An Efficient Algorithm for Finding Optimal Gain-Ratio Multiple-Split Tests on Hierarchical Attributes in Decision Tree Learning

Hussein Almuallim
Information and Computer Science Department
King Fahd University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
hussein@ccse.kfupm.edu.sa

Yasuhiro Akiba **Shigeo Kaneda**
NTT Communication Science Labs
1-2356 Take, Yokosuka-shi
Kanagawa 238-03, Japan
{akiba,kaneda}@nttkb.ntt.jp

Abstract

Given a set of training examples S and a tree-structured attribute x, the goal in this work is to find a multiple-split test defined on x that maximizes Quinlan's gain-ratio measure. The number of possible such multiple-split tests grows exponentially in the size of the hierarchy associated with the attribute. It is, therefore, impractical to enumerate and evaluate all these tests in order to choose the best one. We introduce an efficient algorithm for solving this problem that guarantees maximizing the gain-ratio over all possible tests. For a training set of m examples and an attribute hierarchy of height d, our algorithm runs in time proportional to dm, which makes it efficient enough for practical use.

Motivation

Current algorithms for learning decision trees from examples (e.g., CART (Breiman et al. 1984), C4.5 (Quinlan 1986; Quinlan 1993), GID3 (Fayyad & Irani 1992)) assume attributes that are ordered (which may be continuous or discrete) or nominal. Many domains, however, involve attributes that have a *hierarchy* of values, rather than a list of values (Almuallim, Akiba & Kaneda 1995). Figure 1 shows two examples of such *tree-structured* attributes. Each node in the tree associated with a tree-structured attribute is called a *category* and represents one of the values which the attribute may take. Figure 2 shows examples of the concept "colored polygons" described in terms of the *Color* and *Shape* attributes of Figure 1. In this case, the examples are described using *leaf* categories, while the concept itself is best described using the higher level categories *Chromatic* and *Polygon*.

Tests on tree-structured attributes may be binary, or can have multiple outcomes as shown in Figure 3. Searching for the category that gives the best binary split is not difficult and is discussed in (Almuallim, Akiba & Kaneda 1995). This is not the case, however, for multiple-split tests. It can be shown that the number of possible multiple-split tests for a given hierarchy grows exponentially in the number of leaves of the hierarchy. This makes it impractical to enumerate and evaluate all these tests in order to choose the best one. Nunez discusses a "hierarchy-climbing" heuristic within his EG2 algorithm (Nunez 1991) for handling this problem. Quinlan lists the support of tree-structured attributes as a "desirable extension" to his C4.5 package. In order to allow the current C4.5 code to handle such attributes, Quinlan suggests introducing a new nominal attribute for each level of the hierarchy, and encoding the examples using these newly introduced attributes(Quinlan 1993). This essentially means considering only those tests whose outcomes are all lying on one level in the hierarchy. For example, for the attribute *Shape*, only the three tests shown in Figure 4 are considered. This approach, however, has the following problems:

- Although it is true that any multiple-split test can be simulated using Quinlan's "one-level" tests, this comes at the expense of extra unnecessary splitting, and hence, extra complexity of the final decision tree. Moreover, the replication problem (as discussed by Pagallo and Haussler (Pagallo & Haussler 1990)) often arises as a consequence of such simulation.
- In most cases, one-level tests are not well-defined. For the *Color* attribute, for example, unless further background knowledge is available, we do not know whether to associate the category *Achromatic* with the category *Chromatic* or with the categories *Primary* and *Non-primary*, and so on.

In general, attempting to reduce the computational costs by restricting the attention to only a subset of the possible multiple-split tests is obviously associated with the risk of missing favorable tests.

