Relative Unsupervised Discretization for Association Rule Mining

Marcus-Christopher Ludl[1] and Gerhard Widmer[1,2]

[1] Austrian Research Institute for Artificial Intelligence, Vienna,
[2] Department of Medical Cybernetics and Artificial Intelligence,
University of Vienna, Austria

Abstract. The paper describes a context-sensitive discretization algorithm that can be used to completely discretize a numeric or mixed numeric-categorical dataset. The algorithm combines aspects of unsupervised (class-blind) and supervised methods. It was designed with a view to the problem of finding association rules or functional dependencies in complex, partly numerical data. The paper describes the algorithm and presents systematic experiments with a synthetic data set that contains a number of rather complex associations. Experiments with varying degrees of noise and "fuzziness" demonstrate the robustness of the method. An application to a large real-world dataset produced interesting preliminary results, which are currently the topic of specialized investigations.

1 Introduction

Association rules have become one of the most popular objects of study in data mining research. Following the seminal work of e.g. [2], there has been a wealth of research on improvements and extensions of the original algorithms.

One limitation of classical association rule mining algorithms that has been addressed only fairly recently is the fact that they require *categorical data*, i.e., they cannot directly deal with numeric attributes. As numeric information is abundant in real-world databases, this is a problem of practical importance. There have been some attempts at remedying this situation in recent years; section 4 summarizes the most important ones. However, there is still room for improvement, and we hope to show an interesting direction in this paper.

What we present in this paper is a *discretization algorithm* named RUDE that can be used to completely discretize a numeric or mixed numeric-categorical dataset. The algorithm was designed especially with a view to the association rule mining task (see section 2.1). It can be used as a pre-processor to "standard" association rule mining algorithms like Apriori [2]. What distinguishes our algorithm is that it attempts to construct a discretization that as much as possible reflects all the interdependencies between attributes in the database. Experimental results with synthetic data show that RUDE is not only effective, but also quite robust with respect to inaccuracies and noise in the data. We have also applied the algorithm to a large real-world dataset, with interesting preliminary results, which are currently the topic of specialized investigations.

D.A. Zighed, J. Komorowski, and J. Żytkow (Eds.): PKDD 2000, LNAI 1910, pp. 148–158, 2000.

2 RUDE – Relative Unsupervised Discretization

2.1 Discretization for Association Rules

Association rules describe systematic dependencies between fields (attributes) in a relational database. Unlike in classification problems, there is no designated class attribute; rather, any combination of attributes may be related through associations, and many different dependencies involving different sets of attributes may be hidden in the data. The goal is to find all of these. A good discretizer should thus produce discretizations that enable a rule finder both to express and to find all dependencies between attribute ranges, where any attribute may potentially occur in the left-hand or the right-hand side of some rules.

Of course, one could simply use some standard "unsupervised" discretization method, like equal-width, equal-frequency, or k-means [4], and apply it to every numeric attribute. However, these methods simply discretize each attribute in isolation, ignoring any patterns of correlation between attributes; the intervals they construct are thus likely to be irrelevant and unsuitable for describing the hidden dependencies. In association rule mining, some researchers have investigated more sophisticated pre-discretization methods (e.g.,[9]), but again, the attributes are discretized in isolation, the discretization step does not pay attention to systematic dependencies between attributes.

In the field of classification, there has been some research on "supervised" discretization, where, given a numeric attribute to be discretized, one looks at correlations between this attribute and the class attribute in order to create intervals that maximize the numeric attribute's predictivity of the class (see [5] for an overview). While this is not directly applicable for association rule mining, it has inspired the approach to be presented in this paper.

What we propose is a *global* discretization strategy that attempts to construct a discretization of all the numeric attributes which reflects, as much as possible, all the potential dependency patterns between all the attributes simultaneously. The algorithm RUDE (**R**elative **U**nsupervised **D**iscr**E**tization) combines aspects of both unsupervised and supervised discretization. It is an unsupervised method in that it does not require a dedicated class attribute; nonetheless, the split points are not constructed independently of the "other" attributes (hence "relative").

2.2 RUDE – The Top-Level

The basic idea when discretizing a particular attribute (the *target*) is to use information about the value distribution of all attributes other than the target (the *source attributes*). Intuitively, a "good" discretization would be one that creates split points that correlate strongly with changes in the value distributions of the source attributes. The process that tries to accomplish this (the central component of RUDE) is called *structure projection*.

