Building multi-way decision trees with numerical attributes

Fernando Berzal *
berzal@acm.org

Juan-Carlos Cubero *
JC.Cubero@decsai.ugr.es

Nicolás Marín
nicm@decsai.ugr.es

Daniel Sánchez
daniel@decsai.ugr.es

Abstract

Decision trees are probably the most popular and commonly-used classification model. They are recursively built following a top-down approach (from general concepts to particular examples) by repeated splits of the training dataset. When this dataset contains numerical attributes, binary splits are usually performed by choosing the threshold value which minimizes the impurity measure used as splitting criterion (e.g. C4.5 gain ratio criterion or CART Gini's index). In this paper we propose the use of multi-way splits for continuous attributes in order to reduce the tree complexity without decreasing classification accuracy. This can be done by intertwining a hierarchical clustering algorithm with the usual greedy decision tree learning.

Index terms: supervised learning, classification, decision trees, numerical attributes, hierarchical clustering.

*Contact information: Fernando Berzal Galiano (berzal@acm.org) and Juan Carlos Cubero Talavera (JC.Cubero@decsai.ugr.es), Department of Computer Science and Artificial Intelligence, University of Granada, 18071, Spain.

1 Introduction

It is well-known [10] [23] that decision trees are probably the most popular classification model. Commonly-used decision trees are usually built following a top-down approach, from general concepts to particular examples. That is the reason why the acronym TDIDT, which stands for Top-Down Induction of Decision Trees, is used to refer to this kind of algorithms.

The final aim of the decision tree learning process is to build a decision tree which conveys interesting information in order to make predictions and classify previously unseen data.

TDIDT algorithms usually assume the absence of noise in input data and they try to obtain a perfect description of data. This is usually counterproductive in real problems, where management of noisy data and uncertainty is required. Pruning techniques (such as those used in ASSISTANT and C4.5) have proved to be really useful in order to avoid overfitting. Those branches with lower predictive power are usually pruned once the whole decision tree has been built.

The TDIDT algorithm family includes classical algorithms, such as CLS (Concept Learning System), ID3 [22], C4.5 [24] and CART (Classification And Regression Trees) [4], as well as more recent ones, such as SLIQ [21], SPRINT [28], QUEST [18], PUBLIC [27], RainForest [11], BOAT [9], and ART [2].

Some of the above algorithms build binary trees, while others induce multi-way decision trees. However, when working with numerical attributes, most TDIDT algorithms choose

a threshold value in order to perform binary tests. The particular tests which are used to branch the tree depend on the heuristics used to decide which ones will potentially yield better results.

The rest of our paper is organized as follows. Section 2 discusses the heuristics mentioned in the previous paragraph. Section 3 introduces the binary splits which are usually employed to branch decision trees when continuous attributes are present. Section 4 describes how to build multi-way decision trees using a hierarchical clustering algorithm. In Section 5, we present some empirical results we have obtained by applying our alternative approach to build decision trees. Finally, some conclusions and pointers to future work are given in Section 6. There is an appendix available upon request where we describe alternative similarity measures which can be used in our hierarchical clustering algorithm.

2 Splitting criteria

Every possible test which splits the training dataset into several subsets will eventually lead to the construction of a complete decision tree, provided that at least two of the generated subsets are not empty.

Each possible test must be evaluated using heuristics and, as most TDIDT algorithms perform a one-ply lookahead heuristic search without backtracking (i.e. they are greedy), the selected heuristics plays an essential role during the learning process. For instance, most TDIDT algorithms decide how to branch the tree using some measure of node impurity. Such heuristics, splitting rules henceforth, are devised to try to obtain the "best" decision

tree according to some criterion. The objective is usually to minimize the classification error, as well as the resulting tree complexity (following Occam's Economy Principle).

Several splitting rules have been proposed in the literature. CART [4] uses Gini index to measure the class diversity in the nodes of a decision tree. ID3 [22] attempts to maximize the information gain achieved through the use of a given attribute to branch the tree. C4.5 [24] normalizes this information gain criterion in order to reduce the tree branching factor and [25] adjusts C4.5 criterion to improve its performance with continuous attributes. Lopez de Mantaras [19] proposed an alternative normalization based on a distance metrics. Taylor and Silverman [30] proposed the mean posterior improvement (MPI) criterion as an alternative to the Gini rule for binary trees. Berzal et al. [1] introduce two alternative splitting criteria which are dependent only on the most common class in each node and, although simpler to formulate, are as powerful as previous proposals.

All the above-mentioned criteria are impurity-based functions, although there are measures which fall into other categories: some of them measure the difference among the split subsets using distances or angles, emphasizing the disparity of the subsets (on binary trees, typically), while others are statistical measures of independence (a χ^2 test, for example) between the class proportions and the split subsets, emphasizing the reliability of class predictions. Further information about splitting criteria can be found in the references, such as Martin's extensive survey [20] and Shih's study focused on binary decision trees [29].

3 Binary splits for numerical attributes

The splitting criteria reviewed in the previous section provide a mechanism for ranking alternative divisions of the training set when building decision trees. Obviously, there must be some way of generating possible divisions of the training set. In other words, the alternative tests which lead to different decision trees must be enumerated in order to be ranked.

Most TDIDT algorithms define a template of possible tests so that it is possible to examine all the tests which match with the template. Those tests usually involve a single attribute because it makes the trees easier to understand and sidesteps the combinatorial explosion that results if multiple attributes can appear in a single test [24].

