Concurrent Discretization of Multiple Attributes

Ke Wang and Bing Liu

Department of Information Systems and Computer Science
National University of Singapore
{wangk,liub}@iscs.nus.edu.sg

Abstract. Better decision trees can be learnt by merging continuous values into intervals. Merging of values, however, could introduce inconsistencies to the data, or information loss. When it is desired to maintain a certain consistency, interval mergings in one attribute could disable those in another attribute. This interaction raises the issue of determining the order of mergings. We consider a globally greedy heuristic that selects the "best" merging from *all* continuous attributes at each step. We present an implementation of the heuristic in which the best merging is determined in a time independent of the number of possible mergings. Experiments show that intervals produced by the heuristic lead to improved decision trees.

1 Introduction

1.1 Motivation

Continuous values, mainly reals and integers, are linearly ordered. Unlike discrete values, there could be many continuous values and each appears only a few times in the data. Directly applying induction algorithms designed for discrete values to continuous values will generate too many rules with poor predictive power. A common technique for handling continuous values is *discretization*, that is, merging adjacent values into intervals if their distinction contributes little to the structure of the problem. However, such mergings could introduce inconsistencies to the data where two examples with the same attribute values have conflicting classes, consequently, leaves little room for an induction algorithm to do its job. On the other hand, when the inconsistency level is constrained, interval mergings in different attributes are no longer independent of each other. Let us explain using an example.

Example 1. Consider the data in Table 1(A) where the underlying concept for Class star is

$$Age \leq 40 \wedge Salary \geq 50K \rightarrow Class = star.$$

ChiMerge [Kerber:1992] starts with one interval per continuous value and repeatedly merges two adjacent intervals that are most similar in the attribute being considered, measured by the smallest χ^2 value of two adjacent intervals. The merging process is stopped by a threshold on the χ^2 value. If the threshold is too small, similar intervals cannot be merged. If the threshold is too large, an attribute, say Age, may be over-merged, which prevents other attributes, say Salary, from being merged due to a consistency requirement. In particular, if all ages were merged into one interval, to keep the data consistent, the intervals in Salary at the best can be S1=[40K], S2=[50K], S3=[54K], S4=[57K,59K], as in Table 1(B). [1] These intervals give rise to the rules

$$Salary \in S2 \rightarrow Class = star$$
$$Salary \in S4 \rightarrow Class = star,$$

which clearly do not capture the underlying concept. A similar argument applies if Salary is merged into one interval first.

A

Age	Salary	Class
33	50K	star
39	57K	star
40	59K	star
45	54K	non-star
35	40K	non-star

B

Age	Salary	Class
A_1	S_2	star
A_1	S_4	star
A_1	S_4	star
A_1	S_3	non-star
A_1	S_1	non-star

C

Age	Salary	Class
A_2	S_2	star
A_2	S_2	star
A_2	S_2	star
A_1	S_2	non-star
A_2	S_1	non-star

Table 1. A motivating example

Suppose we always merge the two adjacent intervals that have the smallest χ^2 value in the two attributes (rather than in the attribute being considered), the following sequence of mergings can be produced:

step 1. Salary: $[40K] \stackrel{2.0}{\rightarrow} [50K] \stackrel{0.0}{\rightarrow} [57K] \stackrel{0.0}{\rightarrow} [59K]$
step 2. Age: $[33] \stackrel{2.0}{\rightarrow} [35] \stackrel{2.0}{\rightarrow} [39] \stackrel{0.0}{\rightarrow} [40] \stackrel{2.0}{\rightarrow} [45]$
step 3. Salary: $[40K] \stackrel{3.0}{\rightarrow} [50K, 57K] \stackrel{0.0}{\rightarrow} [59K]$
step 4. Salary: $[40K] \stackrel{4.0}{\rightarrow} [50K, 59K]$
step 5. Age: $[33] \stackrel{2.0}{\rightarrow} [35] \stackrel{3.0}{\rightarrow} [39, 40] \stackrel{3.0}{\rightarrow} [45]$
step 6. Age: $[33, 35] \stackrel{4.0}{\rightarrow} [39, 40] \stackrel{3.0}{\rightarrow} [45]$
step 7. Age: $[33, 40] \stackrel{1.33}{\rightarrow} [45]$

where "$\rightarrow$" links all adjacent intervals, and the χ^2 value for two adjacent intervals is written between them. For example, at step 3 intervals [50K] and [57K] are merged into [50K,57K], at step 4 intervals [50K,57K] and [59K] are merged into [50K,59K], etc. Steps 4 and 7 give the final intervals for Salary and Age because further mergings make the data inconsistent. Table 1(C) shows the data

