Machine Learning, 36, 201–244 (1999)

General and Efficient Multisplitting of Numerical Attributes

TAPIO ELOMAA elomaa@cs.helsinki.fi
Department of Computer Science, P.O. Box 26, FIN-00014 University of Helsinki, Finland

JUHO ROUSU Juho.Rousu@vtt.fi
VTT Biotechnology and Food Research, Tietotie 2, P.O. Box 1501, FIN-02044 VTT, Finland

Editor: Robert C. Holte

Abstract. Often in supervised learning numerical attributes require special treatment and do not fit the learning scheme as well as one could hope. Nevertheless, they are common in practical tasks and, therefore, need to be taken into account. We characterize the well-behavedness of an evaluation function, a property that guarantees the optimal multi-partition of an arbitrary numerical domain to be defined on boundary points. Well-behavedness reduces the number of candidate cut points that need to be examined in multisplitting numerical attributes. Many commonly used attribute evaluation functions possess this property; we demonstrate that the cumulative functions Information Gain and Training Set Error as well as the non-cumulative functions Gain Ratio and Normalized Distance Measure are all well-behaved. We also devise a method of finding optimal multisplits efficiently by examining the minimum number of boundary point combinations that is required to produce partitions which are optimal with respect to a cumulative and well-behaved evaluation function. Our empirical experiments validate the utility of optimal multisplitting: it produces constantly better partitions than alternative approaches do and it only requires comparable time. In top-down induction of decision trees the choice of evaluation function has a more decisive effect on the result than the choice of partitioning strategy; optimizing the value of most common attribute evaluation functions does not raise the accuracy of the produced decision trees. In our tests the construction time using optimal multisplitting was, on the average, twice that required by greedy multisplitting, which in its part required on the average twice the time of binary splitting.

Keywords: supervised learning, numerical attributes, optimal partitions, evaluation functions

1. Introduction

In classification learning methods based on conquering strategies the sample is partitioned into smaller subsets in an attempt to discover a final partition in which the class distribution of the subsets reflects the classification of (future) instances. In attribute-based induction the correlation of an instance's class and its attribute values is heuristically approximated by an *evaluation function*, or a *goodness criterion*. In the most basic induction scheme—univariate induction—a single attribute's value is the basis of sample partitioning: the data is divided into subsets according to the value of that attribute which induces the partition that evaluates as the best.

In this paper we use top-down induction of decision trees (TDIDT) as the representative learning scheme. In it a node querying the value of the chosen attribute is inserted into the evolving tree. If the attribute in question has a small nominal domain it is simple to continue tree construction by growing a subtree corresponding to each separate value in the domain. Handling a large discrete domain can turn out to be more problematic, since evaluation functions tend to have biases for, or against, multivalued attributes (Quinlan, 1988; White & Liu, 1994; Kononenko, 1995). Attempts to rectify such unwanted biases can lead to irrelevant attributes or those with low information content gaining an unfair advantage over the preferable attributes (Quinlan, 1986). Many evaluation functions have been developed to address this deficiency (Quinlan, 1986; López de Màntaras, 1991; Fayyad & Irani, 1992a).

The real challenge for decision tree learning, however, is issued by *numerical attributes* with a very large, even infinite domain, which can be either discrete (integer) or continuous (real). Numerical attributes are frequent in real-world induction tasks and their proper treatment is important (Catlett, 1991; Fayyad & Irani, 1992b, 1993; Quinlan, 1993, 1996; Van de Merckt, 1993; Maass, 1994; Dougherty, Kohavi, & Sahami, 1995; Fulton, Kasif, & Salzberg, 1995). For example, Quinlan (1996) was compelled to reconsider this problem in his well-known C4.5 decision tree learner (Quinlan, 1993).

The problem of choosing the interval borders and the right arity for the discretization of a numerical value range remains an open problem in numerical attribute handling. In practical decision tree learners the problem is often solved by using *binarization* (Breiman et al., 1984; Cestnik, Kononenko, & Bratko, 1987; Quinlan, 1993), in which the value range is split into only two intervals in any one node. Further partitioning of the domain is generated by continuing the binarization of previously induced intervals further down the tree. This way the halving approach decides the location of the interval borders and the global properties of the tree decide the number of resulting intervals.

An alternative approach uses *greedy multisplitting* (Catlett, 1991; Fayyad & Irani, 1993) where the value range is similarly partitioned by recursive binarization, but all at once; the resulting multisplit is assigned to a single node of the evolving tree. In Catlett's (1991) work the discretization of numerical value ranges was carried out globally at the preprocessing stage while Fayyad and Irani (1993) partitioned the value ranges dynamically, during the growing of the tree. The depth-first search scheme of these methods does not directly assess the global quality of the multi-partition and thus often requires some method—a stopping criterion—to control the growth of the arity of the final partition. Using greedy breadth-first search with an evaluation function that is balanced with respect to favoring high-arity partitions relieves one from such a requirement (see Section 5). However, neither method guarantees the optimality of the resulting multisplit. Still, their efficiency makes them attractive choices in practical applications.

This paper addresses the problem of finding *optimal multiway partitions* for numerical attribute value ranges. The only known general algorithm for finding them is the exponential brute-force method, which, of course, is not a feasible solution in practice. Fulton, Kasif, and Salzberg (1995) devised an algorithm for finding efficiently optimal multisplits for a class of evaluation functions. In this article we enhance and extend their method and give an exact characterization of the kind of attribute evaluation functions for which the enhanced approach is valid. Henceforth, optimal multisplit always refers to the partitioning of the data that evaluates as the best when using a given goodness criterion. The main contributions of this paper are:

- A characterization of *useful* and *well-behaved* evaluation functions, for which it suffices to inspect only few candidate cut points—the *boundary points*—when searching for a good binary partition or multisplit, respectively. Boundary points were originally defined by Fayyad and Irani (1992b) when studying binary splitting using evaluation functions that are *convex downwards*. Codrington and Brodley (1999) have rigorously studied the convexity properties of many widely used attribute evaluation functions. Convex evaluation functions are a proper subclass of well-behaved evaluation functions. Hence, the results in this paper strictly generalize those of Fayyad and Irani in two ways: the approach of focusing on boundary points becomes applicable to a larger repertoire of attribute evaluation functions and, more importantly, it extends to multisplitting.
- A general and efficient method of finding optimal multiway partitions naturally arises; we adapt the dynamic programming evaluation scheme proposed by Fulton, Kasif, and Salzberg (1995) to take advantage of the boundary points and operate on example intervals instead of individual examples. These enhancements enable an elegant and efficient implementation of the search procedure. The generality of the method means that it can accommodate many different evaluation functions, all those that are *cumulative*. Given an evaluation function with a well-balanced bias (Quinlan, 1986; White & Liu, 1994; Kononenko, 1995) there is no need to employ another function to control the growth of partition arity.
- We demonstrate that well-behaved evaluation functions are common by proving that two goodness criteria stemming from different traditions and backgrounds—the MDL/MML cost function of Wallace and Patrick (1993) and *Training Set Error* (Maass, 1994; Auer, Holte, & Maass, 1995)—are well-behaved. Fayyad and Irani (1992b, 1993) have previously shown the convexity of *Average Class Entropy* and Quinlan's (1983) *Information Gain* functions in binarization tasks. By the results in this paper (Theorem 2) they both are well-behaved. In addition, we study the default evaluation function, *Gain Ratio*, of C4.5 decision tree learner (Quinlan, 1993), and one of its alternatives, *Normalized Distance Measure* (López de Màntaras, 1991). They turn out to be non-convex but still both behave well in multisplitting. However, since they are not cumulative, the above-mentioned dynamic programming approach is not applicable for them.
- Finally, we report an empirical comparison of multisplitting strategies. We contrast the greedy approach with the optimal algorithm. We also include in this comparison, as a baseline, a method that selects the cut points randomly. Our experiments show that better quality partitions are obtained by using optimal multisplitting than by the other two multisplitting strategies. However, the advantage of optimal partitioning over greedy multisplitting is only marginal. To explore the utility of multisplitting in decision tree learning we also compare the greedy and optimal strategies with binarization within the C4.5 framework. This comparison shows that the choice of the evaluation function has a more decisive impact on the result than the choice of the splitting strategy.

We review, in Section 2, the basic problem setting in handling numerical attributes and discuss the most important preceding research on the topic. In particular, we reformulate the definition of a boundary point. In Section 3 we define useful and well-behaved evaluation functions. We show that the desirable properties of useful functions carry over to multisplitting as long as the function is cumulative; such a function is also well-behaved. In addition, we evolve an efficient method of finding optimal multisplits for numerical attributes using cumulative and well-behaved evaluation functions. We explore the extent of this class of functions in Section 4. First we analyze non-cumulative evaluation functions related to the C4.5 decision tree learner. We show that both Gain Ratio function and Normalized Distance Measure are able to take advantage of boundary points in multisplitting. Then we take two cumulative goodness criteria of different nature and prove their well-behavedness. Section 5 reports extensive experiments comparing the methods developed in this paper with each other and with the most common approach to numerical attribute handling. Section 6 reflects upon the outcome of the experiments and discusses related research. Finally, Section 7 presents the concluding remarks and directions for future research. Appendix A contains a detailed description of the search algorithm for optimal multiway partitions and analyzes its time and space complexity. In Appendix B we study, for a large collection of real-world data, the relationship between the number of different values and the number of boundary points in numerical attributes' domains.

2. Problem setting

This section presents the preliminaries of the subsequent work and reviews some of the earlier related research. We examine partitions, boundary points, and evaluation functions that are used to rank partition candidates.

2.1. Partitioning numerical attribute domains

The basic setting that we consider is the following. We have preclassified data acquired from the application domain at our disposal. An *example* is a vector containing values for the attributes at hand together with an associated classification. The aim is to find a good predictor for classifying, on the basis of the given attributes, further instances from the same application domain. To that end the greedy TDIDT construction algorithm needs to estimate how well an attribute's values correlate with the classification of examples. The respective correlations of the attributes are compared and the attribute that appears best is added to the evolving tree. For correlation comparison a fixed evaluation function is applied to each attribute. Prior to comparing a numerical attribute with the nominal ones it is necessary to find the best partition of the attribute's values. The same evaluation function is used to rank the partitions and the attributes.

Partitioning of a numerical attribute A begins by sorting into ascending order the values for attribute A of the n examples. Let $\mathrm{val}_A(s)$ denote the value of the attribute A in the example s, and $\mathrm{val}_C(s)$ denote the class of s. The classes are assumed to be integers $j \in \{1, \dots, m\}$. The *majority* class in the example set S,

$$\mathrm{maj}_C(S) = \arg\max_{1 \le j \le m} |\{s \in S \mid \mathrm{val}_C(s) = j\}|,$$

is the most frequently occurring class. If there are several maximally frequent classes, $\text{maj}_C(S)$ is the smallest one. This tie-breaking is simply to guarantee that $\text{maj}_C(S)$ is unique, it plays no role in any of the algorithms.

A *partition* $\biguplus_{i=1}^{k} S_i$ of the sample S into k intervals has the following properties:

1. it consists of non-empty subsets: for all $i \in \{1, \ldots, k\}$, $S_i \neq \emptyset$,
2. covers the whole domain: $\bigcup_{i=1}^{k} S_i = S$, and
3. the subsets are disjoint: if $s \in S_i$, then $s \notin S_j$, for all $j \neq i$.

Furthermore, if partition $\biguplus_{i=1}^{k} S_i$ has been induced on the basis of attribute A, then for all $i < j$, if $s_i \in S_i$ and $s_j \in S_j$, then $\text{val}_A(s_i) < \text{val}_A(s_j)$. When splitting a set S of examples on the basis of the value of an attribute A, there is a set of thresholds $\{T_1, \ldots, T_{k-1}\} \subseteq \text{Dom}(A)$ that defines a partition $\biguplus_{i=1}^{k} S_i$ for the sample in an obvious manner:

$$S_i = \begin{cases} \{s \in S \mid \text{val}_A(s) \leq T_1\} & \text{if } i = 1, \\ \{s \in S \mid T_{i-1} < \text{val}_A(s) \leq T_i\} & \text{if } 1 < i < k, \text{ and} \\ \{s \in S \mid T_{k-1} < \text{val}_A(s)\} & \text{if } i = k. \end{cases}$$

In this article partitions usually have arity k. When there is no danger of confusion, we write $\biguplus S_i$ in place of $\biguplus_{i=1}^{k} S_i$ to clarify the notation.

A partition of a numerical value range can set the interval thresholds only at points where the value in the ordered sequence changes. These points are the potential *cut points* of the data. A sequence of consecutive examples with the same value for the attribute under consideration is called a *bin*.

2.2. Boundary points

Fayyad and Irani's (1992b) analysis of the binarization technique proved that optimal splits always fall on *boundary points* when the Information Gain function (Quinlan, 1983) is used. Hence, substantial reductions in time consumption can be obtained, since only the boundary points need to be considered as potential cut points.

Definition 1. Let a sequence S of examples be sorted by the value of a numerical attribute A. The set of *boundary points* is defined as follows:

1. The maximum value in S is a boundary point.
2. A value $T \in \text{Dom}(A)$ is a boundary point if and only if there exists a pair of examples $u, v \in S$, having different classes, such that $\text{val}_A(u) = T < \text{val}_A(v)$; and there does not exist another example $w \in S$ such that $\text{val}_A(u) < \text{val}_A(w) < \text{val}_A(v)$.

This definition differs from Fayyad and Irani's (1992b) most notably by including the maximum value as a boundary point. It is included to ensure that every example set has at least one boundary point for each numerical attribute. There is also a difference in the definition of the actual boundary points (case 2 above). In the original definition a boundary point was taken to be a value that is strictly in between the values $\mathrm{val}_A(u)$ and $\mathrm{val}_A(v)$. We follow the common convention that the thresholds defining a partition are values of the attribute appearing in the data (e.g., Quinlan, 1993).

The example sequence in between two consecutive boundary points is called a *block* (of examples). A block can be either class *uniform* or its class distribution can be *mixed*. A uniform block may consist of a single example, but a mixed one always has at least two elements. Observe that the blocks are obtained from the bins by merging only adjacent class uniform bins with the same class label into a block. Mixed bins are never merged into a block with another bin; they always constitute a block of their own.

Example. Consider the set of 27 examples shown in figure 1. For each example its class value, in the set $\{A, B, C\}$, and the value of a numerical attribute—integer-valued in this case—are shown. The examples have been sorted into ascending order according to the value of the numerical attribute. The cut points and the bins of this data are depicted in figure 2.

Let us illustrate how the boundary points can be located with the help of the bins from the categorized version of the data set in figure 2. Boundary points are borders of consecutive bins D and D' such that

1. all examples in bin D belong to class C and all examples in D' to C', and $C \neq C'$, or
2. there is a *mixed* class distribution in either of the bins.

Thus, we can reduce the original set of 27 examples down to the seven blocks depicted in figure 3. Only the class distribution of each block needs to be known in order to be able to compute the impurities of partitions defined on boundary points.

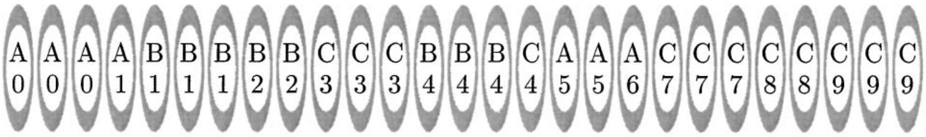

Figure 1. A set of examples sorted into ascending order according to the value of a numerical attribute. The class labels of the examples are also shown.

