Speeding Up the Search for Optimal Partitions

Tapio Elomaa[1] and Juho Rousu[2]

[1] Department of Computer Science, P. O. Box 26 (Teollisuuskatu 23)
FIN-00014 Univ. of Helsinki, Finland, elomaa@cs.helsinki.fi
[2] VTT Biotechnology and Food Research, Tietotie 2, P. O. Box 1501
FIN-02044 VTT, Finland, juho.rousu@vtt.fi

Abstract. Numerical value range partitioning is an inherent part of inductive learning. In classification problems, a common partition ranking method is to use an attribute evaluation function to assign a goodness score to each candidate. Optimal cut point selection constitutes a potential efficiency bottleneck, which is often circumvented by using heuristic methods.

This paper aims at improving the efficiency of optimal multisplitting. We analyze convex and cumulative evaluation functions, which account for the majority of commonly used goodness criteria. We derive an analytical bound, which lets us filter out—when searching for the optimal multisplit—all partitions containing a specific subpartition as their prefix. Thus, the search space of the algorithm can be restricted without losing optimality.

We compare the partition candidate pruning algorithm with the best existing optimization algorithms for multisplitting. For it the numbers of evaluated partition candidates are, on the average, only approximately 25% and 50% of those performed by the comparison methods. In time saving that amounts up to 50% less evaluation time per attribute.

1 Introduction

In inductive processes numerical attribute domains often need to be discretized, which may be time consuming if the domain at hand has a very high number of candidate cut points. This affects both *binarization* [4,14] methods and, in particular, algorithms that need to partition numerical ranges into more than two subsets; e.g., off-line discretization algorithms [5] and optimal [8,11] or greedy [5,10] multisplitters in decision tree learning, rule induction, and nearest neighbor methods. In data mining applications numerical attributes may constitute a significant time consumption bottleneck.

In this paper we continue to explore ways to enhance the efficiency of numerical attribute handling in classification learning. Previous work has shown that the class of *well-behaved* evaluation functions, for which only a part of the potential cut points needs to be examined in *optimal* partition selection, contains all the most commonly used attribute evaluation functions [8].

In this paper we analyze convex attribute evaluation functions. The analysis brings out new opportunities for pruning the set of candidate partitions. Empirical evaluation shows that the speed-up obtained is substantial; on the average,

J.M. Żytkow and J. Rauch (Eds.): PKDD'99, LNAI 1704, pp. 89–97, 1999.

the evaluation of half of the partition candidates can be omitted without sacrificing the optimality of the resulting partition.

2 Preliminaries and an Overview

The processing of a numerical attribute begins by sorting the training data by the value of the attribute. We consider a categorized version of the data, where all examples with an equal value constitute a *bin* of examples.

In supervised learning, the task in 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. The coherence is usually measured by an evaluation function.

Many, though not all [6,8], of the most widely used attribute evaluation functions are either *convex* (upwards) or *concave* (i.e. convex downwards), both are usually referred to as convex functions.

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 said to be strictly convex *if equality holds only if $\rho = 0$ or $\rho = 1$.*

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

Fayyad and Irani's [9] analysis of the binarization technique proved that for the information gain function [13,14] only *boundary points* need to be considered as potential cut points due to the convexity of the function.

Definition 2. *Let a sequence S of examples be sorted by the value of a numerical attribute A. The set of* boundary points *is defined as follows: A value $T \in \mathrm{Dom}(A)$ is a boundary point if and only if there exists a pair of examples $s_1, s_2 \in S$, having different classes, such that $\mathrm{val}_A(s_1) = T < \mathrm{val}_A(s_2)$; and there does not exist another example $s \in S$ such that $\mathrm{val}_A(s_1) < \mathrm{val}_A(s) < \mathrm{val}_A(s_2)$.*

A *block* of examples is the sequence of examples in between two consecutive boundary points. Blocks can be obtained from bins by merging adjacent class uniform bins with the same class label.