C4.5-like algorithms, which build multi-way decision trees for discrete attributes, check the value of such categorical attributes and build a branch for each value of the selected attribute. More complex tests are also allowed, by grouping values of the attribute in order to reduce the tree branching factor. In the extreme case, all attribute values are clustered into two groups in order to build binary trees (as in CART). However, we feel that this strategy makes the trees more complex to understand for human experts.

When an attribute A is continuous, i.e. it has numerical values, a binary test is usually performed. This test compares the value of A against a threshold t: $A \leq t$.

In spite of their apparent complexity, those tests are easy to formulate since you can sort the training cases on the values of A in order to obtain a finite set of values $\{v_1, v_2..v_n\}$. Any threshold between v_i and v_{i+1} will have the same effect when dividing the training set,

so that you just have to check $n-1$ possible thresholds for each numerical attribute A. It might seem computationally expensive to examine so many thresholds, although, when the cases have been sorted, this can be performed in one sequential scan of the training set.

It should be noted that TDIDT performance can also be improved by AVC-sets [11] or any other scalability-oriented implementation technique, such as the middleware described in [5].

Once you can determine that the best possible threshold is between v_i and v_{i+1}, you must choose an accurate value for the threshold to be used in the decision tree. Most algorithms choose the midpoint of the interval $[v_i, v_{i+1}]$ as the actual threshold, that is

$$t_i = \frac{v_i + v_{i+1}}{2}$$

C4.5, for instance, chooses the largest value of A in the entire training set that does not exceed the above interval midpoint:

$$t_i = max \left\{ v \mid v \leq \frac{v_i + v_{i+1}}{2} \right\}$$

This approach ensures that any threshold value used in the decision tree actually appears in the data, although it could be misleading if the value sample in the training dataset is not representative.

Here we propose a slightly different approach which consists of choosing a threshold between v_i and v_{i+1} depending on the number of training instances below and above the threshold. If there are L instances with $v \leq v_i$, and R examples $v \geq v_{i+1}$, then the threshold t_i is

$$t_i = \frac{R * v_i + L * v_{i+1}}{L + R}$$

Let us emphasize that the number of instances whose value of A is greater than the threshold (R) multiplies to the threshold lower bound v_i while the upper bound v_{i+1} is multiplied by the number of examples below the threshold (L).

This way, the threshold will be nearer to v_i than to v_{i+1} if there are less training instances falling in the left branch of the tree, which is the branch corresponding to the examples with $v \leq v_i$. Similarly, the threshold will be nearer to v_{i+1} if the majority of the training examples have values of A below or equal to v_i.

Although this slight modification to the usual algorithm will usually not yield any classifier accuracy improvement, we find that the decision trees it produces are more appealing, specially when the resulting trees are quite unbalanced (which, on the other hand, is a common situation when continuous attributes are present).

4 Multi-way splits for numerical attributes

If we had only categorical attributes, we could use any C4.5-like algorithm in order to obtain a multi-way decision tree, although we would usually obtain a binary tree if our dataset included continuous attributes.

Using binary splits on numerical attributes implies that the attributes involved should be able to appear several times in the paths from the root of the tree to its leaves. Although these repetitions can be simplified when converting the decision tree into a set of rules, they make the constructed tree more leafy, unnecessarily deeper, and harder to understand

for human experts. As stated in [25], "non-binary splits on continuous attributes make the trees easier to understand and also seem to lead to more accurate trees in some domains".

On the other hand, if we use multi-way splits and we are confident enough that the selected split properly discriminates among the problem classes, then we can leave the attribute out once it has been used and the resulting tree will be smaller, easier to understand, and faster to build, while its accuracy will not be significantly hurt.

We could sidestep this problem if we group all numeric attribute values before building the decision tree (a.k.a. global discretization). Those attributes could then be treated as if they were categorical. However, it would be desirable to find some way to cluster the numeric values of continuous attributes depending on the particular situation (i.e. local discretization), since the resulting intervals could widely vary among the different steps in the decision tree induction process.

Our aim here is to propose an alternative method for handling numeric attributes when building decision trees which generates suitable intervals for those attributes depending on the context. As we will see, using multi-way splits will decrease the average tree depth and, therefore, it will lead to trees which are easier to interpret by humans.

4.1 Classical value clustering

Once our objectives are clear, we face the problem of how to group values in a meaningful way. This well-known problem, the clustering of patterns according to a similarity metric, is also known as unsupervised learning, in contrast to the supervised learning that TDIDT

algorithms perform.

Given a set of points in a high-dimensional space, clustering algorithms try to group those points into a small number of clusters, each cluster containing similar points in some sense. In other words, clustering is the non-supervised classification of patterns: the process of generating classes without any a priori knowledge. Good introductions to this topic can be found in [12] and [32].

As TDIDT algorithms, most clustering methods are heuristic, in the sense that local decisions are made which may or may not lead to optimal solutions (i.e. good solutions are usually achieved although no optimum is guaranteed). In their quest for better results, researchers have proposed increasingly complex algorithms such as ISODATA [31], which stands for Iterative Self-Organizing Data Analysis Techniques, with the final A added to make the name pronounceable. However, the results obtained by most iterative algorithms, such as the k-Means algorithm and all its variants, depend on the order in which patterns are presented to the algorithm. Modern search strategies, such as GRASP (Greedy Randomized Adaptive Search Procedure), which can be used in conjunction with the classical k-Means algorithm, and also graph-based methods have been used to solve this problem. While the latter are impractical in real-world problems, the former techniques can yield excellent results when combined with classical methods.