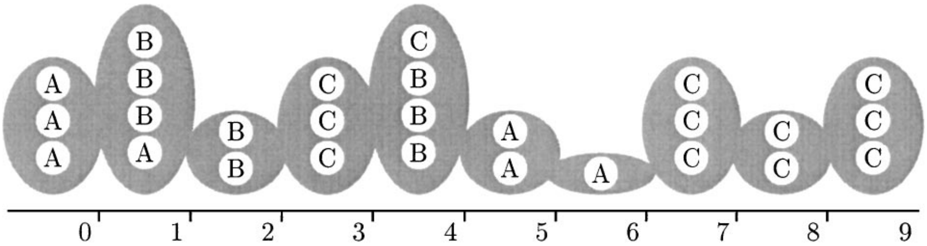

Figure 2. The example set can only be partitioned on threshold values that are bin borders.

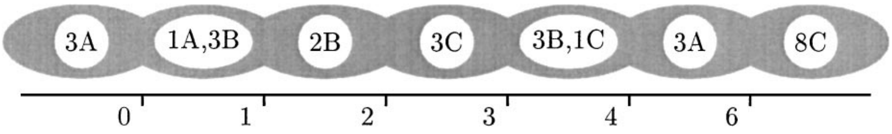

Figure 3. The original set of 27 examples can be reduced to 7 blocks. The block borders coincide with boundary points. Only the class frequencies within blocks need to be known.

Gathering examples into blocks can only be applied for a certain class of evaluation functions. We study the extent and generality of this class in Section 4. Block construction does not depend on duplicate values, hence the processing is effective even for truly continuous ranges. If an attribute in isolation has predictive power, i.e., if it directly correlates with the examples' classification, then the number of the blocks in the range, B, satisfies $B \ll n$ (Fayyad & Irani, 1992b). Subsequently we study in detail the relationship of values the B, V, and n for a large collection of real-world data sets (see Appendix B).

2.3. *Evaluation functions: The average class entropy*

The goodness criteria which are used to evaluate candidate partitions are many (see e.g., Mingers, 1989; Buntine & Niblett, 1992; Kononenko, 1995). The most commonly used attribute evaluation functions build upon *impurity measures* (Breiman et al., 1984), which are functions that try to estimate the class coherence in a given set of examples. An example of such measures is the *entropy* function H,

$$H(S) = -\sum_{j=1}^{m} P(C_j, S) \log_2 P(C_j, S),$$

in which m denotes the number of classes and $P(C, S)$ stands for the proportion of the examples in S that belong to the class C. Attribute evaluation functions based on impurity measures include the Gini Index (Breiman et al., 1984) and Quinlan's (1986) Information Gain and Gain Ratio. Fayyad and Irani (1992b) focused on a particular evaluation function, the *Average Class Entropy*. Let $\biguplus S_i$ be a partition of S, then by $ACE(\biguplus S_i)$ we denote the Average Class Entropy of the partition:

$$ACE\left(\biguplus_i S_i\right) = \sum_i \frac{|S_i|}{|S|} H(S_i) = \frac{1}{|S|} \sum_i |S_i| H(S_i).$$

Definition 2. Let $\biguplus_i S_i$ be a partition of an arbitrary example set S. An evaluation function F is *cumulative* if there exists a function f such that

$$F\left(\biguplus_i S_i\right) = c \cdot \sum_i f(S_i),$$

where c is an arbitrary coefficient whose value may depend on, e.g., the whole data S, but not on its partitions.

For example, *ACE*, defined above, is cumulative. Cumulativity facilitates incremental evaluation of impurity values (Fulton, Kasif, & Salzberg, 1995), but it does not relieve us from the need to examine all candidate cut points when searching for an optimal partition of the data.

In this paper we assume that the aim is to find a partition that minimizes the given evaluation function. All definitions and results in the next section have their natural counterparts that apply for maximization.

Fayyad and Irani (1992b) proved that the Average Class Entropy has the property that it will never favor an obviously bad cut, one that needlessly separates examples of one class. They proved that when searching for the best binary split by choosing a single cut point, we can restrict our attention to boundary points.

Theorem 1 (Fayyad & Irani, 1992b). *If value T defines a binary partition $S_0 \uplus S_1$ of S that minimizes ACE, then T is a boundary point.*

There is one exception to this result, viz., in the degenerate case that the sample S contains only members of one class, then $ACE(S_0 \uplus S_1) = 0$, independent of whether the value defining the partition is a boundary point or not. Recall that in this case the only (additional) boundary point is at the far end of the value range. We can surmount this complication by considering the boundary points and by focusing on such minimum impurity partitions that do not contain superfluous thresholds. Subsequently we give examples of evaluation functions that inherently penalize needless splitting of the data, thus avoiding this complication without further measures.

3. Multisplitting numerical attributes optimally

Fayyad and Irani (1992b) base the proof of Theorem 1 on the fact that Average Class Entropy is *convex downwards* in between any two boundary points. Thus, its minimum values for binary partitions can only occur at boundary points. Convexity is a sufficient but not a necessary requirement for useful behavior. Subsequently, we give examples of non-convex functions that minimize on boundary points.

In the following, by the shorthand notation $F(T)$ we denote the value of the evaluation function F for the binary partition that is defined by the threshold T. Similarly, $F(T_1, \dots, T_{k-1})$ denotes the value of the evaluation function F for the k-partition defined by cut points $T_1, \dots, T_{k-1}$.

Definition 3. Let S be an example set and let $\mathcal{T} = \{T_1, \dots, T_B\}$ be the boundary points in S for numerical attribute A. An evaluation function is *useful* (in binary partitioning) if there exists a value T in $\mathcal{T}$ such that $F(T) \leq F(W)$ for all $W \in \mathrm{Dom}(A)$.

For example, *ACE* is useful. A stronger property, which requires all optimal partitions to be defined on boundary points, has been studied by Codrington and Brodley (1999). They call this the *minima-free property*.

Let us now define *well-behaved* evaluation functions, those that always—not just in the binary case—have an optimal multisplit on boundary points.

Definition 4. Let S be an example set and let $\mathcal{T} = \{T_1, \ldots, T_B\}$ be the boundary points in S for numerical attribute A. An evaluation function is *well-behaved* if for any $1 \leq k \leq B$ there exists a set of at most k values $\mathcal{T}' \subseteq \mathcal{T}$ such that $F(\mathcal{T}') \leq F(\mathcal{W})$ for all $\mathcal{W} \subseteq \text{Dom}(A)$, where $|\mathcal{W}| = k$.

By using a well-behaved function we may concentrate on boundary points independent of whether the partition arity is limited a priori or not. Thus, one can be sure that good partitions can be chosen, no matter which splitting strategy is applied. Definition 3 constrains the search of the optimal binary partition to boundary points. We can draw a connection from useful to well-behaved functions as follows. If an useful evaluation function also is cumulative, then it is also well-behaved with respect to multiway partitions. The following theorem shows that in order to minimize the goodness score of a partition, with respect to any useful and cumulative evaluation function, it suffices to examine only the boundary points.

Theorem 2. *For any cumulative and useful evaluation function F there exists a partition of an arbitrary example set such that it minimizes the value of F and the corresponding cut points are boundary points.*

Proof: Let F be a cumulative and useful evaluation function and let S be an arbitrary example set. Moreover, let $\mathcal{S} = \biguplus_{i=1}^{k} S_i$ be a partition of S such that $F(\mathcal{S})$ is the minimum value of F. Let the thresholds $\{T_1, \ldots, T_{k-1}\}$ be the ones that define $\mathcal{S}$ (see figure 4).

It suffices to show that an arbitrary threshold, which is not a boundary point, can be either removed or replaced by a boundary point. Assume that value T_t in the set $\{T_1, \ldots, T_{k-1}\}$ is not a boundary point. Only the subsets S_t and S_{t+1} in the partition $\mathcal{S}$ depend on the choice of the threshold T_t (see figure 4). Therefore, only the terms $f(S_t)$ and $f(S_{t+1})$ in the cumulative formula $F(\mathcal{S}) = c \cdot \sum_{i=1}^{k} f(S_i)$, giving the total impurity of the partition, depend on the value T_t. Since $\mathcal{S}$ minimizes the total impurity of the partition and because F is cumulative, the impurity of $S_t \uplus S_{t+1}$ also has to be the least possible.

Because F is useful, in the boundary points for the example set $S_t \cup S_{t+1}$ there must exist a value T such that $F(T) \leq F(T_t)$. Substituting the boundary point T for T_t creates a new (possibly trivial) partition for $S_t \cup S_{t+1}$ that has at most the same impurity as the partition defined by T_t. Since T_t was chosen arbitrarily, the claim follows. □

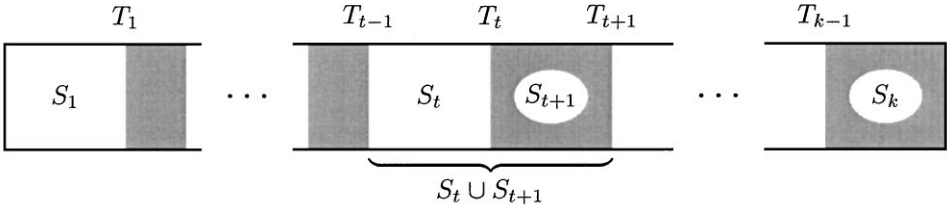

Figure 4. Illustration of the situation in the proof of Theorem 2.

A stronger version of Theorem 2, which states that a minimum impurity multisplit is always defined by boundary points, holds true for well-behaved evaluation functions that do not suffer from the degenerate case (for an example of such see Section 4.2). Nevertheless, even this weaker version allows us to focus on boundary points in searching for the best possible multiway partition. The theorem holds with or without an upper bound for the arity.

Theorem 2 offers only one way to show that a function is well-behaved; usefulness, of course, is a necessary condition for well-behavedness, but cumulativity is not. Indeed, we subsequently show that two non-cumulative functions behave well in multisplitting. The technique given by Theorem 2 is a very convenient one, because it directly implies the existence of an efficient optimization algorithm, as shown below. Moreover, it gives a way to utilize, in multisplitting, the (convexity) results that are known from the more extensively studied binarization scheme.

In any case, one of the optimal multiway partitions of the data has all its cut points at the boundary points if using a well-behaved evaluation function. Therefore, we can focus on the boundary points in searching for the optimal partition. In order to find the optimal k-ary partition of the given data, it suffices to find the $k-1$ boundary points that determine the partition with the lowest impurity. Since for a cumulative evaluation function the best k-partition of a subsample $S' \subset S$ does not depend on the splitting of its complement set $S \setminus S'$, the search strategy can be implemented incrementally by using dynamic programming. Fulton, Kasif, and Salzberg (1995) examined all potential cut points within the example sequence in order to obtain the optimal multisplit by a cumulative evaluation function. They gave the recurrence by which the impurities of the cut point candidates can be calculated from the impurities of shorter intervals and smaller-arity partitions. The computation entails obtaining impurities for lower arity partitions during the process. Hence, given a value k, the method can choose in time $O(kn^2)$ the partition with the lowest impurity from among all those that have at most k intervals (Fulton, Kasif, & Salzberg, 1995).

The same general scheme can also be utilized with (bins or) blocks of examples in place of individual examples. Instead of preparing to cut in between every pair of examples, we only need to inspect those positions where the cut points defining a partition may eventually be located. This can be accomplished by preprocessing the data in (bins or) blocks that are represented only by their class frequency distribution as described in the previous section. As the rationale for assigning examples into blocks shows, the linear order of examples is retained on those parts that matter. Using blocks as the basic processing unit also facilitates and speeds up the implementation and reduces the complexity of the required data structures (see Appendix A).

The base case for the impurity calculation is that if we "split" any part of the data trivially into one subset, then the impurity of that "partition" is directly determined by the impurity measure. When the data has been preprocessed into blocks, the recurrence for impurity calculation is:

$$\mathit{impur}(k, 1, j) = \begin{cases} \min\limits_{k-1 \le i < j} \{\mathit{impur}(k-1, 1, i) + \mathit{impur}(1, i+1, j)\} & \text{if } k \le j, \\ \infty & \text{otherwise,} \end{cases}$$

where $\mathit{impur}(k, i, j)$ denotes the minimum impurity that results when blocks i through j in the sorted sequence of examples are partitioned into k intervals. The best k-split is the

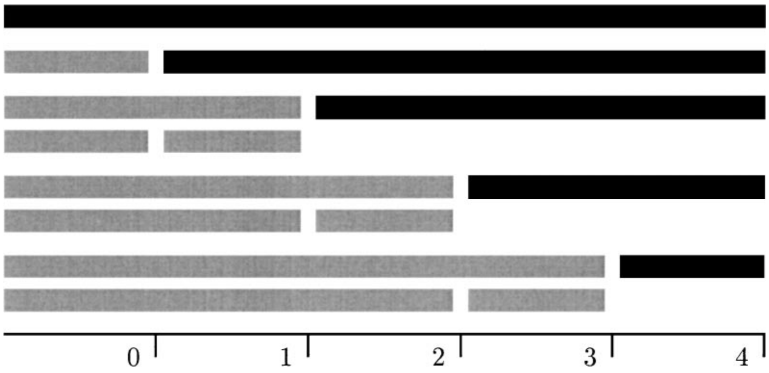

Figure 5. In searching for the best three-way split we only need to evaluate the suffixes of the data in order to recover the optimal splits. Only the black suffixes need to be evaluated when processing the fifth block of examples. The gray partitions have been evaluated previously and the best impurities have been stored. Only the value of the binary partition that evaluated as best has been stored. Combining the new impurities appropriately with the stored ones gives the impurity values for the best partitions of arities 1–3 for this part of the data.

one that minimizes $impur(k, 1, B)$. Using dynamic programming, the time requirement of finding the optimal partitioning into at most k intervals is $O(kB^2)$, where B is the number of blocks in the range. The theoretical worst case is $B = n$, but most typically in practice $B \ll n$ (see Appendix B). The search method is presented in a more algorithmic form in Appendix A. The detailed time- and space-efficiency analysis is also given there.

Example. The efficient impurity calculation requires making use of incremental computation, which can be implemented by dynamic programming. Consider partitioning the example set in figure 3 into (at most) three intervals. Let us illustrate how dynamic programming makes the search efficient. Consider the situation when processing the fifth block of the data. Assume that the best partitions of arities one and two have been evaluated and stored for all prefixes of the data consisting of the first four blocks. The fact that only the best partitions have been stored is depicted in figure 5 by drawing only one binary partition of each prefix, even if there are more alternatives.

In order to obtain the impurities for the best partitions of arity at most three for the extended data set, we only need to evaluate the suffixes of the data corresponding to the rightmost interval of the extended partition and then combine these results appropriately with the best ones obtained earlier for all prefixes of the data (see figure 5). At the end, the optimal partition is the best partition of the whole data with arity 1, 2, or 3.

4. On the well-behavedness of important evaluation functions

From the usefulness of the Average Class Entropy it follows that also the Information Gain function (Quinlan, 1983) is useful (Fayyad & Irani, 1992b). Using our earlier notation the Information Gain can be expressed as

$$IG\left(\biguplus S_i\right) = H(S) - ACE\left(\biguplus S_i\right),$$

where $H(S)$ is the class entropy of sample S prior to partitioning and $ACE(\biguplus S_i)$ is the Average Class Entropy of S, when partitioned by the value of an attribute. The attribute that gains the most increase in information is selected. $H(S)$ is invariant with respect to the attributes and the partitions induced by them. Thus, the objective is, in essence, to maximize the negation of Average Class Entropy. Hence, the well-behavedness of *IG* follows from that of *ACE*.