A *well-behaved* function always has an optimal multisplit on boundary points. All the most commonly used attribute evaluation functions fall into this category [8], including all convex evaluation functions and some non-convex such as the *gain ratio*, *GR* [13,14] and the *normalized distance* measure, *ND* [12].

Table 1 summarizes the current knowledge of optimization algorithms for families of evaluation functions. Brute-force exhaustive search can be used to optimize any evaluation function, but the search method is exponential in the (maximum) arity, k, of the partition for each attribute. With well-behaved functions like *GR* and *ND*, only boundary points need to be examined, which leads to slightly better efficiency.

Table 1. Types of functions, their optimization algorithms, asymptotic time and space requirements for these algorithms, and examples of functions in these categories.

Type	Algorithm	Time	Space	Functions
Any	Brute-force (bins)	$O(mV^k)$	$O(mV)$	
Well-behaved	Brute-force (blocks)	$O(mB^k)$	$O(mB)$	*GR*, *ND*
Cumulative	`Bin-Opt`	$O((k+m)V^2)$	$O((k+m)V)$	
+ Well-behaved	`Block-Opt`	$O((k+m)B^2)$	$O((k+m)B)$	
+ Convex	`Block-Opt-P`	$O((k+m)B^2)$	$O((k+m)B)$	*ACE*, *IG*, *GI*
Monotonic	One-pass	$O(kmn)$	$O(km)$	*TSE*

Cumulative evaluation functions, i.e., functions that compute a (weighted) sum of goodness scores of the subsets, can be optimized in time quadratic in the number of bins using the general algorithm which uses dynamic programming [11,8]. Subsequently we refer to this algorithm as `Bin-Opt`. If the evaluation function, additionally, is well-behaved, then an algorithm called `Block-Opt` [8] can be used to optimize it in time quadratic in the number of blocks.

This paper introduces a pruning method for minimization of concave and cumulative evaluation functions, which improves the efficiency of the `Block-Opt` algorithm. The asymptotic time requirement does not change, but as demonstrated in the subsequent experiments, the practical speed-up is substantial.

Examples of concave and cumulative evaluation functions include the *gini index* (of diversity), *GI* [4] and the *average class entropy*, *ACE*. For a partition $\biguplus_i S_i$ of the data set S, *ACE* is defined to be

$$ACE(\biguplus_i S_i) = (1/|S|) \sum_i |S_i| H(S_i) = (1/n) \sum_i |S_i| H(S_i),$$

where H is the entropy function: $H(S) = -\sum_{j=1}^{m} P(C_j, S) \log_2 P(C_j, S)$, in which m denotes the number of classes and $P(C, S)$ stands for the proportion of examples in S that have class C.

Many other evaluation functions use *ACE* as their building block. Such functions include, e.g., the *information gain* function, *IG* [13,14], *GR*, and *ND*. In the experiments of Section 5 we use *IG*, which is defined as

$$IG(\biguplus_i S_i) = H(S) - ACE(\biguplus_i S_i)\,.$$

Finally, there is only one concave and cumulative evaluation function that is known to be optimizable in linear time by one-pass evaluation [1,2,11]: *training set error*, *TSE*. Unfortunately, the function has many defects, which disqualify it from application in multi-class induction.

3 Pruning Partition Candidates

The algorithm `Block-Opt` uses dynamic programming to efficiently search all boundary point combinations in order to find the best partition [8]. It uses a left-to-right scan over the blocks and tabulates the goodness scores of prefix

partitions to avoid repetitive calculation of the scores. Although the algorithm works well when there is a moderate amount of boundary points in the range, its efficiency suffers when they are more frequent. This section studies how the search space of the algorithm can be restricted by utilizing the convexity properties of the evaluation functions.

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 $p(x) = \mathbf{Pr}\{X = x\}$.

