 4.8, 7.3, 7.65, 8.20, 9.35]$, consequently rearranging $y = [1, 1, 0, 0, 0, 1]$; then, $V_{a_i}(n)$ takes 0 when $a_i(n) \leq \Delta$, and 1 otherwise. Thus, $V_{a_i} = [0, 0, 1, 1, 1, 1]$. The permutation statistic first analyses how many $1-1$ matches (d) vectors V_{a_i} and y have. In this particular example, $d = 1$. Next, it counts how many 1's there are in V_{a_i} (n_a) and in y (n_y). Finally, the permutation statistic can be computed as:

$$\beta^{permutation}(V_{a_i}, y) = \sum_{j=0}^{d} \frac{\binom{n_a}{j}\binom{N_x - n_a}{n_y - j}}{\binom{N_x}{n_y}} - \frac{\binom{n_a}{d}\binom{N_x - n_a}{n_y - d}}{\binom{N_x}{n_y}} U$$

$$\binom{n}{m} = 0 \quad \text{if} \quad n < 0 \text{ or } m < 0 \text{ or } n < m$$

$$= \frac{n!}{m!(n-m)!} \text{otherwise} \tag{2.20}$$

where U is a (continuous) random variable distributed uniformly over $[0, 1]$.

The permutation statistic presents an advantage over the information gain and other criteria: it is not sensitive to the data fragmentation problem.[2] It automatically adjusts for variations in the number of instances from node to node because its distribution does not change with the number of instances at each node.

Quinlan and Rivest [96] propose using the *minimum description length principle* (MDL) as a splitting measure for decision-tree induction. MDL states that, given a set of competing hypotheses (in this case, decision trees), one should choose as the preferred hypothesis the one that minimizes the sum of two terms: (i) the description length of the hypothesis (d_l); and (ii) length of the data given the hypothesis (l_h). In the context of decision trees, the second term can be regarded as *the length of the exceptions*, i.e., the length of certain objects of a given subset whose class value is different from the most frequent one. Both terms are measured in bits, and thus one needs to encode the decision tree and exceptions accordingly.

It can be noticed that by maximizing d_l, we minimize l_h, and vice-versa. For instance, when we grow a decision tree until each node has objects that belong to the same class, we usually end up with a large tree (maximum d_l) prone to overfitting, but with no exceptions (minimum l_h). Conversely, if we allow a large number of exceptions, we will not need to partition subsets any further, and in the extreme case (maximum l_h), the decision tree will hold a single leaf node labeled as the most frequent class value (minimum d_l). Hence the need of minimizing the sum $d_l + l_h$.

MDL provides a way of comparing decision trees once the encoding techniques are chosen. Finding a suitable encoding scheme is usually a very difficult task, and the values of d_l and l_h are quite dependent on the encoding technique used [37]. Nevertheless, Quinlan and Rivest [96] propose selecting the attribute that minimizes $d_l + l_h$ at each node, and then pruning back the tree whenever replacing an internal node by a leaf decreases $d_l + l_h$.

A criterion derived from classical statistics is the *multiple hypergeometric distribution* (P_0) [1, 70], which is an extension of Fischer's *exact test* for two binary variables. It can be regarded as the probability of obtaining the observed data given that the null hypothesis (of variable independence) is true. P_0 is given by:

$$P_0(a_i, \mathbf{X}, y) = \left(\frac{\prod_{l=1}^{k} N_{\bullet,y_l}!}{N_x!}\right) \prod_{j=1}^{|a_i|} \left(\frac{N_{v_j,\bullet}!}{\prod_{m=1}^{k} N_{v_j \cap y_m}!}\right) \tag{2.21}$$

The lower the values of P_0, the lower the probability of accepting the null hypothesis. Hence, the attribute that presents the lowest value of P_0 is chosen for splitting the current node in a decision tree.

Chandra and Varghese [21] propose a new splitting criterion for partitioning nodes in decision trees. The proposed measure is designed to reduce the number of distinct classes resulting in each sub-tree after a split. Since the authors do not name

[2] Data fragmentation is a well-known problem in top-down decision trees. Nodes with few instances usually lack statistical support for further partitioning. This phenomenon happens for most of the split criteria available, since their distributions depend on the number of instances in each particular node.

their proposed measure, we call it CV (from Chandra-Varseghe) from now on. It is given by:

$$CV(a_i, \mathbf{X}, y) = \sum_{j=1}^{|a_i|} \left[p_{v_j,\bullet} \times \frac{D_{v_j}}{D_x} \left(\sum_{l=1}^{D_{v_j}} p_{v_j|y_l} \right) \right] \tag{2.22}$$

where D_x counts the number of *distinct* class values among the set of instances in $\mathbf{X}$, and D_{v_j} the number of distinct class values in partition v_j. The CV criterion must be minimized. The authors prove that CV is strictly convex (i.e., it achieves its minimum value at a boundary point) and cumulative (and thus, well-behaved). The authors argue that the experiments, which were performed on 19 data sets from the UCI repository [36], indicate that the proposed measure results in decision trees that are more compact (in terms of tree height), without compromising on accuracy when compared to the gain ratio and Gini index.

Chandra et al. [20] propose the use of a *distinct class based splitting measure* (DCSM). It is given by:

$$DCSM(a_i, \mathbf{X}, y) = \sum_{j=1}^{|a_i|} \left[p_{v_j,\bullet} D_{v_j} \exp(D_{v_j}) \right. \\ \left. \times \sum_{l=1}^{k} \left[p_{y_l|v_j} \exp\left(\frac{D_{v_j}}{D_x} \left(1 - \left(p_{y_l|v_j}\right)^2\right)\right)\right] \right] \tag{2.23}$$

Note that the term $D_{v_j} \exp D_{v_j}$ deals with the number of distinct classes in a given partition v_j. As the number of distinct classes in a given partition increases, this term also increases. It means that purer partitions are preferred, and they are weighted according to the proportion of training instances that lie in the given partition. Also, note that $\frac{D_{v_j}}{D_x}$ decreases when the number of distinct classes decreases while $(1 - \left(p_{y_l|v_j}\right)^2)$ decreases when there are more instances of a class compared to the total number of instances in a partition. These terms also favor partitions with a small number of distinct classes.

It can be noticed that the value of DCSM increases exponentially as the number of distinct classes in the partition increases, invalidating such splits. Chandra et al. [20] argue that "this makes the measure more sensitive to the impurities present in the partition as compared to the existing measures." The authors demonstrate that DCSM satisfies two properties: convexity and well-behavedness. Finally, through empirical data, the authors affirm that DCSM provides more compact and more accurate trees than those provided by measures such as gain ratio and Gini index.

Many other split criteria for classification can be found in the literature, including *relevance* [3], *misclassification error with confidence intervals* [53], *RELIEF split criterion* [58], *QUEST split criterion* [65], just to name a few.

2.3.1.4 Regression Criteria

All criteria presented so far are dedicated to classification problems. For regression problems, where the target variable y is continuous, a common approach is to calculate the *mean squared error* (MSE) as a splitting criterion:

$$MSE(a_i, \mathbf{X}, y) = N_x^{-1} \sum_{j=1}^{|a_i|} \sum_{x_l \in v_j} (y(x_l) - \bar{y_{v_j}})^2 \tag{2.24}$$

where $\bar{y_{v_j}} = N_{v_j,\bullet}^{-1} \sum_{x_l \in v_j} y(x_l)$. Just as with clustering, we are trying to minimize the within-partition variance. Usually, the sum of squared errors is weighted over each partition according to the estimated probability of an instance belonging to the given partition [12]. Thus, we should rewrite MSE to:

$$wMSE(a_i, \mathbf{X}, y) = \sum_{j=1}^{|a_i|} p_{v_j,\bullet} \sum_{x_l \in v_j} (y(x_l) - \bar{y_{v_j}})^2 \tag{2.25}$$

Another common criterion for regression is the *sum of absolute deviations* (SAD) [12], or similarly its weighted version given by:

$$wSAD(a_i, \mathbf{X}, y) = \sum_{j=1}^{|a_i|} p_{v_j,\bullet} \sum_{x_l \in v_j} abs(y(x_l) - median(y_{v_j})) \tag{2.26}$$

where $median(y_{v_j})$ is the target attribute's median of instances belonging to $\mathbf{X_{a_i=v_j}}$.

Quinlan [93] proposes the use of the *standard deviation reduction* (SDR) for his pioneering system of model trees induction, M5. Wang and Witten [124] extend the work of Quinlan in their proposed system M5', also employing the SDR criterion. It is given by:

$$SDR(a_i, \mathbf{X}, y) = \sigma_X - \sum_{j=1}^{|a_i|} p_{v_j,\bullet} \sigma_{v_j} \tag{2.27}$$

where σ_X is the standard deviation of instances in $\mathbf{X}$ and σ_{v_j} the standard deviation of instances in $\mathbf{X_{a_i=v_j}}$. SDR should be maximized, i.e., the weighted sum of standard deviations of each partition should be as small as possible. Thus, partitioning the instance space according to a particular attribute a_i should provide partitions whose target attribute variance is small (once again we are interested in minimizing the within-partition variance). Observe that minimizing the second term in SDR is equivalent to minimizing wMSE, but in SDR we are using the partition standard deviation (σ) as a similarity criterion whereas in wMSE we are using the partition variance (σ^2).

