Data Mining and Knowledge Discovery, 8, 97–126, 2004

# Efficient Multisplitting Revisited: *Optima-Preserving Elimination of Partition Candidates*

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

**Editors:** Fayyad, Mannila, Ramakrishnan

*Received January 30, 2001; Revised November 6, 2002*

**Abstract.** We consider multisplitting of numerical value ranges, a task that is encountered as a discretization step preceding induction and also embedded into learning algorithms. We are interested in finding the partition that optimizes the value of a given attribute evaluation function. For most commonly used evaluation functions this task takes quadratic time in the number of potential cut points in the numerical range. Hence, it is a potential bottleneck in data mining algorithms.

We present two techniques that speed up the optimal multisplitting task. The first one aims at discarding cut point candidates in a quick linear-time preprocessing scan before embarking on the actual search. We generalize the definition of boundary points by Fayyad and Irani to allow us to merge adjacent example blocks that have the same relative class distribution. We prove for several commonly used evaluation functions that this processing removes only suboptimal cut points. Hence, the algorithm does not lose optimality.

Our second technique tackles the quadratic-time dynamic programming algorithm, which is the best schema for optimizing many well-known evaluation functions. We present a technique that dynamically—i.e., during the search—prunes partitions of prefixes of the sorted data from the search space of the algorithm. The method works for all convex and cumulative evaluation functions.

Together the use of these two techniques speeds up the multisplitting process considerably. Compared to the baseline dynamic programming algorithm the speed-up is around 50 percent on the average and up to 90 percent in some cases. We conclude that optimal multisplitting is fully feasible on all benchmark data sets we have encountered.

**Keywords:** numerical attributes, optimal partitions, convex functions, boundary points

## 1. Introduction

Numerical data is frequently encountered in the data mining task. In most cases, the used learning algorithms require splitting the numerical domains into two or more intervals (Fayyad and Irani, 1992, 1993; Ching et al., 1995; Dougherty et al., 1995; Fulton et al., 1995; Kohavi and Sahami, 1996; Liu and Setiono, 1997; Hong, 1997; Cerquides and López de Màntaras, 1997; Ho and Scott, 1997). Depending on the method, this task is encountered either in preprocessing, where the numerical domain is "discretized," or embedded within, for example, a decision tree learning algorithm.

Numerical attribute handling is critical in inductive learning. For instance, the quadratic time complexity of the popular C4.5 decision tree learning algorithm (Quinlan, 1993) for non-numeric data increases by a logarithmic factor when numerical attributes are processed (Provost et al., 1999; Utgoff, 1989). In the course of decision tree construction numerical value domains are discretized into intervals. Many practical decision tree learning algorithms—including C4.5—use in partitioning techniques that only look for the best halving of the domain (Breiman et al., 1984; Quinlan, 1993). Recently, however, considering *multisplits* of the domain with more than two intervals has gained popularity (Fayyad and Irani, 1993; Fulton et al., 1995; Auer et al., 1995; Zighed et al., 1997; Elomaa and Rousu, 1999).

Numerical domain partitioning algorithms operate on the set of candidate cut points. Therefore, their number is decisive for the efficiency. This is particularly important in algorithms that partition numerical ranges into many subsets; e.g., *greedy* (Catlett, 1991; Fayyad and Irani, 1993) and *optimal* (Fulton et al., 1995; Zighed et al., 1997; Elomaa and Rousu, 1999) multisplitters in decision tree learning, rule induction, and nearest neighbor methods. In data mining applications one has been forced to pay special attention to numerical attributes because of their inefficiency. For example, Ankerst et al. (2000) proposed an user-assisted approach to numerical domain partitioning.

This paper explores ways to enhance the efficiency of numerical attribute handling in classification learning. Previous work has shown that many evaluation functions are *well-behaved* in the sense that optimal partitions do not split class uniform intervals (Fayyad and Irani, 1992; Elomaa and Rousu, 1999). Thus, only the cut points in between class uniform intervals need to be examined in finding an optimal partition for these evaluation functions. In this paper we show that evaluating an even smaller set of cut points suffices for many functions. This set can be extracted in a linear-time scan over the sorted example sequence. The technique can be coupled, e.g., with optimal search algorithms like dynamic programming and brute-force search, and with the greedy best-first search.

As a second contribution, we present a technique for speeding up the generic dynamic programming algorithm (Fulton et al., 1995; Zighed et al., 1997; Elomaa and Rousu, 1999), which runs in time that is quadratic in the number of cut points in the range. The technique is based on dynamically—during partition candidate evaluation—removing partitions of prefixes of the data in an optima-preserving manner. It is applicable to all convex and cumulative evaluation functions, which include, e.g., the Average Class Entropy, Gini Index, and Training Set Error functions.

Our setting is *bounded-arity* discretization in which an upper bound for the number of intervals is given a priori. For many evaluation functions it is necessary to limit the number of intervals, because they would benefit from any splitting of the data and in practice one wants to have a small number of intervals for the domain at hand (see Section 4). There are many possible ways to select the upper bound; for example, the MDL principle has been used in this task (Fayyad and Irani, 1993; Pfahringer, 1995). For other possible ways to decide the upper bound see, e.g. (Catlett, 1991; Ching et al., 1995; Wu, 1996; Ho and Scott, 1997; Cerquides and López de Màntaras, 1997; Liu and Setiono, 1997). In this paper, however, we do not consider the problem of selecting the upper bound.

The remainder of this paper is organized as follows. In Section 2 the situation in partitioning a numerical domain is reviewed. We also introduce different example groupings that retain the possibility to partition the data optimally. In Section 3 we prove for six commonly-used attribute evaluation functions that a large fraction of candidate cut points can be eliminated statically, without evaluating the partition candidates. The background and algorithmic implementation of the dynamic partition pruning is presented in Section 4. The next section reports an empirical evaluation of the proposed partition candidate elimination techniques. Major reductions in the number of examined candidates and evaluation time are obtained by combining both approaches. Finally, Section 6 gives the concluding remarks of this study and outlines some future research directions.

## 2. Cut point candidates in numerical domains

In supervised learning the goal of numerical value range discretization is to find a set of cut points to partition the range into a small number of intervals that have good class coherence, which is usually measured by an evaluation function. Throughout this paper we denote by $S$ the set of examples, $k$ is the (maximum) arity of a partition, and $\biguplus_{i=1}^{k} S_i$ stands for the partitioning of $S$ into $k$ non-empty, disjoint subsets that cover the whole domain.

When splitting a set $S$ of examples on the basis of the value of an attribute $A$, there is a set of thresholds $\{T_1, \ldots, T_{k-1}\} \subseteq \mathrm{Dom}(A)$ that defines a partition $\biguplus_{i=1}^{k} S_i$ for the sample in an obvious manner:

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

where $\mathrm{val}_A(s)$ denotes the value of attribute $A$ in example $s$. The classification of an example $s$ is its value for the class attribute $C$, $\mathrm{val}_C(s)$.

The processing of a numerical attribute usually begins with sorting of the training data by the value of the attribute under consideration. This preprocessing produces a categorized version of the data, where all examples with an equal value for the attribute constitute a *bin* of examples. There are as many bins as distinct values for the attribute; this number is denoted by $V$. By $n$ we denote the total number of examples in the training set. In the worst case $V = n$.

Bin borders are the only possible cut points of the value range. In figure 1 a hypothetical sample has been arranged into bins with respect to an integer-valued attribute $A$. There are only instances of two classes, their numbers within the bins are depicted by the pair of figures inside each bin. To determine the correlation between the value of an attribute and that of the class it suffices to examine their mutual frequencies.

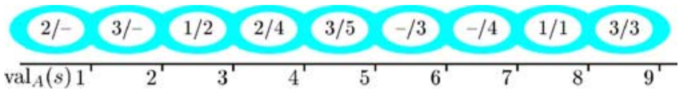

*Figure 1.* The bins in the domain of a numerical attribute $A$ in a hypothetical sample. Bin borders are the potential cut points of the domain. The class frequency distributions of the bins are denoted by the figures within them.

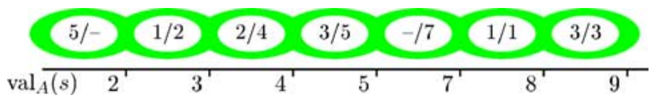

*Figure 2.* The blocks in the domain of attribute *A* in the sample of figure 1. The blocks are separated by the boundary points.

## 2.1. Boundary points

Fayyad and Irani (1992) introduced *boundary points* in analyzing the binarization technique for the *Average Class Entropy*, *ACE*, and *Information Gain*, *IG* (Quinlan, 1986, 1993), functions. They showed that for these functions the optimal binary split is always defined by a boundary point due to the convexity of the functions. Intuitively it means that these functions do not needlessly separate instances of the same class. The result reveals interesting fundamental properties of the functions and it can also be put to use in practice: only boundary points need to be examined as potential cut points to recover the optimal binary split of the data.

We restrain ourselves from giving an unnecessarily complicated formal definition for boundary points. Instead, we introduce them through *blocks* of examples and an illustration. To construct blocks of examples we merge together adjacent class uniform bins with the same class label (see figure 2). The boundary points of the example sequence are the borders of its blocks. The points thus obtained are exactly the same as those that come out from the definition of Fayyad and Irani (1992). Block construction leaves all bins with a mixed class distribution as their own blocks. We denote the number of blocks in a dimension of the data by $B$.

Recently the utility of boundary points has been extended to cover other commonly-used evaluation functions and optimal multisplitting of numerical ranges (Elomaa and Rousu, 1999). Other recent studies concerning the splitting properties of attribute evaluation functions include Breiman's (1996) research of the characteristics of ideal partitions of some impurity functions and Codrington and Brodley's (1997) study of the general requirements of well-behaved splitting functions. Similar research lines for nominal attributes are followed by Coppersmith et al. (1999).