Here is the top level of the algorithm:

1. **Preprocessing:** Discretize (via some unsupervised method – see below) all source attributes that are continuous;
2. **Structure Projection:** Project the structure of each source attribute a_i onto the target attribute t:
   a) Filter the dataset by the different values of attribute a_i.
   b) For each such filtering perform a **clustering** procedure on values of the target attribute (see section 2.4) and gather the split points created.
3. **Postprocessing:** Merge the split points found.

Note that source attributes that are themselves numeric are first subjected to a simple pre-discretization step to turn them into nominal attributes for the purpose of structure projection.

The above algorithm is repeated for every numeric attribute in the database, that is, every numeric attribute in turn gets to act as target and is discretized based on the other attributes. The time required for discretizing one continuous attribute is $O(nm \log m)$, with n the number of attributes and m the number of tuples. A complete discretization of all continuous attributes can therefore be performed in time $O(n^2 m \log m)$.

2.3 The Main Step: Structure Projection

The intuition behind structure projection is best illustrated with an example (see Figure 1). Suppose we are to discretize a target attribute t with a range of, say, [0..1], which happens to be uniformly distributed in our case. The values of t in the tuples of our database have been drawn along the lowest line in Figure 1. The two lines above indicate the same tuples when filtered for the values 1 and 2, respectively, of some particular binary source attribute a. ("Filtering a database for a value v of an attribute a" means retaining only those tuples that have value v for attribute a.) Given the distribution of t, any unsupervised discretizer would return a rather arbitrary segmentation of t that would not reflect the (to us) obvious distribution changes in the source attribute a. The idea of structure projection is to find points where the distribution of the values of a changes drastically, and then to map these "edges" onto the target t. We

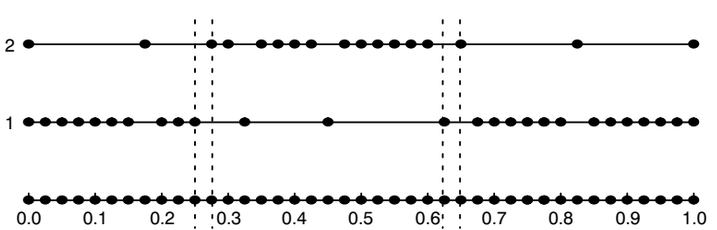

Fig. 1. Structure Projection: An Example.

achieve that by filtering t for each possible value of a separately and clustering the resulting filtered values; the boundaries between clusters are candidates for useful discretization points.

Given:
- a database consisting of m tuples;
- a set of (possibly continuous) source attributes $a_1, \ldots, a_n$
- information on what attribute should be discretized (the target t);

The algorithm:

1. Sort the database in ascending order according to attribute t.
2. For each attribute a_i with $a_i \neq t$ do the following:
   a) If continuous, discretize attribute a_i by *equal-width*
   b) For each symbolic value (interval) v of a_i do the following:
      i. Filter the database for value v in attribute a_i.
      ii. Perform *clustering* on the corresponding values of t in the filtered database.
      iii. Gather the split points thereby created in a *split point list* for attribute t.

Fig. 2. RUDE – The basic discretization algorithm

The basic discretization algorithm is given in fig. 2. RUDE successively maps the "structure" of all source attributes onto the sequence of t's values, creating split points only at positions where some significant distribution changes occur in some of the a_i. For pre-discretizing continuous source attributes in 2(a) above, we have decided to use *equal-width discretization*, because it not only provides a most efficient (linear) method, but also has some desirable statistical properties.

The critical component in all this is the clustering algorithm that groups values of the target t into segments that are characterized by more or less common values of some source attribute a_i. Such segments correspond to relatively densely populated areas in the range of t when filtered for some value of a_i (see Figure 1). Thus, an essential property of this algorithm must be that it tightly delimits such dense areas in a given sequence of values.

2.4 A Characterizing Clustering Algorithm

The algorithm we devised for this purpose has its roots in the concept of *edge detection* in grayscale image processing [7]. The central problem there is to find boundaries between areas of markedly different degrees of darkness. The analogy to our clustering problem is fairly obvious and has led us to develop an algorithm that basically works by opening a "window" of a fixed size around each of the values in an ordered sequence and determining whether this value lies at an "edge", i.e. whether one half of the window is "rather empty" and the other half is "rather full". The notions of "rather full" and "rather empty" are operationalized by two user-defined parameters. For lack of space, we omit the details here.

