An Efficient Data Structure for Decision Rules Discovery*

Raúl Giráldez
Computer Science Dpmt.
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
giraldez@lsi.us.es

Jesús S. Aguilar–Ruiz
Computer Science Dpmt.
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
aguilar@lsi.us.es

José C. Riquelme
Computer Science Dpmt.
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
riquelme@lsi.us.es

ABSTRACT

The increasing amount of information available is encouraging the search for efficient techniques to improve the data mining methods, especially those which consume great computational resources. We present a novel structure, called EES, which helps the data mining algorithms which generate decision rules to reduce the aforementioned cost. Given that decision rules establish conditions for database attributes, EES stores the information in such a way that the search can be carried out by attributes instead of by examples. EES could be useful for any method which generates decision rules. Moreover, it is of particular interest when the search for the solution involves a great many hypothetical solutions. Thus, this structure is designed for speeding up the rule-evaluation process in methods based on Evolutionary Algorithms. The traditional structure, based on vectors of examples (in which the database is stored) is evaluated and compared with EES, including the costs for a stratified set of cases. Finally, the experimental results demonstrate the quality of our proposal, reducing the computational cost by approximately 50%.

Keywords

Data Mining, Decision Rules, Preprocessing

1. INTRODUCTION

Within the context of supervised learning there exist in the literature a large number of methods and algorithms that extract inherent knowledge from a labelled database and build a model which represents the extracted information. A great number of these methods (CN2 [3], RISE [5], OC1 [14], GABIL [4], GIL[12], etc) use probabilistic algorithms to look for solutions in space, with a high computational cost, principally due to the repetitive evaluation of the candidate solutions. This work focus on those systems that define a probabilistic heuristic for the generation of decision rules, especially on those methods that use an Evolutionary Algorithm (henceforth EA) in order to carry out such aim. Thus, we propose a data structure which reduces the computational cost of evaluation in the learning process.

The aforementioned methods usually evaluate the rules directly from the database. That is to explore such database sequentially, taking each of the examples and testing the quality of the rule through the correct classification of those examples. We can see, therefore, that the learning process of these systems is very costly in terms of time and space. Some authors [17, 19] have concentrated their efforts on improving the learning process by speeding up the algorithm, in order to reduce its computational cost. Others have approached the problem from the perspective of scalability [20]. However, the appropriate organisation of the information could also contribute to the reduction of computing time. This aspect, no less important than the previous one, has perhaps been more neglected.

There are in the literature numerous proposals on data structures and organisation of the information which essentially speed up the search for information in multidimensional spaces, such as those known as Multidimensional Access Methods (MAM)[7]: Point Access Methods (Grid File [15], KDB-tree [18], LSD-tree [9] , BV-tree [6], etc) and Spatial Access Methods (K-D-Tree [2], R-tree [8], P-tree [11], SDK-tree [16], etc). However, given the peculiarity of the problem we are facing, MAM do not, in themselves, provide a solution to such problem, since they index a dataset with the goal to speed up the queries on such data. However, we want to reduce the cost of evaluation of decision rules. To do it, we distributes the examples according to the values that they take for each attribute. Thus, we know exactly what examples fulfill these rules. For instance, AD-Tree [13] is the MAM that gives the nearest solution to the problem we want to solve. Nevertheless, if we would want to evaluate a decision rules using this structure, we would have to build the no sparse AD-Tree, that is to store all possible queries in addition to the indexes of the examples that each query includes. This means very high computational cost, since too much redundant information is stored.

In this work we are presenting a data structure called EES (Efficient Evaluation Structure) which is designed specifically for accelerating the evaluation process of rules during the application of data mining algorithms based on EAs. The EES structure organises the information from a database in such a way that it is not necessary to process all the ex-

*This research was supported by the Spanish Research Agency CICYT under grant TIC2001-1143-C03-02.