Buja and Lee [15] propose two alternative regression criteria for binary trees: *one-sided purity* (OSP) and *one-sided extremes* (OSE). OSP is defined as:

$$OSP(a_i, d_1, d_2, \mathbf{X}, y) = \min_{d_1,d_2}(\sigma^2_{d_1}, \sigma^2_{d_2}) \tag{2.28}$$

where $\sigma^2_{d_i}$ is the variance of partition d_i. The authors argue that by minimizing this criterion for all possible splits, we find a split whose partition (either d_1 or d_2) presents the smallest variance. Typically, this partition is less likely to be split again. Buja and Lee [15] also propose the OSE criterion:

$$\begin{aligned} OSE(a_i, d_1, d_2, \mathbf{X}, y) &= \min_{d_1,d_2}(\bar{y_{d_1}}, \bar{y_{d_2}}) \\ &\text{or, conservely:} \\ OSE(a_i, d_1, d_2, \mathbf{X}, y) &= \max_{d_1,d_2}(\bar{y_{d_1}}, \bar{y_{d_2}}) \end{aligned} \tag{2.29}$$

The authors argue that whereas the mean values have not been thought of as splitting criteria, "in real data, the dependence of the mean response on the predictor variables is often monotone; hence extreme response values are often found on the periphery of variable ranges (…), the kind of situations to each the OSE criteria would respond".

Alpaydin [2] mentions the use of the *worst possible error* (WPE) as a valid criterion for splitting nodes:

$$WPE(a_i, \mathbf{X}, y) = \max_j \max_l [abs(y(x_l) - \bar{y_{v_j}})] \tag{2.30}$$

Alpaydin [2] states that by using WPE we can guarantee that the error for any instance is never larger than a given threshold Δ. This analysis is useful because the threshold Δ can be seen as a complexity parameter that defines the *fitting level* provided by the tree, given that we use it for deciding when interrupting its growth. Larger values of Δ lead to smaller trees that could underfit the data whereas smaller values of Δ lead to larger trees that risk overfitting. A deeper appreciation of underfitting, overfitting and tree complexity is presented later, when *pruning* is discussed.

Other regression criteria that can be found in the literature are MPI for regression [112], Lee's criteria [61], GUIDE's criterion [66], and SMOTI's criterion [67], just to name a few. Tables 2.1 and 2.2 show all univariate splitting criteria cited in this section, as well as their corresponding references, listed in chronological order.

2.3.1.5 Multivariate Splits

All criteria presented so far are intended for building univariate splits. Decision trees with multivariate splits (known as *oblique*, *linear* or *multivariate* decision trees) are not so popular as the univariate ones, mainly because they are harder to interpret. Nevertheless, researchers reckon that multivariate splits can improve the performance

Table 2.1 Univariate splitting criteria for classification

Category	Criterion	References
Info theory	GMI	[22, 41, 102, 108]
	Information gain	[18, 44, 92, 122]
	G statistic	[72, 73]
	Gain ratio	[89, 92]
	CAIR	[22]
	Normalized gain	[52]
	Average gain	[123]
	MEE	[100]
Distance-based	KS distance	[38, 99]
	Gini index	[12]
	Twoing	[12]
	χ^2	[72, 125, 130]
	Distance	[26]
	Multi-class KS	[45, 120]
	ORT	[35]
	MPI	[111]
	τ Index	[75]
Other	Permutation	[62]
	Relevance	[3]
	MDL criterion	[96]
	Mis. Error with CI	[53]
	RELIEF	[58]
	QUEST criterion	[65]
	P_0	[1, 70]
	CV	[21]
	DCSM	[20]

Table 2.2 Univariate splitting criteria for regression

Criterion	References
(w)MSE	[12]
(w)SAD	[12]
SDR	[93, 124]
MPI-R	[112]
OSP	[15]
OSE	[15]
Lee’s	[61]
GUIDE’s	[66]
SMOTI’s	[67]
WPE	[2]

of the tree in several data sets, while generating smaller trees [47, 77, 98]. Clearly, there is a tradeoff to consider in allowing multivariate tests: simple tests may result in large trees that are hard to understand, yet multivariate tests may result in small trees with tests hard to understand [121].

A decision tree with multivariate splits is able to produce polygonal (polyhedral) partitions of the attribute space (hyperplanes at an oblique orientation to the attribute axes) whereas univariate trees can only produce hyper-rectangles parallel to the attribute axes. The tests at each node have the form:

$$w_0 + \sum_{i=1}^{n} w_i a_i(x) \leq 0 \tag{2.31}$$

where w_i is a real-valued coefficient associated to the ith attribute and w_0 the disturbance coefficient of the test.

CART (Classification and Regression Trees) [12] is one of the first systems that allowed multivariate splits. It employs a hill-climbing strategy with a backward attribute elimination for finding good (albeit suboptimal) linear combinations of attributes in non-terminal nodes. It is a fully-deterministic algorithm with no built-in mechanisms to escape local-optima. Breiman et al. [12] point out that the proposed algorithm has much room for improvement.

Another approach for building oblique decision trees is LMDT (Linear Machine Decision Trees) [14, 119], which is an evolution of the perceptron tree method [117]. Each non-terminal node holds a linear machine [83], which is a set of k linear discriminant functions that are used collectively to assign an instance to one of the k existing classes. LMDT uses heuristics to determine when a linear machine has stabilized (since convergence cannot be guaranteed). More specifically, for handling non-linearly separable problems, a method similar to simulated annealing (SA) is used (called *thermal training*). Draper and Brodley [30] show how LMDT can be altered to induce decision trees that minimize arbitrary misclassification cost functions.

SADT (Simulated Annealing of Decision Trees) [47] is a system that employs SA for finding good coefficient values for attributes in non-terminal nodes of decision trees. First, it places a hyperplane in a canonical location, and then iteratively perturbs the coefficients in small random amounts. At the beginning, when the temperature parameter of the SA is high, practically any perturbation of the coefficients is accepted regardless of the goodness-of-split value (the value of the utilised splitting criterion). As the SA cools down, only perturbations that improve the goodness-of-split are likely to be allowed. Although SADT can eventually escape from local-optima, its efficiency is compromised since it may consider tens of thousands of hyperplanes in a single node during annealing.

OC1 (Oblique Classifier 1) [77, 80] is yet another oblique decision tree system. It is a thorough extension of CART's oblique decision tree strategy. OC1 presents the advantage of being more efficient than the previously described systems. For instance, in the worst case scenario, OC1's running time is $O(\log_n)$ times greater than the worst case scenario of univariate decision trees, i.e., $O(nN^2 \log_N)$ *versus* $O(nN^2)$. OC1 searches for the best univariate split as well as the best oblique split, and it only employs the oblique split when it improves over the univariate split.[3] It uses both a deterministic heuristic search (as employed in CART) for finding local-optima and a non-deterministic search (as employed in SADT—though not SA) for escaping local-optima.

During the deterministic search, OC1 perturbs the hyperplane coefficients sequentially (much in the same way CART does) until no significant gain is achieved according to an impurity measure. More specifically, consider hyperplane $H = w_0 + \sum_{i=1}^{n} w_i a_i(x) = 0$, and that we substitute an instance x_j in H, i.e., $H = w_0 + \sum_{i=1}^{n} w_i a_i(x_j) = Z_j$. The sign of Z_j indicates whether an instance x_j is above or below the hyperplane H. If H splits $\mathbf{X}$ perfectly, then all instances belonging to the same class will have the same sign of Z. For finding the local-optimal set of coefficients, OC1 employs a sequential procedure that works as follows: treat coefficient w_i as a variable and all other coefficients as constants. The condition that instance x_j is above hyperplane H can be written as:

$$\begin{gathered} Z_j > 0 \\ w_i > \left[\frac{w_i a_i(x_j) - Z_j}{a_i(x_j)} \equiv U_j \right] \end{gathered} \tag{2.32}$$

assuming $a_i(x_j) > 0$, which is ensured through normalization. With the definition in (2.32), an instance is above the hyperplane if $w_i > U_j$ and below otherwise. By plugging each instance $x \in \mathbf{X}$ in (2.32), we obtain N_x constraints on the value of w_i. Hence, the problem is reduced on finding the value of w_i that satisfies the greatest possible number of constraints. This problem is easy to solve optimally: simply sort all the values U_j, and consider setting w_i to the midpoint between each pair of different class values. For each distinct placement of the coefficient w_i, OC1 computes the impurity of the resulting split, and replaces original coefficient w_i by the recently discovered value if there is reduction on impurity. The pseudocode of this deterministic perturbation method is presented in Algorithm 2.

The parameter P_{stag} (stagnation probability) is the probability that a hyperplane is perturbed to a location that does not change the impurity measure. To prevent the stagnation of impurity, P_{stag} decreases by a constant amount each time OC1 makes a "stagnant" perturbation, which means only a constant number of such perturbations will occur at each node. P_{stag} is reset to 1 every time the global impurity measure is improved. It is a user-defined parameter.

After a local-optimal hyperplane H is found, it is further perturbed by a randomized vector, as follows: it computes the optimal amount by which H should be perturbed along the random direction dictated by a random vector. To be more precise, when a hyperplane $H = w_0 + \sum_{i=1}^{n} w_i a_i(x)$ cannot be improved by

[3] OC1 only allows the option of employing oblique splits when $N > 2n$, though this threshold can be user-defined.