The well-behavedness of some other important attribute evaluation functions is also clear from earlier results. For example, the *Gini Index* (of diversity) or the *Quadratic Entropy* (Breiman et al., 1984; Breiman, 1996) is defined as

$$GI\left(\biguplus_i S_i\right) = \sum_i \frac{|S_i|}{|S|} gini(S_i),$$

in which *gini* is the impurity measure $gini(S) = -\sum_{j=1}^{m} P(C_j, S)(1 - P(C_j, S))$, where $P(C, S)$ denotes the proportion of instances of class C in the data S. The evaluation function *GI* is known to be convex and, thus, also useful. Furthermore, since this evaluation function is in addition cumulative, its well-behavedness follows by Theorem 2. There are several other evaluation functions that are based on the entropy function (see e.g., Mingers, 1989; Kononenko, 1995). Not all of them, however, are necessarily well-behaved.

The following subsections probe into the extent of the class of well-behaved evaluation functions. First, we analyze two non-cumulative evaluation functions, the *Gain Ratio* (Quinlan, 1986) and the *Normalized Distance Measure* (López de Màntaras, 1991). Then we utilize Theorem 2 in proving the well-behavedness of two cumulative functions. In the proofs we will treat the evaluation functions and their component functions as continuous and twice differentiable, even though they are defined to be discrete. Observe that this causes no harm, since we only consider proving the *absence* of certain local extremas.

Recall that the only possible cut points of the data are the points in between the bins; no partition can occur within a bin. When the bins are further combined into blocks, mixed bins remain as separate blocks, but consecutive uniform bins of the same class get combined into a common block. Thus, the only possible cut points in addition to boundary points are within uniform blocks. Therefore, when examining the usefulness or well-behavedness of an evaluation function, it is enough to show that the function does not have an optimum within uniform blocks (Fayyad & Irani, 1992b).

4.1. Evaluation functions of C4.5

The Information Gain function does not penalize a partition for having a large arity. Therefore, it favors excessively multi-valued nominal attributes and multisplitting numerical attribute value ranges. To correct this deficiency Quinlan (1986) suggested dividing the *IG* score of a partition by the term

$$\kappa\left(\biguplus_i S_i\right) = -\sum_i \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

The resulting evaluation function is known as the *Gain Ratio*

$$GR\left(\biguplus S_i\right) = IG\left(\biguplus S_i\right) \Big/ \kappa\left(\biguplus S_i\right).$$

As with *IG*, the intent is to maximize this function. Notice that *GR* is not a cumulative evaluation function, because the value of the coefficient $1/\kappa(\biguplus S_i)$ depends on the chosen partition by referring to the sizes of the partition subsets.

C4.5 incorporates the Gain Ratio as the default choice for attribute evaluation. It has been observed to have some difficulties in particular in connection with numerical attributes. In order to overcome those problems López de Màntaras (1991) has proposed to use another, but closely related evaluation function, and Quinlan (1996) has recently been compelled to change the evaluation of numerical attributes in C4.5. Still, analysis of these functions has been surprisingly scarce (López de Màntaras, 1991; Dietterich, Kearns, & Mansour, 1996). In the following we study the convexity and well-behavedness of these functions.

Theorem 3. *The evaluation function Gain Ratio is not convex.*

Proof: Consider the data in figure 6. For each example its class label (A or B) and the value of a numerical (integer in this case) attribute is shown. There are three boundary points in this data, $BP_1, \ldots, BP_3$, and one further possible cut point, *CP*. Let $g(p)$, $0 \leq p \leq 1$, denote the Gain Ratio of a binary partition in which a fraction p of the examples belongs to the left subset and the rest of the data to the right one.

In order to see that Gain Ratio is not convex, the behavior of its second derivative g'' was inspected on the leftmost block (two bins) consisting of instances of the class A. It was seen that in the Gain Ratio curve there is an inflection point $x \approx 0.12468$ such that

$$g''(y) = \begin{cases} <0 & \text{when } y < x, \\ 0 & \text{when } y = x, \text{ and} \\ >0 & \text{when } x < y. \end{cases}$$

Since $0 < x < 3/11 \approx 0.27273$, this example shows that the Gain Ratio function is not convex over the first block. □

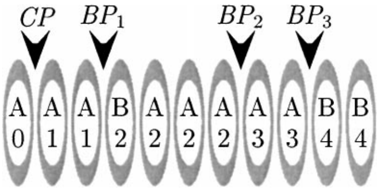

Figure 6. An example set ordered by the value of an integer-valued attribute. The examples are members of two classes, *A* and *B*. There are three boundary points and a further possible cut point in this data along the chosen dimension.

Although convexity is a desirable property, the lack thereof does not imply that a function would not be useful. Elomaa and Rousu (1997) gave an explicit usefulness proof in earlier work; it utilizes the same proof techniques as the following proof. Independently from our research, Codrington and Brodley (1999) have obtained essentially the same result. They have also examined the usefulness of evaluation functions more generally and have formalized the properties which make evaluation functions, with a similar rational formula as the Gain Ratio, exhibit useful behavior.[1] It turns out that the properties are just the ones utilized in the following proof. Keep in mind that the intent is to maximize the Gain Ratio rather than to minimize it.

Theorem 4. *The Gain Ratio optimal partitions are defined on boundary points.*

Proof: Let $\mathcal{S} = \biguplus_{i=1}^{h} S_i$, $h \le k$, be a partition of an arbitrary example set S along the dimension of a numerical attribute A. Assume that $\mathcal{S}$ is a gain-ratio-optimal partition of S among those partitions that have arity of at most k, $k \ge 1$. If $h = 1$ there is nothing to prove, therefore, we assume that $h > 1$. Let $\mathcal{T} = \{T_1, \ldots, T_{h-1}\}$ be the set of cut points that defines $\mathcal{S}$. It is enough to show that if in $\mathcal{T}$ there is a threshold, which is not a boundary point, then we can slide the threshold onto a boundary point without decreasing the Gain Ratio. Therefore, assume that threshold T_t, $1 \le t \le h-1$, is not a boundary point (see figure 7).

Let us define two further points from the sorted example sequence S. The cut point D is the lower border (a boundary point) of the uniform block which contains T_t, if the border is positioned in between the thresholds T_{t-1} and T_t. Otherwise, if cut point T_{t-1} is within the same block as T_t, we define D to be T_{t-1}. Similarly, we define U to be either the boundary point which is the upper border of the uniform block that contains T_t or the threshold T_{t+1}. In any case, there are only instances of one class in between the cut points D and U.

The cut points T_{t-1}, D, U, and T_{t+1} induce the partition $X \uplus P \uplus Q \uplus R \uplus Y$ of the data such that $X = \{s \in S \mid \mathrm{val}_A(s) \le T_{t-1}\}$, $P = \{s \in S \mid T_{t-1} < \mathrm{val}_A(s) \le D\}$, $Q = \{s \in S \mid D < \mathrm{val}_A(s) \le U\}$, $R = \{s \in S \mid U < \mathrm{val}_A(s) \le T_{t+1}\}$, and $Y = \{s \in S \mid \mathrm{val}_A(s) > T_{t+1}\}$. All that follows applies also in the case where any of the sets X, P, R, and Y is an empty set. Let b, $b \in \{1, \ldots, m\}$, be the unique class of the examples in Q, and

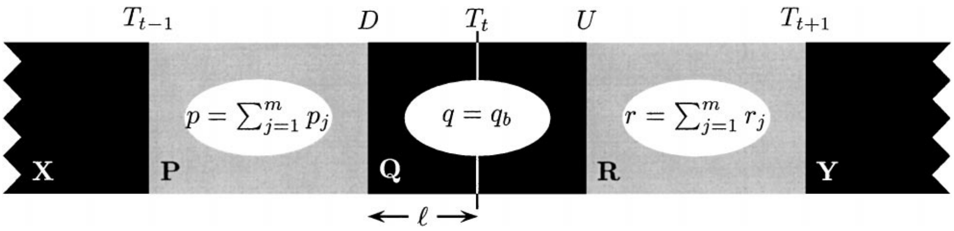

Figure 7. An illustration of the situation in the proof of Theorem 4. The cut points T_{t-1}, D, U, and T_{t+1} induce the partition $X \uplus P \uplus Q \uplus R \uplus Y$ of the data. There are only instances of one class, b, in Q. The proof examines what are the consequences of positioning, within the interval Q, the tth threshold of a multiway partition of the data.

let $p = |P|$, $q = |Q|$, and $r = |R|$. Moreover, by p_j we denote the number of instances of class j, $j \in \{1, \dots, m\}$, in the set P, r_j is the respective number for R (see figure 7). In particular, p_b and r_b are the numbers of instances of class b in the sets P and R, respectively.

We consider what happens to the value of the Gain Ratio if the threshold T_t is shifted within the interval Q. Therefore, we review the case where the threshold T_t is positioned in Q after the ℓth instance, where $0 < \ell < q$. That leaves $q - \ell$ elements of Q to the right side of T_t. By *GR*(ℓ), *IG*(ℓ), *ACE*(ℓ), and $\kappa(\ell)$ we denote the related function values corresponding to this situation. Furthermore, $M(j, S)$ is the number of instances of class j in the set S.

Let us define that

$$\begin{aligned} E(\ell) &= \frac{1}{|S|}\Bigg(|S_t|\log_2|S_t| + |S_{t+1}|\log_2|S_{t+1}| \\ &\qquad - \sum_{j=1}^{m}(M(j,S_t)\log_2 M(j,S_t) + M(j,S_{t+1})\log_2 M(j,S_{t+1}))\Bigg) \\ &= \frac{1}{|S|}\Bigg((p+\ell)\log_2(p+\ell) + (r+q-\ell)\log_2(r+q-\ell) - (p_b+\ell) \\ &\qquad \cdot \log_2(p_b+\ell) - (r_b+q-\ell)\log_2(r_b+q-\ell) \\ &\qquad - \sum_{j\neq b}(p_j\log_2 p_j + r_j\log_2 r_j)\Bigg), \end{aligned}$$

which is the part of the numerator of *GR* formula whose value depends on the exact location of the threshold T_t in Q. Moving T_t within the uniform interval Q only affects these values. Similarly in the denominator κ of the *GR* formula only two terms are affected by the positioning of T_t; let us define

$$\begin{aligned} K(\ell) &= \frac{1}{|S|}((|S_t| + |S_{t+1}|)\log_2|S| - |S_t|\log_2|S_t| - |S_{t+1}|\log_2|S_{t+1}|) \\ &= \frac{1}{|S|}((p+q+r)\log_2|S| - (p+\ell)\log_2(p+\ell) \\ &\quad - (r+q-\ell)\log_2(r+q-\ell)). \end{aligned}$$

We can now write out, using $E(\ell)$, the Information Gain of the partition $\mathcal{S}$, when T_t is placed after the ℓth example of Q:

$$IG(\ell) = H(S) - ACE(\ell) = H(S) - \left(\sum_{i=1}^{t-1}\frac{|S_i|}{|S|}H(S_i) + E(\ell) + \sum_{i=t+2}^{h}\frac{|S_i|}{|S|}H(S_i)\right).$$

Similarly, the denominator κ can be rewritten as

$$\kappa(\ell) = -\sum_{i=1}^{t-1} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} + K(\ell) - \sum_{i=t+2}^{h} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}.$$

Taking the first derivative of IG with respect to ℓ, all the constant terms that do not depend on the value of ℓ differentiate to zero. Hence,

$$IG'(\ell) = \frac{d}{d\ell} IG(\ell) = -E'(\ell).$$

Therefore, the second derivative is $IG''(\ell) = -E''(\ell)$. The same happens with the derivative of $\kappa(\ell)$, i.e., $\kappa'(\ell) = K'(\ell)$ and $\kappa''(\ell) = K''(\ell)$. These derivatives are not affected by the data in the subsets X and Y, except by summing their elements to the total number of the examples.

The first derivative of $GR(\ell)$ is given by

$$GR'(\ell) = \frac{d}{d\ell} GR(\ell) = \frac{IG'(\ell)\kappa(\ell) - \kappa'(\ell)IG(\ell)}{\kappa^2(\ell)}.$$

Let us define $N(\ell) = IG'(\ell)\kappa(\ell) - \kappa'(\ell)IG(\ell)$, and note that

$$\begin{aligned} N'(\ell) &= IG''(\ell)\kappa(\ell) + \kappa'(\ell)IG'(\ell) - \kappa''(\ell)IG(\ell) - \kappa'(\ell)IG'(\ell) \\ &= -E''(\ell)\kappa(\ell) - K''(\ell)IG(\ell). \end{aligned}$$

Since for each $0 < \ell < q$ it holds, by definition, that $\kappa(\ell) > 0$ and $IG(\ell) \geq 0$, we only have to check that $-E''(\ell) \geq 0$ and $K''(\ell) < 0$ in order to see that $N'(\ell) \geq 0$: The second derivative of $E(\ell)$ is

$$E''(\ell) = \frac{1}{|S|}\left(\frac{1}{p+\ell} - \frac{1}{p_b+\ell} + \frac{1}{r+q-\ell} - \frac{1}{r_b+q-\ell} \right) \leq 0,$$

because $p_b \leq p$ and $r_b \leq r$. Hence, $-E''(\ell) \geq 0$. Furthermore,

$$K''(\ell) = \frac{1}{|S|}\left(\frac{-1}{p+\ell} + \frac{-1}{r+q-\ell} \right) < 0.$$

Using the shorthand notation the second derivative of $GR(\ell)$ is expressed by

$$GR''(\ell) = \frac{d}{d\ell} GR'(\ell) = \frac{d}{d\ell} \frac{N(\ell)}{\kappa^2(\ell)} = \frac{N'(\ell)\kappa^2(\ell) - 2\kappa(\ell)\kappa'(\ell)N(\ell)}{\kappa^4(\ell)}.$$

Let, now, $\psi \in]0, q[$ be a potential location for a local maximum, i.e., such a point that $GR'(\psi) = 0$. Then also $N(\psi) = 0$ and the expression for $GR''(\psi)$ is further simplified to

$$GR''(\psi) = N'(\psi)/\kappa^2(\psi),$$

which is larger than zero because $N'(\psi) \geq 0$ and $\kappa^2(\psi) > 0$. In other words, $GR(\psi)$ is not a local maximum. Since ψ was chosen arbitrarily, we have shown that $GR(\ell)$ can only obtain its maximum value when the threshold T_t is placed at either of the cut points D and U, where $\ell = 0$ and $\ell = q$, respectively. □

Thus, the Gain Ratio function is neither convex nor cumulative, but it still behaves well. However, the non-cumulativity of the Gain Ratio means that no efficient evaluation scheme is known for this function.

The above analysis also applies to the *Normalized Distance Measure*, *ND*, proposed by López de Màntaras (1991) as an alternative to the Information Gain and Gain Ratio functions. The measure can be expressed with the help of the Information Gain as

$$ND\Big(\biguplus S_i\Big) = 1 - IG\Big(\biguplus S_i\Big)\Big/\lambda\Big(\biguplus S_i\Big),$$

where

$$\lambda\left(\biguplus_{i=1}^{k} S_i\right) = -\sum_{i=1}^{k}\sum_{j=1}^{m} \frac{M(j, S_i)}{|S|}\log_2\frac{M(j, S_i)}{|S|},$$

in which k is the number of intervals and m is the number of classes, as in the preceding notation, and $M(j, S)$ stands for the number of instances of class j in the set S. The intent is to minimize the value of distance $ND(\biguplus S_i) \in [0, 1]$ or, equally, maximize the value of $1 - ND(\biguplus S_i)$. Since this has a similar form to the definition of the Gain Ratio function, a similar proof shows that also *ND* behaves well and the same counterexample serves to prove its non-convexity.