This paper addresses the problem of how one can efficiently optimize over the *whole* set of possible multiple-split tests. We assume that the gain-ratio criterion (Quinlan 1986) is used to evaluate tests. It is well-known that tests with too many outcomes (those defined on low level categories) have more "splitting power" than those with few outcomes (defined on higher level categories). This necessitates the use of

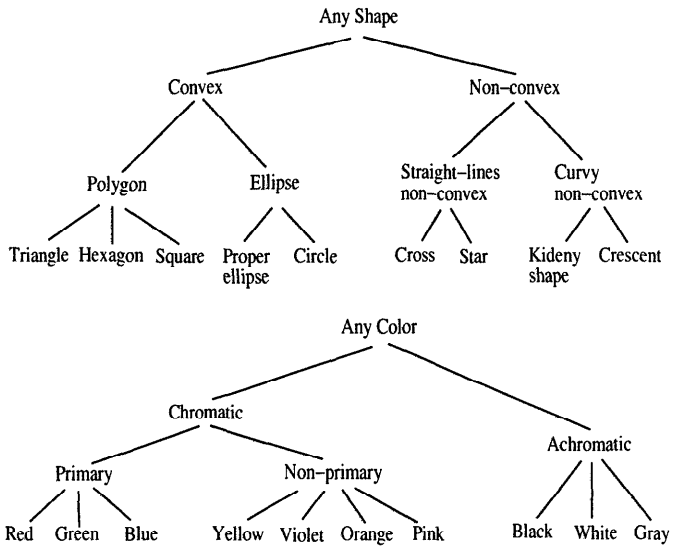

Figure 1: The *Shape* and *Color* hierarchies

$$
\begin{array}{llll}
\langle & [\text{ Yellow}, & \text{Square} &], + \rangle \\
\langle & [\text{ Green}, & \text{Hexagon} &], + \rangle \\
\langle & [\text{ White}, & \text{Cross} &], - \rangle \\
\langle & [\text{ Red}, & \text{Circle} &], - \rangle \\
\langle & [\text{ Black}, & \text{Circle} &], - \rangle
\end{array}
$$

Figure 2: Examples of the concept "colored polygons"

a criterion such as the gain-ratio that involves some penalty for excessive splitting.

We introduce an algorithm that, given a set of examples and a tree-structured attribute x, finds a multiple-split test defined on x with maximized gain-ratio over all possible multiple-split tests defined on x. Our algorithm employs a computational technique introduced by Breiman et al. in the context of decision tree pruning (Breiman et al. 1984). The proposed algorithm can be called from any top-down decision tree learning algorithm to handle tree-structured attributes. We show that when the number of examples is m and the depth of the hierarchy is d, our algorithm runs in time proportional to dm in the worst case, which is efficient enough from a practical point of view.

In the next section, we start by giving a precise definition of the problem studied in this paper. An overview of the pruning algorithm of Breiman et al. is then given, followed by an outline of our algorithm and a discussion of its time complexity. Finally, we conclude with a discussion of future research directions.

