A Tool to Obtain a Hierarchical Qualitative Rules from Quantitative Data

Jesús Aguilar, José Riquelme y Miguel Toro.

Departamento de Lenguajes y Sistemas Informáticos.
Facultad de Informática y Estadística. Universidad de Sevilla.
Avda. Reina Mercedes s/n. 41012 Sevilla.
e-mail: {aguilar, riquelme, mtoro}@lsi.us.es

Abstract. A tool to obtain a classifier system from labelled databases is presented. The result is a hierarchical set of rules to divide the space in n-orthohedrons. This hierarchy means that obtained rules must be applied in specific order, that is, an example will be classify by i-rule only if it isn't matched the conditions of the i-1 preceding rules. It is used a genetic algorithm with real codification as searching method. Logically, computation time will be greater than other systems like C4.5, but it will provide flexibility to the user because it is always possible to produce rules set with 0% of error rate and, from here, to relax the error rate for having less rules. Afterwards, a qualitative approach is made to obtain a linguistic rule set. Finally, several results are summarized in section 4.

1 Introduction

C4.5 [1] is one of the most extended programs for supervised learning by several qualities: easy use, low computation time, clear interpretation of the results and acceptable error rate. Its algorithm learns decision trees by constructing them top-down, beginning with the question "which attribute should be tested at the root of the tree?" The best attribute is selected by using a statistical test to determine how well it alone classifies the training examples. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node. This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices. Therefore, it is susceptible to the usual risks of hill-climbing search without backtracking: converging to locally optimal solutions that are not globally optimal.

Genetic algorithms employ a randomized search method to seed a maximally fit hypothesis. This search is quite different from other learning methods. The genetic algorithm search can move much more abruptly, replacing a parent hypothesis by an offspring less likely to fall into the same kind of local minima that can happen with other methods, like C4.5.

Some training files are very difficult to classify. Below, it is presented a file with two parameters and two classes: white or black circles. The most immediate classification of these data is to select three rectangles of black circles and the remainder of white circles,

that is, four rules. However, C4.5 produces a set of thirteen rules, which are presented in the figure on the right.

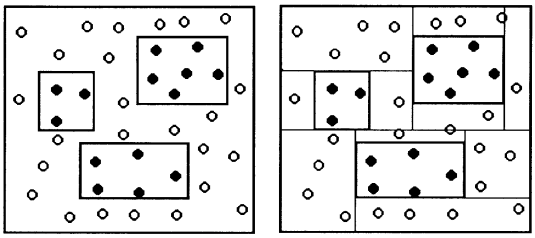


Fig. 1. Hierarchical rules vs C4.5

Sometimes it can be useful that user sets a maximum error rate for having a rules set with the least number of rules, according to the initial error rate. Or vice versa, this is, user establishes the number of rules that he wants and the system produces a rules set with the minimum error rate for that number of rules.

Our approach for solving the disadvantages of C4.5 is to produce as solution a hierarchic rules set. Here, rules must be applied in specific order. With this organization, the number of rules is substantially reduced. The shape of these rules would be:

```
If conditions then A-class
else  If conditions then B-class
      else If ..................................
                        else Z-class.
```

The objective is to design a system able to obtain a decision rules set from a labelled database. We propose a genetic algorithm [2] with real codification for the individuals [3]. GABIL [4] and GIL [5] are the most known tools. Both have binary codification but these can not work well with continuous spaces.

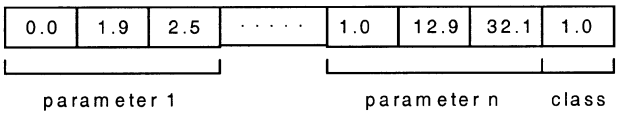
Finally, in [6] we presented a first version of this work, which produces a rules set without errors by using genetic algorithms with binary codification. Here, in addition to use real codification, it is possible to relax the error rate what we wants for having less rules and for making more comprehensible the linguistic model that rules set offers.