Theorem 5. *The evaluation function Normalized Distance Measure is not convex.*

Proof: The Normalized Distance Measure has an inflection point $x \approx 0.13192$ over the data in figure 6, such that its second derivative turns from negative to positive in the point x. Hence, the claim follows. □

Theorem 6. *The optimal Normalized Distance Measure partitions are defined on boundary points.*

The proof of Theorem 6 is almost identical to that of Theorem 4. Thus, we omit it here. The outline for the proof is the following. We prove instead that the claim holds for

$$ND_1\Big(\biguplus S_i\Big) = 1 - ND\Big(\biguplus S_i\Big) = IG\Big(\biguplus S_i\Big)\Big/\lambda\Big(\biguplus S_i\Big),$$

from which it directly follows that *ND* behaves well. In order to prove that optimal ND_1 partitions are defined on boundary points, we consider the same situation as in the proof of Theorem 4 (see figure 7) and examine what happens to the ND_1 value of the optimal partition when a threshold, which is not a boundary point, is shifted within an uniform interval. By decomposing the numerator and the denominator of ND_1 formula in this situation and differentiating them twice, we can show that the uniform interval does not contain a local maximum, which renders the threshold, which is not a boundary point, superfluous.

Example. With the functions *GR* and *ND* we can demonstrate one property of well-behavedness. It does not guarantee that an optimal partition exists on boundary points regardless of the the arity. Neither of these well-behaving functions does have the optimal three-way partition defined on boundary points for the example set shown in figure 6. Both of them rank the partition defined by the cut point *CP* and boundary point BP_3 as the best among those with arity three.

However, since both *GR* and *ND* are well-behaved, they give a better score to a binary partition of the data than to the best partition of arity three. The Gain Ratio of the best three-way partition is 0.411 and that of the best binary partition 0.634 (defined by boundary point BP_3). The respective figures for *ND* are 0.396 and 0.698.

4.2. A MDL/MML cost function

In machine learning research based on the *Minimum Description Length Principle*, MDL (Rissanen, 1989), or the *Minimum Message Length Principle*, MML (Wallace & Freeman, 1987), the central theme is coding of examples. Rissanen (1989, 1995), Quinlan and Rivest (1989) as well as Wallace and Patrick (1993) have explored MDL/MML-based decision tree learning. The intent is to minimize the coding length of examples, hence, there is a natural cost function for evaluating attributes: the attribute, which gives least rise to the total coding length of examples, is chosen to the evolving tree.

In the incremental exception coding scheme of Wallace and Patrick (1993) an instance's class is encoded on the fly while going through the sample. If the instance under consideration is a member of class j, $1 \leq j \leq m$, of which there have been n_j observations out of the total n examples seen so far, then we use

$$-\log_2 \frac{n_j + \alpha}{n + m\alpha}$$

bits to encode it. The parameter $\alpha > 0$ ensures that all classes have a positive probability, and biases, depending on its value, for or against impurity in the resulting intervals. Subsequently we refer to this method by the name *WP*.

The coding scheme *WP* falls into the category of adaptive arithmetic data compression methods (Witten, Neal, & Cleary, 1987). Given the class frequency distribution for a sample S of n examples, we can compute its exact code length (Howard & Vitter, 1992):

$$WP(S, \alpha) = \log_2 \frac{(m\alpha)^{\bar{n}}}{\prod_{j=1}^{m} \alpha^{\bar{n}_j}},$$

where $a^{\bar{b}}$ is a shorthand notation for the increasing power $a(a+1)\cdots(a+b-1)$.

A natural way of defining the MDL/MML cost of a k-split is

$$WP_{\text{cost}}\left(\biguplus_{i=1}^{k} S_i\right) = \log_2(V-1) + \log_2\binom{V-1}{k-1} + \sum_{i=1}^{k} WP(S_i, \alpha).$$

The two first terms define the cost of transmitting the positions of the $k-1$ cut points. As before, V is the number of unique values for the numerical attribute.

Theorem 7. *The cost function WP_{cost} is useful.*

Proof: Since the two first terms in the formula for WP_{cost} do not depend on the positions of the cut points, in order to prove the usefulness of the function, it is enough to show that the coding scheme $WP(S, \alpha)$ in the third term behaves usefully. In order to do that, it suffices to show the useful behavior of the coding cost $WP(S_0, \alpha) + WP(S_1, \alpha)$ of a binary partition $S_0 \uplus S_1$ of a data set S.

It is sufficient to show that WP_{cost} is useful in between two consecutive boundary points T_1 and T_2, such that in the sample S, when ordered by the value of a numerical attribute A, there are only instances of one class in between them. These two thresholds induce a tripartition $P \uplus Q \uplus R$ of the sample (see figure 8):

$$\begin{aligned} P &= \{s \in S \mid \text{val}_A(s) \le T_1\}, \\ Q &= \{s \in S \mid T_1 < \text{val}_A(s) \le T_2\}, \quad \text{and} \\ R &= \{s \in S \mid T_2 < \text{val}_A(s)\}. \end{aligned}$$

Observe that R, or in the degenerate case even both P and R, may be an empty set. All that follows applies in those cases, too.

For all classes j, $j \in \{1, \ldots, m\}$, let p_j denote the number of instances in P of class j and let $p = \sum_{j=1}^{m} p_j$. Respectively, define $r_1, \ldots, r_m$ and r for the numbers of instances in R. By assumption, all instances in Q belong to the same class. Hence, $q = q_b$ for some $b \in \{1, \ldots, m\}$.

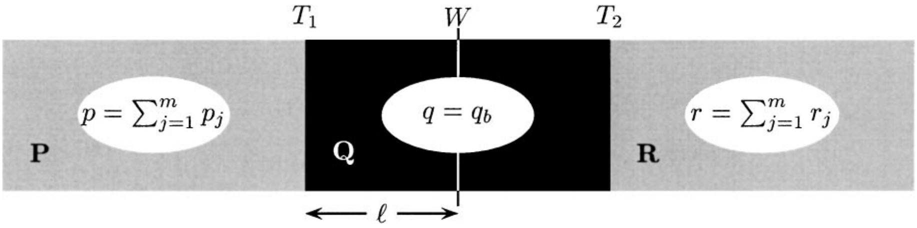

Figure 8. Illustration of the situation in the proofs of Theorems 7 and 8. Boundary points T_1 and T_2 induce a tripartition $P \uplus Q \uplus R$ of the data such that there are only instances of the class b in Q. The proofs examine what are the consequences of partitioning the data on a threshold W, $T_1 \le W \le T_2$, which has ℓ members of Q to its left side.

If $q \leq 1$, the proof is trivial, so subsequently we assume that $q > 1$. Now, consider the partition $S_0 \uplus S_1$ of S induced by a point W, such that $T_1 \leq W \leq T_2$. Let there be ℓ instances $s \in Q$ such that $\text{val}_A(s) \leq W$, which leaves $q - \ell$ examples of Q in the range $W < \text{val}_A(s) \leq T_2$. If we move the cut point one example to the right from W, the cost of coding S_0 increases by

$$-\log_2 \frac{(p_b + \ell) + \alpha}{(p + \ell) + m\alpha},$$

and the cost of S_1 decreases by

$$-\log_2 \frac{(r_b + (q - \ell - 1)) + \alpha}{(r + (q - \ell - 1)) + m\alpha}.$$

Hence, the net change in the total cost of encoding $WP(S_0, \alpha) + WP(S_1, \alpha)$ is

$$\Delta_\ell = \log_2 \frac{p + \ell + m\alpha}{p_b + \ell + \alpha} - \log_2 \frac{r + q - \ell - 1 + m\alpha}{r_b + q - \ell - 1 + \alpha}.$$

No harm happens in taking Δ_ℓ to be a differentiable function. The first derivative of Δ_ℓ with respect to ℓ is

$$\begin{aligned} \Delta'_\ell &= \frac{\alpha + p_b - m\alpha - p}{(m\alpha + p + \ell)(\alpha + p_b + \ell)} + \frac{\alpha + r_b - m\alpha - r}{(\alpha + r_b + q - \ell - 1)(m\alpha + r + q - \ell - 1)} \\ &= \frac{(1 - m)\alpha + p_b - p}{(m\alpha + p + \ell)(\alpha + p_b + \ell)} + \frac{(1 - m)\alpha + r_b - r}{(\alpha + r_b + q - \ell - 1)(m\alpha + r + q - \ell - 1)}. \end{aligned}$$

Since $m \geq 2$, $p_b \leq p$, and $r_b \leq r$, the numerators of both terms in Δ'_ℓ are negative and their denominators are positive when $0 \leq \ell \leq q$. Hence, the value of Δ_ℓ decreases as ℓ grows. Thus, in between T_1 and T_2 the cost function WP_{cost} is useful and since the two boundary points were arbitrarily selected, the claim follows. □

The cost function WP_{cost} is cumulative because its value for a partition is obtained by summation over the encodings of subsets. Therefore, its well-behavedness follows from the previous result by Theorem 2.

Corollary 1. *The cost function WP_{cost} is well-behaved.*

This and other MDL/MML cost functions are interesting candidates as evaluation functions for multisplitting, since they inherently penalize the growth of the arity of the partition (Fayyad & Irani, 1993; Quinlan, 1996). For WP_{cost} needless binarization of a subsample, which contains only members of one class, always incurs an increase in cost. Our earlier experience (Rousu, 1996), however, lets us conjecture that drastic improvements in prediction accuracy are not to be expected by using these techniques. Similar empirical evidence has been reported by Quinlan and Rivest (1989) as well as Wallace and Patrick (1993).

4.3. *Minimum training set error*

In learning decision trees of limited depth—e.g., one- and two-level decision trees (Landeweerd et al., 1983; Iba & Langley, 1992; Holte, 1993; Elomaa, 1994; Maass, 1994; Auer, Holte, & Maass, 1995)—repetitive binary splitting is not a viable way of partitioning numerical value ranges. For instance, the T2 algorithm (Auer, Holte, & Maass, 1995) exhaustively searches for the optimal decision tree of depth at most two. The process entails optimal multisplitting of the numerical domains. The evaluation function being optimized is *Training Set Error, TSE*: the number of training instances falsely classified by the decision tree.

Let S be a set of examples and C the class attribute. The number of *disagreeing* instances, those in the set S not belonging to its majority class, is given by

$$\delta(S) = |\{s \in S \mid \mathrm{val}_C(s) \neq \mathrm{maj}_C(S)\}|.$$

Training set error of a partition $\biguplus S_i$ of S can now be defined as the cumulative sum

$$TSE\left(\biguplus_i S_i\right) = \sum_i \delta(S_i).$$

Theorem 8. *The evaluation function TSE is useful.*

Proof: Let the basic setting be the same as in the previous usefulness proof (see figure 8). Boundary points T_1 and T_2, which have only instances of one class in between them, define a tripartition of the sample: $S = P \uplus Q \uplus R$. Again, let W, $T_1 \leq W \leq T_2$, be a threshold with ℓ examples in Q to the left of it and $q - \ell$ to the right. Let $S_0 \uplus S_1$ be the binary partition of S induced by W.

Let C be the class attribute and let $a = \mathrm{maj}_C(P)$, $b = \mathrm{maj}_C(Q)$, and $c = \mathrm{maj}_C(R)$, $a, b, c \in \{1, \ldots, m\}$. We define that $w_0 = p_a - p_b$. Intuitively, $w_0 + 1$ is the minimum number of examples from the set Q that have to be to the left of the threshold W before b becomes the unique most frequent class in the set S_0. Respectively, we define $w_1 = q - (r_c - r_b)$ for the set S_1: while there are more than $q - w_1$ (resp. less than w_1) elements of Q to the right (resp. to the left) of the threshold W, b will be the unique majority class in the set S_1.

Let us now consider the numbers of disagreeing examples in the sets S_0 and S_1. While the number of examples in Q to the left of the threshold W, ℓ, is below value w_0, the majority class of P, a, continues to be the majority class of S_0. All the ℓ members of the set Q belonging to S_0 have to be counted as disagreeing instances, since they do not belong to the majority class. Therefore, the number of disagreeing examples is $\delta(P) + \ell$. Once there are (strictly) more than w_0 examples of Q to the left of W, b becomes the (unique) majority class of S_0 and the number of the disagreements ceases to increase, because all members of Q now belong to the majority class. To see what is the number of the disagreements then, observe that when $\ell = w_0$, there are exactly as many instances of classes a and b in the set S_0. Thus, the numbers of the disagreements with these classes also have to be equal. At this point, the number of the disagreements with class a, obviously, is $\delta(P) + w_0$. Since, from that situation on, no new disagreements (with the new majority class b) arise, it must be the number of the disagreements also when $\ell > w_0$. Hence, the following holds

$$\delta(S_0) = \begin{cases} \delta(P) + \ell & \text{if } \ell \leq w_0, \text{ and} \\ \delta(P) + w_0 & \text{if } \ell > w_0. \end{cases}$$

If $a = b$, then $w_0 = 0$ and $\delta(S_0)$ reduces to $\delta(P)$.

The corresponding equation for S_1 is

$$\delta(S_1) = \begin{cases} \delta(R) + q - w_1 & \text{if } \ell < w_1, \text{ and} \\ \delta(R) + q - \ell & \text{if } \ell \geq w_1. \end{cases}$$

Notice that if P is empty, then $\delta(P) = w_0 = 0$ and $\delta(S_0)$ also reduces to value 0. Respectively, if R is empty, then $w_1 = q$ and $\delta(S_1) = 0$. Thus, these equations also hold if there are no natural boundary points in the value range.

Depending on the relation between w_0 and w_1, the total number of errors is derived differently, although the resulting equations, as we will see, are very much alike. In the following let d be a constant such that $d = \delta(P) + \delta(R) + q$. We have two cases:

a) If $w_0 \leq w_1$, combination of the equations for $\delta(S_0)$ and $\delta(S_1)$ leads to

$$\begin{aligned} TSE(S_0 \uplus S_1) &= \begin{cases} \delta(P) + \ell + \delta(R) + q - w_1 & \text{if } \ell \leq w_0, \\ \delta(P) + w_0 + \delta(R) + q - w_1 & \text{if } w_0 < \ell < w_1, \text{ and} \\ \delta(P) + w_0 + \delta(R) + q - \ell & \text{if } w_1 \leq \ell. \end{cases} \\ &= \begin{cases} d + \ell - w_1 & \text{if } \ell \leq w_0, \\ d + w_0 - w_1 & \text{if } w_0 < \ell < w_1, \text{ and} \\ d + w_0 - \ell & \text{if } w_1 \leq \ell. \end{cases} \end{aligned}$$

Since ℓ is the only variable in these equations, the training set error is monotonically increasing for $\ell \leq w_0$, stays constant for $w_0 < \ell < w_1$, and is monotonically decreasing for $w_1 \leq \ell$. Hence, $TSE(S_0 \uplus S_1)$ receives its minimum value when ℓ has either value 0 or q. In this case the value of TSE minimizes at a boundary point.

b) If $w_0 > w_1$, we obtain

$$\begin{aligned} TSE(S_0 \uplus S_1) &= \begin{cases} \delta(P) + \ell + \delta(R) + q - w_1 & \text{if } \ell < w_1, \\ \delta(P) + \ell + \delta(R) + q - \ell & \text{if } w_1 \leq \ell \leq w_0, \text{ and} \\ \delta(P) + w_0 + \delta(R) + q - \ell & \text{if } w_0 < \ell. \end{cases} \\ &= \begin{cases} d + \ell - w_1 & \text{if } \ell < w_1, \\ d & \text{if } w_1 \leq \ell \leq w_0, \text{ and} \\ d + w_0 - \ell & \text{if } w_0 < \ell. \end{cases} \end{aligned}$$

The function is increasing for $\ell < w_1$, stays constant for $w_1 \leq \ell \leq w_0$, and decreases monotonically for $w_0 < \ell$. Again, the minimum value of $\mathit{TSE}(S_0 \uplus S_1)$ is obtained in one of the boundary points T_1 and T_2.