A great number of clustering algorithms have been proposed in the literature, most of them from the Pattern Recognition research field. In general, we can classify clustering algorithms into centroid-based methods and hierarchical algorithms.

- Centroid-based clustering algorithms characterize clusters by central points (a.k.a. centroids) and assign patterns to the cluster of their nearest centroid. The k-Means algorithm is the most popular algorithm of this class, and also one of the simplest.

- Hierarchical clustering methods are incremental. Agglomerative (a.k.a. merging or bottom-up) hierarchical clustering methods begin with a cluster for each pattern and merge nearby clusters until some stopping criterion is met. Alternatively, divisive (a.k.a splitting or top-down) hierarchical clustering methods begin with a unique cluster which is split until the stopping criterion is met. In this paper we propose a hierarchical algorithm in order to build multi-way decision trees using numeric attributes, although we use a novel approach to decide how to cluster data.

4.2 Discretization techniques

Splitting a continuous attribute into a set of adjacent intervals, also known as discretization, is a well-known enabling technique to process numerical attributes in Machine Learning algorithms [16]. It not only provides more compact and simpler models which are easier to understand, compare, use, and explain; but also makes learning more accurate and faster [6]. In other words, discretization extends the borders of many learning algorithms [16].

Discretization methods used as an aid to solve classification tasks are usually univariate (for the sake of efficiency) and can be categorized according to several dimensions:

- **Supervised vs. unsupervised** (depending on their use of class information to cluster data). Supervised methods employ the class information to evaluate and

choose the cut-points, while unsupervised methods do not.

- **Dynamic (local) vs. static (global)**. In a global method, discretization is performed beforehand. Local methods discretize continuous attributes in a localized region of the instance space (as in C4.5) and their discretization of a given attribute may not be unique for the entire instance space.

The simplest discretization methods create a specified number of bins (e.g. equiwidth and equidepth). Supervised variants of those algorithms include Holte's One Rule Discretizer [15].

Information Theory also provides some methods to discretize data (as the entropy and gain ration splitting criteria used in decision trees). Fayyad and Irani's multi-interval discretization based on Rissanen's Minimum Description Length Principle (MDLP) is a prominent example [8]. Mantaras Distance can be used as a variation of the original Fayyad and Irani's proposal [19] and Van de Merckt's contrast measure provides another alternative [33].

Other approaches include Zeta [14], which employs a measure of strength of association between nominal values: the maximum achievable accuracy when each value of a feature predicts a different class value. The χ^2 statistics has also been used in discretization algorithms such as ChiMerge, Chi2, and ConMerge. A survey of such methods an a taxonomy proposed to categorize them can be found in [16]. Dougherty, Kohavi and Sahami's paper [6] can also be a good starting point for the interested reader.

4.3 An alternative approach: Taking context into account

Once we have described how classical clustering algorithms work and some previous work on discretization algorithms, we will present a different viewpoint in order to formulate a clustering algorithm which will help us to group numeric values into intervals. Those intervals will then be used to build multi-way decision trees in classification tasks.

The naive approach to build multi-way decision trees using numerical attributes consists of using any well-known classical clustering algorithm with the set of one-dimensional patterns which correspond to the values of the continuous attribute we evaluate. For example, we could use a graph-based theoretical algorithm which finds a minimum spanning tree for the training set and removes its longest edges, as described in [3]. Although it is easy to implement in the one-dimensional case and optimal from a theoretical point of view (that is, it maximizes inter-cluster distance), this approach is useless for our purposes, since it does not take into account whether the generated split leads to a better decision tree or to a worse one. In other words, it does not consider our classification problem context.

The actual value distribution of a continuous attribute we would use in a traditional unsupervised clustering algorithm is completely meaningless for clustering individual values into intervals given our classification purposes.

Supervised discretization techniques (see Section 4.2) do take class information into account, although they usually focus on some kind of purity measure associated to each interval resulting from the discretization process. Here we prefer to measure the similarity between adjacent intervals in order to choose the appropriate cut-points which delimit.

Henceforth, we will consider the class distribution for each value in order to group adjacent values and build the intervals that lead to the discretization of a continuous attribute.

In order to make our multi-way TDIDT algorithm context-sensitive, we redefine the notion of pattern used by our clustering algorithms, which will be a standard hierarchical clustering algorithm.

Clustering algorithms usually compute the distance between patterns using their feature space. Here we prefer to use the distance between class distributions corresponding to adjacent values because adjacency relationships are established beforehand by the attribute values. In fact, any distance metrics could be used between attribute values in order to determine adjacency relationships. Since adjacent values in the training dataset will always be the nearest ones in the traditional sense, it is unnecessary to compute these actual distances. Obviously, such implicit distance measurements are not enough to solve our problem. We need to introduce a criterion to decide which pair of adjacent values or intervals to combine (if we follow a bottom-up approach) or where to split a given interval (if we decide to employ a top-down algorithm). The differences between class distributions for attribute values in the training set will help us decide how to merge adjacent values or where to split a given interval. That is, intervals will be merged/split using a classical hierarchical clustering algorithm.

It should be noted that this technique can be applied to any of the existing TDIDT algorithms, from C4.5 to RainForest. Our method actually constitutes a general-purpose supervised discretization technique by itself.

