Finding Optimal Multi-Splits for Numerical Attributes in Decision Tree Learning

Tapio Elomaa Juho Rousu
Department of Computer Science
P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 University of Helsinki, Finland
{elomaa, rousu}@cs.helsinki.fi

NeuroCOLT Technical Report Series

NC-TR-96-041

March 1996[1]

Produced as part of the ESPRIT Working Group
in Neural and Computational Learning,
NeuroCOLT 8556

NeuroCOLT Coordinating Partner

Department of Computer Science
Egham, Surrey TW20 0EX, England

For more information contact John Shawe-Taylor at the above address
or email neurocolt@dcs.rhbnc.ac.uk

[1]Received March 15, 1996

Abstract

Handling continuous attribute ranges remains a deficiency of top-down induction of decision trees. They require special treatment and do not fit the learning scheme as well as one could hope for. Nevertheless, they are common in practical tasks and, therefore, need to be taken into account.

This topic has attracted abundant attention in recent years. In particular, Fayyad and Irani showed how optimal binary partitions can be found efficiently. Later, they based a greedy heuristic multipartitioning algorithm on these results. Recently, Fulton, Kasif, and Salzberg attempted to develop algorithms for finding the optimal multi-split for a numerical attribute in one phase.

We prove that, similarly as in the binary partitioning, only boundary points need to be inspected in order to find the optimal multipartition of a numerical value range. We develop efficient algorithms for finding the optimal splitting into more than two intervals. The resulting partition is guaranteed to be optimal w.r.t. the function that is used to evaluate the attributes' utility in class prediction.

We contrast our method with alternative approaches in initial empirical experiments. They show that the new method surpasses the greedy heuristic approach of Fayyad and Irani constantly in the goodness of the produced multi-split, but, with small data sets, cannot quite attain the efficiency of the greedy approach. Furthermore, our experiments reveal that one of the techniques proposed by Fulton, Kasif, and Salzberg is of scarce use in practical tasks, since its time consumption falls short of all demands. In addition, it categorically fails in finding the optimal multi-split because of an error in the rationale of the method.

1 Introduction

Continuous numerical attributes do not fit the top-down induction scheme of decision trees quite as naturally as discrete attributes do. In real-world induction tasks numerical attribute ranges, nevertheless, regularly appear, and therefore, they also need to be taken into account in decision tree construction. Conventionally, continuous ranges have been handled by dividing them into two in a single node of the evolving tree (Breiman *et al.*, 1984). Further down the tree the division process may continue in another node. However, there are two major drawbacks in this approach: First, the continuous attributes are handled in a different manner than discrete attributes and the induced trees are no longer *reduced* (i.e., an attribute may appear more than once on a path leading from the root to a leaf), which certainly damages the intelligibility of the result, if nothing else. More importantly, the repetitive binarization of an attribute range cannot guarantee optimal multi-way partitioning even if the binary splits were optimal (cf. Fayyad and Irani, 1992, 1993).

This paper tackles the problem of finding optimal (w.r.t. a given evaluation function) multi-way partitionings (up to k intervals) for continuous attribute value ranges. The usual setting in this task is the following. The underlying assumption is that we have sorted our N examples into (ascending) order according to the value of a numerical attribute A. For truly continuous values sorting takes $O(N \log N)$ time, but could be performed in linear time for

(small) integer ranges using bucket sort.[1] With this sorted sequence of examples in hand we try to elicit the best possible (w.r.t. class prediction) multi-split along the dimension determined by attribute A. A further prerequisite is that we have a fixed evaluation function for measuring the goodness of candidate splits; i.e., evaluating their correlation to class label prediction.

At least in the root level of the evolving tree, the data needs to be sorted according to the values of all numerical attributes, since they all need to be considered as the potential label of the root. When using multi-splits, it is not necessary to reconsider an attribute, which already appears on the path, deeper down the tree. On the contrary, it is possible to construct reduced decision trees. However, those attributes that were not chosen to the root have to be re-evaluated in the second level of the tree, and possibly still after that. Even though the example set, which determines the goodness of an attribute, changes dynamically as the construction process proceeds, we do not have to keep sorting the examples over and over again, but can rely on careful bookkeeping instead (Fayyad and Irani, 1993; Fulton, Kasif, and Salzberg, 1995).