Given:
• A split point list $s_1, s_2, \ldots$
• A merging parameter (minimal difference s)

The algorithm:

1. Sort the sequence of split points in ascending order.
2. Run through the sequence until you find two splits s_i and s_{i+1} with $s_{i+1} - s_i \leq s$.
3. Starting at $i + 1$ run through the sequence until you find two split points s_j and s_{j+1} with $s_{j+1} - s_j > s$.
4. Calculate the *median* m of $[s_i, \ldots, s_j]$.
   – If $s_j - s_i \leq s$ **merge** all split points in $[s_i, \ldots, s_j]$ to m.
   – If $s_j - s_i > s$ **triple** the set of split points in $[s_i, \ldots, s_j]$ to $\{s_i, m, s_j\}$.
5. Start at s_{j+1} and go back to step 2.

Fig. 3. Merging the split points.

One advantage of the clustering algorithm is that it autonomously determines the appropriate number of clusters/splits, which is in contrast to simpler clustering methods like, e.g., k-means clustering [4]. In fact, the algorithm may in some cases refuse to produce any clusters if it cannot find any justification for splitting. Some numeric attributes may thus be lumped into just one big interval if they do not seem to be correlated with other attributes.

2.5 Post-processing: Merging the Split Points

Of course, due to the fact that RUDE projects multiple source attributes onto a single target attribute, usually many "similar" split points will be formed during the projections. It is therefore necessary to *merge* the split points in a post-processing phase. Figure 3 shows an algorithm for doing that. In step 3, we find a subset of split points with successive differences lower than or equal to a certain pre-defined value s. If all these split points lie closer than s (very dense), they are merged down to only one point (the median). If not, the region is characterized by the median and the two outer borders. (Admittedly, this is a rather *ad hoc* procedure that lacks a strong theoretical foundation.)

3 Experiments

We have evaluated our discretization algorithm as a pre-processing step to the well-known Apriori association rule mining algorithm [2]. In section 3.1, we present a systematic study on a synthetic database that contains a number of rather complex associations. Section 3.2 then briefly hints at some interesting preliminary results on a large, real-world dataset.

3.1 Systematic Experiments with Synthetic Data

As a first testbed for systematic experiments, we used the data and dependencies originally described in [1] and also used in experiments in [10]. In this hypothetical problem, there are six numeric attributes (*salary*, *commission*, *age*, *hvalue*, *hyears*, *loan*) and 3 categorical attributes (*elevel*, *car*, *zipcode*). All attributes except *commission* and *hvalue* take values randomly from a uniform distribution over certain numeric or nominal ranges. *Commission* and *hvalue* are made to depend on other attributes: *commission* is uniformly distributed between $10K$ and $75K$ if *salary* $< 75K$, otherwise *commission* $:= 0$. *Hvalue* is made to lie in different numeric ranges depending on the value of *zipcode*.

In addition to these attribute dependencies, Agrawal et al. [1] defined a set of 10 binary functions of these attributes, of increasing complexity. We used the first four of these for our experiments:

$$
\begin{aligned}
f1 = \mathit{true} \Leftrightarrow\ & (\text{age} < 40) \vee (60 \leq \text{age}) \\
f2 = \mathit{true} \Leftrightarrow\ & ((\text{age} < 40) \wedge ((50K \leq \text{salary} \leq 100K)) \vee \\
& ((40 \leq \text{age} < 60) \wedge (75K \leq \text{salary} \leq 125K)) \vee \\
& ((\text{age} \geq 60) \wedge (25K \leq \text{salary} \leq 75K)) \\
f3 = \mathit{true} \Leftrightarrow\ & ((\text{age} < 40) \wedge (\text{elevel} \in [0..1])) \vee \\
& ((40 \leq \text{age} < 60) \wedge (\text{elevel} \in [1..3])) \vee \\
& ((\text{age} \geq 60) \wedge (\text{elevel} \in [2..4])) \\
f4 = \mathit{true} \Leftrightarrow\ & ((\text{age} < 40) \wedge (\textbf{if}\ \ \text{elevel} \in [0..1] \\
& \quad \textbf{then}\ \ (25K \leq \text{salary} \leq 75K)\ \textbf{else}\ (50K \leq \text{salary} \leq 100K))) \vee \\
& ((40 \leq \text{age} < 60) \wedge (\textbf{if}\ \ \text{elevel} \in [1..3] \\
& \quad \textbf{then}\ \ (50K \leq \text{salary} \leq 100K)\ \textbf{else}\ (75K \leq \text{salary} \leq 125K))) \vee \\
& ((\text{age} \geq 60) \wedge (\textbf{if}\ \ \text{elevel} \in [2..4] \\
& \quad \textbf{then}\ \ (50K \leq \text{salary} \leq 100K)\ \textbf{else}\ (25K \leq \text{salary} \leq 75K)))
\end{aligned}
$$