Given a set of adjacent intervals I_1, I_2.. I_n for attribute A, we will characterize each

one of them with the class distribution of the training examples it covers. If there are J different problem classes, each interval I will have an associated characteristic vector $V_I = (p_1, p_2, .., p_J)$, where p_j is the probability of the jth class in the training examples which have their A value included in the interval I.

4.3.1 An example

Let us consider a three-class classification problem. Let a be a continuous attribute with four observed attribute values (namely v_1, v_2, v_3, and v_4). In this case, every value v_i of the continuous attribute a is characterized by a 3-dimensional vector V_i which contains the relative frequency for each class in the instances belonging to the training dataset whose attribute a takes the value v.

Let us suppose that our characteristic vectors are $V_1 = (0.3, 0.4, 0.3)$, $V_2 = (0.2, 0.6, 0.2)$, $V_3 = (0.8, 0.1, 0.1)$, and $V_4 = (0.6, 0.4, 0.0)$. We evaluate the construction of a 4-way decision tree using the subsets corresponding to each attribute value and, after that, we merge the most similar pair of adjacent value distributions. If we use the squared Euclidean distance $d_2^2(V_i, V_j)$ to measure vector dissimilarity, we obtain $d_2^2(V_1, V_2) = 0.06$, $d_2^2(V_2, V_3) = 0.62$, and $d_2^2(V_3, V_4) = 0.14$. Since (V_1, V_2) is the most similar pair of adjacent vectors, we merge them and obtain $V_{12} = (0.25, 0.5, 0.25)$ if both v_1 and v_2 represent the same number of training instances. We then evaluate the 3-way decision tree which would result from using $\{v_1, v_2\}$, $\{v_3\}$, and $\{v_4\}$. Again, we compute the distance between adjacent vectors in order to obtain $d_2^2(V_{12}, V_3) = 0.3475$ and $d_2^2(V_3, V_4) = 0.14$, so we decide to merge v_3 and v_4. Our current clusters are, therefore, $\{v_1, v_2\}$ and $\{v_3, v_4\}$. We evaluate the corresponding

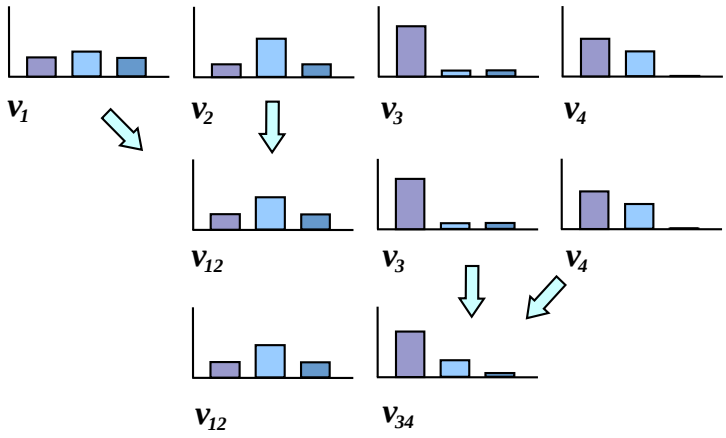

Figure 1: Iterative grouping of adjacent intervals.

binary decision tree and terminate the execution of our algorithm, since there is no need to continue merging adjacent intervals. The whole process is depicted in Figure 1.

It should be noted that only two distances must be recomputed in each iteration of our algorithm. Once v_i is merged with v_{i+1}, the resulting class distribution $V_{i,i+1}$ is computed. We just need to evaluate the similarity between $V_{i,i+1}$ and its neighbours (V_{i-1} and V_{i+2}) because the other class distributions and similarity measurements remain unchanged.

4.3.2 Measuring similarity

Although we have used the Euclidean distance in the example above, others similarity measures are also feasible.

Any feasible similarity measure should encourage the combination of adjacent intervals when their most common class is the same, although this naive criterion is useless in problems where the class populations are quite unbalanced. To state it more clearly, if

95% of the training examples belong to the same class, such a simple similarity criterion would eventually merge all attribute values into a single interval in a meaningless way, since the most common class would probably be the same for all the possible interval partitions.

In order to avoid such a pathological behaviour, we will have to use similarity functions which take into account the class probability for every problem class, not just the most common one.

Table 1 summarizes several similarity measures which have been proposed in the literature and can be used to solve our problem. A more in-depth description of those similarity measures can be found in an extended version of this paper available upon request.

4.3.3 Interleaving the hierarchical clustering algorithm with the TDIDT evaluation process

We have chosen a hierarchical clustering algorithm because it allows us to intertwine its execution with the evaluation of alternative training set partitions in order to build a multi-way decision tree. Hierarchical agglomerative clustering methods (HACM) have been widely used in several fields, including information retrieval and a good survey of them can be found at [26].