One remaining topic is the arity, k, of the resulting partition. Usually we would like to penalize increase in k; that is, we would like to keep the arity of the partition relatively low because of the following reasons: The utility of an attribute in class prediction may be impaired because of too large arity (Quinlan, 1988) and both the comprehensibility of the result and the efficiency of finding it remain better, when k is not allowed to grow too much. In general, we can consider k to be an upper bound for the arity of the resulting partition. The algorithms that we present give us the possibility to select the best partition from among those with arity at most k with only small amount of extra work. Moreover, we show that, in practice, we do not need to set the upper bound to be $k = N$, but each application domain has a natural upper bound, which usually is significantly lower than the number of separate examples. In any case, the number of different values appearing in the data for an attribute gives the maximum arity of any partition along the dimension determined by the attribute. In the empirical experiments reported in this paper we circumvented taking a stand to this issue by assuming k to be a given value.

We continue the presentation by giving, in Section 2, a review of related work conducted hitherto; in particular, we recapitulate the recent work of Fulton, Kasif, and Salzberg (1995), which has laid the ground for our method, and the formal results of Fayyad and Irani (1992), which constitute the underpinnings for our work. Thereafter, in Section 3, we introduce our main contribution: An efficient method of finding optimal multi-splits in one shot. Furthermore, we extend the main result of Fayyad and Irani (1992) and prove that only boundary points need to be considered as possible cut points in multi-splitting as well. Section 4 reports the results of an initial empirical comparison between the different multi-splitting techniques on several real-world applica-

[1] Quite often the "real-valued" measurements recorded into the values of an attribute come from a device with a limited measuring accuracy. Such semi-continuous ranges could be mapped into an integer range. Furthermore, discrete numerical (integer) ranges are common in learning domains. The bucket sort ordering, nevertheless, is only applicable to relatively small ranges.

tion domains. The experiments show that alternative methods often fail in finding the optimal discretization of a numerical value range and they cannot attain substantial advantage in running time. We conclude this paper by considering future development possibilities of multi-splitting algorithms.

2 Related work

The problem of continuous attribute range discretization has received abundant attention recently. The standard technique is to sort the examples into order according to the value of a numerical attribute, and then consider each successive pair of examples in the sorted list as a potential *cut point*. The cut points are evaluated according to the *goodness criterion*, which, in binarization, is used to pick the best cut point for further comparison. Fayyad and Irani's (1992) analysis of the binarization technique proved that substantial reductions in time consumption can be obtained, since only *boundary points* need to be considered as potential cut points, because optimal splits always fall on boundary points. Intuitively, a boundary point is the point between two successive examples in the sorted list such that they belong to different classes. We review their result in detail subsequently.

Let us already at this point fix some notation. Let $val_A(s)$ denote the value of the attribute A in the example s. A *partition* $\bigcup_{i=1}^{k} S_i$ of sample S into k intervals has the following properties:

(i) for all $i \in \{1, \ldots, k\}$, $S_i \neq \emptyset$,

(ii) $\bigcup_{i=1}^{k} S_i = S$, and

(iii) if $s_i \in S_i$, then $s_i \notin S_j$, for all $j \neq i$.