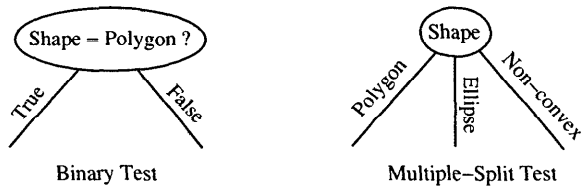

Figure 3: Binary and multiple outcome tests on attribute *Shape*

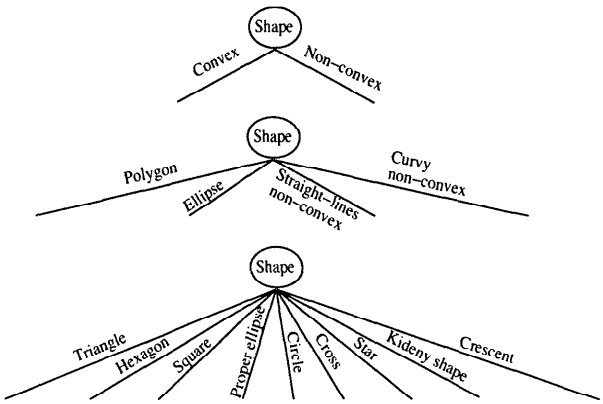

Figure 4: The three "one-level" tests for attribute *Shape*

Problem Definition

Let S be a set of training examples each having the form $[\ \langle a_1, a_2, \cdots, a_n \rangle, c\ ]$, where $a_1, a_2, \cdots, a_n$ are the values of attributes $x_1, x_2, \cdots, x_n$, and c denotes a *class*. Given such a set S, the basic operation in top-down induction of decision trees is to compute a score for each $x_i, 1 \leq i \leq n$, that measures how good it is to use x_i for the test at the root of the decision tree being learned for S. The attributes x_i may be continuous, discrete, and/or nominal. The objective of this work is to extend this to *tree-structured* attributes such as the *Shape* and *Color* attributes shown in Figure 1. A tree-structured attribute $x \in \{x_1, x_2, \cdots, x_n\}$ is associated with a hierarchy which we will denote by *x-tree*. Each node in *x-tree* is called a *category* of x. For simplicity, we assume that only the categories at the leaves of *x-tree* appear in the examples as values of x. (See Figure 2.)

Following (Haussler 1988), we define a *cut*, C, of *x-tree* as a subset of the categories of x satisfying the following two properties: (i) For any leaf ℓ of *x-tree*, either $\ell \in C$ or ℓ is a descendant of some category $g \in C$. (ii) For any two categories $i, j \in C$, i is not a descendant (nor an ancestor) of j. Figure 5 shows some cuts for the *Shape* hierarchy.

Each cut C of *x-tree* can be turned into a multiple-split test defined on x in a natural way. Namely, the

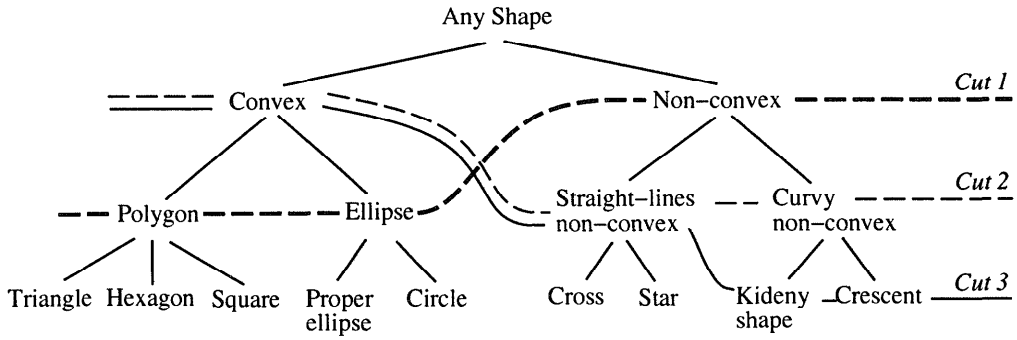

Figure 5: Examples of cuts for the *Shape* hierarchy

test would have $|C|$ outcomes—an outcome for each $g \in C$, where an example in which $x = v$ is of outcome g iff v is a descendant of g, or v is g itself. For example, *Cut 3* in Figure 5 represents a test with the four outcomes {*Convex, Straight-lines-non-convex, Kidney shape, Crescent*}. Clearly, the value of *Shape* in any example would belong to exactly one of these categories.