Any hierarchical agglomerative clustering method can be described by the following general algorithm:

1. Create one cluster per pattern in the training dataset.

2. Identify the two closest clusters and combine them in a cluster.

Distance-based models	
Minkowski r-metric	$d_r(x,y) = \left(\sum_{j=1}^{J} \lvert x_j - y_j \rvert^r\right)^{\frac{1}{r}}, \quad r \geq 1$
- Euclidean distance	$d_2(x,y) = \sqrt{\sum_{j=1}^{J} (x_j - y_j)^2}$
- Manhattan distance	$d_1(x,y) = \sum_{j=1}^{J} \lvert x_j - y_j \rvert$
- Dominance metric	$d_\infty(x,y) = \max_{j=1..J} \lvert x_j - y_j \rvert$
Bhattacharyya distance	$R(x,y) = \sqrt{1 - \sum_{j=1}^{J} \sqrt{x_j, y_j}}$
Correlation-like similarity measures	
Dot-product	$S_\cdot(x,y) = x \cdot y = \sum_{j=1}^{J} x_j y_j$
Correlation-like index	$\rho(x,y) = 1 - \left(\frac{4}{N_x + N_y}\right) d_2^2$ where $N_v = \sum_{j=1}^{J} (2v_j - 1)^2$
Set-theoretical approaches	
Tversky's model	$s(a,b) = \theta f(A \cap B) - \alpha f(A - B) - \beta f(B - A)$, where $\theta, \alpha, \beta \geq 0$
- Restle's model	$-S_{Restle}(A,B) = \lvert A \square B \rvert$ $-S_\square(A,B) = \sup_x \mu_{A \square B}(x)$
- Intersection model	$S_{MinSum}(A,B) = \lvert A \cap B \rvert$ $-S_{Enta}(A,B) = 1 - \sup_x \mu_{A \cap B}(x)$
Ratio model	$s(a,b) = \dfrac{f(A \cap B)}{f(A \cap B) + \alpha f(A - B) + \beta f(B - A)}$ where $\alpha, \beta \geq 0$
- Gregson's model	$S_{Gregson}(A,B) = \dfrac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$

Notes:

$|A| = \sum_x \mu_A(x)$
$\mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
$\mu_{A \cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$
$\mu_{A \square B}(x) = \max\{\min\{\mu_A(x), 1 - \mu_B(x)\}, \min\{1 - \mu_A(x), \mu_B(x)\}\}$

Table 1: Some similarity measures

3. If more than one cluster remains, return to step 2.

Individual HACMs differ in the way in which the most similar pair is defined, and in the means used to represent a cluster.

In our problem, we start with each separate attribute value as an individual interval and we merge the two most similar adjacent intervals. If we had N intervals, we are left with $N-1$ intervals. This process is repeated until there are only two intervals left, which would lead to a binary decision tree if used to branch it.

Each time we merge two adjacent intervals, we can check the impurity measure associated with the decision tree we would build using the current set of intervals to branch the tree with the numerical attribute we are analyzing. If that measure improves the best one obtained so far, we record the current interval set in order to use it to branch the tree if no better splits are attained in subsequent iterations. Obviously, this technique is orthogonal to the splitting criterion used to evaluate altenative splits, let it be C4.5 gain ratio, Gini index of diversity, or whatever.

The resulting algorithm can be expressed as shown in Figure 2 and allows us to build multi-way decision trees without setting a tree branching factor beforehand. We should, however, set a maximum branching factor in order to reduce the number of alternative tree evaluations performed and, therefore, speed up the construction of the decision tree. This maximum value can be determined by the usual threshold TDIDT algorithms employ to stop the decision tree learning process as a pre-pruning technique: when the resulting subsets of the training dataset contain only a few examples, there is no need to branch

1. Create one interval per pattern in the training dataset.
2. Identify the two closest adjacent intervals and combine them (according to their class distributions).
3. Evaluate the decision tree which would result from using the current set of intervals to branch the tree. If it is the best split known so far, remember it.
4. If more than one interval remains, return to step 2.

Figure 2: Context discretization interleaved within the TDIDT process.

the tree further. Analogously, we can use that threshold to avoid checking partitions of the training dataset during the first stages of the agglomerative clustering algorithm (i.e. when almost every value constitutes an interval of its own).

For every continuous attribute we would perform the above clustering algorithm in order to find the most promising split of the decision tree.

Although traditional HACMs can be usually implemented using algorithms that are $O(N^2)$ in time and $O(N)$ in space, our special-purpose HACM is only $O(N \log N)$ in time. Since interval adjacency is established beforehand and only two similarity measurements have to be computed in each iteration, a heap-like data structure can be employed to perform each iteration in $O(\log N)$ steps.

Alternatively, we could use a divisive top-down hierarchical clustering algorithm starting from a unique interval covering the whole instance space and splitting it in each iteration. Note that the top-down approach is less sensitive to noise in the training set, since

the agglomerative bottom-up algorithm starts with the local class distributions for each value of the continuous feature. The noise effect could also be reduced using a standard binning discretization algorithm beforehand (such as equidepth) and it could also improve our method performance since less similarity measures would need to be computed to get our algorithm started.

In the following section we present the empirical results we have attained using our algorithm to build multi-way decision trees with numerical attributes and compare them with the results obtained using binary splits and previous discretization methods for this kind of attributes.

5 Experimental results

Table 3 summarizes the results we have obtained building decision trees using different local discretization methods for 16 standard small and medium-sized datasets, most of them from the UCI Machine Learning Repository. Those datasets can be downloaded from the following URL:

http://www.ics.uci.edu/∼mlearn/MLRepository.html

All the results reported in this section were obtained using 10-CV (ten-fold cross-validation), Quinlan's gain ratio criterion, and pessimistic pruning (with $CF = 0.25$), as described in [24].