Furthermore, if partition $\bigcup_{i=1}^{k} S_i$ 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 $val_A(s_i) < val_A(s_j)$.[2] When splitting a set S of examples on the basis of the value of an attribute A, then there is a set of values $\{T_1, \ldots, T_{k-1}\}$ that define a partition $\bigcup_{i=1}^{k} S_i$ for the sample in an obvious manner:

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

The goodness criteria that are used to evaluate candidate partitions are many. The most commonly used functions belong to the family of *impurity measures* (Breiman *et al.*, 1984). They include such well-known functions as the gini index (Breiman *et al.*, 1984) and Quinlan's (1993) information gain and gain ratio. Impurity measures tend to be *additive*; i.e., the impurity of the whole data is obtained by summation over the subsets that results due to the discretization. Fayyad and Irani (1992) focused on a particular impurity

[2] Note that this really is a strict relation, since two examples with the same value for the attribute A are inseparable when considering only this one value. Hence, two examples with an equal value for the attribute in question can never fall into different intervals.

measure, the *average class entropy*: Let $\bigcup_{i=1}^{k} S_i$ be a partition of S, then by $E(\bigcup_{i=1}^{k} S_i)$ we denote the average class entropy of the partition:

$$E(\bigcup_{i=1}^{k} S_i) = \sum_{i=1}^{k} \frac{|S_i|}{|S|} H(S_i) = \frac{1}{|S|} \sum_{i=1}^{k} |S_i| H(S_i),$$

where $H(S)$ is the entropy function,

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

in which m denotes the number of classes and $P(C, S)$ stands for the proportion of examples in S that have class C. Fayyad and Irani (1992) proved that this measure is well-behaved in the sense that it will never favor an obviously bad cut, one that needlessly disperses examples of one class into different sides of a cut. In this paper we concentrate on the same impurity measure; the same consequences as in binary splitting follow in the general case studied here.

A single binary partition along the dimension determined by an attribute seldom suffices; usually the range needs to be partitioned into several intervals. This led Fayyad and Irani (1993) later to propose a greedy top-down method of discretizing continuous ranges into multiple intervals. In this straightforward approach the binary splitting scheme is recursively applied to a numerical range (in one node) as to obtain division into multiple subranges. Minimum Description Length (MDL) criterion is used in Fayyad and Irani's technique to control the number of intervals produced. Even though this is based on a method that produces optimal binary partitions and has been reported to give good results in practice (Fayyad and Irani, 1993), no guarantees can be given to the resulting multi-split induced by the recursive algorithm.

Catlett (1991) has put forward essentially the same technique as Fayyad and Irani (1993) for handling numerical attribute value ranges. He, however, does not give any formal motivation for the method and uses an entropy-based interval controlling function.

Pfahringer (1995) combined the two above-mentioned methods: He suggested using entropy in candidate split point selection and, then, using MDL (together with best-first search) in the final splitting determination.

Very recently Fulton, Kasif, and Salzberg (1995) introduced two algorithms for finding a set of intervals in a single node of the evolving tree. They consider the case, where given are a function for measuring the goodness of a split and an upper bound k for the number of intervals. They present two solutions: A quadratic-time general solution, that works for any additive impurity measure, and a linear-time special algorithm for multi-splitting a given range optimally (mod k). Subsequently, we examine only the former method and refer to it, henceforth, as the algorithm FKS.

Other related work includes, for instance, that of Chou (1991), which gives a meticulous account of the history of discretization problem within the statistical community in regression tree construction (Breiman *et al.*, 1984). The partitioning technique developed by Chou, however, concerns grouping of values of a categorical attribute. Maass (1994) proposed a method that produces

minimum training set error partitioning in time $O(N(\log N + k^2))$. As far as we know, no experimental evaluation of this method exists.

For a comprehensive review and empirical experiments with different (supervised and unsupervised) methods of continuous value range discretization we refer the reader to the paper of Dougherty, Kohavi, and Sahami (1995).

3 Efficient optimal multi-splitting of numerical attribute ranges

Let us begin by recapitulating the motivation of the quadratic (in N) general solution of Fulton *et al.* (1995). The method is general in the sense that it works for any additive impurity measure. In the following, we use the average class entropy as the goodness criterion.