Algorithm 2 Deterministic OC1's procedure for perturbing a given coefficient. Parameters are the current hyperplane H and the coefficient index i.

```
1: procedure PERTURB($H, i$)
2:     for $j = 1$ to $N_x$ do
3:         Compute $U_j$ (32)
4:     end for
5:     Sort $U_1..U_{N_x}$ in non-decreasing order
6:     $w'_i$ = best split of the sorted $U_j$s
7:     $H_1$ = resulting of replacing $w_i$ by $w'_i$ in $H$
8:     if ($impurity(H_1) < impurity(H)$) then
9:         $w_i = w'_i$
10:        $P_{move} = P_{stag}$
11:    else if ($impurity(H_1) = impurity(H)$) then
12:        $w_i = w'_i$ with probability $P_{move}$
13:        $P_{move} = P_{move} - 0.1 P_{stag}$
14:    end if return $w_i$
15: end procedure
```

deterministic perturbation (Algorithm 2), OC1 repeats the following loop J times (where J is a user-specified parameter, set to 5 by default):

- Choose a random vector $R = [r_0, r_1, \ldots, r_n]$;
- Let α be the amount by which we want to perturb H in the direction R. More specifically, let $H_1 = (w_0 + \alpha r_0) + \sum_{i=1}^{n}(w_i + \alpha r_i)a_i(x)$;
- Find the optimal value for α;
- If the hyperplane H_1 decreases the overall impurity, replace H with $H1$, exit this loop and begin the deterministic perturbation algorithm for the individual coefficients.

Note that we can treat α as the only variable in the equation for H_1. Therefore each of the N examples, if plugged into the equation for H_1, imposes a constraint on the value of α. OC1 can use its own deterministic coefficient perturbation method (Algorithm 2) to compute the best value of α. If J *random jumps* fail to improve the impurity measure, OC1 halts and uses H as the split for the current tree node. Regarding the impurity measure, OC1 allows the user to choose among a set of splitting criteria, such as information gain, Gini index, twoing criterion, among others.

Ittner [51] proposes using OC1 over an augmented attribute space, generating *non-linear decision trees*. The key idea involved is to "build" new attributes by considering all possible pairwise products and squares of the original set of n attributes. As a result, a new attribute space with $(n^2 + 3n)/2$ is formed, i.e., the sum of n original attributes, n squared ones and $(n(n-1))/2$ pairwise products of the original attributes. To illustrate, consider a binary attribute space $\{a_1, a_2\}$. The augmented attribute space would contain 5 attributes, i.e., $b_1 = a_1, b_2 = a_2, b_3 = a_1a_2, b_4 = a_1^2, b_5 = a_2^2$.

A similar approach of transforming the original attributes is taken in [64], in which the authors propose the BMDT system. In BMDT, a 2-layer feedforward neural network is employed to transform the original attribute space in a space in which the new attributes are linear combinations of the original ones. This transformation is performed through a hyperbolic tangent function at the hidden units. After transforming the attributes, a univariate decision-tree induction algorithm is employed over this new attribute space. Finally, a procedure replaces the transformed attributes by the original ones, which means that the univariate tests in the recently built decision tree become multivariate tests, and thus the univariate tree becomes an oblique tree.

Shah and Sastry [103] propose the APDT (Alopex Perceptron Decision Tree) system. It is an oblique decision tree inducer that makes use of a new splitting criterion, based on the level of non-separability of the input instances. They argue that because oblique decision trees can realize arbitrary piecewise linear separating surfaces, it seems better to base the evaluation function on the degree of separability of the partitions rather than on the degree of purity of them. APDT runs the Perceptron algorithm for estimating the number of non-separable instances belonging to each one of the binary partitions provided by an initial hyperplane. Then, a correlation-based optimization algorithm called Unnikrishnan et al. [116] is employed for tuning the hyperplane weights taking into account the need of minimizing the new split criterion based on the degree of separability of partitions. Shah and Sastry [103] also propose a pruning algorithm based on genetic algorithms.

Several other oblique decision-tree systems were proposed employing different strategies for defining the weights of hyperplanes and for evaluating the generated split. Some examples include: the system proposed by Bobrowski and Kretowski [11], which employs heuristic sequential search (combination of sequential backward elimination and sequential forward selection) for defining hyperplanes and a dipolar criterion for evaluating splits; the omnivariate decision tree inducer proposed by Yildiz and Alpaydin [128], where the non-terminal nodes may be univariate, linear, or nonlinear depending on the outcome of comparative statistical tests on accuracy, allowing the split to match automatically the complexity of the node according to the subproblem defined by the data reaching that node; Li et al. [63] propose using tabu search and a variation of linear discriminant analysis for generating multivariate splits, arguing that their algorithm runs faster than most oblique tree inducers, since its computing time increases linearly with the number of instances; Tan and Dowe [109] proposes inducing oblique trees through a MDL-based splitting criterion and the evolution strategy as a meta-heuristic to search for the optimal hyperplane within a node. For regression oblique trees please refer to [27, 48, 60].

For the interested reader, it is worth mentioning that there are methods that induce oblique decision trees with optimal hyperplanes, discovered through *linear programming* [9, 10, 68]. Though these methods can find the optimal hyperplanes for specific splitting measures, the size of the linear program grows very fast with the number of instances and attributes.

For a discussion on several papers that employ evolutionary algorithms for induction of oblique decision trees (to evolve either the hyperplanes or the whole tree), the reader is referred to [**Barros2012**]. Table 2.3 presents a summary of some systems proposed for induction of oblique decision trees.

Table 2.3 Multivariate splits

System	Criterion	Hyperplane strategy
CART [12]	Gini index/twoing	Hill-climbing with SBE
LMDT [14]	Misclassification error	Linear machine with thermal training
SADT [47]	Sum-minority	Simulated annealing
OC1 [77, 80]	Info gain, gini index, twoing, etc	Hill-climbing with randomization
BMDT [64]	Gain ratio	2-Layer feedforward neural network
APDT [103]	Separability[a]	Alopex
Bobrowski and Kretowski [11][b]	Dipolar criterion	Heuristic sequential search
Omni [128]	Misclassification error	MLP neural network
LDTS [63]	Info gain	LDA with tabu search
MML [63]	MDL-based	Evolution strategy
Geometric DT [69]	Gini-index	Multisurface proximal SVM

[a]The authors of the criterion do not explicitly name it, so we call it "Separability", since the criterion is based on the degree of linear separability
[b]The authors call their system "our method", so we do not explicitly name it

2.3.2 Stopping Criteria

The top-down induction of a decision tree is recursive and it continues until a stopping criterion (or some stopping criteria) is satisfied. Some popular stopping criteria are [32, 98]:

1. Reaching class homogeneity: when all instances that reach a given node belong to the same class, there is no reason to split this node any further;
2. Reaching attribute homogeneity: when all instances that reach a given node have the same attribute values (though not necessarily the same class value);
3. Reaching the maximum tree depth: a parameter *tree depth* can be specified to avoid deep trees;
4. Reaching the minimum number of instances for a non-terminal node: a parameter *minimum number of instances for a non-terminal node* can be specified to avoid (or at least alleviate) the data fragmentation problem;
5. Failing to exceed a threshold when calculating the splitting criterion: a parameter *splitting criterion threshold* can be specified for avoiding *weak* splits.

Criterion 1 is universally accepted and it is implemented in most top-down decision-tree induction algorithms to date. Criterion 2 deals with the case of contradictory instances, i.e., identical instances regarding A, but with different class values. Criterion 3 is usually a constraint regarding tree complexity, specially for those cases in which comprehensibility is an important requirement, though it may affect complex classification problems which require deeper trees. Criterion 4 implies that small disjuncts (i.e., tree leaves covering a small number of objects) can be ignored since they are error-prone. Note that eliminating small disjuncts can be harmful to exceptions—particularly in a scenario of imbalanced classes. Criterion 5 is heavily dependent on the splitting measure used. An example presented in [32] clearly indicates a scenario in which using criterion 5 prevents the growth of a 100 % accurate decision tree (a problem usually referred to as *the horizon effect* [12, 89]).

The five criteria presented above can be seen as *pre-pruning* strategies, since they "prematurely" interrupt the growth of the tree. Note that most of the criteria discussed here may harm the growth of an accurate decision tree. Indeed there is practically a consensus in the literature that decision trees should be overgrown instead. For that, the stopping criterion used should be as *loose* as possible (e.g., until a single instance is contemplated by the node or until criterion 1 is satisfied). Then, a *post-pruning* technique should be employed in order to prevent *data overfitting*—a phenomenon that happens when the classifier *over-learns* the data, that is, when it learns all data peculiarities—including potential noise and spurious patterns—that are specific to the training set and do not generalise well to the test set. Post-pruning techniques are covered in the next section.

2.3.3 Pruning

This section reviews strategies of pruning, normally referred to as *post-pruning* techniques. Pruning is usually performed in decision trees for enhancing tree comprehensibility (by reducing its size) while maintaining (or even improving) accuracy. It was originally conceived as a strategy for tolerating noisy data, though it was found that it could improve decision tree accuracy in many noisy data sets [12, 92, 94].

A pruning method receives as input an unpruned tree T and outputs a decision tree T' formed by removing one or more subtrees from T. It replaces non-terminal nodes by leaf nodes according to a given heuristic. Next, we present the six most well-known pruning methods for decision trees [13, 32]: (1) reduced-error pruning; (2) pessimistic error pruning; (3) minimum error pruning; (4) critical-value pruning; (5) cost-complexity pruning; and (6) error-based pruning.