Theorem 1 (Jensen's inequality [7, pp. 25–26]). *If f is a convex function and X is a random variable, then*

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

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

$$\textstyle\sum_i \alpha_i f(t_i) \leq f\left(\sum_i \alpha_i t_i\right) \tag{1}$$

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 [4]. The interval scores are weighted relative to the sizes of the intervals. Thus, a common form of an evaluation function F is

$$F\left(\biguplus_i S_i\right) = \textstyle\sum_i (|S_i|/|S|) I(S_i), \tag{2}$$

where I is an impurity function. Now, $|S_i|/|S| \geq 0$ and $\sum_i (|S_i|/|S|) = 1$. If the impurity function I is concave, then by Eq. 1:

$$\textstyle\sum_i (|S_i|/|S|) I(S_i) \leq I\left(\sum_i (|S_i|/|S|) S_i\right) \Leftrightarrow F\left(\biguplus_i S_i\right) \leq F(S), \tag{3}$$

in which $F(S)$ is the score of the unpartitioned data. Observe that, since I 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. Hence, in practice, the arity of the partition needs to be bounded, either a priori or by using some penalizing term.

For example, the evaluation function *ACE* fulfills the requirements of the function F above; the entropy function, H, is concave, because function $x \log x$ is convex [7].

Theorem 2. *Let F be the evaluation function defined in Eq. 2 and let I be a concave impurity function. Let S be a sequence of examples consisting of consecutive intervals $S_1, S_2, \ldots, S_m$. 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*

$$|S_1 \cup S_2| F(P_2) - |S_1| F(P_1) - |S_2| I(S_2) \leq 0, \tag{4}$$

then for any example set S_*

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

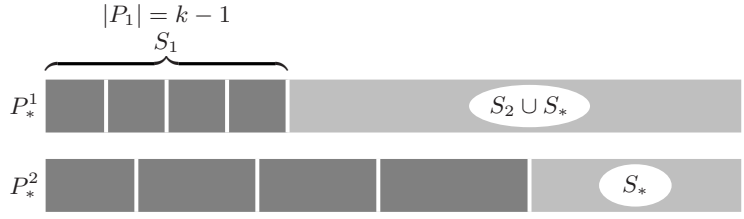

Fig. 1. P_1 and P_2 are two $(k-1)$-partitions of the prefixes of the data set. They can be extended into k-partitions P_*^1 and P_*^2, respectively, for a larger sample by augmenting a new interval to them.

Proof. Let us now consider different k-partitions of $S_1 \cup S_2 \cup S_*$, where S_* is a combination of any number of 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 Fig. 1). Assume that the inequality 4 holds.

According to the inequality of Eq. 3

$$\begin{aligned} |S|F(P_*^1) &= |S_1|F(P_1) + |S_2 \cup S_*|I(S_2 \cup S_*) \\ &\geq |S_1|F(P_1) + |S_2|I(S_2) + |S_*|I(S_*) \text{ and} \\ |S|F(P_*^2) &= |S_1 \cup S_2|F(P_2) + |S_*|I(S_*). \end{aligned}$$

The difference of these two candidates can be bound from above by the inequality 4

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

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

The theorem gives us a possibility of pruning the search space significantly: 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.

4 The Algorithm for Finding Optimal Partitions

We incorporate the candidate pruning method to algorithm `Block-Opt`, which uses a dynamic programming scheme similar to that suggested by Fulton et al. [11]. The main modification is that the algorithm works on blocks of examples rather than on individual examples. The blocks are extracted in two-pass preprocessing. That entails bin construction from the sorted example sequence and merging of adjacent class uniform bins (of the same class) into blocks. The time and space complexity of preprocessing is $O(n + mV)$.

The search algorithm inputs a sequence $\boldsymbol{\mu}$ of class distributions of the blocks $b_1, \ldots, b_B$, an upper limit for the arity of the partition and an evaluation function of the form $g(\mu) = |S|I(S)$, where I is a concave function, μ is the class distribution of the set S.