Consider the following set of examples, ordered by the value of a numerical attribute (integer-valued in this case). The alphabets above the examples are their class values and the integers below correspond to the value of the attribute in question.

```
      $ @             *     @       $
  A A A A B B B B B C C C B B B C A A A C C C C C C C C
 |-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
  0 0 0 1 1 1 1 2 2 3 3 3 4 4 4 4 5 5 6 7 7 7 8 8 9 9 9
```

In the brute-force solution proposed by Fulton *et al.*, every point in between two successive examples is evaluated. In the dynamic programming implementation the impurities of cut point candidates can very conveniently be calculated recursively from the impurities of shorter intervals and smaller arity partitions:

$$impurity(k, 1, i) = \min_{1 \leq j < i} (impurity(k-1, 1, j) + impurity(1, j+1, i)),$$

where $impurity(k, j, i)$ denotes the minimum impurity that results when examples j through i are partitioned into k intervals. The best k-split is the one that minimizes $impurity(k, 1, N)$. This leads to asymptotic time requirement of $O(kN^2)$, when choosing a discretization of arity at most k.

Incidentally, this example reveals an error in the method of Fulton *et al.* We have marked some cut points above the sequence of examples: Those defining the optimal 3-way partition of this data are denoted by dollar signs (**$**). The leftmost cut point in the partition chosen by the FKS algorithm coincides with that of the optimal one. Thus, these two partitions share the first interval. The other cut point defining the 3-way partition of the FKS algorithm is denoted by an asterisk (*). The error in the FKS algorithm is to evaluate points in between two examples that have the same value for the attribute in question, when in practice the data cannot be partitioned there. Hence, the actual cut point will slide—depending on whether the inequality, that determines the threshold, is strict or not—to the left or to the right from the point, where it was evaluated. Therefore, a different partition than that, which was intended, comes out as a result.

In this particular example, the FKS algorithm has found points marked with at signs (@) to define a partition with impurity 0.70. Since strict inequality is used to determine the actual thresholds, the final cut points shift to the left and a partition with impurity of 1.21 results. (Using non-strict inequality would not help here either: If the cut points were to slide to the right, a better, but still suboptimal impurity 0.93 would result.) The best 3-way partition that can be obtained in this example (denoted by the dollar signs) has impurity 0.89.

Algorithm 1

This integer-valued example brings clearly out a possibility for a simple rectification and optimization to the FKS method: In any case, it is impossible to divide the data in between two examples that have the same value for the attribute in question. Therefore, computing the impurity for all pairs of successive examples renders superfluous. We can, as well, consider a categorized version of the data, rather than examine all midpoints in between separate examples. We can throw all examples, that have the same value for the example in question, into a common *bin* and consider only points in between bins as potential cut points. Thus, the above depicted situation would reduce to that below.

```
 AAA ABBB BB CCC BBBC AA A CCC CC CCC
|---|----|--|---|----|--|-|---|--|---|
 0   1    2  3   4    5  6 7   8  9
```

All is well and fine when handling an integer-valued attribute range, but is this ordering of examples into bins relevant when dealing with real-valued variables. We already discussed this question and according to our experience the answer is positive: Truly continuous attribute ranges are rare and discrete numerical (integer) attributes appear quite often. In practice, often $V \ll N$, where V denotes the number of different values in an attributes value range. In any case, we do not lose anything by examining bins rather than handling the examples separately. The problem arises: What is the class label of a bin of examples? We do not have to worry about that; to compute the impurities on the borders in between bins, it suffices to know the number of instances from each of the classes within each bin, but their order is immaterial. Therefore, only class distribution of each bin has to be known. We could denote the situation as follows.

```
    1A       3B
 3A 3B 2B 3C 1C 2A 1A 3C 2C 3C
|--|--|--|--|--|--|--|--|--|--|
 0  1  2  3  4  5  6  7  8  9
```

This algorithm, as well as our implementation of the FKS algorithm, consists of three components: *preprocessing*, *interval extraction*, and *search* using dynamic programming. Preprocessing entails sorting the examples; it is common to both algorithms. The algorithms differ only in the interval extraction

phase. The output of the interval extraction phase is a sorted array of *initial intervals*. For the changed algorithm the initial intervals are the bins of examples having the same value for the numerical attribute in question and for the FKS algorithm each example constitutes its own initial interval. An initial interval is represented by a pair (D, i), in which D is the class frequency distribution of the interval and i is the index of the example that ends the interval.