Elomaa and Rousu (1999) defined well-behaved evaluation functions as those that always have a bounded-arity optimal multisplit on boundary points. The definition requires that for any upper bound $k$, $2 \leq k \leq B$, there exists an optimal partition of arity at most $k$ that is defined on boundary points only. All the most commonly used attribute evaluation functions fall into the category of well-behaved functions, including all convex evaluation functions (see Section 4) and some non-convex such as the *Gain Ratio*, *GR* (Quinlan, 1986) and the *Normalized Distance* measure, *ND* (López de Màntaras, 1991). However, a stricter notion of well-behavedness, requiring for each $k$ the existence of an optimal $k$-partition on boundary points, is not possessed by *GR* and *ND*.

## 2.2. Segment borders

Concentrating on boundary points represents a bias that refrains from splitting class uniform intervals. What about intervals with mixed class distributions? Can we say anything about the minima of the evaluation functions on those? In Section 3 we show that the answer is affirmative for many well-known evaluation functions.

The idea is to generalize the concept of an example block. It is easy to see that a boundary point occurs between two adjacent blocks $S'$ and $S''$ when

1. $S'$ and $S''$ have different relative class distributions, or
2. both $S'$ and $S''$ have mixed class distributions.

Let us consider the two conditions separately. The first condition seems to be necessary. Consider a situation where $S'$ and $S''$ are both class uniform consisting of instances of different classes. Then, the partition $S' \uplus S''$ separates the classes perfectly. But without the first condition there would not be a candidate cut point in between $S'$ and $S''$.

The second condition, on the other hand, does not seem as important as the first one. If the first condition does not hold, the subsets $S'$ and $S''$ have the same relative class distribution. In other words, the class distribution is independent of the numerical attribute's value in the interval $S' \cup S''$. Therefore, there is no reason to separate the two sets with a cut point, independent of whether the blocks have mixed class distributions or not.

By combining adjacent bins with an equal relative class distribution we obtain *example segments* (see figure 3). Segments group together adjacent mixed-distribution bins that have equal relative class distribution. Also adjacent class uniform bins fulfill this condition; hence, uniform blocks are a special case of segments and segment borders are a subset of boundary points.

Bins, blocks, and segments can all be identified in the same single scan over the sorted sample. Thus, taking advantage of them only incurs a linear computational cost. It is majorized by the usually unavoidable $O(n \log n)$ time requirement of sorting. In multiway partitioning the partition candidate evaluation often takes at least quadratic, even exponential, time in the number of candidate cut points. Our subsequent tests demonstrate up to 75% savings in time consumption by preprocessing the data into segments instead of running the algorithms on the example bins.

In the next section we show that it suffices to examine segment borders in optimizing the value of the best-known attribute evaluation functions. Hence, the changes in class distribution, rather than absolute impurities of the subsets, define the potential locations of the optimal cut points (cf. López de Màntaras, 1991). Two of the examined functions are non-convex. Thus, the property of splitting on segment borders is not only a consequence of the convexity of a function.

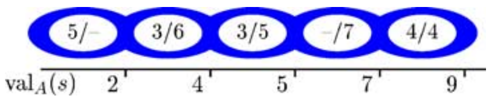

*Figure 3.* The segments and their borders in the domain of attribute $A$ in the sample of figure 1. Segment borders are a subset of the boundary points.

## 3. Most common evaluation functions minimize on segment borders

The time requirement of numerical range partitioning depends on the number of candidate cut points. Therefore, reducing their number makes numerical attribute handling more efficient. Cut points, however, cannot be removed arbitrarily; we want to preserve the possibility to find the optima of the value range as determined by some evaluation function. In this section we show that many commonly-used attribute evaluation functions have their local optima on segment borders. Hence, partitions with intra-segment cut points can be disregarded.

### *3.1. Proof setting*

All the following proofs have the same setting. The sample $S$ contains three subsets, $P$, $Q$, and $R$ with class frequency distributions

$$p = \sum_{j=1}^{m} p_j, \quad q = \sum_{j=1}^{m} q_j, \quad \text{and} \quad r = \sum_{j=1}^{m} r_j,$$

where $p$ is the total number of examples in $P$ and $p_j$ is the number of instances of class $j$ in $P$. Furthermore, $m$ is the number of classes. The notation is similar also for $Q$ and $R$. By $w_j$ we denote the proportion of instances of class $j$ in $Q$; $w_j = q_j/q \in [0, 1]$.

We consider the $k$-ary partition $\biguplus_{i=1}^{k} S_i$ of the sample $S$, where subsets $S_h$ and $S_{h+1}$ consist of the set $P \cup Q \cup R$, so that the split point is inside $Q$, on the border of $P$ and $Q$, or that of $Q$ and $R$ (see figure 4). Let $\ell$ be an integer, $0 \leq \ell \leq q$. We assume that splitting the set $Q$ so that $\ell$ examples belong to $S_h$ and $q - \ell$ to $S_{h+1}$ results in identical class frequency distributions for both subsets of $Q$ regardless of the value of $\ell$. In other words, for all $j$ and $\ell$ it holds that $q_j(\ell) = w_j \ell$, where $q_j(\ell)$ is the frequency of class $j$ in $S_h$.

The underlying idea in the proof setting is that $Q$ is a segment of examples; no matter from which cut point within it we divide $Q$ into two, the resulting parts will always have the same relative class frequency distribution. The objective of the proofs is to show that the evaluation functions that are considered receive their optimal value on a segment border rather than on a cut point within a segment.

The proofs treat the evaluation functions and their component functions as continuous in $[0, q]$ and twice differentiable, even though they are defined to be discrete. Observe

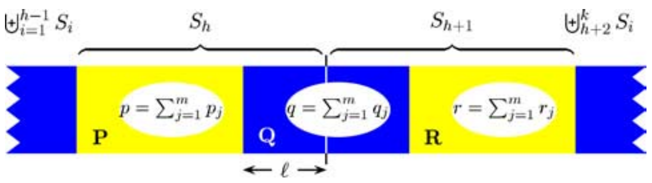

*Figure 4.* The following proofs consider partitioning of the example set $P \cup Q \cup R$ into two subsets $S_h$ and $S_{h+1}$ within $Q$. No matter where, within $Q$, the cut point is placed, equal class distributions result.

that this causes no harm, since we only consider proving the *absence* of certain local extrema.

The proofs show in the multisplitting situation that the cut point in between two arbitrarily chosen partition subsets $S_h$ and $S_{h+1}$ is on a segment border. The remaining partition subsets are not affected by the placement of the cut point within $S_h \cup S_{h+1}$. Therefore, their impact usually disappears when the proof involves differentiation of the function.

We take all logarithms in this paper to be natural logarithms; it makes the manipulation and notation simpler. It is easy to check that our proofs can be worked through with binary logarithms as well.

## *3.2. Average class entropy*

The evaluation functions that as used in class-driven partitioning tasks are many. Perhaps the most important of them is the Average Class Entropy

$$ACE\left(\biguplus_{i=1}^{k} S_i\right) = \frac{1}{|S|}\sum_{i=1}^{k} |S_i| H(S_i) = \frac{1}{n}\sum_{i=1}^{k} |S_i| H(S_i),$$

where $H$ is the entropy function:

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

in which $m$ denotes the number of classes and $P(C, S)$ stands for the proportion of examples in $S$ that have class $C$. Since the function $x \log x$ is convex (Cover and Thomas, 1991), the entropy function is concave.

Many other evaluation functions use *ACE* as their building block. Such functions include, e.g., *IG*, *GR*, and *ND*. For instance, *IG* is defined as

$$IG\left(\biguplus_{i=1}^{k} S_i\right) = H(S) - ACE\left(\biguplus_{i=1}^{k} S_i\right).$$

The entropy of the data set $S$ prior to partitioning it by any of the attributes, $H(S)$, is constant with respect to the dimensions of the data. Therefore, the maximum value of *IG* coincides with the minimum value of *ACE*.

**Theorem 1.** *The Average Class Entropy optimal partitions are defined on segment borders.*

**Proof:** Let $L(\ell)$ denote the value of $\sum_{i=1}^{h} |S_i| H(S_i)$ when $S_h$ contains $P$ and the first $\ell$ examples from $Q$, and $R(\ell)$ the value $\sum_{i=h+1}^{k} |S_i| H(S_i)$ in the same situation. To simplify and unify the notation, we write $t = r + q$ and $t_j = r_j + q_j$ for all $j \in \{1, \ldots, m\}$.

Now,

$$\begin{aligned}
L(\ell) &= \sum_{i=1}^{h-1} |S_i| H(S_i) - \sum_{j=1}^{m} (p_j + w_j \ell) \log \frac{p_j + w_j \ell}{p + \ell} \\
&= \sum_{i=1}^{h-1} |S_i| H(S_i) + \log(p + \ell) \sum_{j=1}^{m} (p_j + w_j \ell) \\
&\quad - \sum_{j=1}^{m} (p_j + w_j \ell) \log(p_j + w_j \ell) \\
&= \sum_{i=1}^{h-1} |S_i| H(S_i) + (p + \ell) \log(p + \ell) - \sum_{j=1}^{m} (p_j + w_j \ell) \log(p_j + w_j \ell)
\end{aligned}$$

and, similarly,

$$R(\ell) = (t - \ell) \log(t - \ell) - \sum_{j=1}^{m} (t_j - w_j \ell) \log(t_j - w_j \ell) + \sum_{i=h+2}^{k} |S_i| H(S_i).$$

Since the first sum in the formula of $L(\ell)$ is independent of the placing of the $h$-th cut point, it differentiates to zero and the second derivative of $L(\ell)$ is

$$\begin{aligned}
L''(\ell) &= \frac{\mathrm{d}}{\mathrm{d}\ell} \left( \log(p + \ell) - \sum_{j=1}^{m} w_j \log(p_j + w_j \ell) \right) \\
&= \frac{1}{p + \ell} - \sum_{j=1}^{m} \frac{w_j^2}{p_j + w_j \ell} \\
&= \frac{1}{p + \ell} - \sum_{j=1}^{m} \frac{w_j}{p_j / w_j + \ell}.
\end{aligned}$$

The remaining sum can be interpreted as the weighted arithmetic mean of the terms $1/(p_j/w_j + \ell)$, $1 \leq j \leq m$, and be bound, by the arithmetic-harmonic mean inequality (Hardy et al., 1934; Meyer, 1984), from below by the corresponding harmonic mean

$$\sum_{j=1}^{m} w_j \frac{1}{p_j / w_j + \ell} \geq \frac{1}{\sum_{j=1}^{m} w_j (p_j / w_j + \ell)} = \frac{1}{\sum_{j=1}^{m} (p_j + w_j \ell)} = \frac{1}{p + \ell}.$$

Thus, $L''(\ell) \leq 0$.