Table 2. The search algorithm for multisplits. After executing the algorithm, for each i and k, $P_{i,k}$ is the cost of the optimal k-split of first i blocks and $L_{i,k}$ is the index to the block which is situated immediately left from the rightmost cut point.

```
procedure Search(μ, g, aritymax)
/* μ = {μ1,...,μB} contains the class frequency distributions of blocks
   b1,...,bB, g(μj) = |Sj|I(Sj), where I is a concave function */
method:
1. for i ← 1 to B do
2.     for j ← 1 to i-1 do μj ← μj + μi; costj ← g(μj) od;
3.     Pi,1 ← cost1; Ni,1 ← i-1;
4.     if i = B then limit ← aritymax else limit ← aritymax-1 fi;
       /* Compute the best k-split of b1∪...∪ bi for each k */
5.     for k ← 2 to min(i,limit) do
6.         minimum ← ∞; rejectlevel ← Pi,k−1;
7.         l ← i; j ← Nl,k−1;
8.         while j ≥ k do /* Scan the remaining candidate (k-1)-splits */
9.             current ← Pj,k−1 + costj+1;
10.            if current ≥ rejectlevel then Nl,k−1 ← Nj,k−1 /* prune */
11.            else /* This candidate could be the optimal one */
12.                if current < minimum then
13.                   minimum ← current; indexofmin ← j fi;
14.                l ← j fi;
15.            j ← Nj,k−1 od;
16.        Pi,k ← minimum; Li,k ← indexofmin; Ni,k ← i-1 od od
```

The search algorithm (Table 2) scans the blocks $b_1, \ldots, b_B$ from left to right. 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.

At step i, array P is updated according to the formula:

$$P_{i,k} \leftarrow \min_{j \in N_{i,k-1}} \{P_{j,k-1} + g(\mu_{j+1})\},$$

which denotes that the optimal partitioning of $b_1, \ldots, b_i$ into k subsets is the minimum cost over all combinations—remaining in the search space—of fixing the last interval $\bigcup_{l=j+1}^{i} b_l$ and adding the cost of the best $(k-1)$-split of $b_1 \cup \cdots \cup b_j$.

As the scan proceeds, the distributions of blocks are merged, so that at point i, each $\mu_j, j \leq i$, represents the class distribution of $b_j \cup \cdots \cup b_i$. The corresponding evaluation function score is stored in array *cost*.

Array L stores the corresponding cut points: $L_{i,k}$ is an index to the block that contains the rightmost cut point of the multisplit having the cost $P_{i,k}$.

The search space is pruned incrementally by comparing the best $(k-1)$-split of the intervals processed so far with each remaining candidate k-split of the same range: if $P_{i,k-1} \leq P_{j,k-1} + cost_{j+1}$, the candidate is eliminated. The connection to Theorem 2 is the following: $P_{i,k-1}$ corresponds to P_2, $P_{j,k-1}$ to

P_1, $b_{j+1} \cup \cdots \cup b_i$ to S_2, and an empty set to S_*. The array N stores the search space of remaining partition candidates in linked lists: $j = N_{i,k-1}$ 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 blocks $b_1 \cup \cdots \cup b_i$, is the optimal $(k-1)$-split of the blocks $b_1, \ldots, b_j$.

Note that testing against the best candidate so far is conditioned on passing the pruning test. 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 asymptotic time and space complexities of the algorithm are the same as the algorithm `Block-Opt` [8]. The algorithm takes the time $O((k+m)B^2)$ because the incremental merging of the class distributions take the time $O(mB^2)$ and scanning the table P takes the time $O(kB^2)$ in the worst case. The tables P, L and N are of size $O(kB)$ and the class distributions of the blocks allocate the space $O(mB)$, which leads to the total space complexity of $O((k+m)B)$.