[1] Note that we simplify the presentation by considering only observed values for the boundary of an interval.

discretized by these final intervals, from which we can easily induce the correct rules

$$Age \in A_2 \wedge Salary \in S_2 \rightarrow Class = star.$$

This example shows that at each step merging the best pair of adjacent intervals, whatever attributes they come from, leads to better rules.

1.2 Main results

Given an inconsistency threshold, mergings of intervals are no longer independent of each other. We propose a globally greedy heuristic that at each step merges the "best" pair of intervals chosen from *all* continuous attributes, rather than the attribute being considered. The idea is simple: given the limited tolerance of inconsistency, the best merging in all continuous attributes should be considered first. We consider two goodness measures of mergings, the χ^2 value and the change in entropy. A distinctive feature of the global greediness is that mergings in several attributes are *concurrent*, in the sense that mergings in attribute A can be performed without completing all mergings in attribute B.

Two questions need to be answered. First, is it really a good idea to be globally greedy when the goodness of mergings itself is only a heuristic. In other words, can the quality of rules learnt really be improved by being globally greedy. We conducted several experiments to answer this question.

The second question is: can the globally greedy heuristic be implemented efficiently, especially for large datasets. At each step, a critical operation is to find the best merging across all continuous attributes. It does not work to sort all possible mergings by their goodness and perform mergings in the sorted order because early mergings will affect the goodness of later ones. Scanning all pairs of adjacent intervals for each merge is not acceptable because the merging is performed frequently, in the worst case, equal to the number of distinct values. We propose the *Merge-tree* to find the best merging in a time independent of the number of intervals.

2 ConMerge Algorithm

The proposed algorithm, called *ConMerge* (for Concurrent Merger), consists of an initialization step and a bottom-up merging process. In the initialization step, we put each continuous value into its own interval. In the merging process, we repeatedly select the best pair of adjacent intervals from *all* continuous attributes according to a goodness criterion. The selected pair are merged if doing so does not exceed a user-specified inconsistency threshold. (By default, the inconsistency in the original data is used, but can be overridden by a larger value.) If the pair are not merged, the merging of this pair is excluded from further consideration. The merging process is repeated until no more merging is possible. This is described below.

Conceptual **ConMerge**:
Initialization:
  sort observed values for each continuous attribute;
  put each continuous value into its own interval;
Merging process:
  while there are interval pairs to merge **do**
    select the best interval pair from all continuous attributes;
    if merging the pair does not exceed the inconsistency threshold
      then merge the pair into one interval
      else exclude the pair from further merging;

Several questions remain to be answered: (a) how is the inconsistency formally defined, (b) what is a goodness criterion of mergings, (c) how is the above algorithm implemented efficiently, (d) does it produce good intervals. We answer (a) and (b) in the rest of this section. (c) and (d) will be addressed in sections on Implementation and Empirical Evaluation, respectively.

2.1 Inconsistency rate

Inconsistency refers to conflicting class information for examples that agree on all attributes. For a set of examples agreeing on all attributes, called a *matching pattern*, the *inconsistency count* is the number of examples in the set minus the number of examples belonging to a majority class in the set. For example, suppose that, for a set of n examples having the same matching pattern, c_1, c_2, c_3 are the numbers of examples for class 1, 2, and 3, where $c_1 + c_2 + c_3 = n$. If c_1 is the largest, the inconsistency count for the matching pattern is $c_2 + c_3$. The *inconsistency rate* is the sum of all inconsistency counts (for all matching patterns) divided by the total number of examples. The following monotonicity of inconsistency rate implies that if two adjacent intervals cannot be merged because of execeding the inconsistency threshold, they cannot be merged later. (The proof is straightforward, so omitted.)