Correspondingly, the second derivative of $R(\ell)$ can be approximated by majorizing the second term by the harmonic mean

$$R''(\ell) = \frac{1}{t - \ell} - \sum_{j=1}^{m} \frac{w_j^2}{t_j - w_j \ell} \leq \frac{1}{t - \ell} - \frac{1}{\sum_{j=1}^{m} w_j (t_j / w_j - \ell)} = 0.$$

Hence, we have shown that the second derivative of *ACE*,

$$ACE''(\ell) = \frac{L''(\ell) + R''(\ell)}{|S|},$$

is non-positive for all $\ell$. This forces all local extrema of *ACE* within $Q$ to be maxima. □

## 3.3. Information gain

Since *IG* is a simple modification of *ACE*, proving that it does not partition within segments is straightforward.

**Theorem 2.** *The Information Gain optimal partitions are defined on segment borders.*

**Proof:** The Information Gain of the partition $\biguplus_{i=1}^{k} S_i$, when the $h$-th cut point is placed after the $\ell$-th example of $Q$, is

$$IG(\ell) = H(S) - ACE(\ell).$$

The constant term $H(S)$ that does not depend on the value of $\ell$ differentiates to zero. Therefore, $IG'(\ell) = -ACE'(\ell)$ and the second derivative of *IG* is $IG''(\ell) = -ACE''(\ell)$. From the proof of Theorem 1 we know that $ACE''(\ell) \leq 0$, which means that $IG''(\ell) \geq 0$. Hence, *IG* cannot have a local maximum within the segment $Q$. □

## 3.4. Gain ratio

To penalize against *IG*'s excessive favoring of multi-valued nominal attributes and multi-splitting numerical attribute value ranges (Quinlan, 1986, 1988) suggested dividing the *IG* score of a partition by the term

$$\kappa\left(\biguplus_{i=1}^{k} S_i\right) = -\sum_{i=1}^{k} \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|}.$$

The resulting evaluation function is the Gain Ratio

$$GR\left(\biguplus_{i=1}^{k} S_i\right) = IG\left(\biguplus_{i=1}^{k} S_i\right) \Big/ \kappa\left(\biguplus_{i=1}^{k} S_i\right).$$

The nominator of this formula—the *IG* function—was already inspected above. Therefore, the following proof concentrates on the denominator $\kappa$.

**Theorem 3.** *The Gain Ratio optimal partitions are defined on segment borders.*

**Proof:** The denominator $\kappa$ of the *GR* formula in our proof setting is

$$\kappa(\ell) = \kappa\left(\biguplus_{i=1}^{h-1} S_i\right) + K(\ell) + \kappa\left(\biguplus_{i=h+2}^{k} S_i\right),$$

where

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

Since the first and the last term of $\kappa(\ell)$ are independent of placing $\ell$ within $Q$, the first derivative of $\kappa(\ell)$ with respect to $\ell$ is $\kappa'(\ell) = K'(\ell)$ and its second derivative is

$$\begin{aligned} \kappa''(\ell) &= \frac{\mathrm{d}}{\mathrm{d}\ell}\left(\frac{1}{|S|}(-\log(p+\ell) - \log(t-\ell))\right) \\ &= \frac{1}{|S|}\left(\frac{-1}{p+\ell} + \frac{-1}{t-\ell}\right) \\ &< 0. \end{aligned} \tag{1}$$

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

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

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

$$\begin{aligned} N'(\ell) &= IG''(\ell)\kappa(\ell) + \kappa'(\ell)IG'(\ell) - \kappa''(\ell)IG(\ell) - \kappa'(\ell)IG'(\ell) \\ &= IG''(\ell)\kappa(\ell) - \kappa''(\ell)IG(\ell) \\ &\geq 0, \end{aligned}$$

because for each $0 < \ell < q$ it holds by definition that $\kappa(\ell) > 0$ and $IG(\ell) \geq 0$. Furthermore, by Theorem 2 we know that $IG''(\ell) \geq 0$ and by Eq. (1) that $\kappa''(\ell) < 0$.

Now the second derivative of $GR(\ell)$ can be expressed as

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

Let $\psi \in ]0, q[$ be a potential location for a local maximum of $GR$, 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 $\psi$ was chosen arbitrarily, we have shown that $GR(\ell)$ can only obtain its maximum value when the threshold is placed at either of the segment borders, where $\ell = 0$ and $\ell = q$, respectively. □

## 3.5. *Normalized distance measure*

Normalized Distance Measure (López de Màntaras, 1991) can be expressed with the help of $IG$ as

$$ND\left(\biguplus_{i=1}^{k} S_i\right) = 1 - IG\left(\biguplus_{i=1}^{k} S_i\right) \Big/ \lambda\left(\biguplus_{i=1}^{k} S_i\right),$$

where

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

in which $M(j, S)$ stands for the number of instances of class $j$ in the set $S$.

The following proof concerns, instead, the function

$$ND_1\left(\biguplus_{i=1}^{k} S_i\right) = 1 - ND\left(\biguplus_{i=1}^{k} S_i\right) = IG\left(\biguplus_{i=1}^{k} S_i\right) \Big/ \lambda\left(\biguplus_{i=1}^{k} S_i\right),$$

from which the claim directly follows for $ND$.

The $ND_1$ formula resembles that of $GR$. Therefore, the proof outline is also the same.

**Theorem 4.** *The Normalized Distance Measure optimal partitions are defined on segment borders.*

**Proof:** Let $L(\ell)$ denote the value of $\lambda(\biguplus_{i=1}^{h} S_i)$ and $R(\ell)$ the value $\lambda(\biguplus_{i=h+1}^{k} S_i)$.

$$\begin{aligned} L(\ell) &= \lambda\left(\biguplus_{i=1}^{h-1} S_i\right) - \sum_{j=1}^{m} \frac{p_j + w_j\ell}{|S|} \log \frac{p_j + w_j\ell}{|S|} \\ &= \lambda\left(\biguplus_{i=1}^{h-1} S_i\right) + \frac{1}{|S|}\left((p+\ell)\log|S| - \sum_{j=1}^{m}(p_j + w_j\ell)\log(p_j + w_j\ell)\right) \end{aligned}$$

and

$$R(\ell) = \frac{1}{|S|}\left((t-\ell)\log|S| - \sum_{j=1}^{m}(t_j - w_j\ell)\log(t_j - w_j\ell)\right) + \lambda\left(\biguplus_{i=h+2}^{k} S_i\right).$$

The second derivative of $L(\ell)$ is given by

$$\begin{aligned} L''(\ell) &= \frac{\mathrm{d}}{\mathrm{d}\ell}\frac{1}{|S|}\left(\log|S| - \sum_{j=1}^{m} w_j \log(p_j + w_j\ell)\right) \\ &= \frac{-1}{|S|}\sum_{j=1}^{m}\frac{w_j^2}{p_j + w_j\ell} \\ &\le 0, \end{aligned}$$

because $|S|$, $w_j$, $p_j$, and $\ell$ are all non-negative.

Correspondingly, the second derivative of $R(\ell)$ is

$$R''(\ell) = \frac{-1}{|S|}\sum_{j=1}^{m}\frac{w_j^2}{t_j - w_j\ell} \le 0.$$

Thus, we have proved that the second derivative of $\lambda$, $\lambda''(\ell) = L''(\ell) + R''(\ell)$ is non-positive for all $\ell$.

The proof for $ND_1$ is easy to complete similarly as the proof for the Gain Ratio. Thus, the local extrema of $ND_1$ within $Q$ are minima, which makes them local maxima of $ND(\ell) = 1 - ND_1(\ell)$. Hence, Normalized Distance measure does not obtain its minimum value within a segment. □

## 3.6. Gini index

The *Gini Index* of diversity, or the *Quadratic Entropy*, (Breiman et al., 1984; Breiman, 1996) is defined as

$$GI\left(\biguplus_{i=1}^{k} S_i\right) = \sum_{i=1}^{k}\frac{|S_i|}{|S|} gini(S_i),$$

in which *gini* is the impurity measure

$$gini(S) = \sum_{j=1}^{m} P(C_j, S)(1 - P(C_j, S)) = 1 - \sum_{j=1}^{m} P^2(C_j, S),$$

where $P(C, S)$ again denotes the proportion of instances of class $C$ in the data $S$. For the properties of Gini Index see also Coppersmith et al. (1999).