In all our experiments, we generated a predefined number of tuples, described by the 9 attributes. Then the binary functions were computed and the corresponding labels (*true* or *false*) added to the dataset as additional columns.

Qualitative Results: In a first run, RUDE+Apriori were applied to a database of 10.000 tuples; RUDE generated the following 11 split points:

salary:	**50.5**, **75**, 88.5, **101**, 113.5, **125.5**
commission:	**9.5**
age:	**40**, **59.5**
hvalue:	518.5
hyears:	–
loan:	245.5

It is obvious that RUDE has found all the relevant splits but one. These are printed in bold. Based on this discretization, Apriori succeeded in finding near perfect definitions of functions $f1$ and $f3$ and good approximations of $f2$ and $f4$ (see also Table 1 below). For instance, here are the rules that relate to $f1$:

$\{\text{age} \in [19.5...40)\} \Rightarrow f1 = true$
$\{\text{age} \in [40...59.5)\} \Rightarrow f1 = false$
$\{\text{age} \in [59.5...81.5)\} \Rightarrow f1 = true$

It should be pointed out that RUDE+Apriori also discovered the dependencies between the numeric attributes *commission* and *salary*, and, at least partly, between *hvalue* and *zipcode*. Here are the rules for *commission*:

$\{\text{salary} \in [19.5...50.5)\} \Rightarrow commission \in [9.5...76.5)$
$\{\text{salary} \in [50.5...75)\} \Rightarrow commission \in [9.5...76.5)$
$\{\text{salary} \in [75...88.5)\} \Rightarrow commission \in [0...9.5)$
$\{\text{salary} \in [88.5...101)\} \Rightarrow commission \in [0...9.5)$
$\{\text{salary} \in [101...113.5)\} \Rightarrow commission \in [0...9.5)$
$\{\text{salary} \in [113.5...125.5)\} \Rightarrow commission \in [0...9.5)$
$\{\text{salary} \in [125.5...151.5)\} \Rightarrow commission \in [0...9.5)$

For the more complex dependency between *hvalue* and *zipcode*, RUDE+Apriori found a rather crude approximation that maps a subset of the *zipcodes* to an interval representing low *hvalues*, and another subset to a range of high *hvalues*.

Quantitative Results of Systematic Study: Now we study how imprecision and noise affect the algorithm's ability to extract and represent the hidden dependencies. We vary two factors systematically:

1. *"Fuzziness"*: To model fuzzy boundaries between numeric ranges, the values of numeric attributes are perturbed (see also [1]): if the value of attribute A_i for a tuple t is v and the range of A_i is a, then the value of A_i after perturbation becomes $v + r \times \alpha \times a$, where r is a uniform random variable between -0.5 and $+0.5$, and α is our fuzziness parameter.
2. *Noise*: At a given noise level β, the value of each attribute (numeric or nominal) is replaced by a random value from its domain with probability β.

To get objective quantitative measures, we run RUDE+Apriori on the data, extract all the association rules that involve the attributes in the definitions of the hidden functions (remember our primary interest is in whether the algorithm can find the dependencies), and regard these rules as a *classifier*. Let p be the total number of positive examples (i.e., of class *true*) in the dataset, n the number of negative examples, p_c the number of positive examples covered by the rules, and n_c the number of negative examples erroneously covered. Then we measure the classifier's *coverage* $C = p_c/p$ and the fraction of *false positives* $FP = n_c/n$. This gives a more detailed picture than simply the total error of the classifier.

Table 1 shows the results for three different levels of fuzziness ($\alpha \in \{0, 0.05, 0.1\}$) and noise ($\beta \in \{0, 0.05, 0.1\}$), respectively. In each case, we used a database of size 10.000. To save on experimentation time, APriori was then run on a random sample of 200 tuples from the discretized version of the database.