Theorem 1. *Merging two adjacent intervals does not decrease the inconsistency rate of the data.*

To check the inconsistency threshold, in the merging process we can keep track of inconsistency counts for each interval. Each time two intervals I_1 and I_2 are merged, we find inconsistency counts for the merged interval $I_1 \cup I_2$ by sorting examples in $I_1 \cup I_2$ on all attribute values. If the sorting was kept for each of I_1 and I_2, the sorting for $I_1 \cup I_2$ can be obtained by merging the sorted lists for I_1 and I_2. Therefore, for each merging operation the inconsistency threshold can be checked by a linear scan of examples in the two intervals merged.

In the presence of unknown values, the inconsistency count is defined as follows. If example e has known values on attributes $A_1, \ldots, A_p$, e will match the pattern that agrees with e on $A_1, \ldots, A_p$ and has the largest number of examples. The inconsistency rate is defined as before.

2.2 Merging criteria

We consider two goodness criteria for the merging of two intervals.

The χ^2 value. The χ^2 value of two adjacent intervals, first used in [Kerber:1992], is a statistic measure about how the class is independent of the choice of the two intervals. A smaller χ^2 value implies more independence, or equivalently, less significance in distinguishing the two intervals. Therefore, the smaller the χ^2 value, the more similar the two intervals. The χ^2 value of two adjacent intervals is computed by

$$\chi^2 = \Sigma_{i=1}^{m} \Sigma_{j=1}^{k} \frac{(C_{ij} - E_{ij})^2}{E_{ij}}, \text{ where}$$

$m = 2$ (the 2 intervals being compared)
$k =$ number of classes
$C_{ij} =$ number of examples in ith interval, jth class
$R_i =$ number of examples in ith interval $= \Sigma_{j=1}^{k} A_{ij}$
$C_j =$ number of examples in jth class $= \Sigma_{i=1}^{m} C_{ij}$
$N =$ total number of examples $= \Sigma_{j=1}^{k} C_j$
$E_{ij} =$ expected frequency of $C_{ij} = \frac{R_i * C_j}{N}$

The change in entropy. The entropy for the ith interval I_i is defined as

$$ent(I_i) = -\Sigma_j \frac{C_{ij}}{R_i} log_2 \frac{C_{ij}}{R_i},$$

where C_{ij} and and R_i are as before. The more mixed the classes in interval I_i, the larger $ent(I_i)$. Let $I_1 \cup I_2$ denote the merged interval of I_1 and I_2. The change of the entropy after merging I_1 and I_2 is given by

$$\Delta = ent(I_1 \cup I_2) - \frac{R_1}{R_1 + R_2} ent(I_1) - \frac{R_2}{R_1 + R_2} ent(I_2).$$

(Note that Δ is non-negative [Quinlan:1993].) Δ is the information gain by splitting the merged interval into the two original intervals, or equivalently, the information loss by merging the two intervals. Therefore, merging the two intervals with the minimum Δ minimizes the information loss or maximize the pureness of classes. In the literature, entropy has only been used in the top-down splitting approach.

Suppose that I_0, I_1, I_2, I_3 are 4 adjacent intervals and that I_1 and I_2 are merged into a single interval. The count information C_{ij}, C_j, R_i for the new interval $I_1 \cup I_2$ can be computed from those for I_1 and I_2. Therefore, the χ^2 value or Δ for the affected pairs $(I_0, I_1 \cup I_2)$ and $(I_1 \cup I_2, I_3)$ can be computed efficiently. In the rest of the paper, the χ^2 value and Δ are called *goodness values*.