The standard binary-split C4.5 algorithm is compared with several multi-way techniques: our contextual discretizer, three previous supervised discretizers (Holte's One Rule,

Dataset	#Instances	#Features	#Classes
ADULT	48842	14	2
AUSTRALIAN	690	14	2
BREAST	699	9	2
BUPA	245	7	2
CAR	1728	6	4
GLASS	214	10	6
HAYESROTH	160	4	3
HEART	270	13	2
IONOSPHERE	351	34	2
IRIS	150	4	3
PIMA	768	8	2
SPAMBASE	4601	57	2
THYROID	2800	29	2
WAVEFORM	5000	21	3
WINE	178	13	3
YEAST	1484	8	10

Table 2: Datasets used in our experiments

Fayyad and Irani's MDLP, and Ho and Scott's Zeta), and three standard unsupervised discretization methods (the ubiquitous K-Means and two binning algorithms: Equiwidth and Equidepth). To improve our algorithm robustness in the presence of noise, we use an agglomerative clustering algorithm (using the Euclidean distance to measure dissimilarity between class distributions) preceded by Equidepth binning (using 25 bins as a starting point for our hierarchical clustering method).

	C4.5	Context	OneRule	MDLP	Zeta	KMeans	Equiwidth	Equidepth
CV Accuracy	82.00%	82.04%	76.90%	81.44%	74.87%	81.53%	81.06%	81.01%
- Measured error	18.00%	17.96%	23.10%	18.56%	25.13%	18.47%	18.94%	18.99%
- Estimated error	12.74%	14.34%	16.88%	16.44%	19.99%	16.36%	17.27%	17.20%
Training time (seconds)	6.93	12.17	7.30	7.92	15.53	20.82	8.61	8.26
Tree size	110.9	83.8	96.1	42.5	116.8	84.3	93.5	89.0
- Internal nodes	49.9	32.1	21.8	15.8	15.3	24.0	29.0	27.0
- Leaves	60.9	51.7	74.3	26.7	101.6	60.3	64.5	62.0
Average tree depth	3.91	3.40	2.37	3.02	1.84	2.80	4.43	2.70

Table 3: Local discretization experiments summary

Table 3 shows that our method is competitive in terms of classification accuracy and it tends to build small decision trees. It is remarkable that our discretization technique outperforms any other discretization method in its estimated error (that is, the estimation of error used by Quinlan's pessimistic pruning) although there are no big differences in the 10-CV measured error. Regarding to the tree size, our method obtains trees which contain 84% of the nodes required by the corresponding binary tree on average and it is only outperformed by Fayyad and Irani's MDLP discretizer.

The 10-CV classifier accuracy results obtained for each dataset and discretization method are displayed in Table 4. Multi-way decision trees for numerical attributes improved TDIDT accuracy in some experiments. Our method significantly improved binary-decision tree accuracy in three datasets (Bupa, Pima, and Yeast) and was slightly worse in only two datasets (Glass and Wine). Other discretization methods obtain similar results, although our contextual discretization tends to outperform them. Our method did not always obtain accuracy improvements, although the accuracy loss was not significant either. It should be noted, however, that the pessimistic pruning might bias the results in some cases (such as in Bupa when the MDLP criterion is used and the tree is aggresively pruned).

Dataset	C4.5	Context	OneRule	MDLP	Zeta	KMeans	Equiwidth	Equidepth
ADULT	82.60%	82.58%	77.10%	85.27%	80.56%	84.73%	82.92%	82.42%
AUSTRALIAN	85.65%	84.93%	85.80%	85.07%	84.78%	85.51%	84.64%	85.65%
BREAST	93.99%	93.84%	94.85%	94.28%	95.57%	95.28%	95.28%	94.27%
BUPA	66.10%	67.83%	59.76%	57.71%	57.71%	64.71%	62.96%	66.72%
CAR	92.88%	92.88%	92.88%	92.88%	92.88%	92.88%	92.88%	92.88%
GLASS	70.06%	68.77%	59.70%	66.34%	49.46%	70.58%	67.38%	63.46%
HAYESROTH	73.75%	73.75%	73.75%	73.75%	73.75%	73.75%	73.75%	73.75%
HEART	77.04%	76.67%	74.07%	76.67%	73.33%	78.89%	78.52%	79.63%
IONOSPHERE	90.60%	90.32%	87.21%	91.16%	79.76%	88.31%	89.17%	88.60%
IRIS	94.00%	93.33%	94.00%	93.33%	93.33%	95.33%	94.67%	94.67%
PIMA	74.85%	76.81%	68.74%	74.72%	58.19%	74.20%	76.03%	72.76%
SPAMBASE	90.52%	90.24%	87.09%	92.00%	87.83%	91.68%	82.22%	92.11%
THYROID	96.18%	95.61%	96.46%	97.29%	96.04%	95.32%	96.04%	94.54%
WAVEFORM	76.80%	76.50%	58.86%	76.80%	55.36%	74.38%	75.74%	74.58%
WINE	92.71%	91.57%	75.07%	90.92%	74.51%	87.09%	92.68%	88.20%
YEAST	54.25%	56.94%	45.15%	54.79%	44.81%	51.76%	52.09%	51.89%
Average	82.00%	82.04%	76.90%	81.44%	74.87%	81.53%	81.06%	81.01%
Win-Loss-Tie (1%)		3-2-11	0-10-2	3-3-10	1-10-5	5-5-6	3-7-6	2-7-7

Table 4: Classifier accuracy

The accuracy results are more relevant when we take into account that the resulting tree complexity diminishes when multi-way splits are used. The tree size (internal nodes plus leaves) and the average tree depth are reduced. Although it is commonly assumed that binary decision trees have less leaves but are deeper than multi-way trees [20], we have found that this fact does not hold (at least after pruning) since the pruned binary tree can have more leaves than its multi-way counterpart as Table 3 shows for our discretizer and MDLP: you can end up with simpler trees if you use multi-way splits even for numerical attributes.