**Theorem 5.** *The Gini Index optimal partitions are defined on segment borders.*

**Proof:** Let $L(\ell)$ denote the value of $\sum_{i=1}^{h} |S_i| gini(S_i)$ when $S_h$ contains $P$ and the first $\ell$ examples from $Q$. Respectively, $R(\ell)$ is the value $\sum_{i=h+1}^{k} |S_i| gini(S_i)$. Now,

$$\begin{aligned} L(\ell) &= \sum_{i=1}^{h-1} |S_i| gini(S_i) + \sum_{j=1}^{m} (p_j + w_j \ell)\left(1 - \frac{p_j + w_j \ell}{p + \ell}\right) \\ &= \sum_{i=1}^{h-1} |S_i| gini(S_i) + (p + \ell) - \sum_{j=1}^{m} \frac{(p_j + w_j \ell)^2}{p + \ell}. \end{aligned}$$

The second derivative of $L(\ell)$ is:

$$\begin{aligned} L''(\ell) &= \frac{\mathrm{d}}{\mathrm{d}\ell}\left(1 - \sum_{j=1}^{m} \frac{2(p_j + w_j\ell)w_j(p+\ell) - (p_j + w_j\ell)^2}{(p+\ell)^2}\right) \\ &= \frac{\mathrm{d}}{\mathrm{d}\ell}\left(1 - \sum_{j=1}^{m} \frac{2(p_j + w_j\ell)w_j}{(p+\ell)} + \sum_{j=1}^{m} \frac{(p_j + w_j\ell)^2}{(p+\ell)^2}\right) \\ &= -\sum_{j=1}^{m} \frac{2w_j^2(p+\ell) - 2(p_j + w_j\ell)w_j}{(p+\ell)^2} \\ &\quad + \sum_{j=1}^{m} \frac{2(p_j + w_j\ell)w_j(p+\ell)^2 - 2(p+\ell)(p_j + w_j\ell)^2}{(p+\ell)^4} \\ &= \frac{1}{(p+\ell)^3} \sum_{j=1}^{m} \big(2(p_j + w_j\ell)w_j(p+\ell) - 2w_j^2(p+\ell)^2 \\ &\quad + 2(p_j + w_j\ell)w_j(p+\ell) - 2(p_j + w_j\ell)^2\big) \\ &= \frac{-2}{(p+\ell)^3} \sum_{j=1}^{m} (p_j + w_j p)^2 \\ &\le 0. \end{aligned}$$

By symmetry we determine that $R''(\ell) \le 0$ as well. Thus, $GI''(\ell) = (L''(\ell) + R''(\ell))/|S| \le 0$ and, therefore, $GI$ does not obtain its minimum value within the segment $Q$. □

## 3.7. *Training set error*

Sometimes, instead of even trying to minimize the error of future instances, one is simply satisfied with minimizing the error of the already available training examples. We call this evaluation function as *Training Set Error*.

The *majority* class of sample $S$ is its most frequently occurring class:

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

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 is the number of training instances falsely classified in the partition. For a partition $\biguplus S_i$ of $S$ it is defined as

$$TSE\left(\biguplus_{i=1}^{k} S_i\right) = \sum_{i=1}^{k} \delta(S_i).$$

The number of instances in $S_h$ from other classes than $j$, $(p - p_j) + (1 - w_j)\ell$, is linearly increasing for any $j$, since the first term is constant and $0 \leq w_j \leq 1$. Respectively, in $S_{h+1}$ the number of those instances, $(r - r_j) + (1 - w_j)(q - \ell)$, decreases with increasing $\ell$.

In our proof setting, the majority class of $S_h$ depends on the growth rates of classes in $Q$ and the number of their instances in $P$. First, when $\ell = 0$, the majority class of $P$, say $u$, is also the majority class of $S_h$. Subsequently an other class $x$, with strictly larger proportion of instances in $Q$, $w_x > w_u$, may become the majority class of $S_h$ (see figure 5). Observe that $p_x \leq p_u$. As a combination of non-decreasing functions, $\delta(S_h)$ is also non-decreasing.

**Theorem 6.** *The Training Set Error optimal partitions are defined on segment borders.*

**Proof:** Let us examine the value of $TSE(l) = \delta(S_h) + \delta(S_{h+1})$ at an arbitrary cut point $\ell = l$, $0 \leq l \leq q$. Let $u$ and $v$ be the majority classes of $S_h$ and $S_{h+1}$, respectively, in this situation. Then,

$$TSE(l) = (p - p_u) + (1 - w_u)l + (r - r_v) + (1 - w_v)(q - l).$$

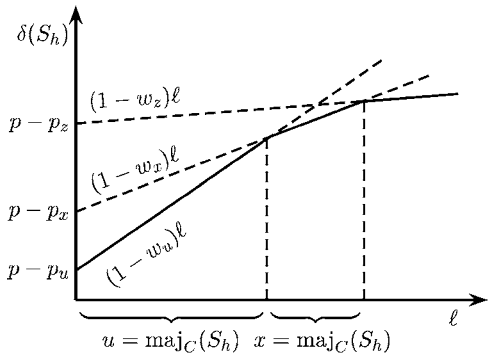

*Figure 5.* In $Q$ the number of instances of other classes than $j$, $(1 - w_j)\ell$, grows linearly with increasing $\ell$ for all $j$. The number of disagreements $\delta(S_h)$ is convex.

We now show that a smaller Training Set Error is obtained by moving the cut point to the left or to the right from $l$. There are four possible scenarios for the changes of majority classes of $S_h$ and $S_{h+1}$ when the cut point is moved: (i) neither of them changes, only the majority class of (ii) $S_h$ or (iii) $S_{h+1}$ changes, or (iv) both of them change. Let $x$ and $y$, when needed, be the new majority classes of $S_h$ and $S_{h+1}$, respectively.

Assume, for now, that $w_u \leq w_v$. Let us consider the four scenarios mentioned when moving the cut point one example to the left.

(i) Now, $\text{maj}_C(S_h) = u$ and $\text{maj}_C(S_{h+1}) = v$ independent of whether $\ell = l - 1$ or $\ell = l$. Then,

$$\begin{aligned} TSE(l-1) &= (p - p_u) + (1 - w_u)(l-1) + (r - r_v) + (1 - w_v)(q - l + 1) \\ &= TSE(l) + w_u - w_v \\ &\leq TSE(l), \end{aligned}$$

because $w_u - w_v \leq 0$ by the assumption.

(ii) The majority class of $S_h$ becomes $x$ and $v$ remains to be the majority class of $S_{h+1}$. Then, the error in $S_h$ with respect to $x$ must be at most the same as that with respect to $u$; $(p - p_x) + (1 - w_x)(l-1) \leq (p - p_u) + (1 - w_u)(l-1)$. Hence,

$$\begin{aligned} TSE(l-1) &= (p - p_x) + (1 - w_x)(l-1) + (r - r_v) + (1 - w_v)(q - l + 1) \\ &\leq (p - p_u) + (1 - w_u)(l-1) + (r - r_v) + (1 - w_v)(q - l + 1) \\ &= TSE(l) + w_u - w_v, \end{aligned}$$

Since $w_u \leq w_v$ by the assumption, we have shown that $TSE(l-1) \leq TSE(l)$.

(iii) The majority class of $S_h$ remains to be $u$ and $y$ becomes the majority class of $S_{h+1}$. Observe that then $(r - r_y) + (1 - w_y)(q - l + 1) \leq (r - r_v) + (1 - w_v)(q - l + 1)$, by $y$ being the majority class of $S_{h+1}$. Thus,

$$TSE(l-1) \leq TSE(l) + w_u - w_v \leq TSE(l).$$

(iv) If both majority classes change, then by combining (ii) and (iii) we see that $TSE(l-1) \leq TSE(l)$.

Hence, in all scenarios a smaller value of $TSE$ is obtained by moving the cut point. Similarly, if $w_u \geq w_v$ we can obtain a smaller training set error for $S_h \uplus S_{h+1}$ by sliding the cut point forward in $Q$.

In any case, the cut point can be slid all the way to one of the borders of $Q$. Because $l$ was chosen arbitrarily and

$$TSE\left(\biguplus_{i=1}^{k} S_i\right) = \sum_{i=1}^{h-1} \delta(S_i) + TSE(l) + \sum_{i=h+2}^{k} \delta(S_i),$$

we have proved the claim. $\square$

### *3.8. Discussion*

Given a numerical value range that has been sorted into (ascending) order, we can preprocess it in linear time into segments. If we are using one of the common evaluation functions examined above, we only need to examine the segment borders to find an optimal multisplit of the value range.

An alternative proof for the *ACE*, *IG*, and *GI* functions is given by Elomaa and Rousu (2003). The proof applies to concave evaluation functions, hence the proofs for the non-concave *GR* and *ND* functions are not covered by it. In addition, it is shown that no segment border can be ignored without risking the optimality.

Incidentally, the result above is not the best possible for *TSE*. Elomaa and Rousu (2003) show that *TSE*-optimal cut points lie on a subset of segment borders, called *alternation points*.

Even though all of the above-examined functions have optimal multisplits defined on segment borders, the time required to compute the function values varies. Those functions that take the form of a weighted sum (*ACE*, *IG*, *GI*, and *TSE*) can be optimized in quadratic time by using dynamic programming (Fulton et al., 1995; Zighed et al., 1997; Elomaa and Rousu, 1999). In fact, *TSE* can be optimized even in linear time (Maass, 1994; Fulton et al., 1995; Auer, 1997; Birkendorf, 1997), which is a consequence of the function being a combination of several monotonic functions (Elomaa and Rousu, 2001). For the remaining functions—*GR* and *ND*—no better optimization method than the exponential-time exhaustive search is known. In any case, linear, quadratic, and exponential search algorithms can all take advantage of example segments. The benefits of the preprocessing are, of course, the larger the more demanding the subsequent search phase is.