3 Implementation

At each merging, a critical operation is finding the best pair of adjacent intervals from all continuous attributes. The implementation will affect the efficiency of the algorithm significantly. For large datasets, scanning all pairs of adjacent

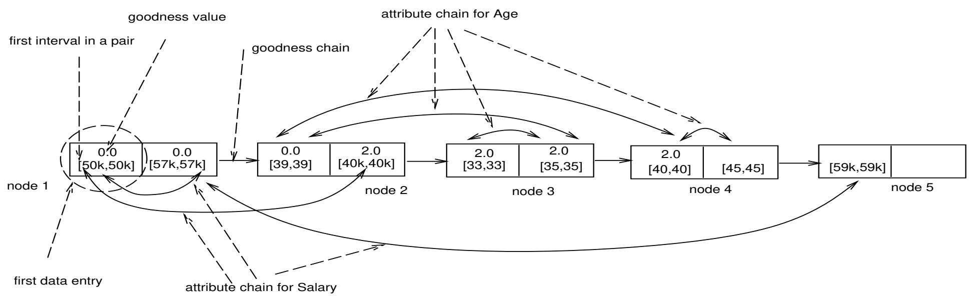

Fig. 1. The leaf level of the Merge-tree

intervals for each merging operation adds one more order to the complexity. A minimum requirement is that the best pair be found in a time independent of the number of intervals. We propose a *Merge-tree* structure, a modified B-tree, to achieve this goal. This is not "just an implementation issue", but an issue that determines how useful the method is in real-world applications.

The Merge-tree. We modify the B-tree into a structure, called the *Merge-tree*, for finding the best merging at the cost of B-tree operations. Each data entry represents a potential merging of two adjacent intervals. (I_1, I_2) denotes the data entry for the potential merging of I_1 and I_2. The search key value for (I_1, I_2) is the goodness value (either χ^2 value or Δ of entropy) of merging I_1 and I_2. As in the B-tree, all leaf nodes are chained in the ascending order of the goodness value, called the *goodness chain*. The best merging is thus represented by the first data entry on the goodness chain.

There are two differences from the B-tree. The first difference is that all data entries for the same continuous attribute are doubly chained according to the adjacency of intervals. This chain is called an *attribute chain*. After merging intervals I_1 and I_2 into a larger interval $I_1 \cup I_2$, by following the two attribute chain pointers in (I_1, I_2), we can find the two affected data entries of the form (I_0, I_1) and (I_2, I_3), which must be replaced with new data entries $(I_0, I_1 \cup I_2)$ and $(I_1 \cup I_2, I_3)$ because I_1 and I_2 were replaced with the new $I_1 \cup I_2$. The second difference is that there are two kinds of data entries in the Merge-tree. Initially, all data entries are *unexamined*. When a pair of adjacent intervals is examined and no merging can be done (because of the inconsistency threshold), the corresponding data entry becomes *non-mergeable*. Since there is no need to search non-mergeable data entries, they will be deleted from the goodness chain. However, a non-mergeable data entry (I_1, I_2) is still kept on the attribute chain. This is because the boundary of I_1 (or I_2) needs to be updated if a merging of the form (I_0, I_1) (or (I_2, I_3)) is performed.

Let us look at one example. Figure 1 shows leaf nodes for attributes Age and Salary in Example 1 before any merging. (Non-leaf nodes are omitted for simplicity.) Each node contains 2 data entries, though typically much more. Each interval, except the first and last for an attribute, is involved in two data entries, one for "left-merging" and one for "right-merging". Instead of storing each interval twice, only the first interval I_1 is stored at data entry (I_1, I_2); I_2 can be found at data entry (I_2, I_3) by following the attribute chain in (I_1, I_2). For example, the first data entry represents the potential merging of intervals [50K,50K] and [57K,57K], the second represents the potential merging of intervals [57K,57K] and [59K,59K], etc.

Suppose that [50K,50K] and [57K,57K] (represented by the first data entry) are merged into [50K,57K]. We need to delete the first data entry and update affected data entries. The affected data entries are ([40K,40K],[50K,50K]) and ([57K,57K],[59K,59K]) because intervals [50K,50K] and [57K,57K] are replaced with the new interval [50K,57K]. By following the attribute chain in the first data entry, we find these affected data entries, delete them, and insert new data entries ([40K,40K],[50K,57K]) and ([50K,57K],[59K,59K]). The insertions are guided by the new goodness values, thus, not necessarily going back to the old places. Since deletion and insertion are standard B-tree operations, we omit the detailed description.