5 Empirical Evaluation

We contrast the multisplitting algorithms `Bin-Opt` and `Block-Opt` with and without the new candidate pruning technique. The pruning version is called `Block-Opt-P`. As baseline we use a breadth-first implementation of Fayyad and Irani's [10] widely used heuristic greedy multisplitting method. Keep in mind that this method does not produce optimal partitions, even though the scores of the resulting partitions often are very close to optimal [8]. As the evaluation function we use information gain [13], which is convex (thus also well-behaved) and cumulative.

In the experiment we partition the numerical dimensions of 31 test domains, which come mainly from the UCI repository [3], using all four partitioning strategies. For each domain we record the number of candidate partitions evaluated in processing each numerical attribute.

Fig. 2 depicts the results of this experiment. The figures on the top are the average number of evaluations per numerical attribute performed by the algorithm `Bin-Opt`, which operates on example bins, the white bars represent the relative number of evaluations per attribute for the algorithm `Block-Opt` operating on blocks, the gray bars are those of `Block-Opt-P`, where the new candidate pruning is employed, and the black ones correspond to those of the greedy heuristic selection.

We can see that the average reduction in the number of examined partitions between `Block-Opt` and `Bin-Opt` is close to 50%. An average reduction of the same size is obtained when pruning is employed. Hence, the total average saving in candidate evaluations between `Bin-Opt` and `Block-Opt-P` is approximately 75%. These reductions do not correspond linearly to the search times. For instance, in the domain Adult pruning only filters 15% of the candidate partitions examined by `Block-Opt`, but in time that amounts to a relative saving of 32%. The time consumption of `Block-Opt-P` is only 14% of that of `Bin-Opt`. As ano-

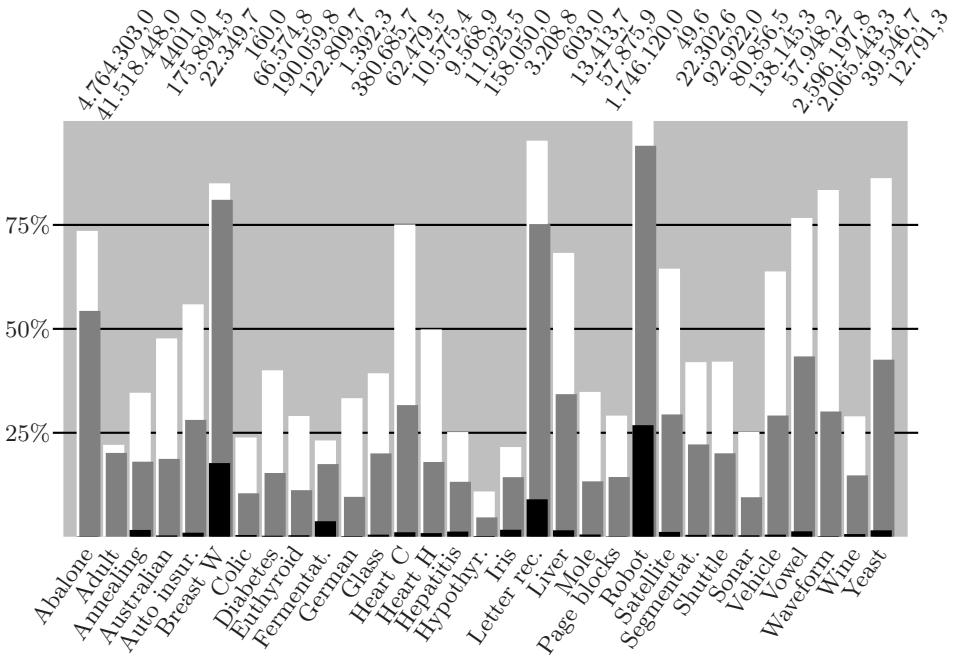

Fig. 2. The relative average numbers of partition candidate evaluations per attribute performed by the the algorithms `Block-Opt`(white bars), `Block-Opt-P`(gray bars), and the greedy approach (black bars). The figures on the top are the absolute averages for the algorithm `Bin-Opt`.

ther example, in the domain Vowel the filtering leaves unevaluated 43% of the partitions examined by `Block-Opt`, but the relative time saving is only 5%.