In search phase of the new algorithm we apply the recursive impurity evaluation of Fulton *et al.* (1995) to the initial intervals rather than to individual examples. Thus, by $impurity(k, j, i)$ we denote the minimum impurity of bins j through i when partitioned into k intervals, otherwise there is no change in the recurrence of FKS. The calculation of the recurrence is done as follows: We maintain two data structures, a two-dimensional matrix for values of type $impurity(k-1, 1, j)$ and an array for values of type $impurity(1, j+1, i)$, i.e., for the kth interval of the resulting partition. From the values stored into these data structures we can easily obtain the values $impurity(k, 1, i)$. Finding the optimal partition of the N examples into at most k intervals now takes $O(kV^2)$ time, which in the worst case remains equivalent with that of the method of Fulton *et al.*, but in practice gives substantial reductions in time consumption (see Section 4).

Implementation of the above described discretization method will subsequently be referred to as Algorithm 1. Clearly, this method is general in the sense that it works for any additive impurity measure.

Algorithm 2

In order to evolve Algorithm 1 further, we need to develop theory of discretization a step further. Let us, first, recapitulate the exact definition of a boundary point.

Definition 1 (Fayyad and Irani, 1992) *A value T in the range of the attribute A is a boundary point iff in the sequence of examples sorted by the value of A, there exist two examples $s_1, s_2 \in S$, having different classes, such that $val_A(s_1) < T < val_A(s_2)$; and there exists no other example $s' \in S$ such that $val_A(s_1) < val_A(s') < val_A(s_2)$.*

Fayyad and Irani proved that when searching for the best binary split by choosing a single cut point, we can restrict our attention to boundary points. Their theorem can be restated as follows:

Theorem 1 (Fayyad and Irani, 1992) *If value T minimizes the impurity $E(S_0 \cup S_1)$, where $S_0 \cup S_1$ is the partition of S defined by T, then T is a boundary point.* □

In the following we show that Theorem 1 generalizes to the situation, where a k-ary partition of examples is generated by choosing $k-1$ cut points.

Theorem 2 *If partition $\bigcup_{i=1}^{k} S_i$ of S minimizes the impurity $E(\bigcup_{i=1}^{k} S_i)$, then the corresponding values $\{T_1, \ldots, T_{k-1}\}$ are boundary points.*

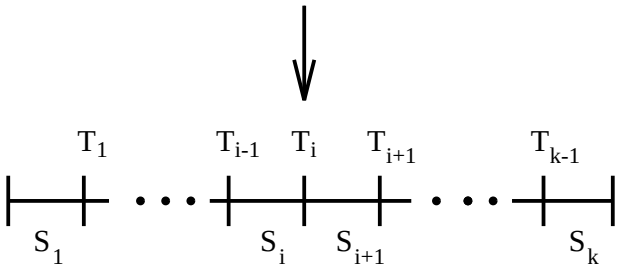

Figure 1 Illustration of the situation in the proof of Theorem 2.

Proof Assume that, contrary to the claim, value T_i in set $\{T_1, \ldots, T_{k-1}\}$ is not a boundary point. Replacing T_i with another value $T_i' \in (T_{i-1}, T_{i+1})$ clearly affects only the sets S_i and S_{i+1} in the partition $\bigcup_{i=1}^{k} S_i$. Therefore, changes happen only to the terms $|S_i|H(S_i)$ and $|S_{i+1}|H(S_{i+1})$, in the equation giving the total impurity of the partition[3] (see Fig. 1). Hence, in order to minimize the total impurity of the partition, we should also minimize the impurity $|S_i|H(S_i) + |S_{i+1}|H(S_{i+1})$. Theorem 1 says, that in order to minimize the impurity $|S_i|H(S_i) + |S_{i+1}|H(S_{i+1})$, one should partition the set $S_i \cup S_{i+1}$ on a boundary point. Since, by assumption, T_i is not a boundary point, the impurity of the partition $\bigcup_{i=1}^{k} S_i$ is not minimum possible. Because T_i was chosen freely, the claim follows. □