2 Description of the system

2.1 The environment and its codification

Information of the environment come from a data file, where each example has a class and a number of parameters. The number of parameters is fixed but undefined, like the number of classes. The genetic algorithm uses real codification, that is, an individual is

formed by a n-tuple of real, including class, that it takes real values beginning with 1 and adding a unit to the successive. We consider that an example is a member of k-class if it satisfies its conditions, which are defined with upper and lower boundaries for each parameter. So, the representation of an individual takes the following form:



Each parameter is defined by,

- A value belonging to the set {0, 1, 2}, that identifies the type of operation:
  - 0 means "if p1 ≤ 1.9..."
  - 1 means "if p1 ≥2.5..."
  - 2 means "if 1.9 ≤ p1 ≤ 2.5..."
- The two following values are the limits for each parameter. If the operator is ≤ only makes sense the first value; if the operator is ≥, the second value is used; and, in the third case the two values indicates lower and upper boundaries.
- The last value identifies the class. It will exists as many possible values as classes, that is to say, if there are five classes, the value will belong to the set {0,1,2,3,4}.

2.2 The algorithm

The algorithm is:

```
initialize
repeat
  evolution
  select rule
  adapt environment
until stop condition
```

The *initialize* module calculates the number of data of the training file, the number of parameters per datum and the number of classes. It also calculates the maximum and minimum values for each parameter. The *evolution* module is a classic evolutionary algorithm with real codification which is seeking a rule to minimize the fitness function. The *select rule* module chooses the best individual of the evolutionary process, transforming it into a rule. This rule is used in *adapt environment* to eliminate data from the training file that fulfill its premise (though they do not satisfy the conditions). In this way, the training file is reduced for the following iteration. *Stop condition* can be reached when the training file have been totally covered or when a number of rules have been obtained, or when it has been reached a precision. It is a typical sequential covering algorithm [7].

2.3 Initial population

The initial population is randomly generated. First, two values are randomly obtained between the boundaries of each parameter. An valid individual is formed by the values that belongs to the set of intervals before obtained. However, the examples that are near to the limits are hard to cover during the evolutive process. For solving it, the search space is increased. For thus, the lower bound is decreased a 5% of the range, and the upper bound is increased in the same value. To the end of the process, if the bound of a interval is out of range, this is replaced by the original limit of the database search space.

2.4 Genetic operations

The main feature of the evolution module presents is elitism, in the sense that the best of every generation is replicated to the next one and a set of children are obtained from copies of the parents, selecting it randomly but depending on their fitness values. The rest of individuals are formed through crossovers and after applying a mutation, depending on a probability of mutation.

We used two kind of crossover which are alternatively applied depending on a probability. On the one hand, it has been preserved the "n-point random crossover", that takes advantage of parameter values which can be good for some individual. The n-point random crossover chooses randomly a number k of crosspoints and the offspring (two individuals) inherits pieces between the crosspoints. The usefulness of this crossover is justified since can exist individual whose n-1 parameters may have good values but not the remaining parameters; then, they are crossed these values with other individuals before rejecting the defective individual. When n-point random crossover is applied with real codification do not generate new values for the parameters, on the contrary that binary codification. On the other hand, it has been used a real crossover specifically designed. The real crossover uses three weighted types depending on the significance that they could have in relation to the solution:

- intermediate segmented crossover: it obtains a random value belonging to the defined segment by two values which occupies the same location in both selected individuals (it is applied approximately the 40 % of the times).
- segmented crossover forced to the minimal (maximal): it obtain a random value belonging to the defined segment by the lower (upper) bound of the range and the smaller (greater) of the two values which occupies the same location in both selected individuals (it is applied approximately the 40 % of the times each one).

Mutation makes to grow the region covered by the individual so that it could take in more examples of the database. Two kinds of mutation have been used:

- incremental mutation: if it is right value of the parameter, the value is incremented a small quantity (1 % of the range); and if it is the left value of the parameter, it is decreased the same percentage.
- mutation forced to the boundary: one of the boundaries is mutated depending on whether it is the right value or the left value of the parameter range. This will make with very small probability (5% of the mutations).