SAC 2003 Melbourne, Florida, USA

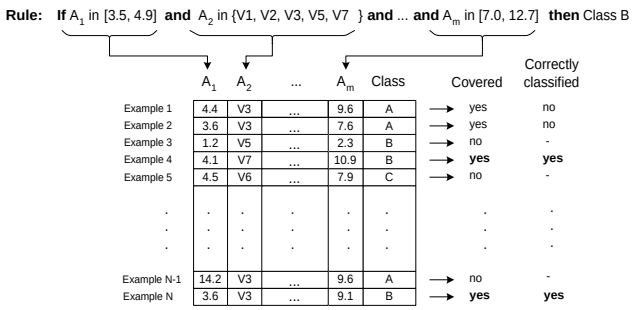


Figure 1: Evaluation using a vector of examples.

amples in order to evaluate decision rules generated by a supervised learning system.

An EA codifies decision rules as individuals in the genetic population. Once the population is constructed the individuals are evaluated and depending on the quality of each one of them, the crossover and mutation operators are applied, thus constructing the next generation. It is simple to see that those systems that apply EAs need to carry out constant evaluations throughout the learning process, since they have to evaluate each one of the individuals of each one of the populations that they generate. For example, an EA that carries out 300 generations, each containing 100 individuals, needs to carry out at least 30,000 evaluations. If the database contains 1000 examples the process will be carried out 30 million times. In the particular case of EAs applied to the generation of rules, the database is stored in a vector of examples (see Figure 1), which is used for evaluating of every individual. Evaluation by means of a linear search processes each and every one of the examples in the database independently of the conditions established by the rule. As can be seen in Figure 1, not all the covered examples are correctly classified (only Examples 4 and N).

Therefore the computational cost of a single evaluation is O(NM), where N is the number of examples and M is the number of attributes in the database. The rule-learning methods that EAs use invest approximately 85% of their time in evaluating the individuals (the mean of the executions of a 10-fold cross-validation with 20 UCI Repository databases [1]). This is the reason why we propose the EES structure in order to organise the information in such a way that only the necessary examples from the database will be dealt with, and not all data.

2. DESCRIPTION

EES distributes the information from the database in such a way it is possible to carry out a search in the space by attribute instead of by example. The data structure must be capable of storing this information independent of the type of attribute (continuous or discrete). In the case of continuous attributes it is convenient to apply a method to transform them into discrete which reduces the cardinality of the set of values that this type of attributes can eventually take. In certain cases the methods of rule-generation apply a method of discretization as pre-processing of the data in order to calculate such intervals. EES must be constructed using the same sets of intervals obtained by the discretization method applied by the rule-generation method. This does not imply any kind of limitation as regards the data structure, since the latter is totally flexible as opposed to the discretization used by the continuous attributes. The only restriction demanded of the discretization method is that the generated intervals be disjoint, given that if this were not the case there could exist in the database a value pertaining to various intervals. In any case we will be able to maintain infinite rank, that is to say, we will not need to apply any discretization method. The EES structure would still be valid: in this case the number of nodes would increase and the size of the lists associated with the nodes would be reduced even to length equal to one.

In general, for every attribute A_i in the database we will denote the finite set of values that A_i can take by Ω_i. In the case of A_i being a discrete attribute Ω_i will contain values which we will represent as V_{ij} $(1 \leq j \leq |\Omega_i|)$. On the other hand, if we are dealing with a continuous attribute, Ω_i will contain intervals which we will call I_{ij} $(1 \leq j \leq |\Omega_i|)$, the lower and upper bounds of which we will denote by l_{ij} and u_{ij} respectively. In this manner EES will store the information of the continuous attributes in a similar way to that which it does for the discrete attributes: the lower and upper limits of each interval will be saved in place of the unique values that are stored for the discrete attributes.