In this work, a test (cut) is evaluated using Quinlan's *gain-ratio* criterion. For a given set S of examples with q classes, let

$$Ent(S) = -\sum_{j=1}^{q} \frac{Freq(j,S)}{|S|} \times \log_2\left(\frac{Freq(j,S)}{|S|}\right),$$

where $Freq(j,S)$ denotes the number of examples of class j in S. The *mutual information* of a cut C for attribute x, denoted $MI(C)$, is defined as follows:

$$MI(C) = \sum_{g \in C} \frac{|S_g|}{|S|} \times Ent(S_g),$$

where S_g is the subset of S having the outcome g of the cut C for the attribute x. The *split information* for the cut is defined as

$$Sp(C) = -\sum_{g \in C} \frac{|S_g|}{|S|} \times \log_2\left(\frac{|S_g|}{|S|}\right).$$

Finally, the gain-ratio score for the cut C with respect to a training set S is given by

$$GR(C) = \frac{Ent(S) - MI(C)}{Sp(C)}.$$

With the above definitions, our problem can now be stated as follows:

> Given a set of examples S and a tree-structured attribute x with hierarchy *x-tree*, find a cut C of *x-tree* such that $GR(C)$ is maximum over all possible cuts of *x-tree*.

Note that cuts consisting of "specific" categories (those that appear at low levels of *x-tree*) give tests with too many outcomes and consequently yield good *MI* scores compared to cuts with more "general" categories which naturally have fewer outcomes. The use of gain-ratio (rather than the pure gain or other similar measures) helps in avoiding the tendency towards those tests with too many outcomes (Quinlan 1986).

It can be shown that the number of possible cuts for a given hierarchy grows exponentially in the number of leaves of the hierarchy. Thus, the challenge here is to solve the above optimization problem within affordable computational costs. It turns out that this task is very similar to the task of decision tree pruning. A natural one-to-one correspondence exists between cuts and trees obtained by pruning *x-tree*. Namely, a cut C is mapped to the pruned tree in which the subtrees rooted at each $g \in C$ are removed (substituted by leaves). Conversely, a pruned tree is mapped to the cut $C = \{g \mid g \text{ is a leaf in the pruned tree}\}$. This view allows employing a decision tree pruning technique introduced by Breiman et al. (Breiman et al. 1984) in solving our problem.

Breiman et al.'s Pruning Algorithm

Breiman et al. present an efficient optimal pruning algorithm in which the goal is to minimize what they call the *cost-complexity* of a decision tree. They assume that a decision tree T is given in which each test node t is associated with an error estimate $e(t)$ which measures the error introduced if the subtree below t is substituted by a leaf. The error of a tree T' obtained by pruning subtrees from T is then defined as $error(T') = \sum_{\ell: \text{leaf of } T'} e(\ell)$. They also define $size(T')$ as the number of leaves of T'. The *quality* of a pruned decision tree T' is measured as a linear combination of its size and error: $Score_\alpha(T') = error(T') + \alpha\ size(T')$, for a constant $\alpha \geq 0$.

The goal in Breiman et al.'s work is to find for each $\alpha \geq 0$ a pruned tree that minimizes $Score_\alpha$. Such a tree is said to be optimally pruned with respect to α. Although α runs through a continuum of values, only a finite sequence of optimally pruned decision trees ex-

ists, where each tree minimizes $Score_\alpha$ over a range of α. Breiman et al. show that such a sequence can be generated by repeatedly pruning at node t for which the quantity

$$\frac{e(t) - \sum_{\ell \in L(t)} e(\ell)}{|L(t)| - 1}$$

is minimum, where $L(t)$ denotes the set of leaves of the subtree rooted at node t in the *current* tree. They call this approach *weakest link* cutting.

Although Breiman et al. consider binary decision trees only, extending their algorithm to decision trees with multiple branches is straightforward (Bohanec & Bratko 1994). Moreover, in the setting of Breiman et al., each leaf in a pruned tree contributes a uniform amount (exactly 1) to the size of the tree. Nevertheless, the same algorithm can be easily generalized by associating a *weight*, $w(t)$, with each node t in the given tree T, and then letting $size(T') = \sum_{\ell:\text{ leaf of } T'} w(\ell)$, for a pruned tree T'. In this generalized setting, the node t at which pruning occurs is the node which minimizes the quantity

$$\frac{e(t) - \sum_{\ell \in L(t)} e(\ell)}{\sum_{\ell \in L(t)} w(\ell) - w(t)}.$$