The merging process stops when the Merge-tree becomes empty, at which time all data entries are non-mergeable and are linked by attribute chains. Non-mergeable entries contain the boundary information of all final intervals, which will be used to discretize the testing data. The time for the merging process is $\Sigma_i(c + n_i)$, where c is the constant time for updating the Merge-tree for each merging, as discussed above, and n_i is the number of examples in the two intervals for the ith merging.

4 Empirical Evaluation

To evaluate the effectiveness of the proposed algorithm, we compare three methods: (a) Release 8 of C4.5, denoted C4.5(R8), (b) ConMerge using χ^2, denoted ConMerge(χ^2), and (c) ConMerge using the change of entropy, denoted ConMerge(Δ). Unlike all previous releases, C4.5(R8) improves the performance on continuous values by employing an MDL-inspired penalty to adjust the gain of a binary split of continuous values. [Quinlan:1996] shows that C4.5(R8) compares favorably with other discretization methods. Therefore, we choose C4.5(R8) as a benchmark.

The three methods are applied to 15 datasets chosen from the UCI repository [Merz and Murphy:1996] based on the variety of involvement of continuous attributes. For ConMerge, the procedure is as follows. We partition a dataset into 10 runs using 10-fold cross validation. For each run, ConMerge is applied to the training set to produce intervals, C4.5 is applied to the discretized training set to produce the (pruned) decision tree, and the error rate is collected for the testing set and averaged over 10 runs. In all cases C4.5 was run using the default setting. The default inconsistency threshold is 0% because all original datasets are consistent. The result is shown in Table 2. The numbers following $\pm$ are standard errors.

Dataset	C4.5(R8)		ConMerge(χ^2)		ConMerge(Δ)	
	size	error (%)	size	error (%)	size	error (%)
Anneal	66.8±2.3	7.5±0.5	**46.6± 3.7**	12.2±0.6	**66.3±2.0**	9.5±0.8
Australian	35.7±3.5	14.8±1.1	**29.9±4.3**	15.9±1.2	39.6±3.2	15.5±1.3
Breast-c	28.2±1.7	5.6±0.8	**18.5±1.3**	**5.3±0.6**	**22.2±1.2**	**4.4±0.6**
Bupa	43.8±4.0	34.8±1.5	**41.7± 2.2**	39.1±2.5	48.0±2.7	35.6±1.9
Cleve	77.9±3.8	47.2±2.0	**62.3±3.7**	48.2±2.2	**68.7±5.6**	47.9±2.0
Crx	32.1±4.2	15.0±1.2	39.2±3.6	**13.6±1.5**	**27.4±1.9**	**14.0±1.5**
Diabetes	46.6±4.4	26.1±1.5	**22.7± 1.4**	**24.2±1.4**	**37.8±1.8**	**24.0±1.3**
German	142.8±6.3	26.2±0.7	**90.2± 3.6**	**24.4±0.6**	**122.0±7.5**	28.8±1.6
Heart	36.4±1.8	21.8±1.6	**15.9± 1.1**	**18.1±2.1**	**20.1±6.7**	**16.3±2.2**
Hepatitis	18.2±1.7	18.2±2.0	**5.4±0.5**	**16.9±2.1**	**8.6±2.2**	**17.6±2.1**
Iris	8.8±0.7	4.7±2.0	**6.4±0.4**	**4.7±1.7**	10.0±0.0	5.3±2.4
Labor	5.9±0.9	22.3±5.5	**3.0±0.0**	**12.5±4.2**	**5.1±0.3**	25.0±9.1
Sick-euthyroid	24.1±1.7	2.4±0.3	60.8±1.8	2.6±0.3	**21.1±2.3**	3.8±0.3
Glass	11.0±0.0	2.8±1.0	**9.0±0.0**	**0.5±0.5**	23.1±4.8	8.9±1.6
Wine	9.2±0.2	7.3±2.9	11.2±0.2	**2.2±0.9**	10.2±0.3	**6.2±1.8**
Average	39.2	17.1	**30.9**	**16.0**	**35.3**	17.5