EES arranges the information from the database in a vector of binary and balanced search-trees in such a way that the i^{th} element of the vector will contain information about the i^{th} attribute (A_i) in the database. Specifically, the different values or intervals that A_i can take are stored in the tree, which we will denote by T_i. In addition to V_{ij} or I_{ij} , each node N_{ij} of the tree T_i contains a list (L_{ij}) of numbers which indicate the positions of examples in the database. If A_i is discrete, the indexes contained in the list L_{ij} will correspond to those examples, the i^{th} attributes of which take the j^{th} value (V_{ij}) within the Ω_i set of possible values of such attribute. If A_i is continuous, the indexes contained in L_{ij} will correspond to those examples, the value of which for the i^{th} attribute is included in the j^{th} interval (I_{ij}) within the Ω_i set of possible intervals of such attribute. Figure 2 shows a example of the data structure for a database with 15 examples and 2 attributes, where the 1^{st} attribute is continuous while the 2^{nd} is discrete. It is important to note that in the case of continuous attributes the tree is sorted (inorder) by (disjoint) intervals and in the case of discrete attributes it is arranged alphabetically by the discrete value. In this manner, any search within the tree has a logarithmic cost.

2.1 Construction of the EES

Let us start from the information contained in a labelled database with N examples indexed from one to N, each of them with M attributes of any type (see Figure 2). In these M attributes we are not including the class. Previously a discretization method similar to the method 1R described by Holte in [10] has been applied, although any other supervised discretization method could have been used. This method obtains a set of disjoint intervals for each one of the continuous attributes in the database. For each attribute (A_i) a balanced search-tree is created and inserted in the structure in the corresponding position (i). The nodes of these trees contain the values or intervals according respectively to whether the attribute is discrete or continuous. Once all the trees have been created and inserted, the lists for each node

N	A_1:Size	A_2:Colour	Class
1	2.0	blue	A
2	1.0	green	A
3	1.2	green	A
4	1.4	blue	A
5	1.0	red	A
6	1.4	red	A
7	1.7	blue	B
8	1.8	green	B
9	1.2	red	B
10	2.1	blue	A
11	2.2	green	A
12	1.0	blue	A
13	1.2	blue	B
14	2.0	green	A
15	1.6	blue	B

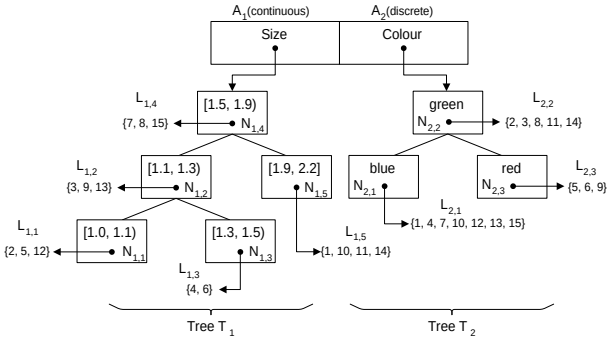


Figure 2: Example.

are completed. A linear search of the database is carried out, passing through all the examples. For each attribute A_i of a specific example, the node corresponding to the value of such attribute will be searched for in the tree T_i. Once the node has been located, the index of the example being processed is inserted into the list of such node and the next attribute of this example is processed. When all the attributes of an example have been treated we say that the example has been inserted into the structure, and we go on to process the next example. At the point at which all the examples have been inserted, the structure contains the same information as the database, excepting the class of each example. However, this information is directly accessible during the use of the structure, since the latter stores the indexes of examples. The computational cost of the construction process of the data structure is $O(NMlog_2|\overline{\Omega}|)$, where $\overline{\Omega}$ is the mean number of tree-nodes.