Thus, generalized as above, the algorithm of Breiman et al. can be characterized as follows:

- **Input:** A decision tree T, with error estimate $e(t)$ and a weight $w(t)$ at each node t in T.
- **Output:** A sequence $\{(T_1, \alpha_1), (T_2, \alpha_2), (T_3, \alpha_3), \cdots, (T_r, \alpha_r)\}$, such that each T_i is a pruned tree that minimizes $Score_\alpha$ in the range $\alpha_{i-1} \leq \alpha < \alpha_i$, where $\alpha_0 = 0$, and $\alpha_r = \infty$.

Although Breiman et al. only address the case of binary trees and uniform weight, their arguments can be extended to our generalized case of multiple branch trees and non-uniform weights. Details are omitted here, however, for lack of space.

Finding a Multiple-Split Test with Optimal Gain-Ratio

We now outline our algorithm for finding a test with maximum gain-ratio for a given set of examples S, and a given attribute x with hierarchy *x-tree*. The first step of the algorithm is an initialization step:

Step I: For each category g in *x-tree*, attach an array CD_g to be used to store the class distribution at that category. This array has an entry for each class which is initially set to 0. We repeat the following steps for each example $e \in S$:

1. Let v be the value of attribute x in e.
2. Let c_1 be the class of e.
3. Increment $CD_v[c_1]$.
4. Climbing from v towards the root, increment $CD_g[c_1]$ for every ancestor g of v in *x-tree*.

At the end of Step I, each array CD_g will be storing the class distribution for those examples in S in which the value of the attribute x is a descendant of the category g. The next step computes the amounts each category g would contribute to $MI(C)$ and $Sp(C)$ if g were a member in a cut C.

Step II: For each category g in *x-tree*, let $|S_g| = \sum_c CD_g[c]$ (that is, the number of examples in which the value of attribute x is a descendant of g), and compute the following two quantities:

$$i(g) = -\frac{|S_g|}{|S|} \sum_c \frac{CD_g[c]}{|S_g|} \times \log_2 \left(\frac{CD_g[c]}{|S_g|} \right)$$

$$s(g) = -\frac{|S_g|}{|S|} \times \log_2 \left(\frac{|S_g|}{|S|} \right)$$

Now for any cut C, it is obvious that $MI(C)$ and $Sp(C)$ (as defined in Section 2) can be computed as $\sum_{g \in C} i(g)$ and $\sum_{g \in C} s(g)$, respectively. The next step is a call to the generalized algorithm of Breiman et al.:

Step III: Pass the tree *x-tree* to the generalized Breiman et al.'s algorithm, viewing each $i(g)$ and $s(g)$ as the error estimate and the weight of node g, respectively.

As explained previously, there is a one-to-one correspondence between the set of all possible cuts and the set of all possible pruned decision trees. Since we are passing $i(g)$ and $s(g)$ to Breiman et al.'s algorithm as the error estimates and the weights at the nodes, $error(T')$ and $size(T')$ are respectively equivalent to $MI(C)$ and $Sp(C)$, for the cut C corresponding to T'. This then justifies the following view:

Step IV: View the tree sequence returned by Breiman et al.'s algorithm as a sequence $\{(C_1, \alpha_1), (C_2, \alpha_2), (C_3, \alpha_3), \cdots, (C_r, \alpha_r)\}$, in which each C_i minimizes $Score_\alpha(C) = MI(C) + \alpha\ Sp(C)$ over all cuts C, within the range $\alpha_{i-1} \leq \alpha < \alpha_i$, where $\alpha_0 = 0$ and $\alpha_r = \infty$.

