Mining Numeric Association Rules with Genetic Algorithms

Mata, J.*, Alvarez, J.L.†, Riquelme, J.C.‡

*Dept. Ing. Electronica, Sist. Informaticos y Autom. Universidad de Huelva. Spain. email:mata@uhu.es
†Dept. Ing. Electronica, Sist. Informaticos y Autom. Universidad de Huelva. Spain. email:alvarez@uhu.es
‡Dept. Lenguajes y Sistemas Informaticos. Universidad de Sevilla. Spain. email:riquelme@lsi.us.es

Abstract

In this last decade, association rules are being, inside Data Mining techniques, one of the most used tools to find relationships among attributes of a database. Numerous scopes have found in these techniques an important source of qualitative information that can be analyzed by experts in order to improve some aspects in their environment.

Nowadays, there are different efficient algorithms to find these rules, but most of them are demanding of databases containing only discrete attributes. In this paper we present a tool, GENAR (GENetic Association Rules), that discover association rules in databases containing quantitative attributes. We use an evolutionary algorithm in order to find the different intervals. We also make use of the evolutionary methodology of iterative rule learning to not evolve always to the same rule. By means of this we get to discover the different association rules. In our approach we present a tool that obtain association rules with an undetermined number of numeric attributes in the antecedent of the rule.

1 Introduction

Association rules were introduced in [1] as a method to find relationships among attributes in a database. These techniques allow us to obtain a very interesting qualitative information with which we can take later decisions. In general terms, an association rule is a relationship between attributes in the way $C_1 \Rightarrow C_2$, where C_1 and C_2 are pair conjunctions (attribute, value) in the way $A_1 = v_1$ if it is a discrete attribute, or $A_1 \epsilon [x_1, y_1]$ if it is a continuous or numeric attribute.

Obviously, many associations of this kind can be found in a database but only the most interesting ones will be used for their later study. To define this notion of interesting two fundamental concepts were introduced [1]: **Support.** We will say that a rule $C_1 \Rightarrow C_2$ has a support value s, if a $s\%$ of the records contain C_1 and C_2. **Confidence.** We will say that a rule $C_1 \Rightarrow C_2$ has a confidence value c, if a $c\%$ of the records that contain C_1 also contain C_2.

For example, we suppose that we have a database with different measures taken from a river, from which we store among other information, temperature, pH, salinity, chlorophyll level, etc. The rule: $T \epsilon [12.1, 15.4] \wedge pH \epsilon [7.84, 7.96] \Rightarrow chlorophyll \epsilon [11.44, 16.41]$ has a *support* of 0.7 if the 70% of the measures realised have a temperature between 12.1 and 15.4, a pH between 7.84 and 7.96, and a chlorophyll level between 11.44 and 16.41. On the other hand, the rule has a *confidence* of 0.85 if the 85% of records where the temperature is between 12.1 and 15.4 and pH between 7.84 and 7.96, then the chlorophyll level is between 11.44 and 16.41.

The problem to solve consists in finding all the association rules that overcome some levels or minimum thresholds of support and confidence, defined by the user, called generally *minsup* and *minconf*. The usual algorithms of association rules work in two phases: firstly they find pair sets attribute-value which overcome *minsup* minimum threshold, secondly, departing from these sets, they discover the rules that overcome *minconf* minimum threshold. There are different algorithms to discover these association rules, that can be found in [2, 9].

The process to find frequent itemsets consists, basically, in building different combinations of attribute-value pairs and verifying that they are produced in a determined number of records. This process can be relatively easy and efficient when attributes are discrete and their domains span few values. For this reason the tools proposed in the previously referred articles, work with databases in which the domains of their attributes are formed by a finite set of values. But in the real world there are numerous databases where the information is numeric. In these databases, attributes have thousands of possibilities of taking one value, by this reason the process described above is unthinkable

from a computational point of view.

There are some studies in which tools that handle with attributes with continuous domains are presented. In [6] the author propose to divide the quantitative attributes into a fixed number of intervals of the same size and to discover the rules departing from such intervals. One of the main problems is that rules are only discovered departing from such intervals. In [9] they go a step further allowing the union of consecutive intervals. In order that intervals do not cover all the domain of the attribute, the authors propose a new measure, *partial completeness*, that the intervals must not overcome. In [3], the concept of optimized association rule was introduced. Rastogi and Shim followed this line in [7] and [8], in which they permit rules that contain disjunctions over uninstantiated numeric attributes. The optimized association rules have the form: $U \wedge C_1 \Rightarrow C_2$, where U is a conjunction of one or two conditions over numeric attributes, and C_1 and C_2 are instantiated conditions.

Our contribution lies in the fact that these rules can have an undetermined amount of numeric attributes in the antecedent and a unique numeric attribute in the consequent. In this paper we present a technique to find association rules in numeric databases by using evolutionary algorithms (EA) [4].