2.3.3.1 Reduced-Error Pruning

Reduced-error pruning is a conceptually simple strategy proposed by Quinlan [94]. It uses a pruning set (a part of the training set) to evaluate the goodness of a given subtree from T. The idea is to evaluate each non-terminal node $t \in \zeta_T$ with regard to the classification error in the pruning set. If such an error decreases when we replace the subtree $T^{(t)}$ by a leaf node, than $T^{(t)}$ must be pruned.

Quinlan imposes a constraint: a node t cannot be pruned if it contains a subtree that yields a lower classification error in the pruning set. The practical consequence of this constraint is that REP should be performed in a bottom-up fashion. The REP pruned tree T' presents an interesting optimality property: it is the smallest most accurate tree resulting from pruning original tree T [94]. Besides this optimality property, another advantage of REP is its linear complexity, since each node is visited only once in T. An obvious disadvantage is the need of using a pruning set, which means one has to divide the original training set, resulting in less instances to grow the tree. This disadvantage is particularly serious for small data sets.

2.3.3.2 Pessimistic Error Pruning

Also proposed by Quinlan [94], the pessimistic error pruning uses the training set for both growing and pruning the tree. The apparent error rate, i.e., the error rate calculated over the training set, is optimistically biased and cannot be used to decide whether pruning should be performed or not. Quinlan thus proposes adjusting the apparent error according to the continuity correction for the binomial distribution (cc) in order to provide a more realistic error rate. Consider the apparent error of a pruned node t, and the error of its entire subtree $T^{(t)}$ before pruning is performed, respectively:

$$r^{(t)} = \frac{E^{(t)}}{N_x^{(t)}} \tag{2.33}$$

$$r^{T^{(t)}} = \frac{\sum_{s \in \lambda_{T^{(t)}}} E^{(s)}}{\sum_{s \in \lambda_{T^{(t)}}} N_x^{(s)}}. \tag{2.34}$$

Modifying (2.33) and (2.34) according to cc results in:

$$r_{cc}^{(t)} = \frac{E^{(t)} + 1/2}{N_x^{(t)}} \tag{2.35}$$

$$r_{cc}^{T^{(t)}} = \frac{\sum_{s \in \lambda_{T^{(t)}}} E^{(s)} + 1/2}{\sum_{s \in \lambda_{T^{(t)}}} N_x^{(s)}} = \frac{\frac{|\lambda_{T^{(t)}}|}{2} \sum_{s \in \lambda_{T^{(t)}}} E^{(s)}}{\sum_{s \in \lambda_{T^{(t)}}} N_x^{(s)}}. \tag{2.36}$$

For the sake of simplicity, we will refer to the adjusted number of errors rather than the adjusted error rate, i.e., $E_{cc}^{(t)} = E^{(t)} + 1/2$ and $E_{cc}^{T^{(t)}} = (|\lambda_{T^{(t)}}|/2) \sum_{s \in \lambda_{T^{(t)}}} E^{(s)}$. Ideally, pruning should occur if $E_{cc}^{(t)} \leq E_{cc}^{T^{(t)}}$, but note that this condition seldom holds, since the decision tree is usually grown up to the homogeneity stopping criterion (criterion 1 in Sect. 2.3.2), and thus $E_{cc}^{T^{(t)}} = |\lambda_{T^{(t)}}|/2$ whereas $E_{cc}^{(t)}$ will very probably be a higher value. In fact, due to the homogeneity stopping criterion, $E_{cc}^{T^{(t)}}$ becomes simply a measure of complexity which associates each leaf node with a cost of 1/2. Quinlan, aware of this situation, weakens the original condition

$$E_{cc}^{(t)} \leq E_{cc}^{T^{(t)}} \tag{2.37}$$

to

$$E_{cc}^{(t)} \leq E_{cc}^{T^{(t)}} + SE(E_{cc}^{T^{(t)}}) \tag{2.38}$$

where

$$SE(E_{cc}^{T^{(t)}}) = \sqrt{\frac{E_{cc}^{T^{(t)}} * (N_x^{(t)} - E_{cc}^{T^{(t)}})}{N_x^{(t)}}}$$

is the standard error for the subtree $T^{(t)}$, computed as if the distribution of errors were binomial.

PEP is computed in a top-down fashion, and if a given node t is pruned, its descendants are not examined, which makes this pruning strategy quite efficient in terms of computational effort. As a point of criticism, Esposito et al. [32] point out that the introduction of the continuity correction in the estimation of the error rate has no theoretical justification, since it was never applied to correct over-optimistic estimates of error rates in statistics.

2.3.3.3 Minimum Error Pruning

Originally proposed by Niblett and Bratko [82] and further extended by Cestnik and Bartko [19], minimum error pruning is a bottom-up approach that seeks to minimize the *expected error rate* for unseen cases. It estimates the expected error rate in node t ($EE^{(t)}$) as follows:

$$EE^{(t)} = \min_{y_l} \left[\frac{N_x^{(t)} - N_{\bullet,y_l}^{(t)} + (1 - p_{\bullet,y_l}^{(t)}) \times m}{N_x^{(t)} + m} \right]. \tag{2.39}$$

where m is a parameter that determines the importance of the a priori probability on the estimation of the error. Eq. (2.39), presented in [19], is a generalisation of the expected error rate presented in [82] if we assume that $m = k$ and that $p_{\bullet,y_l}^{(t)} = 1/k, \forall y_l \in Y$.

MEP is performed by comparing $EE^{(t)}$ with the weighted sum of the expected error rate of all children nodes from t. Each weight is given by $p_{v_j,\bullet}$, assuming v_j is the partition corresponding to the jth child of t. A disadvantage of MEP is the need of setting the ad-hoc parameter m. Usually, the higher the value of m, the more severe the pruning. Cestnik and Bratko [19] suggest that a domain expert should set m according to the level of noise in the data. Alternatively, a set of trees pruned with different values of m could be offered to the domain expert, so he/she can choose the best one according to his/her experience.

2.3.3.4 Critical-Value Pruning

The critical-value pruning, proposed by Mingers [73], is quite similar to the pre-pruning strategy *criterion 5* in Sect. 2.3.2. It is a bottom-up procedure that prunes a given non-terminal node t in case the value of its splitting measure is below a pre-determined threshold cv. Mingers [73] proposes the following two-step procedure for performing CVP:

1. Prune T for increasing values of cv, generating a set of pruned trees;
2. Choose the best tree among the set of trees (that includes T) by measuring each tree's accuracy (based on a pruning set) and significance (through the previously presented G statistic).

The disadvantage of CVP is the same of REP—the need of a pruning set. In addition, CVP does not present the optimality property that REP does, so there is no guarantee that the best tree found in step 2 is the smallest optimally pruned subtree of T, since the pruning step was performed based on the training set.

2.3.3.5 Cost-Complexity Pruning

Cost-complexity pruning is the post-pruning strategy of the CART system, detailed in [12]. It consists of two steps:

1. Generate a sequence of increasingly smaller trees, beginning with T and ending with the root node of T, by successively pruning the subtree yielding the lowest *cost complexity*, in a bottom-up fashion;
2. Choose the best tree among the sequence based on its relative size and accuracy (either on a pruning set, or provided by a cross-validation procedure in the training set).

The idea within step 1 is that pruned tree T_{i+1} is obtained by pruning the subtrees that show the lowest increase in the apparent error (error in the training set) per pruned leaf. Since the apparent error of pruned node t increases by the amount $r^{(t)} - r^{T^{(t)}}$, whereas its number of leaves decreases by $|\lambda_{T^{(t)}}| - 1$ units, the following ratio measures the increase in apparent error rate per pruned leaf:

$$\alpha = \frac{r^{(t)} - r^{T^{(t)}}}{|\lambda_{T^{(t)}}| - 1} \tag{2.40}$$

Therefore, T_{i+1} is obtained by pruning all nodes in T_i with the lowest value of α. T_0 is obtained by pruning all nodes in T whose α value is 0. It is possible to show that each tree T_i is associated to a distinct value α_i, such that $\alpha_i < \alpha_{i+1}$. Building the sequence of trees in step 1 takes quadratic time with respect to the number of internal nodes.

Regarding step 2, CCP chooses the smallest tree whose error (either on the pruning set or on cross-validation) is not more than one standard error (SE) greater than the

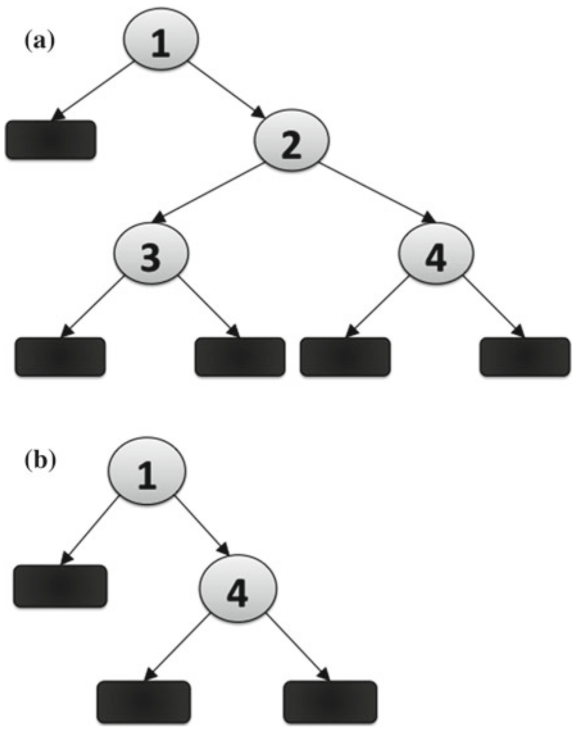

Fig. 2.2 Grafting of subtree rooted in 4 onto the place of parent 2. In **a** the original tree T and in **b** the pruned tree T'

lowest error observed in the sequence of trees. This strategy is known as "1-SE" variant since the work of Esposito et al. [33], which proposes ignoring the standard error constraint, calling the strategy of selecting trees based only on accuracy of "0-SE". It is argued that 1-SE has a tendency of overpruning trees, since its selection is based on a conservative constraint [32, 33].