2.2 Use of the EES

The fundamental property that the data structure offers is the possibility of accessing the information from the database through attributes instead of through examples. The main aim that we are looking for in the use of the data structure is not having to process those examples, the values of which are not covered by the rule which is being evaluated. If we take a node N_{ij} of each T_i, the intersection of the lists L_{ij} will be the set of the indexes of examples which fulfill that each one of its attributes A_i take values which are covered by each corresponding node N_{ij}. The advantage that the EES offers lies in the fact that the intersections are carried out in an incremental manner - that is to say, firstly the intersection of the list for the attribute A_1 and the list for the attribute A_2 is carried out. If such intersection is not empty, the list for the attribute A_3 is searched for and a new intersection between this and the result of the previous intersection is created. This process is repeated until all the attributes are completed, or until one of the intersections remains empty. If the process concludes, the resulting list will contain the indexes of the examples which fulfill the values or intervals of all the selected nodes N_{ij}. If these nodes N_{ij} are searched for according to the conditions established by a decision rule, we will be evaluating such rule. The pseudocode of the evaluation algorithm for decision rules using the EES is shown in Figure 3.

The notation used in the algorithm in Figure 3 is the same

```
Function EVALUATE
   Inputs:  R: Decision Rule; E: Data Structure
   Output: L: List of indexes (examples covered by R)
 begin
   i := 1
   L_i := ListUnion(R, E, i)
   while i < NumberOfAttributes(E) ∧ L_i ≠ ∅
      i := i + 1
      L_i := L_{i-1} ⋂ ListUnion(R, E, i)
   end while
   L := L_i
end EVALUATE

Function ListUnion
   Inputs:  R: Decision Rule; E: Data Structure
            k: Integer (attribute location)
   Output: Lu: List of indexes(examples covered by R_k)
begin
   Lu := ∅; T_k := E[k];
   if A_k is Continuous
      for every I_kj ∈ T_k | I_kj ⊆ R_k
         Lu := Lu ⋃ L_kj
      end for
   else /* A_k is Discrete */
      for every V_kj ∈ T_k | V_kj ∈ R_k
         Lu := Lu ⋃ L_kj
      end for
   end if
end ListUnion
```

Figure 3: Algorithm for the use of EES.

as that used in the general structure of Figure 2, excepting the following symbols: E represents the data structure and R is the rule to be evaluated, while R_i is the condition that R establishes for the attribute A_i. The symbol L_i (with a single subindex) represents the list of accumulated intersections until the i^{th} iteration, that is to say, until the i^{th} attribute. The algorithm is divided in two functions. The first, *EVALUATE*, is the main function and has as its input parameters the decision rule R and the data structure E. The rule R has the structure that is shown in Figure 1, the condition that R establishes for the k^{th} attribute in the database being represented by R_k. The function *EVALUATE* offers a single output parameter (L), which is the list of indexes of examples resulting from the evaluation of the rule R on the data structure E. In this way, the function *EVALUATE* searches the rule R and the structure E simultaneously, calculating the intersections of the lists and returning the final list (L). The main function uses an auxiliary function named *ListUnion*, that has the rule R, the structure E and a inte-

ger which indicates the attribute which it has to deal with. The only output parameter of this function is the list of indexes (Lu) corresponding to the union of the lists L_{kj} for the attribute A_k (with $1 \leq j \leq |\Omega_k|$). This union of lists is necessary owing to the fact that the condition of the rule R for the attribute A_k can include various nodes of the tree T_k. In the case of continuous attributes the first node included in the condition that the rule establishes is searched for in the tree. Once located, a inorder-walk of the tree starting from such node and stopping when it finds the first node which is not covered by the condition is carried out. In the case of discrete, it is necessary to search for all the values that the rule establishes for such attribute in the corresponding tree. When dealing with both cases of balanced search trees, all the searches carried out in them have a logarithmic cost.

3. EXPERIMENTS

The experiments carried out to test the efficiency of the structure consist of evaluating various sets of decision rules for 15 different databases from the UCI Repository, as opposed to the linear search habitually used for the evaluation of decision rules. For each database groups of rules are generated in an aleatory manner, through use of a method which assures the uniform distribution of these rules. Subsequently, such rules are evaluated using the linear method and the data structure EES and the results obtained are compared.