## 4. Dynamic pruning of partition candidates

As shown above, preprocessing of the data makes it possible to reduce the number of partition candidates that need to be examined in order to find the optimal multisplit of the data. However, even after the static pruning of partition candidates, there still are many candidates out of which only a small part are promising.

In this section we present a technique to improve the efficiency of the quadratic-time generic dynamic programming algorithm, which is the fastest known algorithm for the optimization of, e.g., *ACE*, *IG*, and *GI* functions. As mentioned above, no efficient optimization scheme is known for the non-cumulative *GR* and *ND*.

The technique is based on removing, during the left-to-right scan over the data, such partitions of the prefixes of the data that cannot be part of an optimal multisplit of the whole data. A test for such condition is developed that requires only a constant time per partition of a prefix. Depending on the stage of the search, when the removal happens, up to $O(G)$ candidate comparisons are avoided, where $G$ is the number of example segments in the range. The technique is applicable to all convex and cumulative functions; that is, functions that take the form

$$F\left(\biguplus_{i=1}^{k} S_i\right) = \sum_{i=1}^{k} \frac{|S_i|}{|S|} Imp(S_i),$$

for some convex impurity function *Imp*. The functions *ACE*, *IG*, *GI*, and *TSE* all belong to this group of functions.

### *4.1. Convex evaluation functions*

Many, though not all, of the most widely used attribute evaluation functions are either *convex* or *concave* (Breiman, 1996; Hickey, 1996; Codrington and Brodley, 1997; Elomaa and Rousu, 1999). Some functions are not convex themselves, but their values can be computed in constant or, at worst, linear time from the value of a convex component of the function. It is common to call also concave functions as convex, since a function that is concave upwards is at the same time convex downwards.

*Definition 1.* A function $f(x)$ is said to be *convex* over an interval $(a, b)$ if for every $x_1, x_2 \in (a, b)$ and $0 \leq \rho \leq 1$,

$$f(\rho x_1 + (1-\rho)x_2) \leq \rho f(x_1) + (1-\rho) f(x_2).$$

A function $f$ is *concave* if $-f$ is convex.

Let $X$ be a variable with domain $\mathcal{X}$. Let $E$ denote the expectation. In the discrete case $EX = \sum_{x\in\mathcal{X}} p(x)x$, where $p(x) = \mathbf{Pr}\{X = x\}$, and $EX = \int x f(x) dx$ in the continuous case.

**Theorem 7** (*Jensen's inequality*; Cover and Thomas, 1991). *If $f$ is a convex function and $X$ is a random variable, then*

$$Ef(X) \geq f(EX).$$

As mentioned already, many commonly-used attribute evaluation functions are convex and, thus, fulfill Jensen's inequality. Out of the functions examined in Section 3 only *GR* and *ND* are non-convex. In the following we show how Jensen's inequality can be utilized to dynamically prune unfruitful partition candidates from further evaluation. In practice, significant speed-up is obtained, although the worst-case asymptotical time-complexity is not helped by the technique (Elomaa and Rousu, 2001).

The convexity of an evaluation function is known to manifest itself in two ways. First, adding cut points to a partition always decreases the impurity, which is a direct consequence of Jensen's inequality. For this reason, in practice, the arity of the partition needs to be bounded, either a priori or by using some penalizing term. Second, if the evaluation function is convex in between potential cut points, then the function cannot obtain its optimal value within that segment of examples (cf. Section 3).

Figure 6 depicts these effects for the data of figure 3: the vertical axis is the value of the *ACE* function relative to the entropy of the data and the horizontal axis corresponds to the locations of cut points in the data. We see that the curve is at its highest in the end points

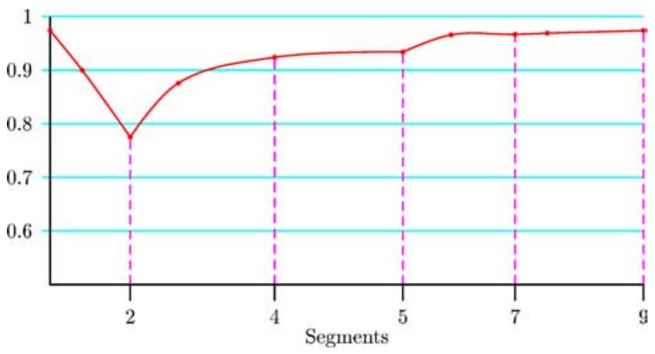

*Figure 6.* The two manifestations of convexity of the *ACE* function on the data of figure 3. All cut points lead to improved partition quality. Best values for segments are obtained on borders.

of the range. This follows from the Jensen's inequality: A (non-trivial) partition of the data can only decrease entropy.

Furthermore, there are five potential cut points in between which the curve takes a concave (downward convex) form. The cut points are exactly the segment borders.

## 4.2. *Pruning partition candidates dynamically with convex evaluation functions*

Jensen's inequality does not restrict the probability distribution underlying the expectation. Hence, for a concave function $f$ it holds that

$$\sum_i \alpha_i f(t_i) \leq f\left(\sum_i \alpha_i t_i\right) \tag{2}$$

for $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$.

Typically, partition ranking functions give each interval a score using an other function, which tries to estimate the class coherence of the interval. A common class of such functions are the *impurity* functions (Breiman et al., 1984). The interval scores are weighted relative to the sizes of the intervals. Thus, a common form of an evaluation function $F$ is

$$F\left(\biguplus_i S_i\right) = \sum_i \frac{|S_i|}{|S|} Imp(\mu_{S_i}), \tag{3}$$

where *Imp* is an impurity function and $\mu_S$ is the class frequency distribution of the set $S$. Now, $|S_i|/|S| \geq 0$ and $\sum_i(|S_i|/|S|) = 1$. If the impurity function *Imp* is concave, then by Eq. (2):

$$F\left(\biguplus_i S_i\right) = \sum_i \frac{|S_i|}{|S|} Imp(\mu_{S_i}) \leq Imp\left(\sum_i \frac{|S_i|}{|S|}\mu_{S_i}\right) = F(S), \tag{4}$$

in which $F(S)$ is the score of the unpartitioned data. Observe that, since *Imp* is concave, any splitting of the data can only decrease the value of $F$. Thus splitting on all cut points will lead to best score. This fact can be utilized in estimating the goodness scores of partitions.

**Theorem 8.** *Let $F$ be the evaluation function defined in Eq.* 3 *and let Imp be a concave impurity function. Let $S$ be a sequence of examples consisting of consecutive intervals $S_1$, $S_2, \dots, S_l$. Let $P_1$ be, for some fixed $k \geq 2$, a $(k-1)$-partition for the interval $S_1$ and $P_2$ be a $(k-1)$-partition for $S_1 \cup S_2$. If*

$$|S_1 \cup S_2| F(P_2) - |S_1| F(P_1) - |S_2| Imp(S_2) \leq 0, \tag{5}$$

*then for any example set $S_*$*

$$F(P_2 \uplus S_*) \leq F(P_1 \uplus \{S_2 \cup S_*\}).$$

**Proof:** Let us consider different $k$-partitions of $S_1 \cup S_2 \cup S_*$, where $S_*$ is a combination of any number of intervals (bins) immediately following $S_2$. $P_*^1 = P_1 \uplus (S_2 \cup S_*)$ and $P_*^2 = P_2 \uplus S_*$ are two $k$-partitions of $S_1 \cup S_2 \cup S_*$ (see figure 7). Assume that Inequality 5 holds.

According to Inequality 4

$$\begin{aligned} |S| F(P_*^1) &= |S_1| F(P_1) + |S_2 \cup S_*| Imp(S_2 \cup S_*) \\ &\geq |S_1| F(P_1) + |S_2| Imp(S_2) + |S_*| Imp(S_*) \end{aligned}$$

and

$$|S| F(P_*^2) = |S_1 \cup S_2| F(P_2) + |S_*| Imp(S_*).$$

The difference of these two candidates can be bound from above by Inequality 5

$$|S| F(P_*^2) - |S| F(P_*^1) \leq |S_1 \cup S_2| F(P_2) - |S_1| F(P_1) - |S_2| Imp(S_2) \leq 0,$$

from where the claim follows by dividing by $|S|$. □

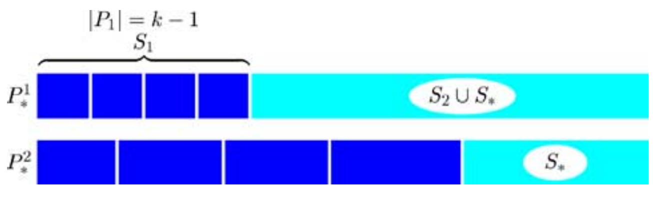

*Figure 7.* $P_1$ and $P_2$ are two $(k-1)$-partitions of the prefixes of the data set. They can be extended into $k$-partitions $P_*^1$ and $P_*^2$, respectively, for a larger sample by augmenting a new interval to them.

Inequality 5 is independent of $S_*$. Hence, we can test the bound for empty $S_*$ and, if the pruning condition is satisfied, subsequently drop all partitions containing $P_1$ from further consideration.

This pruning does not improve the asymptotic time requirement of partition evaluation, but its practical significance is high. Subsequent experiments show that on the average as much as half of the partition candidates can be omitted.

### *4.3. An algorithm for optimal partitioning*

As noted above, convex evaluation functions always obtain their optimal value by splitting on all possible cut points. A partition with the same score for the functions of our interest can be obtained by splitting on all segment borders. However, in practice one wants to have partitions with a relatively small number of subsets. Therefore, bounded-arity partitioning is of practical interest. We now apply the above-presented pruning approach to finding optimal bounded-arity partitions.