2.3.3.6 Error-Based Pruning

This strategy was proposed by Quinlan and it is implemented as the default pruning strategy of C4.5 [89]. It is an improvement over PEP, based on a far more pessimistic estimate of the expected error. Unlike PEP, EBP performs a bottom-up search, and it performs not only the replacement of non-terminal nodes by leaves but also the *grafting*[4] of subtree $T^{(t)}$ onto the place of parent t. Grafting is exemplified in Fig. 2.2.

Since grafting is potentially a time-consuming task, only the child subtree $T^{(t')}$ of t with the greatest number of instances is considered to be grafted onto the place of t.

[4] *Grafting* is a term introduced by Esposito et al. [34]. It is also known as *subtree raising* [127].

For deciding whether to replace a non-terminal node by a leaf (subtree replacement), to graft a subtree onto the place of its parent (subtree raising) or not to prune at all, a pessimistic estimate of the expected error is calculated by using an upper confidence bound. Assuming that errors in the training set are binomially distributed with a given probability p in $N_x^{(t)}$ trials, it is possible to compute the exact value of the upper confidence bound as the value of p for which a binomially distributed random variable P shows $E^{(t)}$ successes in $N_x^{(t)}$ trials with probability *CF*. In other words, given a particular confidence *CF* (C4.5 default value is $CF = 25\,\%$), we can find the upper bound of the expected error (EE_{UB}) as follows:

$$EE_{UB} = \frac{f + \frac{z^2}{2N_x} + z\sqrt{\frac{f}{N_x} - \frac{f^2}{N_x} + \frac{z^2}{4N_x^2}}}{1 + \frac{z^2}{N_x}} \tag{2.41}$$

where $f = E^{(t)}/N_x$ and z is the number of standard deviations corresponding to the confidence *CF* (e.g., for $CF = 25\,\%$, $z = 0.69$).

In order to calculate the expected error of node t ($EE^{(t)}$), one must simply compute $N_x^{(t)} \times EE_{UB}$. For evaluating a subtree $T^{(t)}$, one must sum the expected error of every leaf of that subtree, i.e., $\sum_{s \in \lambda_{T^{(t)}}} EE^{(s)}$. Hence, given a non-terminal node t, it is possible to decide whether one should perform subtree replacement (when condition $EE^{(t)} \leq EE^{T^{(t)}}$ holds), subtree raising (when conditions $\exists j \in \zeta_t,\ EE^{(j)} < EE^{(t)} \wedge \forall i \in \zeta_t,\ N_x^{(i)} < N_x^{(j)}$ hold), or not to prune t otherwise.

An advantage of EBP is the new *grafting* operation that allows pruning useless branches without ignoring interesting lower branches (an elegant solution to the horizon effect problem). A drawback of the method is the parameter *CF*, even though it represents a confidence level. Smaller values of *CF* result in more pruning.

2.3.3.7 Empirical Evaluations

Some studies in the literature performed empirical analyses for evaluating pruning strategies. For instance, Quinlan [94] compared four methods of tree pruning (three of them presented in the previous sections—REP, PEP and CCP 1-SE). He argued that those methods in which a pruning set is needed (REP and CCP) did not perform noticeably better than the other methods, and thus their requirement for additional data is a weakness.

Mingers [71] compared five pruning methods, all of them presented in the previous sections (CCP, CVP, MEP, REP and PEP), and related them to different splitting measures. He states that pruning can improve the accuracy of induced decision trees by up to 25 % in domains with noise and residual variation. In addition, he highlights the following findings: (i) MEP (the original version by Niblett and Bratko [82]) is the least accurate method due to its sensitivity to the number of classes in the data; (ii) PEP is the most "crude" strategy, though the fastest one—due to some bad results, it should be used with caution; (iii) CVP, CCP and REP performed well, providing consistently low error-rates for all data sets used; and (iv) there is no evidence of an interaction between the splitting measure and the pruning method used for inducing a decision tree.

Buntine [16], in his PhD thesis, also reports experiments on pruning methods (PEP, MEP, CCP 0-SE and 1-SE for both pruning set and cross-validation). Some of his findings were: (i) CCP 0-SE versions were marginally superior than the 1-SE versions; (ii) CCP 1-SE versions were superior in data sets with little apparent structure, where more severe pruning was inherently better; (iii) CCP 0-SE with cross-validation was marginally better than the other methods, though not in all data sets; and (iv) PEP performed reasonably well in all data sets, and was significantly superior in well-structured data sets (mushroom, glass and LED, all from UCI [36]);

Esposito et al. [32] compare the six post-pruning methods presented in the previous sections within an extended C4.5 system. Their findings were the following: (i) MEP, CVP, and EBP tend to underprune, whereas 1-SE (both cross-validation and pruning set versions) and REP have a propensity for overpruning; (ii) using a pruning-set is not usually a good option; (iii) PEP and EBP behave similarly, despite the difference in their formulation; (iv) pruning does not generally decrease the accuracy of a decision tree (only one of the domains tested was deemed as "pruning-averse"); and (v) data sets not prone to pruning are usually the ones with the highest base error whereas data sets with a low base error tend to benefit of any pruning strategy.

For a comprehensive survey of strategies for simplifying decision trees, please refer to [13]. For more details on post-pruning techniques in decision trees for regression, we recommend [12, 54, 85, 97, 113–115].

2.3.4 Missing Values

Handling missing values (denoted in this section by "?") is an important task not only in machine learning itself, but also in decision-tree induction. Missing values can be an issue during tree induction and also during classification. During tree induction, there are two moments in which we need to deal with missing values: splitting criterion evaluation and instances splitting.

During the split criterion evaluation in node t based on attribute a_i, some common strategies are:

- Ignore all instances belonging to the set $M = \{x_j | a_i(x_j) = ?\}$ [12, 38];
- Imputation of missing values with the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t [24];
- Weight the splitting criterion value (calculated in node t with regard to a_i) by the proportion of missing values, i.e., $|M|/N_x^{(t)}$ [95].
- Imputation of missing values with the mode (nominal attributes) or the mean/median (numeric attributes) of all instances in t whose class attribute is the same of the instance whose a_i value is being imputed [65].

For deciding which child node training instance x_j should go to, considering a split in node t over a_i, and that $a_i(x_j) = ?$, some possibilities are:

- Ignore instance x_j [92];
- Treat instance x_j as if it has the most common value of a_i (mode or mean/median) [95];
- Weight instance x_j by the proportion of cases with known value in a given partition, i.e., $N_x^{(v_l)}/(N_x^{(t)} - |M|)$ (assuming t is the parent node and v_l is its lth partition) [57];
- Assign instance x_j to all partitions [38];
- Build an exclusive partition for missing values [95].
- Assign instance x_j to the partition with the greatest number of instances that belong to the same class that x_j. Formally, if x_j is labeled as y_l, we assign x_j to $\arg\max_{v_m}[N_{v_m,y_l}]$ [65].
- Create a *surrogate split* for each split in the original tree based on a different attribute [12]. For instance, a split over attribute a_i will have a surrogate split over attribute a_j, given that a_j is the attribute which most resembles the original split. Resemblance between two attributes in a binary tree is given by:

$$res(a_i, a_j, \mathbf{X}) = \frac{|\mathbf{X}_{\mathbf{a_i \in d_1(a_i) \wedge a_j \in d_1(a_j)}}|}{N_x} + \frac{|\mathbf{X}_{\mathbf{a_i \in d_2(a_i) \wedge a_j \in d_2(a_j)}}|}{N_x} \tag{2.42}$$

  where the original split over attribute a_i is divided in two partitions, $d_1(a_i)$ and $d_2(a_i)$, and the alternative split over a_j is divided in $d_1(a_j)$ and $d_2(a_j)$. Hence, for creating a surrogate split, one must find attribute a_j that, after divided by two partitions $d_1(a_j)$ and $d_2(a_j)$, maximizes $res(a_i, a_j, \mathbf{X})$.

Finally, for classifying an unseen test instance x_j, considering a split in node t over a_i, and that $a_i(x_j) = ?$, some alternatives are:

- Explore all branches of t combining the results. More specifically, navigate through all ambiguous branches of the tree until reaching different leaves and choose class k with the highest probability, i.e., $\arg\max_y[\sum_{s\in\lambda_t}[N_{\bullet,y_l}^{(s)}]/N_x^{(t)}]$ [90];
- Treat instance x_j as if it has the most common value of a_i (mode or mean/median);
- Halt the classification process and assign instance x_j to the majority class of node t [95].