In any case $\mathit{TSE}(S_0 \uplus S_1)$ is useful in between boundary points T_1 and T_2. Since the two boundary points were arbitrarily selected, the claim follows. □

Again, from the cumulativity of *TSE* and the previous result it follows, by Theorem 2, that it is well-behaved as well.

Corollary 2. *The evaluation function TSE is well-behaved.*

The quadratic-time optimization algorithm is not the fastest one for this simple evaluation function; several authors have devised linear-time optimization algorithms for *TSE*. In addition to proposing the general, quadratic-time evaluation scheme for cumulative functions, Fulton, Kasif, and Salzberg (1995) put forward a special solution for computing optimal *TSE* partitions for two-class problems in time $O(kn)$ per attribute. Birkendorf (1997) has presented an algorithm that attains the same efficiency. His algorithm, however, has the additional advantage of being dynamic in the sense that it does not require the data to be preprocessed into the ascending order prior to partitioning. In multiclass problems the time requirement is $O(kmn)$, but the space requirement can still be kept low (Auer, 1997). A practical improvement to these linear-time algorithms can be obtained by focusing on boundary points.

The reason for the easy evaluability of *TSE* lies in the fact that, when processing a new block of examples, it suffices only to count the number of the disagreements (with respect to each class) therein and add that number with the disagreement figures obtained earlier. No part of the previously processed data needs to be retouched, because the training set error can only monotonically increase through the addition of the new block. Hence, a single pass through the data manages to reveal the optimal *TSE* partition.

Brodley (1995) and Lubinsky (1995) have both suggested using a mixed strategy, where another (entropy-based) evaluation function is first used in attribute selection high up in the decision tree, but the evaluation is dynamically switched over to *TSE* when the fringe of the decision tree is approached. This strategy has been reported in some cases to produce better results than using either of the evaluation functions throughout the tree growing process. If, in the mixed strategy in addition to *TSE*, another (well-behaved and) cumulative evaluation function is used, then this strategy is easy to implement using the dynamic programming approach. The learning algorithm can be parametrized by the evaluation function, no changes to the actual algorithm are required due to using two separate evaluation functions.

5. Empirical evaluation

We report on two series of empirical tests and explore the properties of commonly used data sets. The first tests contrast two versions of the optimal partitioning strategy with a variant of the greedy multisplitting approach (Catlett, 1991; Fayyad & Irani, 1993) and a random partitioning strategy. In this experiment we use stand-alone versions of the strategies. The second series of tests examines the impact that numerical attribute handling has on decision tree learning. We incorporate the optimal and the greedy multisplitting strategies into the C4.5 (release 5) decision tree learner and compare the results obtained by them with those that are produced by using C4.5's built-in binarization approach.

The contrasted splitting strategies and their implementations are:

Optimal: The dynamic programming implementation of the optimal multisplitting strategy, whose rationale was given in Section 3. As the basic processing units we use bins and blocks of examples. Both versions of Optimal produce optimal partitions; they differ in processing efficiency. We want to test what is, in practice, the advantage gained in the processing speed by inspecting only boundary points in lieu of examining all possible thresholds.

Greedy: An implementation of the greedy top-down multisplitting strategy using breadth-first search. The algorithm constructs a k-split of the data by selecting one interval of the $k-1$-split from the previous iteration to be partitioned into two subintervals. The global quality of the resulting k-split is used as the selection criteria. This procedure is repeated for arities 2 to 10 and the partition that evaluates as the best is chosen. Of this strategy, too, we test two versions; one that utilizes boundary points and another that does not.

Random: This strategy randomly allots $1, 2, \ldots, 9$ thresholds, which define partitions of arities $2, 3, \ldots, 10$, evaluates these, and submits the best one for further comparison with partitions induced by other attributes. Again two versions of this strategy are tested; one where the thresholds are drawn from among boundary points and another that draws them from among all possible values.

Binary: The binarization of a numerical domain as implemented by C4.5. This strategy is inherently intertwined with the decision tree construction. Therefore, it cannot be taken into account in our first test series.

We test these strategies using three evaluation functions:

IG: The Information Gain function as implemented by the C4.5 algorithm.

GR: The Gain Ratio function as implemented by the C4.5 algorithm. Since *GR* is not cumulative, it cannot be optimized as a whole using our dynamic programming approach. Instead, we use the following scheme: for each arity, we search for the *IG* optimizing multisplit and calculate its Gain Ratio. The split whose Gain Ratio evaluates the best is then selected.

BG$_{\text{log}}$: A straightforward attempt to balance the bias of *IG*, which favors multisplits excessively. "Balanced Gain," BG_{log} (Kononenko, Bratko, & Roškar, 1984), assigns a cost to the increase of the arity of a partition: for a k-ary multisplit $BG_{\text{log}}(\biguplus_{i=1}^{k} S_i) = IG(\biguplus_{i=1}^{k} S_i)/\log_2 k$. The well-behavedness of this function is clear. This method is only used for numerical attribute evaluation, C4.5's default evaluation function *GR* is used for nominal attributes in the tests involving decision tree generation. To see that BG_{log} is closely related to the Gain Ratio, observe that the denominator κ in the formula of *GR* is the entropy function H applied to the sizes of the intervals, not to the class distribution.

Hence, $0 < \kappa \leq \log_2 k$. In information theoretical sense $\log_2 k$ is the entropy of the intervals when assigned equal probability regardless of their size. Thus, $BG_{\log}$ penalizes all equal arity partitions uniformly and always maximally as regards *GR*.[2] Note also that $BG_{\log}$ coincides with *IG* in binary splitting, since in that case the denominator $\log_2 k$ has value 1.

In our experiments we use mostly well-known data sets from the UCI repository (Merz & Murphy, 1996). Table 1 summarizes briefly the main characteristics of the data sets. It records the numbers of different attribute types (nominal, integer, real), the average number of different values in a numerical attribute's domain (column $\bar{V}$), the average number of boundary points per each numerical attribute (column $\bar{B}$), the total number of examples, and the number of distinct classes. The experiments were run on a SPARCserver 670MP computer.

The distinction between a nominal and an integer domain is not always clear. Often an integer-valued domain actually enumerates the values of a nominal domain. We have tried to screen out such enumerations from among the numerical attributes as to have a clear picture of the numerical domains that exist in commonly used data sets. Table 1 only gives the average numbers for different values in a numerical attribute's range $\bar{V}$ and the average number of boundary points $\bar{B}$. For these 30 "real-world" data sets it holds with only few exceptions that $\bar{B} < \bar{V} \ll n$. Appendix B studies these decisive relationships more comprehensively.

5.1. *Comparison of splitting strategies*

A multisplitting strategy endeavors to find a split that minimizes or maximizes the value of the evaluation function. If the function is advantageous, optimizing its value will benefit the prediction abilities of the classifier. In any case, the only objective criterion by which the strategies can be judged is the goodness-of-split values that they produce. In order to assess how different multisplitting strategies meet their goal we tested the methods on their own, separate from any learning scheme.

We examined each numerical attribute of a data set in turn and produced a multisplit for the attribute. The evaluation function *IG* was used with an upper bound 10 for the arity of a multisplit. Unknown attribute values were handled in the manner of C4.5: they were distributed into all partition subsets with weight proportional to the subset size. The results are given as the average values of the measured quantities over every numerical attribute of the data set.

The first test series monitors the quality of the splits that are produced by different multisplitting strategies. Figure 9 depicts the *IG* scores obtained by the strategies Greedy and Random relative to the optimal scores. Each mark represents the relative average Information Gain score of a split of arity at most 10 over all numerical attributes. The strategies Greedy and Optimal choose a split on boundary points independent of whether bins or blocks are used as the basic processing unit because function *IG* is well-behaved. Thus, only one relative score—denoted by a square—is depicted for Greedy. For the strategy Random it makes a difference which is the processing unit: the relative average

Table 1. Characteristic figures of the thirty test domains.

	Attributes						
Data set	Nominal	Integer	Real	$\bar{V}$	$\bar{B}$	Examples	Classes
Abalone	1		7	863.7	826.4	4177	29
Adult	8		6	3,673.7	1,668.2	32561	2
Annealing	10	4	6	27.5	17.7	798	5
Australian	9		6	188.2	129.7	690	2
Auto insuran.	10	8	7	61.7	51.1	205	6
Breast W		9		9.9	9.7	699	2
Colic	2	13	8	51.0	39.8	368	2
Diabetes			8	156.8	108.1	768	2
Euthyroid	23	1	5	165.0	91.5	2800	2
Fermentation	1		8	18.5	10.6	100	3
German	13	7		145.9	71.1	1000	2
Glass			9	115.3	70.8	214	6
Heart C			13	30.5	27.3	303	5
Heart H	1	12		61.2	47.6	294	2
Hepatitis		13	6	54.7	30.7	155	2
Hypothyroid	18	1	6	165.4	58.1	3163	2
Iris			4	30.8	15.0	150	3
Letter recogn.		16		16.0	15.7	20000	26
Liver			6	54.7	45.8	345	2
Mole	12	2	18	100.1	63.1	425	2
Page blocks			10	909.2	338.9	5473	5
Robot			22	6.1	6.1	2100	7
Satellite		36		76.3	62.6	4435	6
Segmentation		1	18	137.7	90.1	210	7
Shuttle			9	123.2	85.3	58000	7
Sonar			60	187.6	96.8	208	2
Vehicle		18		79.4	69.1	846	4
Vowel			10	623.5	546.7	990	11
Wine		2	11	98.2	56.3	178	3
Yeast			8	51.5	47.9	1484	10

when bins are used is denoted by a black circle and that when using blocks by a white circle.

Choosing the partition thresholds at random cannot compete with the strategies that aim at maximizing the value of the evaluation function. The strategy's relative performance over all the attributes is the best on the domain Adult. This is a domain that contains attributes for which informed partitioning is difficult; random splitting is almost as good a strategy.

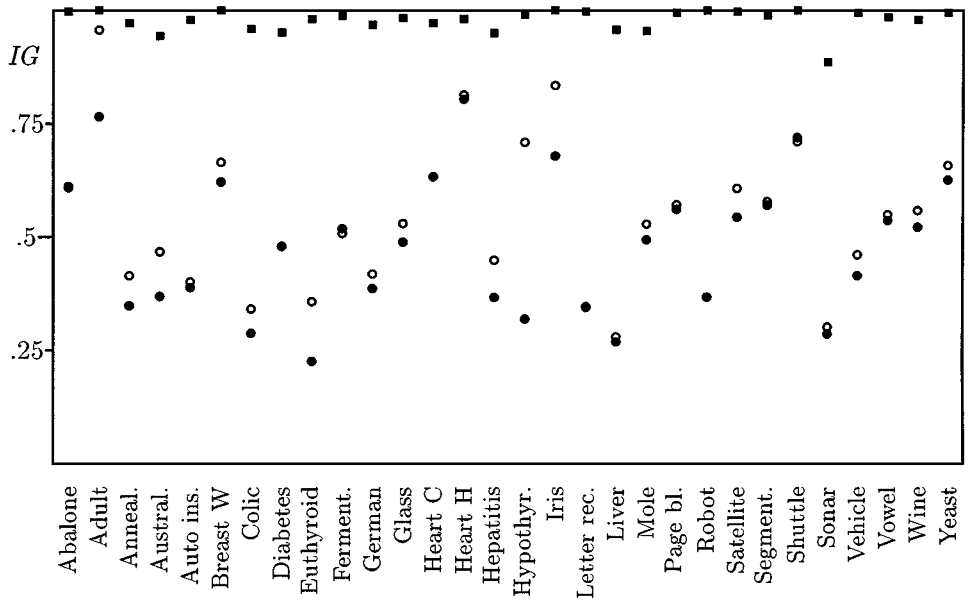

Figure 9. The average relative Information Gain scores obtained by splitting strategies Greedy (denoted by squares) and Random (white circles denote the version operating on blocks and black circles the one that handles bins).

The Greedy strategy regularly fails to obtain optimal Information Gain scores. Only in two domains, those with the least average number of boundary points—Breast W and Robot—was the Greedy able to find the optimal split for every numerical attribute. Nevertheless, Greedy comes up with partitions that have only marginally smaller average gain than optimal partitions.

The domains that cause the most trouble for Greedy in this experiment are those with a high number of truly continuous attributes. A distinctive characteristic of these domains is that they have a high number of bins and blocks with respect to the number of examples in the data (see Appendix B). In these domains Greedy leaves unevaluated a large number of partitions, which may explain the above described behavior.

The other interesting property of multisplitting algorithms is their running time. Table 2 lists the average running time per split required by the algorithms to produce a multisplit. This figure also includes preprocessing time, bin and block construction, which only constitutes a marginal part of the total time required to choose a partition. Time required for sorting is given separately in its own column. In practice, optimal splitting is most often very fast, provided that we use a cumulative, well-behaved evaluation function. For over two thirds of the test sets the average time required to find a partition by the strategy Optimal (the version that utilizes blocks) is below three tenths of a second.

Instead of the arithmetic mean (average) value, which is totally dominated by the heaviest domains, here we use *geometric mean* (in this case the thirtieth root of the product of the thirty domain average values) to summarize the results. Differences in geometric mean

Table 2. Average running times of the multisplitting strategies (in hundredths of a second per split).

		Random		Greedy		Optimal	
Data set	Sorting	Bins	Blocks	Bins	Blocks	Bins	Blocks
Abalone	8.1	53.9	55.2	107.0	104.0	9,342.3	10,105.1
Adult	53.0	81.2	27.8	82.0	64.2	161,300.0	32,433.0
Annealing	0.5	0.9	0.8	1.2	1.0	3.7	1.6
Australian	0.8	1.0	1.1	2.8	2.1	77.5	37.9
Auto insurance	0.2	1.1	0.9	2.0	1.8	16.1	9.6
Breast W	0.4	0.4	0.4	0.4	0.5	0.4	0.4
Colic	0.6	1.5	1.2	2.6	1.9	40.6	10.3
Diabetes	1.0	1.8	1.3	2.8	2.3	83.8	34.5
Euthyroid	6.0	4.2	3.5	5.6	4.5	82.0	26.0
Fermentation	0.0	0.5	0.4	0.6	0.5	1.1	0.4
German	0.9	2.1	1.1	5.0	2.9	291.5	55.8
Glass	0.2	1.8	1.4	3.6	2.9	38.1	20.1
Heart C	0.2	0.7	0.8	1.3	1.1	7.3	5.6
Heart H	0.3	0.9	0.9	1.1	1.0	7.3	4.0
Hepatitis	0.5	0.6	0.4	1.1	0.7	5.3	1.6
Hypothyroid	6.2	4.5	3.6	6.1	4.1	94.9	14.1
Iris	0.0	0.5	0.4	0.6	0.4	1.9	0.6
Letter recogn.	20.0	12.6	12.9	13.5	13.3	13.9	14.6
Liver	0.5	0.7	0.7	1.2	1.0	6.3	4.5
Mole	0.6	1.8	1.5	2.9	2.1	35.2	12.8
Page blocks	12.8	16.1	8.6	35.3	15.7	2,570.4	324.2
Robot	0.9	1.0	1.1	0.9	1.0	0.9	0.9
Satellite	8.2	4.8	4.6	5.9	5.6	25.6	18.1
Segmentation	0.2	2.3	1.9	4.8	3.8	69.0	30.1
Shuttle	106.0	36.9	37.6	39.4	38.1	94.1	60.2
Sonar	0.3	1.4	0.9	3.4	2.1	60.1	15.9
Vehicle	0.9	1.4	1.6	2.4	2.3	33.8	22.8
Vowel	2.0	30.0	24.0	48.9	48.9	2,774.1	2,097.8
Wine	0.2	1.0	0.7	1.9	1.4	20.1	6.0
Yeast	1.3	2.3	1.8	3.0	2.9	13.6	11.8
Geometric mean	1.0	2.4	2.0	3.8	3.1	39.7	18.0

measure the relative value differences as opposed to the absolute differences that arithmetic mean measures.