We incorporate the candidate pruning method to the quadratic-time search algorithm (Elomaa and Rousu, 1999). The algorithm uses a dynamic programming scheme similar to that suggested by Fulton et al. (1995), but works on intervals of examples (bins, blocks, or segments) rather than on individual examples. The intervals are extracted in a linear-time preprocessing phase. The search algorithm takes in as parameters a preprocessed sequence of intervals, an evaluation function $g$, and an upper limit for the arity, i.e., the number of subsets of the partition.

Each interval $I_i$ is represented by its class frequency distribution $\mu_i$. In addition, the index of the last example of the interval is stored to facilitate extracting the result. The function $f(\vec{\mu})$ is required to return $|S| \mathit{Imp}(S)$ for some concave function *Imp*, where $\vec{\mu}$ is the class frequency distribution of $S$.

The search algorithm (Table 1) scans the intervals $I_1, \ldots, I_G$ from left to right and stores the intermediate results into four arrays: Array $P$ stores the costs of the best multisplits; $P[i, k]$ is the minimum cost obtained when the $i$ first 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 segment that contains the rightmost cut point of the multisplit having the cost $P[i, k]$. Array $N$ stores the search space of remaining partition candidates in linked lists; $N[i, k-1] = j$ denotes that the next $(k-1)$-partition to be considered as the prefix of an optimal $k$-split, after the best $(k-1)$-split of the segments $I_1 \cup \cdots \cup I_i$, is the optimal $(k-1)$-split of the segments $I_1, \ldots, I_j$. The lists are pruned during the search by checking against the bound derived in the previous section. The fourth array $cost$ is used to eliminate the repeated calculation of interval impurities.

Lines 5–14 compute the best $k$-split of $I_1, \ldots, I_i$ for each $k$. Lines 9–13 scan the list of remaining candidate $(k-1)$-splits. Line 10 checks whether the current split be pruned from further comparisons and, if so, prunes it. Otherwise (lines 11 and 12) the candidate could be the optimal one.

Since the class distribution of the interval $I_j$ is not needed, as such, after the scan has passed it, the algorithm reuses the space: at point $i$, each $\mu_j$, $j \leq i$, represents the class

*Table 1.* The search algorithm for multisplits.

**procedure** *Search*($\vec{\mu}$, *f*, *arityMax*)

**input:** $\vec{\mu} = \{\mu_1, \ldots, \mu_G\}$ contains the class frequency distributions of intervals $I_1, \ldots, I_G$, $f(\mu_j) = |S_j|Imp(S_j)$, where *Imp* is a concave function, *arityMax* is the maximum number of intervals allowed in the split.

```
(1)  for $i \leftarrow 1$ to $G$ {
(2)      for $j \leftarrow 1$ to $i - 1$ { $\mu_j \leftarrow \mu_j + \mu_i$; $cost[j] \leftarrow f(\mu_j)$; }
(3)      $P[i, 1] \leftarrow cost[1]$; $N[i, 1] \leftarrow i - 1$;
(4)      $limit \leftarrow arityMax$; if $(i < G)$ $limit - -$;
(5)      for $k \leftarrow 2$ to $\min\{i, limit\}$ {
(6)          $min \leftarrow \infty$; $rejLevel \leftarrow P[i, k - 1]$;
(7)          $l \leftarrow i$; $j \leftarrow N[l, k - 1]$;
(8)          while $j \geq k$ {
(9)              $curr \leftarrow P[j, k - 1] + cost[j + 1]$;
(10)             if $(curr \geq rejLevel)$ $N[l, k - 1] \leftarrow N[j, k - 1]$;
(11)             else { if $(curr < min)$ {$min \leftarrow curr$; $minInd \leftarrow j$; }
(12)                 $l \leftarrow j$; }
(13)             $j \leftarrow N[j, k - 1]$; }
(14)         $P[i, k] \leftarrow min$; $L[i, k] \leftarrow minInd$; $N[i, k] \leftarrow i - 1$; }}
```

distribution of the union of the intervals $I_j, \ldots, I_i$. This is performed by merging the distributions and recalculating the costs.

At step $i$, the array $P$ is updated according to a formula, 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 the best $(k-1)$-split of $I_1, \ldots, I_j$ remaining in the search phase. This update needs to be performed for every arity $2 \leq k \leq i$.

The space is pruned incrementally by comparing the best $(k-1)$-split of the processed intervals so far with each remaining candidate $k$-split of the same range. Whenever

$$P[i, k-1] \leq P[j, k-1] + cost[j+1],$$

the candidate is pruned from the space. The connection to Theorem 8 is the following: $P[j, k-1]$ corresponds to $P_1$, $P[i, k-1]$ to $P_2$, $I_{j+1} \cup \cdots \cup I_i$ to $S_2$, and an empty set to $S_*$.

Note that the pruning condition is tested first and, only if the candidate passes the test, we check whether it could be the best candidate so far. The reason for this is that by convexity there is always a $k$-split that is at least as good as the best $(k-1)$-split. Hence, the optimal $k$-split will always pass the first test.

The time complexity of the algorithm is $O((k + m)G^2)$, where $k$ is the maximum arity of the partition, $m$ is the number of classes, and $G$ is the number of example segments. The asymptotic bound is the same as that of the original algorithm (Elomaa and Rousu, 1999). In practice, the time consumption of the dynamic pruning algorithm is clearly lower than that of the original algorithm.

There are many ways to optimize the algorithm in Table 1, including implementation with pointers instead of table indices, the use of integer arithmetic, the use of sentinels to decrease the number of comparisons needed and other techniques (Rousu, 2001).

An optimized implementation of the multisplitting algorithms including the above described one is available from: `www.cs.helsinki.fi/u/rousu/splittER.tgz`.

## 5. Empirical evaluation

This section reports the results of empirical experiments with the speed-up techniques developed in this paper. We test the pruning techniques embedded into the C4.5 decision tree learning algorithm (Quinlan, 1993). As test domains we use twenty-eight data sets from the UCI repository (Blake and Merz, 1998). The data sets are described in Table 2.

### *5.1. Static pruning of partition candidates*

Figure 8 illustrates the average numbers of bin borders (the figures on the right) and the relative portions of boundary points (black bars) and segment borders (white bars) per numerical attribute of the domain.

On the average approximately 50% of bin borders are boundary points. However, the differences between domains are huge. The average number of segment borders per attribute is only marginally smaller than that of the boundary points. By combining bins into segments, in real-world data, almost all reduction in the number of points that need to be examined comes from combination of class uniform bins, only very few mixed bins get combined. The reason for this is obvious: even small changes—caused, e.g., by attribute noise—to the class distribution prevent combining neighboring mixed bins. However, since the segment construction is as efficient as block combination, nothing is lost by taking advantage of the small reduction in the number of cut points examined.

### *5.2. Dynamic pruning of partition candidates*

Our second test isolates the effects of dynamic pruning. We record the average number of partition candidates considered in searching for the optimal multisplit on bins when neither preprocessing nor dynamic pruning is used and when the candidates are dynamically pruned. As a baseline method we use a best-first search implementation of the widely used greedy top-down multisplitting scheme (Fayyad and Irani, 1993). Keep in mind that this method does not produce optimal partitions, even though the scores of the resulting partitions are often very close to optimal (Elomaa and Rousu, 1999). The asymptotic time consumption of the approximative greedy heuristic is linear in the number of potential cut points. As the evaluation function we use *IG*, which is convex (thus also well-behaved) and cumulative.

*Table 2.* Characteristic figures of the twenty-eight test domains.

| Data set | ATTRS | $m$ | $n$ | $\bar{V}$ | $\bar{B}$ | $\bar{G}$ |
|---|---|---|---|---|---|---|
| Abalone | 7/8 | 29 | 4,177 | 863.7 | 826.4 | 823.6 |
| Adult | 6/14 | 2 | 32,561 | 3,673.7 | 1,668.2 | 1,651.0 |
| Annealing | 10/20 | 5 | 798 | 27.5 | 17.7 | 17.7 |
| Australian | 6/15 | 2 | 690 | 188.2 | 129.7 | 125.8 |
| Auto insuran. | 15/25 | 6 | 205 | 61.3 | 50.5 | 50.5 |
| Breast W | 9/9 | 2 | 699 | 9.9 | 9.3 | 9.3 |
| Colic | 21/23 | 2 | 368 | 85.8 | 52.4 | 50.6 |
| Diabetes | 8/8 | 2 | 768 | 156.8 | 108.1 | 104.5 |
| Euthyroid | 6/29 | 2 | 2,800 | 164.0 | 89.8 | 88.0 |
| German | 7/20 | 2 | 1,000 | 203.0 | 98.2 | 97.8 |
| Glass | 9/9 | 6 | 214 | 115.3 | 70.8 | 70.2 |
| Heart C | 13/13 | 5 | 303 | 30.5 | 27.3 | 27.2 |
| Heart H | 12/13 | 2 | 294 | 26.8 | 20.8 | 20.6 |
| Hepatitis | 19/19 | 2 | 155 | 53.8 | 29.3 | 28.8 |
| Hypothyroid | 7/25 | 2 | 3,163 | 184.3 | 67.0 | 66.0 |
| Iris | 4/4 | 3 | 150 | 30.8 | 15.0 | 14.5 |
| Letter recogn. | 16/16 | 26 | 20,000 | 16.0 | 15.7 | 15.7 |
| Liver | 6/6 | 2 | 345 | 54.7 | 45.8 | 44.5 |
| Page blocks | 10/10 | 5 | 5,473 | 909.2 | 338.9 | 337.1 |
| Satellite | 36/36 | 6 | 4,435 | 76.3 | 62.6 | 62.4 |
| Segmentation | 19/19 | 7 | 210 | 145.3 | 95.1 | 94.8 |
| Shuttle | 9/9 | 7 | 58,000 | 123.2 | 85.3 | 85.0 |
| Sonar | 60/60 | 2 | 208 | 187.6 | 96.8 | 96.2 |
| Vehicle | 18/18 | 4 | 846 | 79.4 | 68.6 | 67.8 |
| Vowel | 10/10 | 11 | 990 | 808.7 | 708.8 | 707.2 |
| Waveform | 21/21 | 3 | 5,000 | 714.0 | 653.2 | 636.9 |
| Wine | 13/13 | 3 | 178 | 98.2 | 56.1 | 55.8 |
| Yeast | 8/8 | 10 | 1,484 | 51.5 | 47.9 | 47.8 |

Column $m$ is the number of classes, $n$ is the total number of examples in the domain, ATTRS gives the number of numerical attributes out of the total number of attributes, $\bar{V}$ is the average number of bins for an attribute, $\bar{B}$ is that of blocks, and $\bar{G}$ is that of segments.