We see that RUDE is quite effective in finding a useful discretization. Generally, Apriori reaches high levels of coverage and low levels of error in the

Table 1. Results for different fuzziness and noise levels (size of database = 10.000).

	C	**FP**	**C**	**FP**	**C**	**FP**
dependency	**0% fuzziness**		**5% fuzziness**		**10% fuzziness**	
$[age] \rightarrow [f1]$	100.00%	2.77%	98.60%	3.51%	97.09%	8.06%
$[salary, age] \rightarrow [f2]$	92.83%	3.08%	81.06%	3.00%	78.92%	2.42%
$[age, elevel] \rightarrow [f3]$	99.08%	0.00%	90.05%	2.24%	90.46%	2.18%
$[s, a, e] \rightarrow [f4]$	90.80%	3.20%	69.37%	2.56%	61.05%	0.81%
	0% noise		**5% noise**		**10% noise**	
$[age] \rightarrow [f1]$	100.00%	2.77%	96.21%	7.69%	99.30%	8.78%
$[salary, age] \rightarrow [f2]$	92.83%	3.08%	80.00%	3.50%	71.25%	1.83%
$[age, elevel] \rightarrow [f3]$	99.08%	0.00%	89.62%	4.24%	84.68%	14.60%
$[s, a, e] \rightarrow [f4]$	90.80%	3.20%	63.51%	0.79%	51.93%	0.50%

noise-free case, using RUDE's discretization. The more difficult functions seem to be $f2$ and $f4$, because they depend on the correct identification of more numeric split points than $f1$ and $f3$ (7 for $f2$ and $f4$ vs. 2 for $f1$ and $f3$). As expected, increasing fuzziness and noise leads to a decrease in the coverage of the rules that Apriori can find, and to an increase in the errors of commission (false positives FP). Noise has a stronger detrimental effect than fuzziness; this is because in simulating noise, we replaced attribute values by *random* values, as opposed to values in the *vicinity*. The degradation with increasing fuzziness and noise seems graceful, except for function $f4$; but $f4$ is a complex dependency indeed.

We also performed experiments with datasets of varying sizes, i.e. sets with 1.000, 5.000, 10.000 and 100.000 tuples. There it turned out that the number of intervals constructed by RUDE does not grow with the size of the dataset. On the contrary, there is a slight *decrease* — with more data, RUDE tends to generate fewer spurious splits.

To summarize, the experiments indicate that our discretization algorithm is both effective and robust. It does find most of the relevant split points that are needed to discover the dependencies in the data. Of course, it also generates some unnecessary splits. What should be acknowledged is that RUDE constructs a *globally good* discretization; that is, it defines intervals that allow the association rule learner to uncover all or most of the hidden dependencies *simultaneously*. This is in strong contrast to the results described in [10], where each function was dealt with separately, and where both a template for the type of association rule and the correct number of intervals had to be given for the system to be able to rediscover the function.

3.2 Preliminary Results on Real-World Data

We have also started to apply our algorithm to a large, real-world dataset from a long-term research project being carried out at our institute (see [11]). The data

describes expressive performances of pieces of music by a concert pianist. Every single note is documented and described by various music-theoretic attributes; in addition, for each note we have exact numeric values that specify the *tempo*, *loudness*, and *relative duration* (i.e., degree of *staccato* vs. *legato*) with which the note was played. The goal of the project is to discover general principles of *expressive performance*; we want to find rules that explain and predict how a performer will usually shape aspects like tempo, dynamics, and articulation when playing a piece. Note that these factors are inherently continuous.

When applied to a database representing the performed melodies of 13 complete piano sonatas by W.A. Mozart (more than 44.000 notes), RUDE+Apriori discovered a number of interesting relations between different expression dimensions and other factors. For instance, there seems to be a roughly inverse relationship (at least in terms of ranges) between performed relative tempo and loudness (dynamics), as indicated by association rules like the following:

{duration ∈ [0.012...0.34), tempo ∈ [0.26...1.03)} ⇒ dynamics ∈ [0.99...1.41)
(*support:* 10.63%, *confidence:* 72.73%)
{tempo ∈ [1.03...5.27), articulation ∈ [0.69...1.01)} ⇒ dynamics ∈ [0.59...0.99)
(*support:* 7.13%, *confidence:* 70.40%)

In other words (and simplified), when speeding up and playing notes faster than average, the pianist tends to play notes more softly, and vice versa (at least in some cases). Such associations point to very interesting interrelations between different expression dimensions; these and other discoveries are currently the topic of further specialized investigations (in cooperation with musicologists).