Table 2. Tree size and error rate at default (0%) inconsistency threshold

Tree size. ConMerge(χ^2) and ConMerge(Δ) win over C4.5 12 and 10 out of the 15 cases, respectively, as in bold face. For Hepatitis, Heart, Diabetes, Labor, German, the size produced by ConMerge(χ^2) is only 29%, 44%, 48%, 51%, 63% of the size produced by C4.5. On the other hand, for Sick-euthyroid, the size produced by ConMerge(χ^2) is much larger. This is mainly due to the 0% inconsistency threshold. We will discuss the effect of the threshold below.

Error rate. On the error rate, ConMerge(χ^2) wins over C4.5 10 out of the 15 cases, with the biggest wins for Labor, Glass, and Wine. ConMerge(Δ) wins only 6 out of the 15 cases, therefore, is not more accurate than C4.5.

Effect of inconsistency thresholds. Tables 3 (a) and (b) show tree size and error rate for inconsistency thresholds between 2% and 10%. In general, for both ConMerge algorithms, as the inconsistency threshold is increased, the tree size is reduced and the error rate is increased, as shown by Average in the two tables, because fewer intervals and more inconsistencies are produced. Interestingly, ConMerge(χ^2) performs better at 0% threshold than at 2% threshold, on both error rate and tree size. Compared to C4.5, ConMerge(χ^2) at 2% threshold wins 13 out of 15 cases on tree size, and wins or ties 11 cases out of 15 on error rate.

Dataset	C4.5(R8)	ConMerge(χ^2)					ConMerge($\varDelta$)				
		2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
Anneal	66.8	54.2	18.0	18.0	17.6	17.6	82.4	62.4	62.6	55.0	17.6
Australian	35.7	23.9	43.3	34.2	34.2	34.2	36.0	40.3	34.2	34.2	34.2
Breast-c	28.2	21.2	17.2	7.4	5.0	3.0	23.8	10.0	13.2	5.0	4.8
Bupa	43.8	42.5	37.3	29.3	45.9	35.6	38.7	32.8	27.9	31.9	29.2
Cleve	77.9	63.8	52.5	49.2	50.0	50.4	61.4	60.3	49.5	51.6	50.1
Crx	32.1	33.7	33.7	33.7	33.7	33.7	33.7	33.7	33.7	33.7	33.7
Diabetes	46.6	65.0	66.3	48.1	46.0	33.3	40.0	41.1	28.1	35.6	25.6
German	142.8	73.4	64.1	54.0	49.8	40.6	107.2	103.2	93.6	88.4	76.4
Heart	36.4	22.8	28.2	19.4	22.2	15.4	21.8	21.8	15.8	20.4	16.0
Hepatitis	18.2	3.4	3.4	3.4	3.4	3.4	4.8	3.4	3.4	3.4	3.4
Iris	8.8	6.4	5.0	5.0	5.0	5.0	18.2	13.3	8.4	13.5	12.0
Labor	5.9	3.0	4.4	4.4	4.4	4.4	5.1	4.4	4.4	4.4	4.4
Sick-euthyroid	24.1	11.1	15.4	12.0	12.0	12.0	52.0	16.1	8.8	12.0	12.0
Glass	11.0	10.0	10.0	8.6	8.0	8.0	24.0	24.8	38.2	34.5	32.8
Wine	9.2	11.0	8.4	8.0	8.8	8.8	11.0	10.4	9.2	9.0	7.0
Average	41.4	29.7	27.1	22.3	23.1	20.4	35.3	25.0	26.4	28.8	26.6