We have also tested the analyzed discretization methods using global discretization instead of discretizing continuous attributes at each node of the tree. As previous studies have suggested, global discretization improves TDIDT efficiency, reduces tree complexity, and maintains classification accuracy. Table 5 summarizes our results using a top-down contextual discretizer (which is more robust in the presence of noise) and the other discretization methods described above. Our method also performs well when used as a global discretization technique.

	C4.5	Context	OneRule	MDLP	Zeta	KMeans	Equiwidth	Equidepth
CV Accuracy	82.00%	81.92%	77.24%	82.40%	74.91%	81.25%	79.24%	81.16%
- Measured error	18.00%	18.08%	22.76%	17.60%	25.09%	18.75%	20.76%	18.84%
- Estimated error	12.74%	16.60%	20.53%	15.94%	22.12%	16.96%	19.47%	17.24%
Training time (seconds)	6.93	1.62	6.30	1.02	10.54	1.03	1.05	1.02
Tree size	110.9	70.7	77.6	62.7	92.1	85.5	75.4	82.4
- Internal nodes	49.9	21.8	10.1	20.8	9.0	24.4	23.2	24.1
- Leaves	60.9	48.9	67.4	41.8	83.0	61.1	52.2	58.2
Average tree depth	3.91	2.76	1.70	3.03	1.55	2.80	4.09	2.60

Table 5: Global discretization experiments summary

Further experiments whose results are available from the authors upon request have shown that our method achieves interesting results, regardless of the particular similarity measure employed to group continuous values into intervals (see Table 1). In general, distance-based similarity measures tend to obtain uniform results, while correlation-like measures induce more variability in the experiments outcomes. Among the various set-theoretical approaches to measure similarities, Enta's model stands out. However, there are no significant differences among the similarity measures we have tested in our experiments.

It is important to remark that, in spite of the additional computing resources required by the interleaved hierarchical clustering method we propose to build multi-way decision trees, the overall TDIDT asymptotic complexity is preserved and, therefore, our approach is still applicable to data mining problems where decision trees play an important role [10]. Moreover, as our technique is orthogonal to the underlying TDIDT algorithm, it can be appended to efficient implementations of the TDIDT process, such as PUBLIC or RainForest.

6 Conclusions

It is well-known that no particular splitting rule does significantly increase classification accuracy for decision tree classifiers in TDIDT algorithms. Similarly, great accuracy improvements caused by changes in the tree topology cannot be expected. However, slight modifications in the way that decision trees are built can lead to quite different classification models. Although the classification accuracy obtained will be always similar, the

complexity of the resulting trees can vary widely.

In this paper we have proposed an alternative way of handling numerical attributes when building multi-way decision trees so that the resulting trees tend to be smaller and their accuracy is preserved. In fact, the classifier accuracy is even improved in some cases.

Following Occam's Economy Principle, it is commonly agreed that finding new ways to build smaller decision trees is a desirable goal. Our algorithm, like Fayyad and Irani's MDLP, is remarkable because it dramatically reduces tree complexity without hurting classifier accuracy at a reasonable cost. Moreover, knowledge workers (the executives, analysts, and managers who employ decision support systems) usually feel more comfortable with multi-way splits than with binary trees, and this is also true for numerical attributes.

It is also worthwhile to mention that the method we have proposed does not significantly increase TDIDT computational cost and is orthogonal to the particular TDIDT implementation, so that our algorithm can be easily integrated into the algorithms which have been recently devised to handle huge amounts of data, such as PUBLIC [27] and RainForest [11].

Our method can also be used as an alternative discretization technique to be added to the computer scientist toolkit, since it has proven to be efficient and suitable for both local and global discretization. We feel that our focus of measuring dissimilarity between class distributions between adjacent intervals is preferable to previous supervised discretization methods (such as 1R, MDLP, and Zeta) which use some kind of purity measure to evaluate a particular interval without taking into account the nature of neighboring intervals.

References

[1] Berzal, F., Cubero, J.C., Cuenca, F., and Martín-Bautista, M.J. (2002a). *On the quest for easy-to-understand splitting rules*. To be published in Data & Knowledge Engineering.

[2] Berzal, F., Cubero, J.C., Sánchez, D., and Serrano, J.M. (2002b). *ART: A hybrid classification model*. Submitted to Machine Learning.

[3] Bow, S.-T. (1991). *Pattern Recognition and Image Processing*. Marcel Dekker, 1991. ISBN 0-8247-8583-5.

[4] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth, California, USA, 1984. ISBN 0-534-98054-6.

[5] Chaudhuri, S., Fayyad, U., and Bernhardt, J. (1999). *Scalable Classification over SQL Databases*. IEEE: Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, 23 - 26 March, 1999.

[6] Dougherty, J., Kohavi, R., and Sahami, M. (1995). *Supervised and unsupervised discretization of continuous features*. Proceedings of the 12th International Conference on Machine Learning, Los Altos, CA, Morgan Kaufmann, 1995, pp. 194–202.