2.4 Other Induction Strategies

We presented a thorough review of the greedy top-down strategy for induction of decision trees in the previous section. In this section, we briefly present alternative strategies for inducing decision trees.

Bottom-up induction of decision trees was first mentioned in [59]. The authors propose a strategy that resembles agglomerative hierarchical clustering. The algorithm starts with each leaf having objects of the same class. In that way, a k-class problem will generate a decision tree with k leaves. The key idea is to merge, recursively, the two most similar classes in a non-terminal node. Then, a hyperplane is associated to the new non-terminal node, much in the same way as in top-down induction of oblique trees (in [59], a linear discriminant analysis procedure generates the hyperplanes). Next, all objects in the new non-terminal node are considered to be members of the same class (an artificial class that embodies the two *clustered* classes), and the procedure evaluates once again which are the two most similar classes. By recursively repeating this strategy, we end up with a decision tree in which the more obvious discriminations are done first, and the more subtle distinctions are postponed to lower levels. Landeweerd et al. [59] propose using the *Mahalanobis distance* to evaluate similarity among classes:

$$dist_M(i, j)^2 = (\mu_{\mathbf{y_i}} - \mu_{\mathbf{y_j}})^T \Sigma^{-1} (\mu_{\mathbf{y_i}} - \mu_{\mathbf{y_j}}) \tag{2.43}$$

where $\mu_{\mathbf{y_i}}$ is the mean attribute vector of class y_i and Σ is the covariance matrix pooled over all classes.

Some obvious drawbacks of this strategy of bottom-up induction are: (i) binary-class problems provide a 1-level decision tree (root node and two children); such a simple tree cannot model complex problems; (ii) instances from the same class may be located in very distinct regions of the attribute space, harming the initial assumption that instances from the same class should be located in the same leaf node; (iii) hierarchical clustering and hyperplane generation are costly operations; in fact, a procedure for inverting the covariance matrix in the Mahalanobis distance is usually of time complexity proportional to $O(n^3)$.[5] We believe these issues are among the main reasons why bottom-up induction has not become as popular as top-down induction. For alleviating these problems, Barros et al. [4] propose a bottom-up induction algorithm named BUTIA that combines EM clustering with SVM classifiers. The authors later generalize BUTIA to a framework for generating oblique decision trees, namely BUTIF [5], which allows the application of different clustering and classification strategies.

Hybrid induction was investigated in [56]. The ideia is to combine both bottom-up and top-down approaches for building the final decision tree. The algorithm starts by executing the bottom-up approach as described above until two subgroups are achieved. Then, two centers (mean attribute vectors) and covariance information are extracted from these subgroups and used for dividing the training data in a top-down fashion according to a normalized sum-of-squared-error criterion. If the two new partitions induced account for separated classes, then the hybrid induction is finished; otherwise, for each subgroup that does not account for a class, recursively executes the hybrid induction by once again starting with the bottom-up procedure. Kim and Landgrebe [56] argue that in hybrid induction "It is more likely to converge to classes of informational value, because the clustering initialization provides early

[5] For inverting a matrix, the Gauss-Jordan procedure takes time proportional to $O(n^3)$. The fastest algorithm for inverting matrices to date is $O(n^{2.376})$ (the Coppersmith-Winograd algorithm).

guidance in that direction, while the straightforward top-down approach does not guarantee such convergence".

Several studies attempted on avoiding the greedy strategy usually employed for inducing trees. For instance, *lookahead* was employed for trying to improve greedy induction [17, 23, 29, 79, 84]. Murthy and Salzberg [79] show that one-level lookahead does not help building significantly better trees and can actually worsen the quality of trees induced. A more recent strategy for avoiding greedy decision-tree induction is to generate decision trees through *evolutionary algorithms*. The idea involved is to consider each decision tree as an individual in a population, which is evolved through a certain number of generations. Decision trees are modified by genetic operators, which are performed stochastically. A thorough review of decision-tree induction through evolutionary algorithms is presented in [6].

In a recent work, Basgalupp et al. [7] propose a decision-tree induction algorithm (called Beam Classifier) that seeks to avoid being trapped in local-optima by doing a beam search during the decision tree growth. A beam search algorithm keeps track of w states rather than just one. It begins with w randomly generated states (decision trees). At each step, all the successors of the w states are generated. If any successor is a goal state, the algorithm halts. Otherwise, it selects the w best successors from the complete list, discards the other states in the list, and repeats this loop until the quality of the best current tree cannot be improved. An interesting fact regarding the beam search algorithm is that if we set $w = 1$ we are actually employing the greedy strategy for inducing decision trees.

Beam Classifier starts with n empty decision trees (root nodes), where n is the number of data set attributes, and each root node represents an attribute. Then, the algorithm selects the best w trees according to a given criterion, and each one is expanded. For each expansion, the algorithm performs an adapted pre-order tree search method, expanding recursively (for each attribute) the leaf nodes from left to right. Thus, this expansion results in t new trees,

$$t = \sum_{i=1}^{w} \sum_{j=1}^{|\lambda_i|} m_{ij} \tag{2.44}$$

where w is the beam-width and m_{ij} is the number of available attributes[6] at the jth leaf node of the ith tree. Then, the algorithm selects once again the best w trees considering a pool of p trees, $p = w + t$. This process is repeated until a stop criterion is satisfied.

Other examples of non-greedy strategies for inducing decision trees include: (i) using *linear programming* to complement greedy-induced decision trees [8]; (ii) incremental and non-incremental *restructuring* of decision trees [118]; (iii) *skewing* the data to simulate an alternative distribution in order to deal with problematic cases for decision trees (e.g., the parity-like function) [86]; and (iv) *anytime learning* of decision trees [31].

[6] Nominal attributes are not used more than once in a given subtree.

Even though a lot of effort has been employed in the design of a non-greedy decision-tree induction algorithm, it is still debatable whether the proposed attempts can consistently obtain better results than the greedy top-down framework. Most of the times, the gain in performance obtained with a non-greedy approach is not sufficient to compensate for the extra computational effort.

2.5 Chapter Remarks

In this chapter, we presented the main design choices one has to face when programming a decision-tree induction algorithm. We gave special emphasis to the greedy top-down induction strategy, since it is by far the most researched technique for decision-tree induction.

Regarding top-down induction, we presented the most well-known splitting measures for univariate decision trees, as well as some new criteria found in the literature, in an unified notation. Furthermore, we introduced some strategies for building decision trees with multivariate tests, the so-called *oblique* trees. In particular, we showed that efficient oblique decision-tree induction has to make use of heuristics in order to derive "good" hyperplanes within non-terminal nodes. We detailed the strategy employed in the OC1 algorithm [77, 80] for deriving hyperplanes with the help of a *randomized* perturbation process. Following, we depicted the most common stopping criteria and post-pruning techniques employed in classic algorithms such as CART [12] and C4.5 [89], and we ended the discussion on top-down induction with an enumeration of possible strategies for dealing with missing values, either in the growing phase or during classification of a new instance.

We ended our analysis on decision trees with some alternative induction strategies, such as bottom-up induction and hybrid-induction. In addition, we briefly discussed work that attempt to avoid the greedy strategy, by either implementing *lookahead* techniques, *evolutionary algorithms*, *beam-search*, *linear programming*, *(non-) incremental restructuring*, *skewing*, or *anytime learning*. In the next chapters, we present an overview of *evolutionary algorithms* and *hyper-heuristics*, and review how they can be applied to decision-tree induction.

References

1. A. Agresti, *Categorical Data Analysis*, 2nd edn., Wiley Series in Probability and Statistics (Wiley-Interscience, Hoboken, 2002)
2. E. Alpaydin, *Introduction to Machine Learning* (MIT Press, Cambridge, 2010). ISBN: 026201243X, 9780262012430
3. P.W. Baim, A method for attribute selection in inductive learning systems. IEEE Trans. Pattern Anal. Mach. Intell. **10**(6), 888–896 (1988)
4. R.C. Barros et al., A bottom-up oblique decision tree induction algorithm, in *11th International Conference on Intelligent Systems Design and Applications*, pp. 450–456 (2011)