(a) Tree size

Dataset	C4.5(R8)	ConMerge(χ^2)					ConMerge($\varDelta$)				
		2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
Anneal	7.5	11.7	19.1	19.1	21.5	21.5	13.4	16.7	16.1	16.3	21.5
Australian	14.8	12.9	17.9	15.5	15.5	15.5	14.6	14.9	15.5	15.5	15.5
Breast-c	5.6	4.6	4.9	6.4	7.4	9.3	4.7	5.1	5.3	7.0	10.3
Bupa	34.8	34.2	35.0	39.1	35.1	33.9	32.1	39.4	31.6	34.8	33.0
Cleve	47.2	49.2	48.1	45.2	46.2	46.8	48.5	47.8	48.5	45.2	45.2
Crx	15.0	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3	14.3
Diabetes	26.1	23.6	21.8	22.8	22.3	23.5	24.5	25.7	25.9	25.1	26.3
German	26.2	27.9	27.3	27.1	26.6	27.2	29.3	30.0	28.2	27.9	27.7
Heart	21.8	17.4	17.4	18.1	18.9	18.1	17.4	16.6	15.9	19.2	19.2
Hepatitis	18.2	16.9	16.9	16.9	16.9	16.9	16.9	16.9	16.9	16.9	16.9
Iris	4.7	4.0	4.0	4.0	4.0	4.0	6.0	9.3	8.0	13.3	17.3
Labor	22.3	12.5	27.5	27.5	27.5	27.5	25.0	27.5	28.5	27.5	27.5
Sick-euthyroid	2.4	2.7	4.7	5.6	5.6	9.3	8.0	8.9	35.0	9.3	9.3
Glass	2.8	2.8	2.8	8.8	7.0	7.0	9.8	9.3	24.3	22.9	22.8
Wine	7.3	2.2	3.4	7.3	6.8	9.6	8.4	8.9	7.3	6.7	9.5
Average	17.1	15.8	17.7	18.5	18.4	19.0	18.2	19.4	21.4	20.1	21.1

(b) Error rate

Table 3. Comparison for different inconsistency thresholds

5 Related Work

Existing interval mergings methods include ChiMerge [Kerber:1992] and StatDisc [Richeldi and Rossotto:1995]. ChiMerge merges two adjacent intervals at a time whereas StatDisc merges several. Both algorithms merge intervals for one attribute at a time. The merging for the current attribute is stopped when the similarity of every two adjacent intervals for the attribute drops below a threshold. The similarity measure depends only on the current attribute and the class attribute, thus, the mergings in one attribute do not affect mergings in other attributes. One problem with these methods is that the user has no control over the inconsistency in the discretized data, and a poorly chosen similarity threshold could either under-discretize the data, where intervals are not merged enough, or over-discretize the data, where the data becomes highly inconsistent. Other works on discretization, e.g., those in [Dougherty, et al.:1995], are less related to our work.

6 Summary

The main contribution in this paper is (a) the establishment of an inconsistency threshold as a quality control factor for discretizing continuous data and (b) a discretization method that handles the attribute interaction raised by the inconsistency threshold. Our method selects the best merging of intervals from all continuous attributes, rather than from the one being considered. We proposed an implementation that finds the best merging in a constant time, thus scales up well in large datasets. Experiments show that by constraining the inconsistency and merging the overall best pair of intervals at each step, the discretized data does produce better decision trees, compared to the latest release of C4.5 improved for handling continuous attributes.

References

[Dougherty, et al.:1995] J. Dougherty, R. Kohavi, M. Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *the 12th International Conference on Machine Learning*, 1995.

[Kerber:1992] R. Kerber. ChiMerge: Discretization of Numeric Attributes. In *Ninth National Conference on Artificial Intelligence*, 1992, 123-128.

[Merz and Murphy:1996] C.J. Merz, P.M. Murphy. UCI Repository of machine learning databases [http://www.ics.uci.edu/ m̃learn/MLRepository.html].

[Quinlan:1993] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Los Altos, CA: Morgan Kaufmann, 1993.

[Quinlan:1996] J.R. Quinlan. Improved Use of Continuous Attributes in C4.5. In *Journal of Artificial Intelligence Research 4*, 1996, 77-90

[Richeldi and Rossotto:1995] M. Richeldi and M. Rossotto. Class-driven Statistical Discretization of Continuous Attributes. In *Proc. of European Conference on Machine Learning* 1995, Springer Verlag, 335-338

This article was processed using the LaTeX macro package with LLNCS style