4 Related Work

Extending association rule mining to numeric data has been the topic of quite some research recently. Most related to our approach is the method by Srikant & Agrawal [9]. They first perform a pre-discretization of the numeric attributes into a fixed number of intervals (determined by a "partial completeness" parameter) and then combine adjacent intervals as long as their support in the database is less than a user-specified value. The difference to our approach is that attributes are discretized in isolation; no information about other attributes is taken into account. In terms of results, [9] only report on the effect of the parameters on the number of rules found, and on time complexity.

A second type of methods might be called "template-based" [8,10]. These require a template of an association rule (with right-hand and part of left-hand side instantiated and one or more uninstantiated numeric attributes in the left-hand side) to be given, and then find intervals or hyperrectangles for the numeric attributes that maximize support, confidence, or some other interest measure.

A very different approach is taken by Aumann & Lindell [3]. They do not discretize numeric attributes at all. Rather, they use simple measures describing the distribution of continuous attributes, specifically mean and variance,

directly in association rules. That is, their algorithm can discover rules like $sex = female \Rightarrow wage : mean = 7.90/h$. One could easily imagine combining our approach with theirs.

5 Conclusion

This paper has presented RUDE, an algorithm for discretizing numeric or mixed numeric-categorical datasets. The central ideas underlying the algorithm are *mutual structure projection* — using information about the value distributions of other attributes in deciding how many split points to create, and where — and the use of a novel *clustering algorithm* for finding points of significant changes. The algorithm can be viewed as a combination of supervised and unsupervised discretization. Our experimental results show that RUDE constructs discretizations that effectively reflect several associations simultaneously present in the data, and that perturbations in the data do not affect it in an unreasonable way.

The applicability of RUDE is by no means confined to association rule mining. In [6] we evaluate RUDE in a regression-by-classification setting, and it turns out that discretization by mutual structure projection can produce substantial improvements in terms of classifier size (though not in terms of accuracy).

One of the main problems with the current system is that the user-defined parameters still need to be fine-tuned when dealing with a new dataset. Also, some of the parameters represent absolute values; the problem of defining relative threshold measures (like percentages) is also a current research topic.

Acknowledgments

This research is supported by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung* under grants P12645-INF and Y99-INF (START).

References

1. Agrawal, R., Imielinski, T., and Swami, A. (1993). Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering* 5(6) (Special issue on Learning and Discovery in Knowledge-Based Databases), 914-925.
2. Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules. In *Proc. of the 20th Int.l Conference on Very Large Databases*, Santiago, Chile.
3. Aumann, Y. and Lindell, Y. (1999). A Statistical Theory for Quantitative Association Rules. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*. Menlo Park, CA: AAAI Press.
4. Dillon, W., and Goldstein, M. (1984). *Multivariate Analysis*. New York: Wiley.
5. Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and Unsupervised Discretization of Continuous Features. In *Proceedings of the 12th International Conference on Machine Learning (ML95)*. San Francisco, CA: Morgan Kaufmann.
6. Ludl, M.-C. and Widmer, G. (2000). Relative Unsupervised Discretization for Regression Problems. In *Proceedings of the 11th European Conference on Machine Learning (ECML'2000)*. Berlin: Springer Verlag.

7. Pavlidis T. (1982). *Algorithms for Graphics and Image Processing*. Rockville, MD: Computer Science Press.
8. Rastogi, R. and Shim, K. (1999). Mining Optimized Support Rules for Numeric Attributes. In *Proc. of the International Conference on Data Engineering* 1999.
9. Srikant, R. and Agrawal, R. (1996). Mining Quantitative Association Rules in Large Relational Tables. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, Montreal.
10. Wang, K., Tay, S., and Liu, B. (1998). Interestingness-Based Interval Merger for Numeric Association Rules. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*. Menlo Park: AAAI Press.
11. Widmer, G. (1998). Applications of Machine Learning to Music Research: Empirical Investigations into the Phenomenon of Musical Expression. In R.S.Michalski, I.Bratko and M.Kubat (eds.), *Machine Learning and Data Mining: Methods and Applications*. Chichester, UK: Wiley.