In the experiment we partition the numerical dimensions of the 28 test domains using all three partitioning strategies. For each domain we record the number of candidate partitions evaluated in processing each numerical attribute.

Figure 9 depicts the results of this experiment. The figures on the right are the average number of evaluations per numerical attribute performed by the algorithm that does not prune and operates on example bins, the black bars represent the relative number of evaluations per attribute for the dynamically pruning algorithm, and the white bars are those of the greedy heuristic selection.

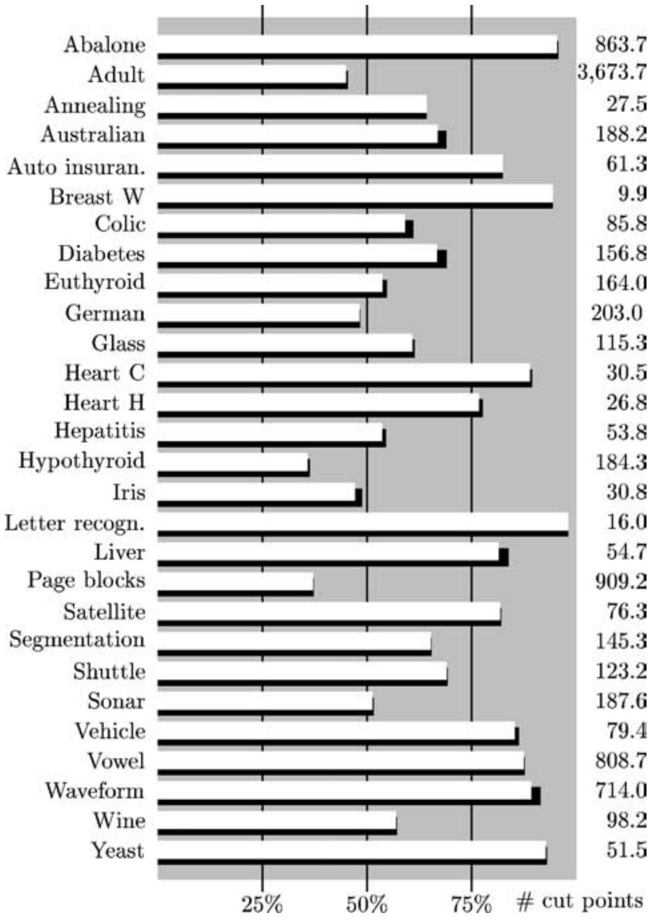

*Figure 8.* The average number of bin borders (the figures on the right) and the relative numbers of boundary points (black bars) and segment borders (white bars) per numerical attribute of the domain.

From figure 9 it can be seen that the average saving in the number of examined partition candidates by dynamic pruning is slightly over 50%. These reductions do not correspond linearly to the search times. The relation between the number of pruned candidates and time consumption is discussed in the next section. Also, the actual time consumption of different evaluation algorithms are reported in the next set of experiments.

Pruning fails to attain any savings in the two domains with the least numbers of initial comparisons, Breast W and Letter recognition. Instead, the overhead involved with dynamic pruning means that on these domains more partition candidate comparisons are performed when dynamic pruning is employed. However, for these domains the time consumption is very low to begin with. Moreover, these domains are exceptions that clearly stand out from the results.

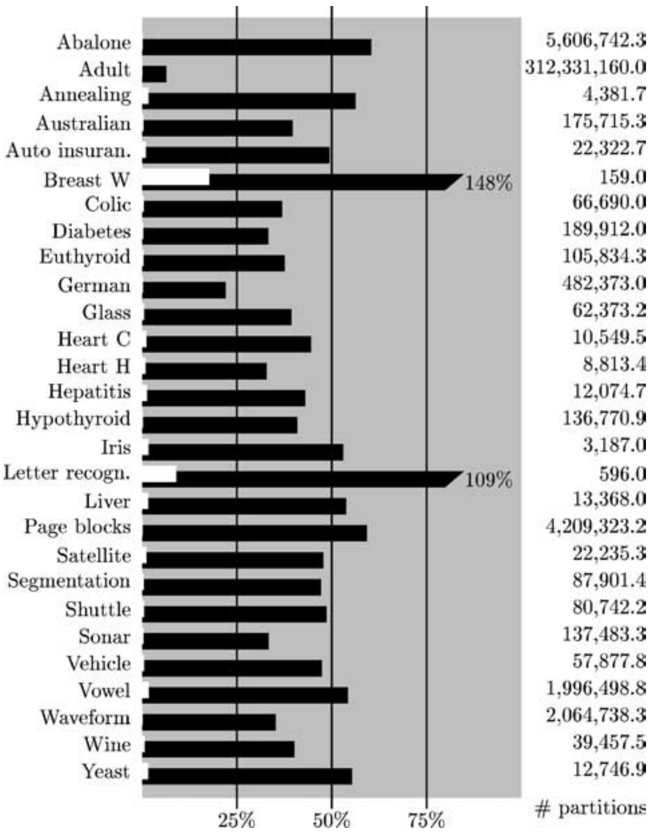

*Figure 9.* The number of partition candidate evaluations. The figures on the right are the average numbers of partition candidate evaluations per attribute performed on bins. Black bars depict the relative numbers of evaluations performed by the algorithm that dynamically prunes partition candidates and white bars correspond to those of the greedy approach. In the domains with a broken bar, the relative time consumption grows over 100%. For these domains the figure on top of the bar gives the relative time consumption.

On other domains better pruning results are observed. Systematically (with the exception of domain Abalone) whenever less than 50% of partition comparisons is saved through dynamic pruning, the bar corresponding to the relative number of comparisons of the greedy method can also be seen figure 9. This indicates that the efficient greedy method executes a comparable number of partition candidate comparisons on these domains. Let us still stress that in these cases the optimal multisplit can be recovered with almost as little work as the suboptimal partition produced by the greedy algorithm.

For most domains less than 50% of initial comparisons are required when dynamic pruning is employed. Clearly the best result is obtained for the most difficult domain to start with, Adult. Dynamic pruning only requires approximately 6% of the initial 313 million comparisons in this particular domain.

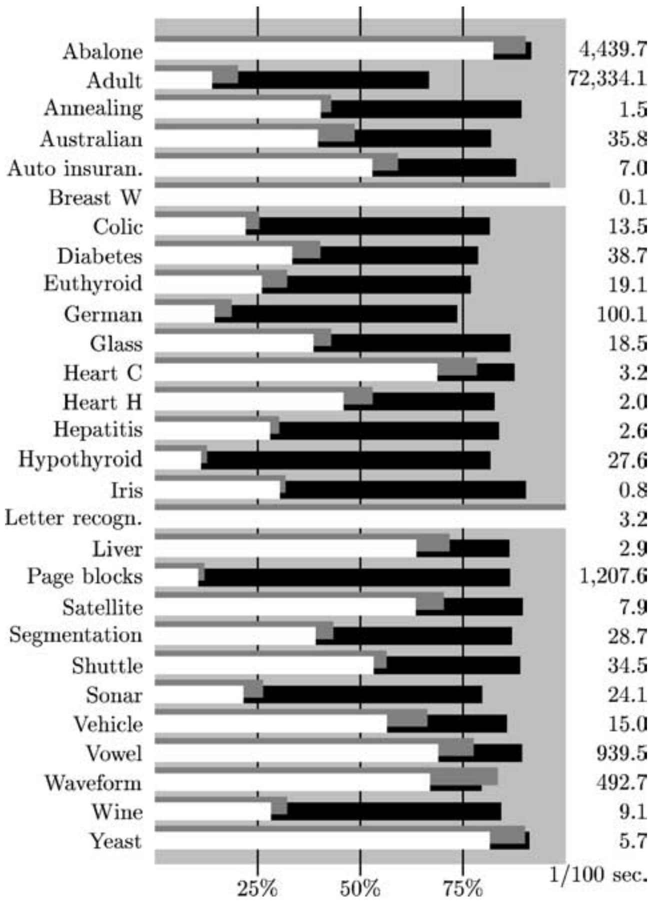

*Figure 10.* The effects of combined static and dynamic pruning on 28 UCI test domains. The figures on the right are the average processing times per numerical attribute when operating on bins. Black bars depict the relative times of dynamic pruning on bins, gray bars are the relative times of processing example segments instead of bins, and white bars correspond to the relative times of the combined approach operating on example segments and using dynamic pruning.

### *5.3. Combined effects of preprocessing and dynamic pruning*

Our final comparison contrasts static and dynamic pruning with each other. Figure 10 shows the relative processing times used by the multisplitting algorithm using dynamic pruning only, static pruning only, and combining both pruning approaches in our 28 test domains.

The figures on the right are the average processing times—in one hundredths of a second per numerical attribute—when using *IG* as the evaluation function and operating on bins. Black bars depict the relative times of dynamic pruning on bins. Gray bars are the relative times when the discretization algorithm operates on example segments instead of bins.

Finally, the white bars correspond to the relative times of the combined approach, which operates on example segments and uses dynamic pruning.