Using the result proved above, we can evolve our categorizing algorithm further. According to Theorem 2, we do not have to consider each border of consecutive bins as a potential cut point: It suffices to inspect boundary points, which in this case are borders of consecutive bins B and B' such that

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

According to this definition further simplifications into our example range can be introduced: We need not examine all borders in between bins, but can combine bins that do not have a boundary point in between them into a common *block*. Of course, the class distribution of each block has to be retained in order to be able to compute the impurity measures. We, however, restraint from marking them down to the following picture, in which `M` denotes mixed class distribution.

`A M B C M A C`

[3] We can ignore the constant coefficients $1/|S|$ from the terms without loss of generality.

Table 1 Descriptions of the data sets used.

Data set	Examples	Attributes		Classes
		Num.	Total	
Iris plant classification	150	4	4	3
Glass type identification	214	9	9	6
Australian credit card assessment	690	6	14	2
Wisconsin **breast** cancer data	699	9	9	2
Diabetes prediction	768	8	8	2
German credit screening	1 000	3	20	2
Identification of **vehicle** silhouettes	846	18	19	4
Obstacle recognition from **robot** sensor input	2 100	22	22	7
Soil type detection from **satellite** image	4 435	36	36	6
Waveform domain by Breiman *et al.*	5 000	40	40	3
Space **shuttle** radiator positioning	43 500	9	9	7

```
|-|-|-|-|-|-|-|
0 1 2 3 4 5 7
          6 9
```

¿From the original 27 separate examples we have reduced the value range now to only 7 blocks. Using the same dynamic programming solution as previously, the time requirement of finding the optimal partitioning into at most k intervals now is $O(kB^2)$, where $B \leq V$ is the number of blocks in the range. Again, in the worst case $B = N$, but most often $B \ll N$. Implementation of the method that works on blocks, rather than bins, is subsequently referred to as Algorithm 2. This algorithm works for well-behaved measures that are guaranteed to minimize on boundary points. (Naturally, well-behaved measures are the ones that we would like to use in any case.)

4 Empirical evaluation

In this initial experiment we compare the splitting techniques on their own, without incorporating them into a decision tree constructing algorithm. Thus, we avoid restricting ourselves to a certain evaluation function for choosing the number of produced intervals. In order to obtain comparable results, we choose the arity of the splitting according to that selected by the greedy approach of Fayyad and Irani (1993).

In our experiments we used mainly well-known data sets from the UCI repository (Murphy and Aha, 1994) and from the StatLog project (Michie *et al.*, 1994). Table 1 summarizes shortly the main characteristics of the data sets. More comprehensive descriptions can be found from the referred sources. The

Table 2 Resulting average split arities and impurities when a multi-split is generated for each numerical attribute in data set.

DATA SET	ARITY	FI	FKS	Alg.1	Alg.2
Iris	3.0	0.6441	0.6578	0.6368	0.6368
Glass	3.3	1.7160	1.7868	1.7149	1.7149
Australian	2.2	0.9022	0.9394	0.9022	0.9022
Breast	3.2	0.4342	0.4975	0.4342	0.4342
Diabetes	2.5	0.8570	0.8740	0.8567	0.8567
German	2.0	0.8603	0.8640	0.8603	0.8603
Vehicle	3.9	1.7332	1.7584	1.7330	1.7330
Robotics	5.4	2.1681	2.3937	2.1681	2.1681
Satellite	10.2	1.5823	1.6258	1.5797	1.5797
Waveform	5.7	1.3304	1.3304	1.3294	1.3294
Shuttle	16.7	0.6317	N/A	0.6314	0.6314

algorithms were implemented in C++ and the tests were run on a SPARCserver 10/402 machine.