For most parts these results are clear: Sorting takes expected $O(n \log V)$ time, so it has a linear dependency on the number of examples, which is clearly reflected in the time consumption of sorting. In case of some of the largest domains—Letter recognition and Shuttle—sorting actually dominates the search times. The relative order of the running speed of the tested strategies is clear and could be expected: Random naturally is the fastest, Greedy comes second, and searching for the optimal split requires the most time. For all strategies restricting partitions to boundary points is faster than considering all possible values as potential cut points. For strategies Random and Greedy the speed-up is on the average little less than 20%; Optimal gains in excess of 50% in speed by examining boundary points instead of all possible cut points.

The running times of the two versions of the Optimal strategy depend quadratically on the number of bins and blocks, respectively. A large number of bins or blocks results in slower running times. The average numbers of bins and blocks given in Table 1 usually serve well to indicate this dependency. The four domains with the highest average number of blocks make Optimal use over one second of time per split: domain Adult requires over 5 min per split, Abalone near to 2 min, Vowel over 20 sec, and “Page blocks” 3 sec. On all the other domains optimal split is chosen within 6 tenths of a second. The Random strategy never takes even a second to choose a split and for Greedy only domain Abalone requires a second per split.

There is one peculiarity in the results; viz., the slow running time of both versions of the Optimal strategy on data set German, which does not have a very high average number of bins nor boundary points and which only contains one thousand examples. The reason is that there is one attribute with 921 bins and 398 blocks, which accounts for the slow processing in this case (see Appendix B). The average figures are kept low for this domain because of the small numbers of bins and blocks along the other numerical dimensions. In fact, the same applies to domains Auto insurance, Colic and Adult, although the effects are not as striking.

The Optimal strategy’s vulnerability to a high number of boundary points in just one numerical dimension cannot be helped. However, since the number of boundary points in the domain of an attribute does not stay constant throughout the tree growing process, one could delay the evaluation of partitions along the dimensions that have a very high number of boundary points, in the hope that the number reduces as the decision tree evolves. The rationale for such a heuristic is that in univariate learning such an attribute alone will not be very useful in class prediction, but due to being part of a multivariate interdependency, can subsequently turn out to be an important one.

In domains with a relatively low number of boundary points the Optimal strategy (using blocks) is within the same order of magnitude in running speed as Greedy. When the (average) number of boundary points grows large, optimal multisplitting is slower than Greedy partitioning; in case of the four hardest domains it is very much slower. On the average the Greedy strategy is almost six times faster than Optimal.

The relative speed of the two versions of the Optimal strategy is, of course, determined by the relationship of the numbers of boundary points and different values in the numerical domains, B/V. Most striking reductions in running time are obtained in the domains where this fraction is the lowest, and vice versa (see Table 1 and Appendix B).

5.2. The impact on decision tree learning

Our second test set explores the practical impact that different splitting strategies have on the result of induction. The strategies Greedy and Optimal were implemented within the

C4.5 algorithm. The test method was 10-fold cross-validation. In addition, each test was repeated ten times. The reported figures are average values over these repetitions. Standard deviations were calculated over the results of the ten repetitions. The default pruning method of C4.5 was applied.

When combined with the multisplitting strategies, the *IG* function systematically fails to match the results that are obtained with the $BG_{\log}$ function. Therefore, we refrain from recording, in the following tables, the results for the strategies Optimal and Greedy combined with *IG*. Recall that in binary partitioning *IG* and $BG_{\log}$ coincide.

Table 3 records the average prediction accuracies (with standard deviations) that were obtained when the trees were not required to be reduced; i.e., a numerical attribute whose value range is split remains to be among the candidate attributes. In individual domains the differences in average accuracy are relatively consistently either in favor or against *GR* throughout the strategies. The only outstanding value difference in Table 3 is the difference on the domain Sonar. In this example set there are only few duplicate values, which appears to be an impairing factor for the *GR* function. Altogether, the results of this experiment lead us to hypothesize that by using $BG_{\log}$ significantly more accurate decision trees are built than by using *GR*, independent of which splitting strategy is used.

We tested whether this hypothesis gets any statistical support from these results. We contrasted the two evaluation functions with each other paired by the splitting strategy. Statistically significant (confidence level 99%) and *almost* significant (confidence level 95%) differences in the prediction accuracy were calculated using the two-tailed Student's t-test.

Table 4 summarizes the result of this comparison. The double plus row gives the number of domains in which $BG_{\log}$ (*IG*) was significantly better in the average prediction accuracy than *GR*. The row entitled with a single plus sums the number of almost significant advantages of the former function(s). The (single and double) minus rows count, respectively, the numbers of domains that were significantly favorable for *GR*. The number of domains where no statistically significant difference in the prediction accuracy could be observed is given in the row without a title. Independent of the splitting strategy used $BG_{\log}$ (*IG*) is clearly more often statistically significantly more accurate than *GR*. The difference is the clearest in binary partitioning—the strategy with which *GR* is typically combined.

These results indicate that the choice of the numerical attribute evaluation function has an effect on the prediction accuracy of the resulting decision tree, no matter which splitting strategy is used. The results in Table 3 also seem to indicate that the choice of partitioning strategy has only a marginal effect on the outcome of induction. In order to test this hypothesis statistically, we also paired the results obtained by all the three strategies with each other according to the evaluation function that was used. Then, the same statistical test as in the previous comparison was applied to all average value pairs.

Table 5 summarizes the results of this comparison. Columns Opt/Grd denote the two comparisons between Optimal and Greedy strategies, Opt/Bin those between Optimal and Binary, and Grd/Bin those between Greedy and Binary. Again, in the double plus row the number of statistically significant differences in favor of the first-mentioned strategy are counted. The row with a single plus stands for almost statistically significant differences in favor of the first strategy. Similarly, rows named by double minus and minus sum the same

Table 3. Average prediction accuracies (with standard deviations) of the decision trees obtained using the different splitting strategies and evaluation functions.

	Binary		Greedy		Optimal	
Data set	*GR*	*IG*	*GR*	$BG_{\log}$	*GR*	$BG_{\log}$
Abalone	21.5 ± 0.5	21.2 ± 0.8	19.9 ± 1.0	20.4 ± 1.0	19.6 ± 0.6	20.5 ± 0.7
Adult	85.4 ± 0.2	84.2 ± 0.3	85.7 ± 0.1	85.6 ± 0.3	85.7 ± 0.1	85.2 ± 0.3
Annealing	91.1 ± 0.6	92.1 ± 0.6	89.4 ± 0.6	88.5 ± 0.7	89.4 ± 0.5	91.3 ± 0.4
Australian	85.0 ± 0.8	84.8 ± 0.4	84.0 ± 1.2	86.0 ± 1.1	83.7 ± 0.4	85.3 ± 0.6
Auto ins.	78.4 ± 1.3	80.3 ± 0.9	78.1 ± 1.5	76.2 ± 2.0	78.5 ± 1.3	75.2 ± 1.6
Breast W	94.7 ± 0.4	94.9 ± 0.4	94.1 ± 0.4	94.6 ± 0.6	94.0 ± 0.4	94.7 ± 0.5
Colic	84.3 ± 0.8	83.9 ± 0.9	84.5 ± 1.1	83.6 ± 0.8	85.8 ± 0.7	84.2 ± 0.7
Diabetes	72.8 ± 1.3	75.0 ± 1.0	73.2 ± 0.8	74.3 ± 0.8	73.2 ± 1.0	74.3 ± 1.0
Euthyroid	98.6 ± 0.1	98.5 ± 0.1	98.4 ± 0.2	98.5 ± 0.1	98.6 ± 0.1	98.6 ± 0.1
Fermentation	90.1 ± 0.7	90.0 ± 1.1	90.3 ± 0.6	90.3 ± 1.0	89.6 ± 0.7	91.2 ± 0.9
German	71.8 ± 0.8	72.7 ± 0.6	71.7 ± 0.7	72.7 ± 0.6	71.1 ± 1.0	72.3 ± 0.7
Glass	68.5 ± 2.4	69.8 ± 1.4	67.8 ± 2.3	72.7 ± 1.5	68.9 ± 2.1	72.3 ± 1.4
Heart C	53.5 ± 1.5	53.3 ± 1.0	52.1 ± 1.1	53.4 ± 2.0	53.2 ± 1.3	53.7 ± 1.7
Heart H	80.1 ± 1.3	77.7 ± 1.6	81.2 ± 0.8	78.7 ± 1.2	82.3 ± 0.7	79.7 ± 1.5
Hepatitis	79.8 ± 1.5	80.8 ± 2.5	80.1 ± 1.7	79.3 ± 2.6	79.8 ± 1.2	81.9 ± 1.7
Hypothyroid	99.1 ± 0.0	99.3 ± 0.0	99.2 ± 0.0	99.2 ± 0.0	99.2 ± 0.1	99.2 ± 0.1
Iris	93.5 ± 1.2	94.3 ± 1.3	92.5 ± 0.9	93.4 ± 0.9	92.5 ± 0.9	93.4 ± 0.9
Letter recogn.	79.9 ± 0.5	80.5 ± 0.5	80.6 ± 0.4	80.4 ± 0.6	80.8 ± 0.7	80.6 ± 0.5
Liver	65.7 ± 2.4	66.8 ± 1.4	64.0 ± 1.2	64.2 ± 1.8	63.6 ± 1.8	61.9 ± 2.1
Mole	81.7 ± 1.2	81.0 ± 1.0	79.9 ± 1.6	79.1 ± 1.5	81.8 ± 1.2	82.5 ± 0.9
Page blocks	96.3 ± 0.5	96.0 ± 0.5	96.0 ± 0.5	96.1 ± 0.4	96.0 ± 0.4	95.9 ± 0.4
Robot	99.3 ± 0.1	99.6 ± 0.1	99.6 ± 0.1	99.5 ± 0.1	99.6 ± 0.1	99.5 ± 0.1
Satellite	86.2 ± 0.3	86.8 ± 0.4	86.6 ± 0.3	86.3 ± 0.3	86.6 ± 0.3	86.0 ± 0.3
Segmentation	86.6 ± 1.1	85.7 ± 1.2	86.5 ± 1.3	85.5 ± 1.0	86.5 ± 1.3	85.4 ± 1.1
Shuttle	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.0	99.9 ± 0.1	99.9 ± 0.0
Sonar	71.7 ± 2.7	75.4 ± 2.7	64.2 ± 2.7	74.7 ± 2.1	65.0 ± 3.1	74.9 ± 2.2
Vehicle	70.4 ± 0.8	73.0 ± 1.2	71.7 ± 1.1	71.9 ± 1.2	71.4 ± 1.2	72.0 ± 0.6
Vowel	63.1 ± 3.3	60.4 ± 2.1	60.4 ± 2.1	63.0 ± 2.2	58.8 ± 2.4	61.7 ± 2.2
Wine	91.2 ± 1.2	91.4 ± 0.6	94.2 ± 1.0	94.4 ± 1.0	94.2 ± 1.0	94.4 ± 1.0
Yeast	57.6 ± 0.6	57.3 ± 0.8	56.9 ± 0.6	57.6 ± 0.6	56.6 ± 0.5	57.4 ± 0.5

numbers for the comparison strategy. The untitled row contains the number of domains where no statistically significant difference exists in this comparison.

The significant differences between every pair of strategies are very nearly balanced. There are very few domains in which there is any significant difference between the two multisplitting strategies. There are more significant differences when a multisplitting strategy is

Table 4. Statistically significant differences in the prediction accuracies of Table 3 when comparing the evaluation functions within splitting strategies.

	Binary	Greedy	Optimal
Difference	*IG* vs. *GR*	$BG_{\log}$ vs. *GR*	$BG_{\log}$ vs. *GR*
++	8	5	11
+	2	3	3
	16	17	10
−	2	3	1
−−	2	2	5

Table 5. Statistically significant differences in the prediction accuracies of Table 3 when comparing the splitting strategies without changing the evaluation function.

	GR			$BG_{\log}$		
Difference	Opt/Grd	Opt/Bin	Grd/Bin	Opt/Grd	Opt/Bin	Grd/Bin
++	2	6	5	2	4	5
+	1	2	1	2	3	0
	25	13	14	23	18	19
−	1	2	6	2	1	0
−−	1	7	4	1	4	6

compared to the Binary strategy, especially with the *GR* evaluation function, but the number of such differences in favor of Binary is almost equal to the number of differences in favor of multisplitting. In sum, this result confirms our earlier observation that the choice of the evaluation function has a greater impact on accuracy than the splitting strategy selection.

Prediction accuracy is undoubtedly the most important variable to monitor. However, there are secondary parameters worth some attention too. The multisplitting strategies produce decision trees that have on the average somewhat more nodes in total than binary decision trees. The average tree sizes over the 30 test sets are: 102.1 for Binary using the Gain Ratio, 103.7 using the Information Gain function, the respective figures for Greedy are 111.7 and 121.0, and for Optimal 130.7 and 123.5. Two factors explain why binary decision trees are slightly smaller on the average: the pruning method of C4.5 deletes a binary partition from the decision tree more eagerly than a multisplit decision node and a binary tree cannot have a large number of leaf nodes.

Geometric mean is used in Table 6 to summarize the average running times of the multisplitting strategies over the 30 test domains. Using the Binary strategy in C4.5 is on average about twice as fast as using greedy multisplitting. Optimizing the multisplit doubles the average time again. The results of our stand-alone test of multisplitting lead one to expect the Greedy strategy to have a greater advantage in running speed over the Optimal strategy. In C4.5 auxiliary tasks like reading in the data, preprocessing and pruning balance the

Table 6. Average running times of C4.5 combined with the different splitting strategies over the 30 test domains (in sec).

	Binary		Greedy		Optimal	
	GR	*IG*	*GR*	$BG_{\log}$	*GR*	$BG_{\log}$
Geometric mean	2.8	2.3	5.3	5.5	12.2	10.5

differences between the strategies. In addition to producing more accurate decision trees, using $BG_{\log}$ instead of *GR* is also beneficial in the execution efficiency.

As a general rule, large domains take the most time to handle, even when using the Binary strategy, since sorting alone takes a longer time for these domains. Hence, the Binary strategy has only a small efficiency advantage in some of the large test sets. The same applies to small domains with a minimal number of (bins and) blocks. Also, the number of the numerical attributes plays a role in decision tree learning, because the more numerical attributes there are the more often one needs to process the data. Moreover, the Binary strategy examines all bin borders. Therefore, its running time is affected by a very high number of bins in a domain. This is most evident in the Adult domain. Nevertheless, the time consumption of Binary never exceeds 4 min. The running times of the multisplitting strategies are hit worse by the Adult domain's high number of boundary points: Greedy (with gain ratio) requires in excess of 19 min to produce a tree and Optimal over 2 hours. However, there are only four domains—Abalone, Adult, Mole, and Vowel—where the time consumption of C4.5 combined with the Optimal strategy is of a different order of magnitude than when C4.5 uses its built-in Binary strategy.

6. Discussion

According to the preceding empirical evidence multisplitting of numerical attribute value ranges using functions $BG_{\log}$ (*IG*) and *GR* does not carry any advantage in prediction accuracy over binary splitting. Similar empirical evidence has been put forward, e.g., by Quinlan (1996). In addition, the greedy selection of the multisplit thresholds in numerical dimensions results in decision trees with as good accuracy as when optimal multisplits are selected. On the other hand, the price that has to be paid for optimizing multisplits remains in most cases low.