Dynamic pruning alone brings about 20% average time saving. This figure is significantly smaller than what could be expected from the 50% reduction in the number of candidate comparisons. This difference is partly explained by the overhead associated with making the pruning test but, more importantly, by the fact that for each pair of candidate cut points the impurity of the subset in between them needs to be computed even when using dynamic pruning. This computation induces the term $O(mG^2)$ in the asymptotic time-complexity $O((k+m)G^2)$ of the algorithm. Dynamic pruning, on the other hand, affects the other term, $O(kG^2)$.

The effects of the preprocessing into example segments surpass these savings clearly. On the average the algorithms run 45% faster when the preprocessing into example segments is used. Combining the two techniques—i.e., running the algorithm on the segment sequence dynamic pruning enabled—reduces the time-consumption below 50% of the baseline. This shows that the two techniques are complementary.

For some of the most time consuming domains (e.g., Adult and Page blocks) static and combined pruning reduce the time consumption below one fifth of the original. However, in the domains, where almost all cut points are boundary points (e.g., Abalone, Breast W, and Letter recognition; see figure 8) the utility of static and dynamic pruning remains low. The same can be observed from the number of partition candidates evaluated (see figure 9).

## 6. Conclusions and further work

Soundness and efficiency are two important aspects of knowledge discovery methods. Data mining often resorts to purely heuristic methods to ensure efficient discovery. This may endanger or, at least, make it harder to assess the quality of the discovered knowledge. For this reason it is important to develop efficient data mining methods that can give guarantees of the quality of the answer.

Partitioning numerical value ranges into a small number of subranges is a task frequently encountered in data mining and machine learning algorithms both as a "discretization" step prior the actual induction phase and also integrated within the learning algorithms. This important task is typically solved by heuristics, since up until recently, the efficiency of optimal algorithms has not been satisfactory.

In this article, we presented techniques for speeding up the discovery of optimal—with respect to an evaluation function—multisplits along numerical value ranges. Both techniques are based on proving some cut points or prefixes of partitions as suboptimal and thus discarding them from the search space of the algorithms.

In the development of the first technique, we introduced segment borders, which are boundary points that separate adjacent example segments with different relative class distributions. Subsequently we proved that many evaluation functions frequently used in class-driven partitioning place their optima on segment borders. There is no need to consider any other cut points in the value range than segment borders. The set of segment borders can be found in linear time, so the time-requirement is negligible compared to the subsequent search, which typically takes at least quadratic time in the number of candidate cut points.

The preprocessing can be coupled with different search algorithms, including dynamic programming, brute-force, and greedy best-first search. Based on our experiments concentrating on segment borders instead of boundary points brings only a marginal advantage, since most of the boundary points are already segment borders. However, when compared to the original set of cut points in the range, the reduction is significant.

To further improve the preprocessing, it would be useful to consider situations where the relative class distributions of the neighboring segments were allowed to differ. The questions for further research include whether the absence of optima can still be guaranteed, how much deviation can be allowed, and which types of deviations make it easier to guarantee the absence of optima within the example segment? A step towards this direction is made for *TSE* by Elomaa and Rousu (2003).

With our second technique, we aimed at speeding up the quadratic-time dynamic programming algorithm by pruning dynamically the search space during the left-to-right scan over the data. The technique is applicable for all convex and cumulative evaluation functions, a set that includes, e.g., the commonly used Average Class Entropy and Gini Index functions. In our experiments, the number of candidate comparisons in the algorithms was halved on the average. The time-usage of the algorithms was improved by ca. 20 percent.

The combined effect of the preprocessing and pruning is significant. Overall, the quadratic-time dynamic programming algorithm runs twice as fast on the average and in some cases up to 90 percent faster than the baseline algorithm. All in all, solving the multisplitting problem optimally is fully feasible even with modest computational resources.

**Acknowledgments**

We thank Jyrki Kivinen for valuable advice and comments.

## References

Ankerst, M., Ester, M., and Kriegel, H.-P. 2000. Toward an effective cooperation of the user and the computer for classification. In Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, R. Ramakrishnan, S. Stolfo, R. Bayardo, and I. Parsa (Eds.), New York, NY: ACM Press, pp. 179–188.

Auer, P. 1997. Optimal splits of single attributes. Technical report, Instute for Theoretical Computer Science, Graz University of Technology. Unpublished manuscript.

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

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

Blake, C.L. and Merz, C.J. 1998. UCI repository of machine learning databases. University of California, Department of Information and Computer Science, Irvine, CA. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Breiman, L. 1996. Some properties of splitting criteria. Machine Learning, 24(1):41–47.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. 1984. Classification and Regression Trees. Pacific Grove, CA: Wadsworth.

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

Cerquides, J. and López de Màntaras, R. 1997. Proposal and empirical comparison of a parallelizable distance-based discretization method. In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy (Eds.), Menlo Park, CA: AAAI Press, pp. 139–142.

Ching, J., Wong, A., and Chan, K. 1995. Class-dependent discretization for inductive learning from continuous and mixed-mode data. IEEE Transactions on Pattern Analysis and Machine Intelligence, 17(7):641–651.

Codrington, C.W. and Brodley, C.E. 1997. On the qualitative behavior of impurity-based splitting rules I: The minima-free property. Technical Report 97-5, School of Electrical and Computer Engineering, Purdue University.

Coppersmith, D., Hong, S.J., and Hosking, J.R.M. 1999. Partitioning nominal attributes in decision trees. Data Mining and Knowledge Discovery, 3(2):197–217.

Cover, T.M. and Thomas, J.A. 1991. Elements of Information Theory. New York, N.Y.: John Wiley & Sons.

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

Elomaa, T. and Rousu, J. 1999. General and efficient multisplitting of numerical attributes. Machine Learning, 36(3):201–244.

Elomaa, T. and Rousu, J. 2001. On the computational complexity of optimal multisplitting. Fundamenta Informaticae, 47(1/2):35–52.

Elomaa, T. and Rousu, J. 2003. Necessary and sufficient pre-processing in numerical range discretization. Knowledge and Information Systems, 5(2):162–182.

Fayyad, U.M. and Irani, K.B. 1992. On the handling of continuous-valued attributes in decision tree generation. Machine Learning, 8(1):87–102.

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

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

Hardy, G.H., Littlewood, J.E., and Pólya, G. 1934. Inequalities. Cambridge, UK: Cambridge University Press.

Hickey, R.J. 1996. Noise modelling and evaluating learning from examples. Artificial Intelligence, 82(1/2):157–179.

Ho, K. and Scott, P. 1997. Zeta: A global method for discretization of continuous variable. In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy (Eds.), Menlo Park, CA: AAAI Press, pp. 191–194.

Hong, S.J., 1997. Use of contextual information for feature ranking and discretization. IEEE Transactions on Knowledge and Data Engineering, 9(5):718–730.

Kohavi, R. and Sahami, M. 1996. Error-based and entropy-based discretization of continuous features. In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, E. Simoudis, J. Han, and U. M. Fayyad (Eds.), Menlo Park, CA: AAAI Press, pp. 114–119.

Liu, H. and Setiono, R. 1997. Feature selction via discretization. IEEE Transactions on Knowledge and Data Engineering, 9(4):642–645.

López de Màntaras, R. 1991. A distance-based attribute selection measure for decision tree induction. Machine Learning, 6(1):81–92.

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

Meyer, B. 1984. Some inequalities for elementary mean values. Mathematics of Computation, 42(1):193–194.

Pfahringer, B. 1995. Compression-based discretization of continuous attributes. In Proceedings of the Twelfth International Conference on Machine Learning, A. Prieditis and S. Russell (Eds.), San Francisco, CA: Morgan Kaufmann, pp. 456–463.

Provost, F., Jensen, D., and Oates, T. 1999. Efficient progressive sampling. In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, S. Chanduri and D. Madigan (Eds.), New York, ACM Press, pp. 23–32.

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

Quinlan, J.R. 1988. Decision trees and multi-valued attributes. In Machine Intelligence 11: Logic and the Acquisition of Knowledge, J.E. Hayes, D. Michie, and J. Richards (Eds.), Oxford, UK: Oxford University Press, pp. 305–318.

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

Rousu, J. 2001. Efficient range partitioning in classification learning. Ph.D. thesis, Department of Computer Science, University of Helsinki. Report A-2001-1.

Utgoff, P.E. 1989. Incremental induction of decision trees. Machine Learning, 4(2):161–186.

Wu, X. 1996. A bayesian discretizer for real-valued attributes. Computer Journal, 39(8):688–694.

Zighed, D., Rakotomalala, R., and Feschet, F. 1997. Optimal multiple intervals discretization of continuous attributes for supervised learning. In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, D. Heckerman, H. Mannila, D. Pregibon, and R. Uthurusamy (Eds.), Menlo Park, CA: AAAI Press, pp. 295–298.

**Tapio Elomaa** is currently Professor of Software Systems at Tampere University of Technology. He received his M.Sc. and Ph.D. degrees from the University of Helsinki in 1989 and 1996, respectively. During 1997–98 he was a Marie Curie Fellow at the Joint Research Centre of the European Commission in Ispra, Italy. Prof. Elomaa's research interests include machine learning, algorithms and data structures, pattern recognition, and artificial intelligence. He can be reached at elomaa@cs.tut.fi.

**Juho Rousu** received his M.Sc. degree in 1996 from University of Helsinki. During 1996–2000 he worked as a research scientist in Technical Research Centre of Finland (VTT). He received his Ph.D. degree in 2001 from University of Helsinki. Currently Dr. Rousu is a Marie Curie Fellow at the Department of Computer Science at the Royal Holloway University of London. His research interests include design and analysis of algorithms, machine learning, and computational biology. He can be reached at juho@cs.rhul.ac.uk.