2 Preliminaries

The tool developed in this approach is based on EA theory. In order to find the optimal rule, we depart from a population where the individuals are potential association rules. These individuals will be evolving by means of crossover and mutation operators, so that, the individual with the best fitness will correspond to the most significant rule in the last generation.

One of the problems we find when using EA theory is that, during the process, all the individuals tend to the same solution. In our study case, this means that all individuals evolve towards the same association rule, so that, rules composing the population of the last generation, provide, in practice, the same information. There are different techniques to solve this problem. Among them, the use of evolutionary algorithm with niches and the iterative rule learning [5]. In this tool we use iterative rule learning to find different association rules inside the database. The process consists in executing the genetic algorithm as many times as rules we want to obtain. In each iteration, we will mark the records covered by the obtained rule. This parameter affects the *fitness* function of the following generations, so that the algorithm will not search for rules that have been previously considered.

3 Practical Implementation

In the following subsections we will describe the general structure of the algorithm, the individuals representation and the meaning of the operators.

3.1 GENAR algorithm

In order to decide the completeness of the intervals that conform the rules, the algorithm only needs to know minimum and maximum values of each attributes domain. This value is needed for intervals not to grow up until spanning the total domain.
Definition 1. We will define *amplitude* as the maximum size the interval of a determined attribute can get. We will obtain this value by 1.

$$amplitude(i) = \frac{M_i - m_i}{k} \qquad (1)$$

Where M_i and m_i are maximum and minimum values of the domain of attribute i, and k is a value definable by the user, which we will call *AF (amplitude factor)*.

```
algorithm_GENAR
1.nRules = 0;
2.while ((nRules < NRULES) or
        (all records covered)) do
3.  nGen = 0;
4.  generate first population P(nGen);
5.  while (nGen < NGENERATIONS) do
6.    process P(nGen);
7.    P(nGen+1) = select individuals of P(nGen);
8.    complete population P(nGen+1) by crossover;
9.    make mutations in P(nGen+1);
10.   nGen++;
11. end_while
12. Rules[nRules] = choose the best of P(nGen);
13. penalize tuples covered by Rules[nRules];
14. nRules++;
15.end_while
end
```

As we can see, the algorithm is repeated until all rules (NRULES) have been obtained. This value is defined by the user depending on the number of rules he wants to obtain in the process. In step 4 the first rule population is generated. Intervals that overcome the *amplitude* defined in 1 are not allowed. The genetic algorithm is located among steps 5 to 11. In step 6, *process* carries out several functions: to calculate the *support*, *confidence* and *fitness* of each individual. In step 7, a percentage of individuals with the best fitness, according to 2 is selected. In 8 crossovers between selected individuals are made in order to complete the population.

Finally, in 9, mutations in individuals are carried out depending on a mutation factor.

In step 12 the best individual is chosen from the population formed in the last generation. This election will depend on the three factors pointed out previously: *support*, *confidence* and *fitness*. This individual is one of the rules that returns the algorithm. The operation made in step 13 is very important. In it records covered by the rule obtained in the previous step are penalized. Due to the fact that this factor is part of the adaptation function of the EA, we achieve that the next population does not repeat its search space, that is, it does not tend to generate the same association rule. To penalize the records we use a value named *PF (penalization factor)*, that will be defined by the user. This parameter takes its values from interval (0,1) and we use it to decrease the fitness of those individuals that are going to cover a record that has already been marked.

3.2 Genetic Algorithm characteristics

GENAR algorithm uses as a search motor an EA with real codification for the individuals. During the evolutionary process a 15% of the individuals pass to the following generation according to the selection operator which will choose those with the best fitness. The rest of individuals to complete the population will be formed by the crossover operator. Besides, the individuals will be affected by a mutation operator depending on a mutation probability. At the end of the evolutionary process, that is, when all fixed generations have been completed, the best individual, depending on its *support*, *confidence* and *fitness* will be the association rule found.

3.3 Structure of individuals

In GENAR algorithm, each individual represents an association rule in which maximum and minimum values of the intervals of each numeric attribute are stored. The last interval is the rules consequent, while the rest of them conform the rules antecedent. Besides, certain additional information for each individual is stored, such as the number of attributes the rule has, *support*, *confidence* and *fitness*. In this paper we consider only those rules which involve all database attributes except the last one which acts as consequent.

In figure 1 we can see a graphic representation.

This individual represents the rule:

$A_1 \in [x_1, y_1] \wedge \ldots \wedge An_{n-1} \in [x_{n-1}, y_{n-1}] \Rightarrow S \in [x_n, y_n]$

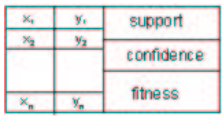

Fig. 1. Structure of an individual

3.4 Evaluation Function

In order that in the different iterations the individuals tend to other search spaces, we have included a penalization factor in the evaluation function. By means of this we also achieve that association rules which our tool return form a non-hierarchical set, since we do not eliminate those cases covered by a rule, but we mark them in order that the same rule will not cover them again.