It is relatively easy to understand why multisplitting does not benefit the prediction accuracy as much as might be expected. The greedy TDIDT construction algorithm does not give any chance of revoking a decision once it has been taken. Hence, the true utility of a local multisplitting decision is actually determined by the subtrees that will be grown subsequently, not at the evaluation time. On the other hand, when basing the multiway partitioning on binarization, the numerical attribute at hand can affect the subtrees all the way to the end. Thus, the final splitting decision is delayed until further knowledge is gathered. The corresponding effect for multisplitting could be obtained by look-ahead, which unfortunately is very time-consuming.

When embedded into the TDIDT framework of C4.5 the difference in the efficiency that exists between strategies Greedy and Optimal reduces because in decision tree construction other tasks, like reading in the data, preprocessing, and pruning, take their part of the total running time. Multisplitting is slower than binarization; greedy multisplitting takes on the average twice the time of binarization in decision tree learning and optimal multisplitting further doubles the average time of greedy multisplitting.

The size of the data set is not a decisive factor between the splitting strategies. It mainly affects sorting, which has to be performed for all methods alike. On the other hand, a large number of boundary points affects the running time of the multisplitting algorithms, but it usually indicates that the attribute at hand is not alone predictive in this situation or it is extremely noisy. Either way, such an attribute is not very applicable in univariate decision tree construction. A filtering mechanism could help to overcome the problem of apparently irrelevant attributes in these situations.

For example, recall from Section 5.1 that there is a single attribute with a high number of blocks in domains German and Adult causing the Optimal strategy to substantially slow down. In an experiment we simply removed those attributes and grew the decision trees anew. Deleting that malignant attributes did not change the prediction accuracies of the produced trees significantly; i.e., the attribute was not used in any case. The running times of all strategies, on the other hand, decreased. Particularly large time-savings were obtained for the Optimal strategy; for the domain Adult over 2 hours of processing time was originally required to come up with a decision tree; after removing the one useless attribute less than 6 min suffices. However, one must be careful in modifying the data; attributes cannot be simply deleted because of their high number of boundary points, without risking to lose complex (multivariate) attribute interrelations prevailing in the data.

For the Optimal strategy there exists many further practical (heuristic) speed-up opportunities. One of them, restricting the split to have at most a constant arity, was used above to counteract the bias of *IG*. We used a global upper bound for the arity of a partition, i.e., it was common to all attributes. One could assign an individual upper bound for each attribute separately. These heuristics intend to take into account the (poor) bias that an evaluation function has. Obviously, with well-balanced evaluation functions (Quinlan, 1986; White & Liu, 1994; Kononenko, 1995) no heuristics are needed.

In these experiments we have intentionally overlooked the problems caused by noisy data; it remains a topic for further research. However, in an initial experiment we observed that increasing the amount of classification noise caused an increase in the number of blocks. For domains with a low $\bar{B}/\bar{V}$ ratio the increase may even be dramatic. This, of course, increases execution time for all splitting strategies based on examining boundary points. Attribute noise increases both the number of bins and blocks, and thus affects the running times of all partitioning strategies. Whether the resulting partition (optimized or approximated) is useful in the noisy induction task is determined by the robustness of the evaluation function rather than by the splitting strategy. Even in a noisy setting, optimizing a good evaluation function ought to be profitable.

7. Conclusion and future work

In this paper we have analyzed the prerequisites of multisplitting of numerical attributes. The starting point for efficient multisplitting is the concept of a boundary point as defined by

Fayyad and Irani (1992b). We have given a characterization of the evaluation functions for which it suffices to focus only on the boundary points in searching for an optimal multiway partition. An efficient method of going through the required boundary point combinations naturally suggests itself. It was previously known that some class entropy-based functions are convex downwards. The class of well-behaved evaluation functions contains these as well as other types of functions. Our empirical evaluation shows that alternative approaches to multisplitting constantly fail to obtain optimal partitions. However, optimizing the value of the most common evaluation functions in choosing a multisplit does not bring any apparent advantage in decision tree learning.

The current work has shown that most of the commonly used attribute evaluation functions are well-behaved in multisplitting. The number of blocks grows sublinearly in the size of the domain; hence, well-behavedness reduces the dependency of the computation of an evaluation function from the size of a data set, but leaves it vulnerable to noisy and irrelevant attributes. Our experiments show that accuracy is considerably influenced by the choice of evaluation function. Well-behavedness is a necessary, but not a sufficient condition for a good evaluation function. In addition, it needs to have a well-balanced bias.

The most interesting candidate for further analysis on efficient attribute evaluation is to explore whether there is any generality in the class of attribute evaluation functions that can be optimized in linear time; is the Training Set Error the only practical instance of such functions? Another interesting research direction is to see whether the definition of a well-behaved function can be tightened in order to ascertain the quality of the decision trees that will be produced and to further reduce the time complexity of the optimization of these functions. As demonstrated in Section 4.1, well-behavedness does not guarantee the existence of an optimal partition of fixed arity such that it would be defined by boundary points. We conjecture that evaluation functions WP_{cost}, *TSE*, and $BG_{\log}$ actually have this property. The functions *GR* and *ND* would be excluded from that class of functions. However, it is unclear whether the tighter definition would rule out all evaluation function that do not have a good bias.

Another interesting research topic is the development of lazy attribute evaluation methods, which postpone the evaluation of attributes with a high number of boundary points until a later stage in the decision tree learning. Such methods could free the optimal partitioning from being very expensive when there are a large number of boundary points even in a single numerical dimension. Moreover, the effect of noise on multisplitting, which was only briefly touched in this paper, ought to be analyzed and empirically explored.

Multisplitting makes it possible to prohibit the use of numerical attributes more than once in a branch of a decision tree. In our empirical experiments we let numerical attributes reappear. It is a topic for further research what is the practical effect of growing reduced trees when using multisplitting of numerical attributes. How does reducedness affect prediction accuracy? Are the reduced trees significantly smaller than those that are not reduced? What about their understandability?

Developing new attribute evaluation functions is an art rather than a science. In particular, the recent “advances” in this field have been ad hoc rectification attempts of well-known evaluation functions. From all that has been reported above, clearly, three guidelines for future evaluation function design emerge:

Use cumulative evaluation functions. As long as the evaluation function is cumulative, the multisplitting strategy given in this paper can be used to find the optimal multisplit for the evaluation function.

Concentrate on developing evaluation functions with the appropriate bias. Focus on developing transparent evaluation functions, where the bias can be better analyzed and understood. Towards that end, the evaluation function should be applied to the whole partition at once, not to just some parts of it at a time, as in some greedy approaches.

Ensure that the function is well-behaved. If a general-purpose evaluation function is well-behaved, then it will work in a beneficial manner in connection with any splitting strategy. Moreover, well-behaved functions clearly are the only reasonable ones to use (Fayyad & Irani, 1992b; Breiman, 1996).

Handling numerical attributes in a preprocessing phase, prior to induction rather than during classifier construction, is an approach that has been often suggested (Cattlet, 1991; Dougherty, Kohavi, & Sahami, 1995). This approach is not quite as time-critical as the on-line discretization of numerical value ranges. Furthermore, preprocessing strives for as good partitions as possible. Obviously, optimal multisplitting is a strategy that suits this approach perfectly. A noteworthy application area for growing optimized, reduced decision trees might be data mining, in which decision tree learning algorithms are important tools.

Appendix A: The search algorithm for optimal multisplits

The optimal multisplit search algorithm described below uses a dynamic programming scheme similar to that suggested by Fulton, Kasif, and Salzberg (1995). The main conceptual and practical difference is that it works on example intervals rather than on individual examples. This leads to more compact data structures and better run-time efficiency. Preprocessing the data into blocks or bins relieves us from checking if the data can be split in between two examples and whether there is a boundary point in between them. Using bins and blocks automatically takes care of those checks, and the search algorithm works much faster in practice.

We use a simple two-phase preprocessing algorithm, which, in essence, was already described in Section 2. In the first phase bins are extracted from the example sequence; this requires a scan over the sorted example sequence and recording the frequencies of the different classes in the bins as well as the associated numerical attribute's value. In the second phase neighboring class uniform bins that contain examples of the same class are compressed into blocks and the highest numerical value in each block is stored. Recall that bins with mixed class distributions are blocks in themselves. Implementation of these algorithms is straightforward and will not be elaborated further. It is possible to implement the preprocessing in a single pass, but even the straightforward two-pass strategy already is very fast in comparison with the following search phase.

In the search algorithm each interval I_i is represented only by its class frequency distribution, denoted by μ_i. The question whether the intervals are blocks or bins is immaterial from the perspective of the search algorithm; preprocessing decides the semantics of the

```
procedure $Search(\vec{\mu},\ f, aritymax)$
    $\vec{\mu} = \{\mu_1, \ldots, \mu_B\}$ contains the class frequency distributions of blocks $I_1, \ldots, I_B$,
    $f$ is an impurity function, and $aritymax$ is the maximum number of intervals
    allowed in the split.

 1. for $i \leftarrow 1$ to $B$ do
 2.     for $j \leftarrow 1$ to $i - 1$ do           /* Augment final intervals with block $I_i$ */
 3.         $\mu_j \leftarrow \mu_j + \mu_i$; $cost_j \leftarrow f(\mu_j)$ od;
 4.     $P_{i,1} \leftarrow cost_1$;
 5.     if $i = B$ then $limit \leftarrow aritymax$
 6.         else $limit \leftarrow aritymax - 1$ fi;
 7.     for $k \leftarrow 2$ to $\min\{i, limit\}$ do                /* Compute best $k$-split of */
 8.         $minimum \leftarrow \infty$;                           /* $I_1, \ldots, I_i$ for each $k \leq i$ */
 9.         for $j \leftarrow k - 1$ to $i - 1$ do
10.             $current \leftarrow P_{j,k-1} + cost_{j+1}$;
11.             if $current < minimum$ then
12.                 $minimum \leftarrow current$; $indexofminimum \leftarrow j$ fi od;
13.         $P_{i,k} \leftarrow minimum$; $L_{i,k} \leftarrow indexofminimum$ od od
```

Figure 10. The search algorithm for multisplits. It fills up two arrays, P and L. After running the algorithm, for each i and k, $P_{i,k}$ is the cost of the optimal k-split of first i blocks and $L_{i,k}$ is the index to the block which is situated immediately left from the rightmost cut point.

intervals. In the following discussion it is assumed that blocks are handled. Corresponding bounds for the case where bins are used instead can be obtained by replacing "V" for "B".

The search algorithm (figure 10) uses a left-to-right scan over the intervals $I_1, \ldots, I_B$ and uses three arrays, P, L, and *cost*, for storing the intermediate results. Array P stores the costs of the best multisplits: $P_{i,k}$ is the minimum cost obtained when first i intervals are split optimally into k subsets. Array L is used for storing the corresponding cut points: $L_{i,k}$ is an index to the block that contains the rightmost cut point of the multisplit having the cost $P_{i,k}$. The third array *cost* is used to eliminate the repeated calculation of interval impurities.

Since the class distribution of the interval I_j is not needed, as such, after the scan has passed the interval j, the algorithm reuses the space: at point i, each μ_j, $j \leq i$, represents the class distribution of the union of the intervals $I_j, \ldots, I_i$. This is performed by merging the distributions and recalculating the costs (lines 2–3).

In iteration i, the array P is updated (lines 4–13) according to the formula:

$$P_{i,k} \leftarrow \min_{k-1 \leq j < i} \{P_{j,k-1} + f(\mu_{j+1})\},$$

which has the following intuitive interpretation. The optimal partitioning of $I_1, \ldots, I_i$ into k subsets is the minimum cost over all combinations of fixing the last interval $\bigcup_{l=j+1}^{i} I_l$ and adding the cost of best $(k-1)$-split of $I_1, \ldots, I_j$. This update needs to be performed for every arity $2 \leq l \leq i$.

Note that no prefix $I_1, \ldots, I_l$, where $l < k$, can be split into k intervals. This fact lets the algorithm restrict the computation to the upper triangular subarray of P. The fact that the impurities of the k-splits of the proper prefixes of the data are not needed for computing the k-split of the data, is utilized as well (lines 5–7).

Finally, let us note that extracting the best k-split from the array L is easy: $L_{B,k}$ gives the index of the block that contains the rightmost cut point. The next cut point to the left is contained in the block $L_{l,k-1}$, where $l = L_{B,k}$. Continuing this iteration we find all of the block indices. Finally, the multisplit is constructed with the help of the boundary points that are stored with each block.

A.1. Time and space requirements of the algorithm

The time complexity of preprocessing is easily determined. The first pass comprises reading the class labels of the examples and summing them up into the class distributions of the bins. All this clearly takes time $O(n + mV)$, where n is the number of the examples, m is the number of the classes, and V is the number of the bins. The cost of the second phase (block construction) is determined by the number of required two bin merges. In the worst case we can have $V - 1$ of them, which leads to time complexity $O(mV)$. Hence, the total worst-case time complexity of preprocessing is $O(n + mV)$.

Let us now analyze the search algorithm. First, consider the loop in lines 2 and 3. Merging two class frequency distributions takes time $O(m)$, where m is the number of classes. For most commonly used evaluation functions, evaluation of the impurity also takes time $O(m)$, thus, the complexity of one iteration is clearly $O(m)$. The number of iterations of the loop, where μ_i is the last item updated, is i. Hence, the total number of iterations is $\sum_{i=1}^{B} i = B(B + 1)/2$, which gives total complexity of $O(mB^2)$ for the loop.

The rest of this analysis is divided into *unbounded* and *bounded* cases, based on whether a prior restriction to the arity is set or not. We begin with analyzing the unbounded case.

In the following keep in mind that the algorithm fills up and consults a triangular area of the array P, that is, no items $P_{i,k}$, where $k > i$, are ever visited by the algorithm. We amortize the cost of running the innermost loop (lines 9–12) among items in that triangle, excluding row 1, which will not be updated in the loop.

Note that the number of iterations of the loop per array item is equal along each diagonal: it is 1 for the diagonal $P_{2,2}, \ldots, P_{B,B}$, which is of length $B - 1$, 2 for the diagonal $P_{3,2}, \ldots, P_{B,B-1}$ of length $B - 2$, and so on. The total number of iterations of the innermost loop is thus given by the sum

$$\sum_{i=1}^{B-1} i(B - i) = B \sum_{i=1}^{B-1} i - \sum_{i=1}^{B-1} i^2 = B(B - 1)\left(\frac{B}{2} - \frac{2B - 1}{6}\right) = \frac{B^3 - B}{6} < \frac{B^3}{6},$$

and so the time complexity of the innermost loop is $O(B^3)$ in the worst case. Finally, observe that lines 4–6, 8 and 13 take constant time per item of the array P and do not affect the worst-case complexity. Hence, the total running time of the algorithm is $O(mB^2 + B^3)$, if no prior upper bound on the arity is given.

Next, let us turn into the case where a prior upper bound k for the arity of the multisplit is given. In that case, the diagonals are cut into the length $k - 2$ (again excluding row 1), with the exception of the $k - 1$ diagonals in the upper right-hand corner of the triangle that are not affected by the bound k. Therefore, the number of iterations of the innermost loop is given by

$$\sum_{i=1}^{B-k} i(k-2) + \sum_{i=B-k+1}^{B-1} i(B-i),$$

where the former sum represent the "cut" diagonals and the latter the rest. This formula simplifies to

$$\frac{1}{2}\left((k-2)B^2 + (4k-k^2-2)B + \frac{5}{3}k - 2k^2 + \frac{1}{3}k^3\right),$$

which is $O(kB^2)$, since $k < B$ by definition. Including the work needed to update the class distributions $\vec{\mu}$, the total running time of the algorithm in the bounded case is then $O((k+m)B^2)$.

The space complexity of the preprocessing algorithm is $O(mV)$, which is the space taken by the class frequency distributions of the bins. The space needed by the search algorithm is determined by the three arrays, $\vec{\mu}$, P, and L. Array $\vec{\mu}$ has size $O(mB)$. The arrays P and L both have size $O(B^2)$ in the unbounded case and $O(kB)$ in the bounded case. Thus, the total amount of space used by the search algorithm is $O((m+k)B)$ in the bounded case and $O(mB + B^2)$ in the unbounded case.