[7] Dubois, D. and Prade, H. (1993). *Fuzzy sets and probability: misunderstandings, bridges and gaps*. Proceedings of the Second IEEE Conference on Fuzzy Systems, 1993, pp. 1059–1068.

[8] Fayyad, U.M., and Irani, K.B. (1993). *Multi-interval discretization of continuous-valued attributes for classification learning*. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1022-1027.

[9] Gehrke, J., Ganti, V., Ramakrishnan, R., and Loh, W.-Y. (1999a). *BOAT - Optimistic Decision Tree Construction*. Proceedings of the 1999 ACM SIGMOD international conference on Management of Data, May 31 - June 3, 1999, Philadelphia, PA USA, pp. 169-180

[10] Gehrke, J., Loh, W.-Y., and Ramakrishnan, R. (1999b). *Classification and regression: money can grow on trees*. Tutorial notes for ACM SIGKDD 1999 international conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, California, USA, pp. 1-73

[11] Gehrke, J, Ramakrishnan, R., and Ganti, V. (2000). *RainForest - A Framework for Fast Decision Tree Construction of Large Datasets*. Data Mining and Knowledge Discovery, Volume 4, Numbers 2/3, July 2000, pp. 127-162

[12] Han, J., and Kamber, M. (2001). *Data Mining. Concepts and Techniques*. Morgan Kaufmann, USA, 2001. ISBN 1-55860-489-8.

[13] Hipp, J., Güntzer, U., and Nakhaeizadeh, G. (2000). *Algorithms for Association Rule Mining - A General Survey and Comparison*. SIGKDD Explorations, Volume 2, Issue 1, June 2000, pp. 58-64

[14] Ho, K.M., and Scott, P.D. (1997). *Zeta: A global method for discretization of continuous variables*. 3rd International Conferemce on Knowledge Discovery and Data Mining (KDD'97), Newport Beach, CA, AAAI Press, pp. 191-194.

[15] Holte, R.C. (1993). *Very simple classification rules perform well on most commonly used datasets*. Machine Learning, 11:63-90.

[16] Hussain, F., Liu, H., Tan, C.L., and Dash, M. (1999). *Discretization: An enabling technique*. The National University of Singapore, School of Computing, TRC6/99, June 1999.

[17] Lee, J.H., Kim, W.Y., Kim, M.H., and Lee, J.L. (1993). *On the evaluation of boolean operators in the extended boolean retrieval framework*. ACM SIGIR'93, Pittsburgh, pp. 291-297

[18] Loh, W.-Y., and Shih, Y.-S. (1997). *Split Selection Methods for Classification Trees*. Statistica Sinica, Vol.7, 1997, pp. 815-840

[19] Lopez de Mantaras, R. (1991). *A Distance-Based Attribute Selection Measure for Decision Tree Induction*. Machine Learning, 6, pp. 81-92.

[20] Martin, J. K. (1997). *An Exact Probability Metric for Decision Tree Splitting and Stopping*. Machine Learning, 28, pp. 257-291.

[21] Mehta, M., Agrawal, R., and Rissanen, J. (1996). *SLIQ: A Fast Scalable Classifier for Data Mining*. Advances in Database Technology - Proceedings of the Fifth International Conference on Extending Database Technology (EDBT'96), Avignon, France, March 25-29, 1996, pp. 18-32

[22] Quinlan, J.R. (1986a). *Induction on Decision Trees*. Machine Learning, 1, 1986, pp. 81-106

[23] Quinlan, J.R. (1986b). *Learning Decision Tree Classifiers*. ACM Computing Surveys, 28:1, March 1986, pp. 71-72

[24] Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. ISBN 1-55860-238-0.

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

[26] Rasmussen, E. (1992). *Clustering algorithms*. In W. Frakes and E. Baeza-Yates (eds.): *Information Retrieval: Data Structures and Algorithms*, Chapter 16.

[27] Rastogi, R., and Shim, K. (2000). *PUBLIC: A Decision Tree Classifier that integrates building and pruning*. Data Mining and Knwoledge Discovery, Volume 4, Number 4, October 2000, pp. 315-344

[28] Shafer, J.C., Agrawal, R., and Mehta, M. (1996). *SPRINT: A Scalable Parallel Classifier for Data Mining*. VLDB'96, Proceedings of 22nd International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India, pp. 544-555

[29] Shih, Y.-S. (1999). *Families of splitting criteria for classification trees*. Statistics and Computing, vol.9, no.4; Oct. 1999; pp. 309-315.

[30] Taylor, P. C., and Silverman, B. W. (1993). *Block diagrams and splitting criteria for classification trees*. Statistics and Computing, vol. 3, no. 4, pp. 147-161.

[31] Tou, J. T., and Gonzalez, R. C. (1974). *Pattern Recognition Principles*. Addison-Wesley, 1974. ISBN 0-201-07587-3.

[32] Ullman, J. (2000). *Data Mining Lecture Notes*. Stanford University CS345 Course on Data Mining, Spring 2000. http://www-db.stanford.edu/∼ullman/mining/mining.html

[33] Van de Merckt, T. (1993). *Decision trees in numerical attribute spaces*. Proceedings of the 13th International Joint Conference on Artificial Intelligence, pp. 1016-1021.

[34] Zwick, R., Carlstein, E., Budescu, D.V. (1987). *Measures of similarity among fuzzy concepts: A comparative analysis*. International Journal of Approximate Reasoning, 1987, 1:221-242.