The cut sequence we now have at hand is not directly maximizing the gain-ratio, but rather optimizing under a different criterion ($MI(C) + \alpha\ Sp(C)$) which involves the unspecified parameter α. However, the following theorem puts things in perspective:

Theorem: *In the sequence $\{C_1, C_2, \cdots, C_{r-1}\}$ of cuts produced by employing the algorithm of Breiman et al., there exists a cut with maximum gain-ratio.*

Proof: This is shown by contradiction. Suppose none of the produced cuts maximizes the gain-ratio. Then, there exists some cut C^* such that, for all $1 \leq i \leq r-1$, we have $GR(C^*) > GR(C_i)$, that is

$$\frac{Ent(S) - MI(C^*)}{Sp(C^*)} > \frac{Ent(S) - MI(C_i)}{Sp(C_i)}, 1 \leq i \leq r-1. \quad (1)$$

Consider now the following value of α:

$$\alpha = \alpha_1 = \frac{Ent(S) - MI(C^*)}{Sp(C^*)}.$$

For any cut C, it is true that $Ent(S) \geq MI(C)$. Therefore, α_1 is a legitimate value for α since it is greater than or equal to 0. At this particular value for α,

$$\begin{aligned} Score_{\alpha_1}(C^*) &= MI(C^*) + \frac{Ent(S) - MI(C^*)}{Sp(C^*)} \times Sp(C^*) \\ &= Ent(S). \end{aligned}$$

On the other hand, for any i, $1 \leq i \leq r-1$,

$$\begin{aligned} Score_{\alpha_1}(C_i) &= MI(C_i) + \frac{Ent(S) - MI(C^*)}{Sp(C^*)} \times Sp(C_i) \\ &> MI(C_i) + \frac{Ent(S) - MI(C_i)}{Sp(C_i)} \times Sp(C_i) \\ &\qquad \text{(from (1))} \\ &= Ent(S). \end{aligned}$$

The above means that at α_1 we have $Score_{\alpha_1}(C^*) < Score_{\alpha_1}(C_i)$ for all C_i in $\{C_1, C_2, C_3, \cdots C_{r-1}\}$. This is a contradiction since for any value of $\alpha \geq 0$, one of the cuts in $\{C_1, C_2, C_3, \cdots C_{r-1}\}$ must minimize $Score_{\alpha_1}$.□

The above theorem leads to the following final step.

Step V: Compute the gain-ratio for each C_i, $1 \leq i \leq r-1$ and return the cut with maximum gain-ratio among these.[1]

Various details have been omitted in the above outline of our algorithm in order to simplify the discussion. In an actual implementation, Steps III and IV (finding the sequence of cuts) and Step V (computing the gain-ratio scores) can be run concurrently—each time a cut is generated, its gain-ratio is computed, and the cut is kept if its gain-ratio is the highest so far. In the appendix, we give a full pseudo-code description of the algorithm in which the weakest-link cutting algorithm of Breiman et al. is embedded.

Time Complexity Analysis

Let m be the number of examples and q the number of classes. Let the number of leaves of *x-tree* be s and let its height be d. Assume that each node in *x-tree* has at most k children. Then, it can be shown that the implementation given in the appendix runs in time $O(dm + (q + kd)s)$. We can, however, assume that $s \leq m$, since if this is not the case, then this means that some of the leaf categories in *x-tree* never show up in any example in S. In such a case, one can reduce the hierarchy by just ignoring these. More precisely, a category in *x-tree* is considered if and only if it is an ancestor of some leaf category that appears in at least one example in S. Reducing *x-tree* in this manner results in a hierarchy of at most m leaves. Thus, the time complexity of our algorithm is in fact $O((q + kd)m)$ in the worst case.

It is interesting to note that the above bound is only linear in the height of the hierarchy. Therefore, when dealing with somewhat balanced hierarchies, d becomes in the order of $\log s$, which is in turn in the order of $\log m$. This then gives time complexity of $O((q + k \log m)m)$. Since the number of classes, q is usually small compared to $k \log m$, this can be viewed as $O(km \log m)$. Interestingly enough, this is similar to the time complexity of $O(m \log m)$ for the task of handling continuous attributes (Quinlan 1986; Fayyad & Irani 1992).