Evaluation function used appears in:

$$fitness = covered - (marked * PF) \qquad (2)$$

Where *covered* is the number of records which fit the rule and *marked* is a binary value which indicates if the record has already been covered by some previous rule.

3.5 Genetic Operators

By means of selection operation the best individuals are chosen, that is, those with the best *fitness*, which are the ones that will go to the following generation. From each crossover between two individuals, two new ones appear whose intervals will be the first intervals of the first individual to cross and the following intervals of the second individual to cross and vice versa. The mutation operator consists in altering one of the intervals of the rule. For each bound of the chosen interval we can have two possibilities, to increase or to decrease its value. In this way we achieve four possible mutations: to shift the complete interval to the left or to the right, and to increase or to decrease its size. We have to be specially careful in not overcoming the fixed value of *amplitude*.

4 Results

To verify that the developed algorithm finds correctly numeric association rules, we have created two artificial databases in which certain rules, previously fixed, are fulfilled in an adequate number of records as to consider them interesting ones. In the first exemplary database, rules have no overlapping, that is, there are not any cases that can belong to two rules. Nevertheless, in order to fix results we have created a second database where there is overlapping among association rules. Due to the fact that

the performance of the tool is based in a EA, we have carried out five times the proofs in both examples and the results fit in with the average values of such proofs.

4.1 Rules without Overlapping

The first example database is formed by four numeric attributes. We have generated 1000 records distributed in 5 association rules. The rules will be formed then by a conjunction of the three first attributes in the antecedent and the fourth one in the consequent. The rules we pretend to find are the following ones:

1. [1,15] Λ [7,35] Λ [60,75] then [0,25]
2. [5,30] Λ [25,40] Λ [10,30] then [25,50]
3. [45,60] Λ [55,85] Λ [20,35] then [25,75]
4. [75,100] Λ [0,20] Λ [40,60] then [75,100]
5. [10,30] Λ [0,30] Λ [75,100] then [100,125]

rule	sup(%)	conf(%)	#r
1 [1,26] [6,35] [54,82] [0,31]	19.1	80.19	191
2 [1,32] [21,42] [10,36] [18,55]	19.9	100	199
3 [37,66] [56,84] [19,44] [23,64]	15.2	81.72	152
4 [70,99] [0,26] [34,61] [69,106]	19.1	100	191
5 [2,32] [1,30] [71,99] [92,124]	18.3	77.99	183

Table 1. Obtained results without overlapping

The exact support of each of the rules artificially defined is 20%, since each of them cover 200 records. In table 1 we can see that the support of the obtained rules is very close to such value. Moreover, the values of the confidence are also close to 100%. In this case the number of covered records (#r) coincide with the support, since the rules have no overlapping.

4.2 Rules with Overlapping

The second example database is formed by three numeric attributes. We have generated 600 records distributed in three association rules. These rules will be formed by a conjunction of the two first attributes in the antecedent and the third one in the consequent. The rules we pretend to find are the following ones:

1. [18,33] Λ [40,57] then [35,47]
2. [1,15] Λ [7,30] then [0,20]
3. [10,25] Λ [20,40] then [15,35]

In this case, the exact support of each of the rules artificially defined is 33,3%, since each of them cover 200 records. Again the obtained rules have a support very close to the expected value. Moreover, good values of confidence are obtained. In spite that the rules are with overlapping, they cover almost all the records assigned to every one of them.

rule	sup(%)	conf(%)	#r
1 [17,32] [35,56] [32,46]	28.5	89.1	166
2 [1,16] [7,30] [0,21]	32.33	82.91	195
3 [10,25] [17,38] [13,35]	31.55	86.25	180

Table 2. Obtained results with overlapping

References

[1] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases", *Proc. of the ACM SIGMOD Conference on Management of Data*, pp. 207-216, Washington, D.C., 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", *Proc. of the VLDB Conference*, pp. 487-489, Santiago (Chile), 1994.

[3] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, "Mining Optimized Association Rules for Numeric Attributes", *Proc. of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Databases Systems*, 1996.

[4] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", *Addison-Wesley*, New York.

[5] A. Gonzlez and F. Herrera, "Multi-stage Genetic Fuzzy System Based on the Iterative Rule Learning Approach", *Mathware & Soft Computing*, 4, 233-249.

[6] G. Piatestsky-Shapiro, "Discovery, Analysis and Presentation of Strong Rules", *Knowledge Discovery in Databases*, AAAI/MIT Press, 1991.

[7] R. Rastogi and K. Shim "Mining Optimized Association Rules for Categorical and Numeric Attributes", *Int'l Conference on Data Engineering*, Orlando, 1998.

[8] R. Rastogi and K. Shim "Mining Optimized Support Rules for Numeric Attributes", *Int'l Conference on Data Engineering*, Sydney, Australia, 1999.

[9] R. Srikant and R. Agrawal "Mining Quantitative Association Rules in Large Relational Tables", *Proc. of the ACM SIGMOD Conference on Management of Data*, 1996.