The linear method of evaluation used in the tests is the most efficient one possible. For each rule this method searches the database, which has previously been stored in an vector of examples, processing each and every one of the examples. Likewise, for each example, the verification that its attributes fulfill the conditions of the rule is also carried out in a linear manner. However, it is not always necessary to process all the attributes of every example. If, during the processing of an example, one of its values does not fulfill the condition that the rule establishes for the corresponding attribute, that example is no longer processed as it will no longer be able to fulfill the rule, independent of the values that the rest of the attributes take, and the next example will be processed. Given that could think that the major advantage of the structure lies in the evaluation of rules which do not cover examples, in order to ensure that the test-method is fair two types of rules have been generated and evaluated: valid and invalid (see the results varying the percentage of valid rules in Figure 4). Let us call a rule valid if it covers at least one example in the database. In comparison, a rule which does not cover any of the examples in the database shall be called invalid. According to these definitions a valid rule will ensure that the structure is completely searched, while in the case of an invalid rule the evaluation process of the data structure will be halted before the search of the latter has been completed. Although, a priori, the evaluation of invalid rules seems to be pointless, this is not so, since the learning methods based on EA generate intermediate rules which in many cases do not cover any examples. It is worth noting to take this into account, seeing that EES is focus on this kind of methods.

Table 1 shows the following: for each database (first column) the average time used by the EES structure (second column) and that used by the linear method (third column), as well as the improvement obtained by EES as compared with evaluation using the vector (final column). For each database the improvement is given by Equation 1, which represents the percentage of time saved by using the EES structure with respect to the time used by the vector in the average-case.

Table 1: Comparison of average results

BD	A.T. ESS	A.T. Vector	Improv.(%)
bupa liver disorder	1.464	4.370	66.5
breast cancer (Wisc.)	5.572	13.421	58.5
cars	2.173	4.870	55.4
cleveland	3.460	6.042	42.7
glass	1.634	2.884	43.3
hayes-roth	0.720	1.499	52.0
heart desease	3.018	6.228	51.5
iris	0.517	1.536	66.3
led7	20.689	34.905	40.7
letter	222.556	842.784	73.6
pima indian	4.283	11.309	62.1
soybean-	1.661	2.808	40.8
tic-tac-toe	7.623	11.031	30.9
vehicle	8.937	18.785	52.4
wine	2.084	4.120	49.4

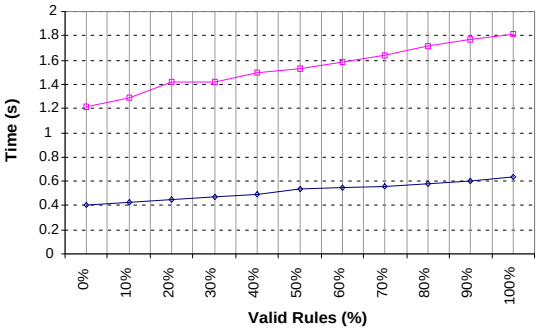
$$Improvement = 100 \times \frac{Time(Vector) - Time(EES)}{Time(Vector)} \quad (1)$$

As can be observed in Table 1, for all the databases the average time taken by the EES is noticeably less than that taken by the vector. This ensures that the improvement will always be positive, that is to say, that the EES always improves on the linear structure. If we calculate the mean improvement for the 15 databases, we obtain the result that the average improvement is 52.4%, which is to say that the structure takes practically half as much time in evaluating the rules.