5. R.C. Barros et al., A framework for bottom-up induction of decision trees, Neurocomputing (2013 in press)
6. R.C. Barros et al., A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. Syst. Man, Cybern. Part C: Appl. Rev. **42**(3), 291–312 (2012)
7. M.P. Basgalupp et al., A beam-search based decision-tree induction algorithm, in *Machine Learning Algorithms for Problem Solving in Computational Applications: Intelligent Techniques*. IGI-Global (2011)
8. K. Bennett, Global tree optimization: a non-greedy decision tree algorithm. Comput. Sci. Stat. **26**, 156–160 (1994)
9. K. Bennett, O. Mangasarian, Multicategory discrimination via linear programming. Optim. Methods Softw. **2**, 29–39 (1994)
10. K. Bennett, O. Mangasarian, Robust linear programming discrimination of two linearly inseparable sets. Optim. Methods Softw. **1**, 23–34 (1992)
11. L. Bobrowski, M. Kretowski, Induction of multivariate decision trees by using dipolar criteria, in *European Conference on Principles of Data Mining and Knowledge Discovery*. pp. 331–336 (2000)
12. L. Breiman et al., *Classification and Regression Trees* (Wadsworth, Belmont, 1984)
13. L. Breslow, D. Aha, Simplifying decision trees: a survey. Knowl. Eng. Rev. **12**(01), 1–40 (1997)
14. C.E. Brodley, P.E. Utgoff, *Multivariate versus univariate decision trees*. Technical Report. Department of Computer Science, University of Massachusetts at Amherst (1992)
15. A. Buja, Y.-S. Lee, Data mining criteria for tree-based regression and classification, in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 27–36 (2001)
16. W. Buntine, A theory of learning classification rules, PhD thesis. University of Technology, Sydney (1992)
17. W. Buntine, Learning classification trees. Stat. Comput. **2**, 63–73 (1992)
18. R. Casey, G. Nagy, Decision tree design using a probabilistic model. IEEE Trans. Inf. Theory **30**(1), 93–99 (1984)
19. B. Cestnik, I. Bratko, On estimating probabilities in tree pruning, *Machine Learning-EWSL-91*, Vol. 482. Lecture Notes in Computer Science (Springer, Berlin, 1991), pp. 138–150
20. B. Chandra, R. Kothari, P. Paul, A new node splitting measure for decision tree construction. Pattern Recognit. **43**(8), 2725–2731 (2010)
21. B. Chandra, P.P. Varghese, Moving towards efficient decision tree construction. Inf. Sci. **179**(8), 1059–1069 (2009)
22. J. Ching, A. Wong, K. Chan, Class-dependent discretization for inductive learning from continuous and mixed-mode data. IEEE Trans. Pattern Anal. Mach. Intell. **17**(7), 641–651 (1995)
23. P. Chou, Optimal partitioning for classification and regression trees. IEEE Trans. Pattern Anal. Mach. Intell. **13**(4), 340–354 (1991)
24. P. Clark, T. Niblett, The CN2 induction algorithm. Mach. Learn. **3**(4), 261–283 (1989)
25. D. Coppersmith, S.J. Hong, J.R.M. Hosking, Partitioning nominal attributes in decision trees. Data Min. Knowl. Discov. **3**, 197–217 (1999)
26. R.L. De Mántaras, A distance-based attribute selection measure for decision tree induction. Mach. Learn. **6**(1), 81–92 (1991). ISSN: 0885–6125
27. G. De'ath, Multivariate regression trees: A new technique for modeling species-environment relationships. Ecology **83**(4), 1105–1117 (2002)
28. L. Devroye, L. Györfi, G. Lugosi, *A Probabilistic Theory of Pattern Recognition* (Springer, New York, 1996)
29. M. Dong, R. Kothari, Look-ahead based fuzzy decision tree induction. IEEE Trans. Fuzzy Syst. **9**(3), 461–468 (2001)
30. B. Draper, C. Brodley, Goal-directed classification using linear machine decision trees. IEEE Trans. Pattern Anal. Mach. Intell. **16**(9), 888–893 (1994)

31. S. Esmeir, S. Markovitch, Anytime learning of decision trees. J. Mach. Learn. Res. **8**, 891–933 (2007)
32. F. Esposito, D. Malerba, G. Semeraro, A comparative analysis of methods for pruning decision trees. IEEE Trans. Pattern Anal. Mach. Intell. **19**(5), 476–491 (1997)
33. F. Esposito, D. Malerba, G. Semeraro, A further study of pruning methods in decision tree induction, in *Fifth International Workshop on Artificial Intelligence and Statistics*. pp. 211–218 (1995)
34. F. Esposito, D. Malerba, G. Semeraro, Simplifying decision trees by pruning and grafting: new results (extended abstract), in *8th European Conference on Machine Learning*. ECML'95. (Springer, London, 1995) pp. 287–290
35. U. Fayyad, K. Irani, The attribute selection problem in decision tree generation, in *National Conference on Artificial Intelligence*. pp. 104–110 (1992)
36. A. Frank, A. Asuncion, *UCI Machine Learning Repository* (2010)
37. A.A. Freitas, A critical review of multi-objective optimization in data mining: a position paper. SIGKDD Explor. Newsl. **6**(2), 77–86 (2004). ISSN: 1931–0145
38. J.H. Friedman, A recursive partitioning decision rule for nonparametric classification. IEEE Trans. Comput. **100**(4), 404–408 (1977)
39. S.B. Gelfand, C.S. Ravishankar, E.J. Delp, An iterative growing and pruning algorithm for classification tree design. IEEE Int. Conf. Syst. Man Cybern. **2**, 818–823 (1989)
40. M.W. Gillo, MAID: A Honeywell 600 program for an automatised survey analysis. Behav. Sci. **17**, 251–252 (1972)
41. M. Gleser, M. Collen, Towards automated medical decisions. Comput. Biomed. Res. **5**(2), 180–189 (1972)
42. L.A. Goodman, W.H. Kruskal, Measures of association for cross classifications. J. Am. Stat. Assoc. **49**(268), 732–764 (1954)
43. T. Hancock et al., Lower bounds on learning decision lists and trees. Inf. Comput. **126**(2) (1996)
44. C. Hartmann et al., Application of information theory to the construction of efficient decision trees. IEEE Trans. Inf. Theory **28**(4), 565–577 (1982)
45. R. Haskell, A. Noui-Mehidi, Design of hierarchical classifiers, in *Computing in the 90s*, Vol. 507. Lecture Notes in Computer Science, ed. by N. Sherwani, E. de Doncker, J. Kapenga (Springer, Berlin, 1991), pp. 118–124
46. H. Hauska, P. Swain, The decision tree classifier: design and potential, in *2nd Symposium on Machine Processing of Remotely Sensed Data* (1975)
47. D. Heath, S. Kasif, S. Salzberg, Induction of oblique decision trees. J. Artif. Intell. Res. **2**, 1–32 (1993)
48. W. Hsiao, Y. Shih, Splitting variable selection for multivariate regression trees. Stat. Probab. Lett. **77**(3), 265–271 (2007)
49. E.B. Hunt, J. Marin, P.J. Stone, *Experiments in Induction* (Academic Press, New York, 1966)
50. L. Hyafil, R. Rivest, Constructing optimal binary decision trees is NP-complete. Inf. Process. Lett. **5**(1), 15–17 (1976)
51. A. Ittner, Non-linear decision trees, in *13th International Conference on Machine Learning*. pp. 1–6 (1996)
52. B. Jun et al., A new criterion in selection and discretization of attributes for the generation of decision trees. IEEE Trans. Pattern Anal. Mach. Intell. **19**(2), 1371–1375 (1997)
53. G. Kalkanis, The application of confidence interval error analysis to the design of decision tree classifiers. Pattern Recognit. Lett. **14**(5), 355–361 (1993)
54. A. Karalič, Employing linear regression in regression tree leaves, *10th European Conference on Artificial Intelligence*. ECAI'92 (Wiley, New York, 1992)
55. G.V. Kass, An exploratory technique for investigating large quantities of categorical data. APPL STATIST **29**(2), 119–127 (1980)
56. B. Kim, D. Landgrebe, Hierarchical classifier design in high-dimensional numerous class cases. IEEE Trans. Geosci. Remote Sens. **29**(4), 518–528 (1991)

57. I. Kononenko, I. Bratko, E. Roskar, Experiments in automatic learning of medical diagnostic rules. Technical Report Ljubljana, Yugoslavia: Jozef Stefan Institute (1984)
58. I. Kononenko, Estimating attributes: analysis and extensions of RELIEF, *Proceedings of the European Conference on Machine Learning on Machine Learning* (Springer, New York, 1994). ISBN: 3-540-57868-4
59. G. Landeweerd et al., Binary tree versus single level tree classification of white blood cells. Pattern Recognit. **16**(6), 571–577 (1983)
60. D.R. Larsen, P.L. Speckman, Multivariate regression trees for analysis of abundance data. Biometrics **60**(2), 543–549 (2004)
61. Y.-S. Lee, A new splitting approach for regression trees. Technical Report. Dongguk University, Department of Statistics: Dongguk University, Department of Statistics (2001)
62. X. Li, R.C. Dubes, Tree classifier design with a permutation statistic. Pattern Recognit. **19**(3), 229–235 (1986)
63. X.-B. Li et al., Multivariate decision trees using linear discriminants and tabu search. IEEE Trans. Syst., Man, Cybern.-Part A: Syst. Hum. **33**(2), 194–205 (2003)
64. H. Liu, R. Setiono, Feature transformation and multivariate decision tree induction. Discov. Sci. **1532**, 279–291 (1998)
65. W. Loh, Y. Shih, Split selection methods for classification trees. Stat. Sin. **7**, 815–840 (1997)
66. W. Loh, Regression trees with unbiased variable selection and interaction detection. Stat. Sin. **12**, 361–386 (2002)
67. D. Malerba et al., Top-down induction of model trees with regression and splitting nodes. IEEE Trans. Pattern Anal. Mach. Intell. **26**(5), 612–625 (2004)
68. O. Mangasarian, R. Setiono, W. H. Wolberg, Pattern recognition via linear programming: theory and application to medical diagnosis, in *SIAM Workshop on Optimization* (1990)
69. N. Manwani, P. Sastry, A Geometric Algorithm for Learning Oblique Decision Trees, in *Pattern Recognition and Machine Intelligence*, ed. by S. Chaudhury, et al. (Springer, Berlin, 2009), pp. 25–31
70. J. Martin, An exact probability metric for decision tree splitting and stopping. Mach. Learn. **28**(2), 257–291 (1997)
71. J. Mingers, An empirical comparison of pruning methods for decision tree induction. Mach. Learn. **4**(2), 227–243 (1989)
72. J. Mingers, An empirical comparison of selection measures for decision-tree induction. Mach. Learn. **3**(4), 319–342 (1989)
73. J. Mingers, Expert systems—rule induction with statistical data. J. Oper. Res. Soc. **38**, 39–47 (1987)
74. T.M. Mitchell, *Machine Learning* (McGraw-Hill, New York, 1997)
75. F. Mola, R. Siciliano, A fast splitting procedure for classification trees. Stat. Comput. **7**(3), 209–216 (1997)
76. J.N. Morgan, R.C. Messenger, *THAID: a sequential search program for the analysis of nominal scale dependent variables*. Technical Report. Institute for Social Research, University of Michigan (1973)
77. S.K. Murthy, S. Kasif, S.S. Salzberg, A system for induction of oblique decision trees. J. Artif. Intell. Res. **2**, 1–32 (1994)
78. S.K. Murthy, Automatic construction of decision trees from data: A multi-disciplinary survey. Data Min. Knowl. Discov. **2**(4), 345–389 (1998)
79. S.K. Murthy, S. Salzberg, Lookahead and pathology in decision tree induction, in *14th International Joint Conference on Artificial Intelligence*. (Morgan Kaufmann, San Francisco, 1995), pp. 1025–1031
80. S.K. Murthy et al., OC1: A randomized induction of oblique decision trees, in *Proceedings of the 11th National Conference on Artificial Intelligence* (AAAI'93), pp. 322–327 (1993)
81. G.E. Naumov, NP-completeness of problems of construction of optimal decision trees. Sov. Phys. Doklady **36**(4), 270–271 (1991)
82. T. Niblett, I. Bratko, Learning decision rules in noisy domains, in *6th Annual Technical Conference on Research and Development in Expert Systems III*. pp. 25–34 (1986)