Appendix B: Relationships between the numbers of examples, different values, and boundary points in the numerical dimensions of the test domains

This appendix analyzes for all our 30 test domains the relationship of the number of different values, V, and the number of boundary points, B, with respect to the total number of examples, n. In addition, we contrast the values V and B with each other.

Figure 11 plots for all test domains the ratios V/n and B/n. The ratios are depicted separately for each numerical dimension. A small circle represents the value for one attribute and the small horizontal lines indicate the median values. In terms of decision tree learning these ratios conform to the situation where the numerical attribute is considered as the label of the root of the evolving tree—when the data has not yet been split by any attribute.

Five domains—Glass, Segmentation, Sonar, Vowel, and Wine—have their median V/n ratio above the value 0.5. These domains with many truly continuous attributes also account for most of those attributes that have on the average at most two examples with the same value. Few attributes from other domains have a ratio higher than 0.5. Only two more domains—Australian and Hepatitis—have their median ratio over 0.25. The number of domains that have their median V/n ratio below 0.1 is 14, almost half of our test domains.

What changes when we consider the ratio B/n? Only one domain, Vowel, keeps its median ratio's value over 0.5. The other four domains that have a median V/n ratio above 0.5 have a median B/n ratio between 0.25 and 0.5. The rest of the domains have a median B/n ratio below 0.25. The number of domains that have this ratio's median below 0.1 is now 19, almost two thirds of our test domains. In domains that only have few attributes

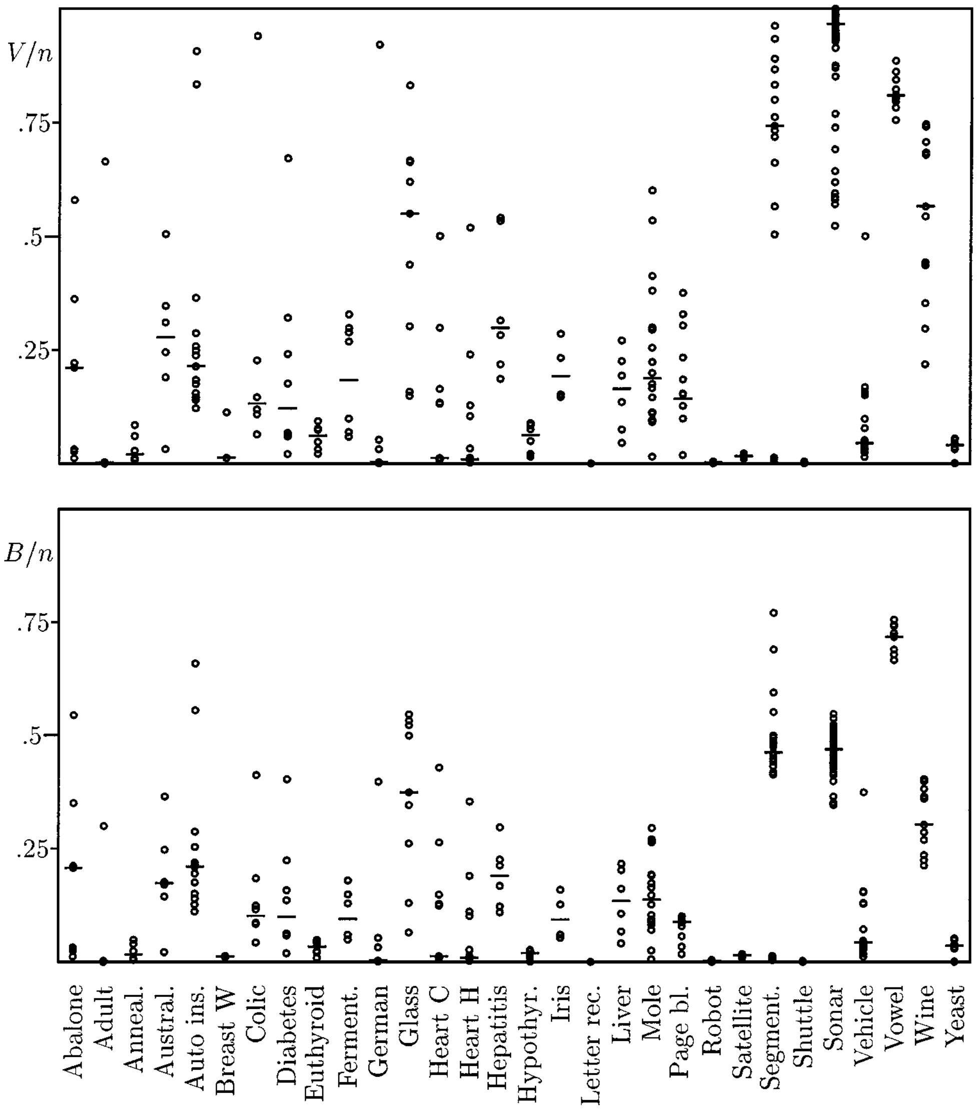

Figure 11. The ratio of the number of different values in a numerical value range, V, and the number of examples, n. Also the ratio between the numbers of boundary points, B, and examples is depicted. Each circle represents one numerical attribute and the horizontal lines indicate the median values of each domain.

with a high V/n ratio—e.g., Adult, Colic, and German—those attributes now have a B/n ratio more like the other attributes.

Based on this comparison we find support for the claim that typically $V \ll n$ for real-world data sets. However, there are domains that have truly continuous attributes for which this does not hold. On the contrary, in such cases we could even claim that $V \approx n$. As

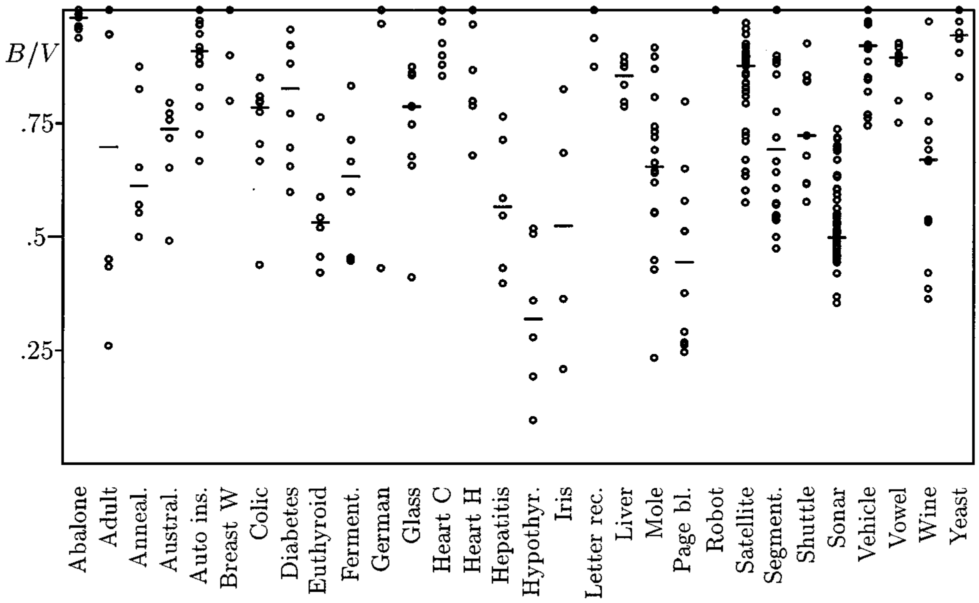

Figure 12. The ratio of the number of boundary points, B, and different values, V, in numerical value ranges. Each small circle represents the B/V ratio of one numerical dimension. Horizontal lines depict the medians of these ratios.

concerns the ratio of boundary points and the number of examples, it is certainly true that typically $B \ll n$, even in domains with truly continuous attributes.

Figure 12 directly compares V and B by plotting the ratio B/V for the 30 test domains. Again, each small circle represents one numerical attribute. A horizontal line denotes the median of these ratios for the numerical attributes of a domain.

In most domains the great majority of attributes have $B/V > 0.5$, meaning that over half of the values are boundary values. This means that examining only boundary points lets us often leave one fourth of the potential cut points without attention, but seldom can more than half of the cut points be overlooked.

Acknowledgments

The insightful comments by the editor Rob Holte helped us to substantially improve the content and presentation of the final version of this article.

The work of T. Elomaa has, in part, been supported by a Marie Curie Fellowship and done at the Joint Research Centre of the European Commission in Ispra, Italy. That of J. Rousu has been partly supported by the Academy of Finland and carried out at the Department of Computer Science of the University of Helsinki.

Notes

1. Strictly speaking, neither the Gain Ratio nor the Normalized Distance Measure conform to our definition of well-behavedness; both are undefined on trivial partitions.
2. Quinlan (1988) motivated the Gain Ratio with experimental results reported by Kononenko, Bratko, and Roškar (1984). In their experiments $BG_{\log}$ was used in evaluating multi-valued nominal attributes, but the results were not satisfactory.

References

Auer, P. (1997). *Optimal splits of single attributes*. Unpublished manuscript, Institute for Theoretical Computer Science, Graz University of Technology.

Auer, P., Holte, R. C., & Maass, W. (1995). Theory and application of agnostic PAC-learning with small decision trees. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference* (pp. 21–29). San Francisco, CA: Morgan Kaufmann.

Birkendorf, A. (1997). On fast and simple algorithms for finding maximal subarrays and applications in learning theory. In S. Ben-David (Ed.), *Proceedings of the Third European Conference on Computational Learning Theory* (pp. 198–209). Lecture Notes in Artificial Intelligence (Vol. 1208). Heidelberg: Springer-Verlag.

Breiman, L. (1996). Some properties of splitting criteria. *Machine Learning, 24*, 41–47.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Pacific Grove, CA: Wadsworth.

Brodley, C. (1995). Automatic selection of split criterion during tree growing based on node location. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference* (pp. 73–80). San Francisco, CA: Morgan Kaufmann.

Buntine, W., & Niblett, T. (1992). A further comparison of splitting rules for decision-tree induction. *Machine Learning, 8*, 75–85.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In Y. Kodratoff (Ed.), *Proceedings of the Fifth European Working Session on Learning* (pp. 164–178). Lecture Notes in Computer Science (Vol. 482). Heidelberg: Springer-Verlag.

Cestnik, B., Kononenko, I., & Bratko, I. (1987). Assistant 86: A knowledge-elicitation tool for sophisticated users. In I. Bratko & N. Lavrač (Eds.), *Progress in Machine Learning, Proceedings of the Second European Working Session on Learning* (pp. 31–45). Wilmslow: Sigma Press.

Codrington, C. W., & Brodley, C. E. (1999). On the qualitative behavior of impurity-based splitting rules I: The minima-free property. *Machine Learning*, to appear.

Dietterich, T., Kearns, M., & Mansour, Y. (1996). Applying the weak learning framework to understand and improve C4.5. In L. Saitta (Ed.), *Machine Learning: Proceedings of the Thirteenth International Conference* (pp. 96–104). San Francisco, CA: Morgan Kaufmann.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference* (pp. 194–202). San Francisco, CA: Morgan Kaufmann.

Elomaa, T. (1994). In defense of C4.5: Notes on learning one-level decision trees. In W. W. Cohen & H. Hirsh (Eds.), *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 62–69). San Francisco, CA: Morgan Kaufmann.

Elomaa, T., & Rousu, J. (1997). On the well-behavedness of important attribute evaluation functions. In G. Grahne (Ed.), *Proceedings of the Sixth Scandinavian Conference on Artificial Intelligence* (pp. 95–106). Frontiers in Artificial Intelligence and Applications (Vol. 40). Amsterdam: IOS Press, Tokyo: Ohmsha Ltd.

Fayyad, U. M., & Irani, K. B. (1992a). The attribute selection problem in decision tree generation. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 104–110). Menlo Park, CA: AAAI Press.

Fayyad, U. M., & Irani, K. B. (1992b). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning, 8*, 87–102.

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1022–1027). San Francisco, CA: Morgan Kaufmann.

Fulton, T., Kasif, S., & Salzberg, S. (1995). Efficient algorithms for finding multi-way splits for decision trees. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference* (pp. 244–251). San Francisco, CA: Morgan Kaufmann.

Holte, R. C. (1993). Very simple classification rules perform well on most commonly used data sets. *Machine Learning, 11*, 63–90.

Howard, P. G., & Vitter, J. S. (1992). Analysis of arithmetic coding for data compression. *Information Processing and Management, 28*, 749–763.

Iba, W. F., & Langley, P. (1992). Induction of one-level decision trees. In D. Sleeman & P. Edwards (Eds.), *Machine Learning: Proceedings of the Ninth International Workshop* (pp. 233–240). San Francisco, CA: Morgan Kaufmann.

Kononenko, I. (1995). On biases in estimating multi-valued attributes. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1034–1040). San Francisco, CA: Morgan Kaufmann.

Kononenko, I., Bratko, I., & Roškar, E. (1984). *Experiments in automatic learning of medical diagnostic rules* (Technical Report). Ljubljana: Josef Stefan Institute, Faculty of Electrical Engineering and Computer Science.

Landeweerd, G. H., Timmers, T., Gelsema, E. S., Bins, M., & Halie, M. R. (1983). Binary tree versus single level tree classification of white blood cells. *Pattern Recognition, 16*, 571–577.

López de Màntaras, R. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning, 6*, 81–92.

Lubinsky, D. J. (1995). Increasing the performance and consistency of classification trees by using the accuracy criterion at the leaves. In A. Prieditis & S. Russell (Eds.), *Machine Learning: Proceedings of the Twelfth International Conference* (pp. 371–377). San Francisco, CA: Morgan Kaufmann.

Maass, W. (1994). Efficient agnostic PAC-learning with simple hypotheses. *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory* (pp. 67–75). New York: ACM Press.

Merz, C. J., & Murphy, P. M. (1996). *UCI repository of machine learning databases* (http://www.ics.uci.edu/∼mlearn/MLRepository.html). Irvine, CA: University of California, Department of Information and Computer Science.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning, 3*, 319–342.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end-games. In R.S. Michalski, J.G. Carbonell, & T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 391–411). Palo Alto, CA: Tioga.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R. (1988). Decision trees and multi-valued attributes. In J. E. Hayes, D. Michie, & J. Richards (Eds.), *Machine intelligence (Vol. 11): Logic and the acquisition of knowledge* (pp. 305–318). Oxford: Oxford University Press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann.

Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research, 4*, 77–90.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227–248.

Rissanen, J. (1989). *Stochastic complexity in statistical inquiry*. River Edge, NJ: World Scientific.

Rissanen, J. (1995). Stochastic complexity in learning. In P. Vitányi (Ed.), *Proceedings of the Second European Conference on Computational Learning Theory* (pp. 196–210). Lecture Notes in Computer Science (Vol. 904). Heidelberg: Springer-Verlag.

Rousu, J. (1996). *Constructing decision trees and lists using the MDL principle* (in Finnish). Master's thesis, Department of Computer Science, University of Helsinki, Finland.

Van de Merckt, T. (1993). Decision trees in numerical attribute spaces. *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1016–1021). San Francisco, CA: Morgan Kaufmann.

Wallace, C. S., & Freeman, P. R. (1987). Estimation and inference by compact coding. *Journal of the Royal Statistical Society (B), 49*, 240–265.

Wallace, C. S., & Patrick, J. D. (1993). Coding decision trees. *Machine Learning, 11*, 7–22.

White, A. P., & Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning, 15*, 321–329.

Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM, 30*, 520–540.

Received November 8, 1996
Accepted June 2, 1999
Final manuscript June 2, 1999