For each numeric attribute, a multi-split was generated using each of the methods in turn. The impurity of the generated split was measured. The average impurity over all numerical attributes is listed in Table 2 for the four test methods. Column ARITY records the average arities of the generated partitions over all numerical attributes. Both the method of Fayyad and Irani (1993) (column FI) and the FKS algorithm often fail in finding the optimal multi-split. On some of the data sets the method of Fayyad and Irani chose a binary split for all or almost all attributes (cf. column ARITY in the table). On those cases our methods could not do any better than the heuristic method, since we let it decide the arity of the split.

Incidentally, we found the method of Fayyad and Irani (1993) to be a bit pessimistic: On many of the smaller data sets, their method decided to refrain from generating any kind of a split on some of the numerical attributes.

As our methods are provably optimal, they found the correct split every time. Though, we must admit that the differences in impurity between the optimal partition and that produced by greedy approach are not large. We did not have enough computing resources to complete the test on the very large Shuttle domain for the FKS algorithm, hence the 'N/A' label in the corresponding cell in Table 2.

The running times of the algorithms are shown in Table 3. Preprocessing time, i.e., time used in sorting the examples is shown separately, since it is common to all the methods. The FKS algorithm's quadratic dependence on the size of the sample is clearly revealed: It is significantly slower than other methods. Limiting the search to only boundary points pays off: The algorithm of Fayyad and Irani and Algorithm 2 are the fastest methods. Their running times do not correlate with the sample size in any way; both methods are fast

Table 3 Average running times (in seconds/split) of methods.

Data set	Preprocess	FI	FKS	Alg.1	Alg.2
Iris	0.006	0.006	0.305	0.019	0.008
Glass	0.009	0.033	0.648	0.364	0.203
Australian	0.034	0.028	5.837	0.535	0.268
Breast	0.036	0.007	6.561	0.007	0.007
Diabetes	0.052	0.029	7.418	0.762	0.312
German	0.083	0.047	11.942	4.886	0.930
Vehicle	0.045	0.029	10.320	0.352	0.229
Robotics	0.129	0.017	71.758	0.019	0.018
Satellite	0.296	0.073	442.398	0.257	0.183
Waveform	0.787	0.231	426.829	12.021	9.937
Shuttle	3.881	0.394	N/A	0.939	0.623

independent of the number of examples in the domain. For small data sets the method of Fayyad and Irani comes out as clear winner in this respect.

As for the large data sets, preprocessing time dominates the total running time, with the exception of the Waveform data set, which consists of truly continuos-valued attributes: Each attribute has over 500 different values in the data and almost as many boundary points. In comparison, the Shuttle domain has on average only little over 100 values and about 80 boundary points per attribute. The large number of boundary points in the noisy Waveform data has a clear effect on the running times of our algorithms. Our conclusion must be that large amounts of attribute noise may affect the running time of our algorithms. However, when contrasted with the FKS algorithm, our approach is clearly better even on the Waveform data.

Sometimes numerical attributes are handled by categorizing them in a preprocessing phase, before submitting the data to the learning algorithm (Langley, 1996). The categorization task, naturally, is related to the problem of finding good multi-splits during decision tree construction. Hence, it is possible to use multi-splitting algorithms to the attribute categorizing task (Catlett, 1991; Pfahringer, 1995): Each multi-split defines a categorization of the numerical attribute in question. It is also plausible that a multi-split algorithm's performance in attribute categorization has at least some correlation to its performance in the duty that it is designed for: Finding good multi-splits.

In our final experiment we categorized each of the data sets using each of the algorithms in turn and let the C4.5 decision tree learner (Quinlan, 1993) build a decision tree based on the categorized data. A 10-fold cross-validation was run 5 times on each data set. Resulting average prediction accuracies are listed in Table 4. For comparison, we have included the accuracy of C4.5 on the original data. Since both our algorithms induce the optimal partition, they produce equal results in this experiment; therefore, we have combined their columns. The best prediction accuracy obtained on each of the data sets is

Table 4 Prediction accuracies of decision trees generated from categorized attributes.