Some of experimental results are graphically shown in Figure 4. This Figure contains 2 graphs (*Iris* and *Pima Diabetes*) which represent the evaluation time in seconds against the percentage of valid rules in the sets of rules evaluated. In turn, each graph contains two curves: the grey line shows the temporal results obtained for the evaluation using the vector of examples, while the black line refers to the results for the evaluation using the EES structure. Both representations show the temporal variation as the percentage of valid rules is increased. As graphs show, for these databases used the behaviour of the EES structure is very favourable in comparison with the vector of examples. The first result that stands out is that for all the databases used, the evaluation time taken by the EES structure is inferior to the time spent by the linear method. The results show that EES is highly efficient independently of the type of rule (valid or invalid), which give an idea of the robustness of the structure.

4. CONCLUSIONS

In this work we are presenting the data structure called EES, the goal of which is to organise the information from a database in such a manner that the efficiency of the methods of generating decision rules based on Evolutionary Algorithms is improved. ESS allows us to process only those examples, the values of which are covered by such rule will be processed, and not the totality of the database. In this



(a) Iris

(b) Pima Diabetes

Figure 4: Results for Iris and Pima Diabetes.

way, the evaluation process of examples is incremental, that is to say, it starts from a number of covered examples that is reduced as we analyse more attributes of the structure. When the number of covered examples is reduced to 0, the rule ceases to be of interest, and therefore is no longer evaluated. Thus EES calculates what examples are within a rule, if any; and not check if each example satisfies all the conditions of a rule, as the vector of examples would do. So this structure is not a method for indexing examples. Finally, the experimental results show the quality of our propose.

5. ADDITIONAL AUTHORS

Daniel Mateos, Computer Science Department, University of Seville, mateos@lsi.us.es

6. REFERENCES

[1] J. S. Aguilar. *Discovering Hierarchical Decision Rules with Evolutionary Algorithms in Supervised Learning.* PhD thesis, University de Seville, 2001.

[2] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, 1979.

[3] P. Clark and R. Boswell. Rule induction with cn2: Some recents improvements. *Machine Learning: Proceedings of EWSL-91*, pp. 51–163, 1991.

[4] K. A. DeJong, W. M. Spears, and D. F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 1(13):161–188, 1993.

[5] P. Domingos. Rule induction and instance-based learning: A unified approach. In *Proceedings of International Joint Conference on Artificial Intelligence*, 1995.

[6] M. Freeston. A general solution of the n-dimensional b-tree problem. In M. J. Carey and D. A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995*, pages 80–91. ACM Press, 1995.

[7] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[8] A. Guttman. R–Trees : A dynamic index structure for spatial searching. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages pp. 47–57, 1984.

[9] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and nonpoint objects. In P. M. G. Apers and G. Wiederhold, editors, *Proceedings of the Fifteenth International Conference on Very Large Data Bases, August 22-25, 1989, Amsterdam, The Netherlands*, pages 45–53. Morgan Kaufmann, 1989.

[10] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11:63–91, 1993.

[11] H. V. Jagadish. Spatial search with polyhedra. In *Proceedings of the Sixth International Conference on Data Engineering, February 5-9, 1990, Los Angeles, California, USA*, pages 311–319. IEEE Computer Society, 1990.

[12] C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 1(13):169–228, 1993.

[13] Andrew W. Moore and Mary S. Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. In *Journal of Artificial Intelligence Research*, vol. 8, pages 67-91, 1998.

[14] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 1994.

[15] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable symmetric multikey file structure. *ACM Transactions on Database Systems, ACM CR 8411-0931*, 9(1), 1984.

[16] B. C. Ooi. Spatial KD–Tree: A data structure for geographic database. In *BWT*, pages 247–258, 1987.

[17] R. Quinlan. See5.0 (http://www.rulequest.com), 1998-2001.

[18] J. T. Robinson. The K–D–B–Tree: A search structure for large multidimensional dynamic indexes. In Y. E. Lien, editor, *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, Michigan, April 29 - May 1, 1981*, pages 10–18. ACM Press, 1981.

[19] S. Ruggieri. Efficient C4.5. Technical Report TR-00-01, 2, 2000.

[20] K. Shim. SIGKDD Explorations. December 2000. Volume 2, Issue 2.