Conclusion and Extensions

For a given tree-structured attribute, the goal of this work is to find a multiple-split test that maximizes Quinlan's gain-ratio measure with respect to a given set of training examples. We presented an algorithm that achieves this goal and runs in time linear in the number of training examples times the depth of the hierarchy associated with the tree-structured attribute.

In no way one can claim any superiority of multiple-split tests in generalization performance over other kinds of tests, such as binary tests that are based on a single value of the attribute (See Figure 3). In fact, multiple-split tests and binary tests should not be viewed as mutually exclusive choices. One indeed can find the best multiple-split test using our method, and in parallel, find the best binary split test using the method of (Almuallim, Akiba & Kaneda 1995), and finally choose from these the test with higher score.

The gain-ratio criterion of Quinlan is "hard-wired" in our algorithm. It would be interesting to generalize the algorithm to cover other similar measures as well. It is also interesting to consider tests that group different values of a tree-structured attribute in a single outcome. This kind of tests is studied in (Fayyad 1994; Fayyad & Irani 1993; Quinlan 1993) for other attribute types. Finally, in certain applications, attributes may be associated with directed acyclic graphs (DAG's) rather than trees as assumed in our work. Studying this generalized problem is an important future research direction.

Acknowledgment

Hussein Almuallim thanks King Fahd University of Petroleum & Minerals for their support. This work was partially conducted during his visit to Prof. Shimura Lab. of Tokyo Institute of Technology, sponsored by Japan's Petroleum Energy Center. Thanks also to Hideki Tanaka of NHK, Japan for a useful discussion.

Appendix

Our algorithm is described below in pseudo code. All the variables are assumed global. $i[g]$ and $s[g]$ are computed for each node g in line 2.1 of *FindBestCut*. At each node g, $\alpha[g]$ stores the value of α above which

[1] Note that the test corresponding to C_r is not interesting since it has only a single outcome and does no splitting.

the subtree rooted at g is pruned. This is initialized at lines 2.2.4 and 2.3.4 of *FindBestCut*. Each call to *PruneOnce* results in pruning the subtree rooted at g for which $\alpha[g]$ is minimum over all (unpruned) nodes of T. At that time, the flag $Pruned[g]$ becomes *True* and $\alpha[s]$ is updated for all ancestors s of g.

The variable $Smallest.\alpha.Below[g]$ stores the smallest α value over all descendant of g. This variable is stored in order to efficiently locate the node with minimum α in the current tree in each pruning iteration. $SubTreeMI[g]$ and $SubTreeSp[g]$ store the sum of $i[\ell]$ and $s[\ell]$, respectively, for all leaves ℓ of the current subtree rooted at g. These are initialized in step 2 of *FindBestCut*, and then updated in step 10 of *PruneOnce* each time pruning occurs at a descendant of g. The current best cut is kept track of by the flag $InBestCut[g]$. A node g is in the current best cut if this flag is *True* for g and *False* for all its ancestors.