DATA SET	C4.5	FI	FKS	Alg.1 / Alg.2
Iris	95.8	**96.7**	95.3	**96.7**
Glass	69.7	**77.4**	74.7	77.2
Australian	81.0	**81.3**	79.9	**81.3**
Breast	**95.1**	94.7	92.9	94.7
Diabetes	68.6	77.1	**78.0**	76.4
German	**68.0**	65.3	67.3	65.3
Vehicle	**72.4**	71.3	68.1	70.9
Robotics	**99.6**	**99.6**	**99.6**	**99.6**
Satellite	**84.5**	81.7	82.7	82.0
Waveform	**75.2**	72.0	72.4	72.6
Shuttle	N/A	99.8	N/A	**99.9**

highlighted by writing it in boldface font.

The results of this experiment are not unambiguos: Depending on the domain, the method that records the best accuracy varies. With the exception of two domains—Glass and Diabetes—C4.5, on average, seems to perform best with uncategorized data. Only quite small increases in prediction accuracy come about due to categorization in other domains except for the two special ones.

In what concerns the relative order of the results obtained using the splitting algorithms, we can only determine that the categorizations produced by the greedy approach are equally good as the optimal categorization using this particular experiment setup. The categorization generated by the FKS algorithm is, on average, less profitable in class prediction than the optimal categorization.

5 Conclusion and future work

We have presented two efficient algorithms for generating optimal multi-splits for numerical attributes. The first one works for any additive impurity measure and the second one for impurity functions that minimize on boundary points, such as average class entropy.

The empirical experiments have demonstrated that the brute-force dynamic programming method of Fulton *et al.* (1995) systematically fails and that, also, the greedy heuristic technique of Fayyad and Irani (1993) in most cases fails in finding the optimal multi-split for a numerical attribute. On the other hand, the heuristic method is extremely fast. However, sorting has turned out to be a major, often dominating, factor in the splitting process and, therefore, hardly any efficiency differences between our method and that of Fayyad and Irani—or any other method that is based on the binary partitioning—would remain visible, when incorporated into a decision tree learning algorithm.

The experiments have also demonstrated that Algorithm 2 improves Algorithm 1 substantially in what concerns time consumption. Hence, in practice blocks consist of several bins, and $B < V \ll N$. In particular, our methods seem to work reasonably well even in the case of truly continuous attributes and they also scale up very well with respect to the size of the data set.

Our initial experiment on the effects that different categorizing techniques have on decision tree construction did not shed much light on the matter. Resting on the heuristic approach on selecting the arity of the partition seems to make the comparison difficult. We expect clear differences in prediction accuracy and, in particular, the size of the produced decision tree to emerge, when the techniques are actually incorporated into a learning algorithm. Furthermore, we expect another goodness criterion—one that can fight the increase in the arity of attributes—for instance, Quinlan's (1993) *gain ratio*, to work significantly better together with the new algorithms than the vulnerable average class entropy does.

Our results also give some evidence for the claim that it is not so far-fetched to consider searching for the optimal k-way partition of the data over all k, since the algorithms that we presented are efficient enough. Moreover, the number of blocks, B, gives an upper bound for the arity, k, of the resulting partition and $B \ll N$ in practice. This might relieve us from the problem of another evaluation function that is needed to control the resulting partition in many other approaches.

References

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

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

Chou, P. (1991). Optimal partitioning for classification and regression trees. *IEEE Trans. Pattern Anal. Mach. Intell.* **4**: 340–354.

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

Fayyad, U. and Irani, K. (1992). On the handling of continuous-valued attributes in decision tree generation. *Mach. Learn.* **8**: 87–102.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1022–1027). San Mateo, CA: Morgan Kaufmann.

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

Langley, P. (1996). *Elements of Machine Learning.* San Francisco, CA: Morgan Kaufmann.

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

Michie, D., Spiegelhalter, D., and Taylor, C. (eds.) (1994). *Machine Learning, Neural and Statistical Classification.* London: Ellis Horwood.

Murphy, P. and Aha, D. (1994). UCI repository of machine learning databases (http://www. ics.uci.edu/~mlearn/MLRepository.html). Univ. California at Irvine.

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

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

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