83. N.J. Nilsson, *The Mathematical Foundations of Learning Machines* (Morgan Kaufmann Publishers Inc., San Francisco, 1990). ISBN: 1-55860-123-6
84. S.W. Norton, Generating better decision trees, *11th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, 1989)
85. K. Osei-Bryson, Post-pruning in regression tree induction: an integrated approach. Expert Syst. Appl. **34**(2), 1481–1490 (2008)
86. D. Page, S. Ray, Skewing: An efficient alternative to lookahead for decision tree induction, in *18th International Joint Conference on Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, 2003), pp. 601–607
87. A. Patterson, T. Niblett, *ACLS User Manual* (Intelligent Terminals Ltd., Glasgow, 1983)
88. K. Pattipati, M. Alexandridis, Application of heuristic search and information theory to sequential fault diagnosis. IEEE Trans. Syst. Man Cybern. **20**, 872–887 (1990)
89. J.R. Quinlan, *C4.5: Programs for Machine Learning* (Morgan Kaufmann, San Francisco, 1993). ISBN: 1-55860-238-0
90. J.R. Quinlan, Decision trees as probabilistic classifiers, in *4th International Workshop on Machine Learning* (1987)
91. J.R. Quinlan, Discovering rules by induction from large collections of examples, in *Expert Systems in the Micro-elect Age*, ed. by D. Michie (Edinburgh University Press, Edinburgh, 1979)
92. J.R. Quinlan, Induction of decision trees. Mach. Learn. **1**(1), 81–106 (1986)
93. J.R. Quinlan, Learning with continuous classes, in *5th Australian Joint Conference on Artificial Intelligent*. **92**, pp. 343–348 (1992)
94. J.R. Quinlan, Simplifying decision trees. Int. J. Man-Mach. Stud. **27**, 221–234 (1987)
95. J.R. Quinlan, Unknown attribute values in induction, in *6th International Workshop on Machine Learning*. pp. 164–168 (1989)
96. J.R. Quinlan, R.L. Rivest, Inferring decision trees using the minimum description length principle. Inf. Comput. **80**(3), 227–248 (1989)
97. M. Robnik-Sikonja, I. Kononenko, Pruning regression trees with MDL, in *European Conference on Artificial Intelligence*. pp. 455–459 (1998)
98. L. Rokach, O. Maimon, Top-down induction of decision trees classifiers—a survey. IEEE Trans. Syst. Man, Cybern. Part C: Appl. Rev. **35**(4), 476–487 (2005)
99. E.M. Rounds, A combined nonparametric approach to feature selection and binary decision tree design. Pattern Recognit. **12**(5), 313–317 (1980)
100. J.P. Sá et al., Decision trees using the minimum entropy-of-error principle, in *13th International Conference on Computer Analysis of Images and Patterns* (Springer, Berlin, 2009), pp. 799–807
101. S. Safavian, D. Landgrebe, A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. **21**(3), 660–674 (1991). ISSN: 0018–9472
102. I.K. Sethi, G.P.R. Sarvarayudu, Hierarchical classifier design using mutual information. IEEE Trans. Pattern Anal. Mach. Intell. **4**(4), 441–445 (1982)
103. S. Shah, P. Sastry, New algorithms for learning and pruning oblique decision trees. IEEE Trans. Syst. Man, Cybern. Part C: Applic. Rev. **29**(4), 494–505 (1999)
104. C.E. Shannon, A mathematical theory of communication. BELL Syst. Tech. J. **27**(1), 379–423, 625–56 (1948)
105. Y. Shih, Selecting the best categorical split for classification trees. Stat. Probab. Lett. **54**, 341–345 (2001)
106. L.M. Silva et al., Error entropy in classification problems: a univariate data analysis. Neural Comput. **18**(9), 2036–2061 (2006)
107. J.A. Sonquist, E.L. Baker, J.N. Morgan, *Searching for structure*. Technical Report. Institute for Social Research University of Michigan (1971)
108. J. Talmon, A multiclass nonparametric partitioning algorithm. Pattern Recognit. Lett. **4**(1), 31–38 (1986)
109. P.J. Tan, D.L. Dowe, MML inference of oblique decision trees, in *17th Australian Joint Conference on AI*. pp. 1082–1088 (2004)

110. P.-N. Tan, M. Steinbach, V. Kumar, *Introduction to Data Mining* (Addison-Wesley, Boston, 2005)
111. P.C. Taylor, B.W. Silverman, Block diagrams and splitting criteria for classification trees. Stat. Comput. **3**, 147–161 (1993)
112. P. Taylor, M. Jones, Splitting criteria for regression trees. J. Stat. Comput. Simul. **55**(4), 267–285 (1996)
113. L. Torgo, Functional models for regression tree leaves, in *14th International Conference on Machine Learning*. ICML'97. (Morgan Kaufmann Publishers Inc., San Francisco, 1997), pp. 385–393
114. L. Torgo, A comparative study of reliable error estimators for pruning regression trees, in *Iberoamerican Conference on Artificial Intelligence* (Springer, Berlin, 1998), pp. 1–12
115. L. Torgo, Error estimators for pruning regression trees, in *10th European Conference on Machine Learning* (Springer, Berlin, 1998), pp. 125–130
116. K.P. Unnikrishnan, K.P. Venugopal, Alopex: A correlation-based learning algorithm for feed-forward and recurrent neural networks. Neural Comput. **6**, 469–490 (1994)
117. P.E. Utgoff, Perceptron trees: a case study in hybrid concept representations. Connect. Sci. **1**(4), 377–391 (1989)
118. P.E. Utgoff, N.C. Berkman, J.A. Clouse, Decision tree induction based on efficient tree restructuring. Mach. Learn. **29**(1), 5–44 (1997)
119. P.E. Utgoff, C.E. Brodley, *Linear machine decision trees*. Technical Report. University of Massachusetts, Dept of Comp Sci (1991)
120. P.E. Utgoff, J.A. Clouse. *A Kolmogorov-Smirnoff Metric for Decision Tree Induction*. Technical Report. University of Massachusetts, pp. 96–3 (1996)
121. P. Utgoff, C. Brodley, An incremental method for finding multivariate splits for decision trees, in *7th International Conference on Machine Learning*. pp. 58–65 (1990)
122. P.K. Varshney, C.R.P. Hartmann, J.M.J. de Faria, Application of information theory to sequential fault diagnosis. IEEE Trans. Comput. **31**(2), 164–170 (1982)
123. D. Wang, L. Jiang, An improved attribute selection measure for decision tree induction, *in: 4th International Conference on Fuzzy Systems and Knowledge Discovery*. pp. 654–658 (2007)
124. Y. Wang, I.H. Witten, Induction of model trees for predicting continuous classes, in *Poster papers of the 9th European Conference on Machine Learning* (Springer, Berlin, 1997)
125. A.P. White, W.Z. Liu, Technical note: Bias in information-based measures in decision tree induction. Mach. Learn. **15**(3), 321–329 (1994)
126. S.S. Wilks, *Mathematical Statistics* (Wiley, New York, 1962)
127. I.H. Witten, E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations* (Morgan Kaufmann, San Francisco, 1999). ISBN: 1558605525
128. C.T. Yildiz, E. Alpaydin, Omnivariate decision trees. IEEE Trans. Neural Netw. **12**(6), 1539–1546 (2001)
129. H. Zantema, H. Bodlaender, Finding small equivalent decision trees is hard. Int. J. Found. Comput. Sci. **11**(2), 343–354 (2000)
130. X. Zhou, T. Dillon, A statistical-heuristic feature selection criterion for decision tree induction. IEEE Trans. Pattern Anal. Mac. Intell. **13**(8), 834–841 (1991)