Algorithm *FindBestCut*
Input: A sample S, an attribute x, its hierarchy T
1. Initialize the arrays *CD* as in STEP I.
2. Traverse T in post-order. For each g in T:
2.1. Compute $i[g]$ and $s[g]$ as in STEP II.
2.2. If g is a leaf then
2.2.1. $Pruned[g] = True$
2.2.2. $SubTreeMI[g] = i[g]$
2.2.3. $SubTreeSp[g] = s[g]$
2.2.4. $\alpha[g] = \infty$
2.2.5. $Smallest.\alpha.Below[g] = \infty$
2.2.6. $InBestCut[g] = True$
2.3. else
2.3.1. $Pruned[g] = False$
2.3.2. $SubTreeMI[g] = \sum_{y:\text{child of } g} SubTreeMI[y]$
2.3.3. $SubTreeSp[g] = \sum_{y:\text{child of } g} SubTreeSp[y]$
2.3.4. $\alpha[g] = \frac{i[g] - SubTreeMI[g]}{SubTreeSp[g] - s[g]}$
2.3.5. $Smallest.\alpha.Below[g] = \min\{\alpha[g], \min\{Smallest.\alpha.Below[y] | y \text{ is a child of } g\}\}$
2.3.6. $InBestCut[g] = False$
3. $BestGR = \frac{i[\text{root of } T] - SubTreeMI[\text{root of } T]}{SubTreeSp[\text{root of } T]}$
4. $p = PruneOnce$
5. While $p \neq$ root of T:
5.1. $ThisGR = \frac{i[\text{root of } T] - SubTreeMI[\text{root of } T]}{SubTreeSp[\text{root of } T]}$
5.2. If $ThisGR \geq BestGR$ then
5.2.1. $BestGR = ThisGR$
5.2.2. $InBestCut[p] = True$
5.2.3. $p = PruneOnce$
6. Report $BestGR$ as the best gain-ratio over all cuts of T
7. Report the set $\{g \mid InBestCut[g] = True$ and for every ancestor g' of g, $InBestCut[g'] = False\}$ as the best cut.

Procedure *PruneOnce*
1. Let $g =$ root of T
2. While $\alpha[g] > Smallest.\alpha.Below[g]$:
2.1. $g' =$ child of g such that $Smallest.\alpha.Below[g']$ is minimum
2.2. $g = g'$
3. $Pruned[g] = True$
4. $\alpha[g] = \infty$
5. $Smallest.\alpha.Below[g] = \infty$
6. $SubTreeMI[g] = i[g]$
7. $SubTreeSp[g] = s[g]$
8. $p = g$
9. If $p =$ root of T, return p
10. Repeat
10.1. $g =$ parent of g
10.2. $SubTreeMI[g] = \sum_{y:\text{child of } g} SubTreeMI[y]$
10.3. $SubTreeSp[g] = \sum_{y:\text{child of } g} SubTreeSp[y]$
10.4. $\alpha[g] = \frac{i[g] - SubTreeMI[g]}{SubTreeSp[g] - s[g]}$
10.5. $Smallest.\alpha.Below[g] = \min\{\alpha[g], \min\{Smallest.\alpha.Below[g'] \mid g' \text{ is a kid of } g\}\}$
10.6. until $g =$ the root of T
11. Return p

References

Almuallim, H.; Akiba, Y.; and Kaneda, S. 1995. On Handling Tree-Structured Attributes in Decision Tree Learning. In Proceedings of the 12th International Conference on Machine Learning, p. 12–20. San Francisco, California: Morgan Kaufmann.

Bohanec, M.; and Bratko, I. 1994. Trading Accuracy for Simplicity in Decision Trees. *Machine Learning*, 15:223–250.

Breiman, L.; Friedman, J.H.; Olshen, R.A.; and Stone, C.J. 1984. *Classification and Regression Trees.* Belmont: Wadsworth.

Fayyad, U. M.; and Irani, K. B. 1992. On the Handling of Continuous Valued Attributes in Decision Tree Generation. *Machine Learning*, 8:87–102.

Fayyad, U. M.; and Irani, K. B. 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence, p. 1022–1027.

Fayyad, U. M. 1994. Branching on Attribute Values in Decision Tree Generation. In Proceedings of the 12th National Conference on Artificial Intelligence, p. 601–606.

Haussler, D. 1988. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221.

Nunez, M. 1991. The Use of Background Knowledge in Decision Tree Induction. *Machine Learning*, 6: 231–250.

Pagallo, G.; and Haussler, D. 1990. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, 5(1):71–100.

Quinlan, J. R. 1986. Induction of Decision Trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*, p. 104. San Mateo, CA: Morgan Kaufmann.