Pruning attains only small savings in domains with the least numbers of initial comparisons (e.g., Breast W and Robot). In these domains the time consumption is low to begin with. On other domains better pruning results are observed. Unfortunately, the relative reduction in the number of examined partition candidates is small also for some of the hardest domains to evaluate (Abalone and Adult). The actual time saving, though, can be larger as demonstrated above.

Only in some domains, those where the number of comparisons per attribute is the least, is the pruning technique's efficiency comparable with that of the $O(kB)$ time greedy multisplitting method. However, the greedy method is not guaranteed to find the optimal partition.

6 Conclusion

Multipartition optimization lacks an efficient general solution. However, specific subclasses of attribute evaluation functions can be optimized in polynomial time. In particular, cumulative and well-behaved evaluation functions can be optimized in time quadratic in the number of blocks in the numerical domain. It

seems unlikely that this asymptotic bound could be improved without trading off generality. Linear-time optimization would seem to require that the goodness score of the best partition changes monotonically during the search procedure, as happens with *TSE*.

The class of convex evaluation functions is a large one, including many of the commonly used functions. In this paper we bound the value that can be obtained by a partition determined by a convex evaluation function. With the analytical bound we were able to reduce the number of partition candidates that need to be evaluated in optimizing any convex evaluation function. The pruning technique does not improve the asymptotic time requirement of optimizing a convex function, but it does have a great impact on the practical time consumption of the search algorithm.

References

1. Auer, P.: Optimal splits of single attributes. Unpublished manuscript, Institute for Theoretical Computer Science, Graz University of Technology (1997)
2. Birkendorf, A.: On fast and simple algorithms for finding maximal subarrays and applications in learning theory. In: Ben-David, S. (ed.): Computational Learning Theory, Third European Conference. Lecture Notes in Artificial Intelligence, Vol. 1208, Springer-Verlag, Berlin Heidelberg New York (1997) 198–209
3. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html` (1998)
4. Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J.: Classification and Regression Trees. Wadsworth, Pacific Grove, CA (1984)
5. Catlett, J.: On changing continuous attributes into ordered discrete attributes. In: Kodratoff, Y. (ed.): Machine Learning – EWSL-91, Fifth European Working Session on Learning. Lecture Notes in Computer Science, Vol. 482. Springer-Verlag, Berlin Heidelberg New York (1991) 164–178
6. Codrington, C. W., Brodley, C. E.: On the qualitative behavior of impurity-based splitting rules I: The minima-free property. Mach. Learn. (to appear)
7. Cover, T., Thomas, J.: Elements of Information Theory. Wiley, New York (1991)
8. Elomaa, T., Rousu, J.: General and efficient multisplitting of numerical attributes. Mach. Learn. **36** (1999) to appear
9. Fayyad, U. M., Irani, K. B.: On the handling of continuous-valued attributes in decision tree generation. Mach. Learn. **8** (1992) 87–102
10. Fayyad, U. M., Irani, K. B.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann, San Mateo, CA (1993) 1022–1027
11. Fulton, T., Kasif, S., Salzberg, S.: Efficient algorithms for finding multi-way splits for decision trees. In: Prieditis, A., Russell, S. (eds.): Machine Learning: Proceedings of the Twelfth International Conference. Morgan Kaufmann, San Francisco, CA (1995) 244–251
12. López de Màntaras, R.: A distance-based attribute selection measure for decision tree induction. Mach. Learn. **6** (1991) 81–92
13. Quinlan, J. R.: Induction of decision trees. Mach. Learn. **1** (1986) 81–106
14. Quinlan, J. R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)