2.5 Fitness function

The evolutionary algorithm minimizes the fitness function f for each individual. It is given by

$$f(i) = \frac{1}{1 + ND - g(i)} \quad (1)$$

where ND is the cardinality of the training file and g is a penalty function. This function presents two relevant features: first, the difference between class and data errors; and second, the use of a new factor called coverage. Class errors are produced when an example is covered by a rule even though the classes aren't the same. Class errors are produced when an example isn't covered by a rule, independently of the class.

Due to the different influence of that errors types in the learning task, it is necessary to introduce two penalty factors, one for each error type, called class error and data error penalty, respectively. The rule coverage is the side of a n-dimensional hypercube which volume is equivalent to the volume of the covered n-dimensional region by the rule. This new operator, applied to the population, allows to increase or to reduce the region of a rule without loosing matched examples. In this way, every rule can best adapt to the space by reducing its volume or, on the contrary, it can quickly expand for finding more examples. Particularly, our approach rewards to individuals that covers more space with same number of matched examples.

2.6. Relaxing coefficient

Databases uses as training files have not areas clearly differentiated in n-orthohedrons, for that, to obtain a rules system totally coherent involves a high number of rules. We show in previous paper a system (that we call COGITO) capable of producing a rules set exempt from error rate; however sometimes, it is interesting to reduce the number of rules for having a rules set that it can be used like a comprehensible linguistic model. When databases present a distribution of examples very hard to classify, then it is advisable to use a relaxing coefficient. Many times, we are more interested to understand the structure of databases than error rate. In this way, it could be better a system with less cardinality (despite some errors) than too many rules (with 0% of error rate). Then, it could be interesting to introduce the relaxing coefficient for understanding the behaviour of databases by decreasing the number of rules. Relaxing coefficient indicates what percentage of examples inside of a indicated region can have different class to the individual. For example, if we allow a relaxing coefficient of 10%, it means that an individual covering 83 examples of the class 'A' could make 8 errors as maximum. Relaxing coefficient behaves like the upper bound of the error with respect to the training file.

The present version of COGITO allows to limit the number of rules that we want to obtain and it can fix a maximum for the error rate. Consequently, two kinds of questions we could tackle: users can fix a maximum error rate and system produces a rules set minimizing the number of these; or users fix a maximum number of rules and the system produces a classifier minimizing the error rate. Results are very satisfactory

in both cases since user directly controls the error rate. For example, if we fix the relaxing coefficient (R=20) and it obtains a error rate equal to 16 with 23 rules, the user can decide: whether to increase R if the number of rules is too high, or to decrease R if the error rate is too high. In this way, users can get the rules sets more beneficial for its application.

3 Qualitative Rules

3.1 Linguistic terms definition

The technique to obtain a qualitative information of a spatial arrangement, expressed in the previous paragraphs can be in a way simple converted automatically in a model based on linguistic terms.

To express through a linguistic term a value range of a variable, is a relatively easy task. Only it must take into account two considerations: the first one is that the number of terms that are defined must be enough to attempt to cover most of linguistic nuances of the possible value ranges. The second is that the number should not be excessive so as to complicate the understanding of the model. To choose the number of terms together with the nomenclature of these is, then, the only difficulty of this approximation. In this case the linguistic model would follow the following grammar:

```
<model>          ::= <list_rules>
<list_rules>     ::= <list_rules> <rule>
<rule>           ::=IF <list_premises> THEN <conclusion>
                   | <rule>
<list_premises>  ::= <list_premises> AND <premise>
                   | <premise>
<premise>        ::= PARAMETER IS <list_terms>
<list_terms>     ::= <list_terms> OR <TERM> | <TERM>
                   | not <TERM>
<conclusion>     ::= <CLASS>
```

For the assignment between a set of values and what in the grammar has been expressed as <TERM> will have to be followed then steps:

1) The parameter p is supposed to have a range of values that remains defined by the minimal and maximum values of that parameter in the database. Those values will be m and M, respectively.

2) The interval of values is supposed to be divided into L linguistic terms, that may have an equivalence in L ranges of equal length values for the parameter p.

3) The central values of those ranges would come to consider the values succession:

$$m+\frac{M-m}{2L},\ m+\frac{3(M-m)}{2L},\ \ldots,\ m+\frac{(2L-1)(M-m)}{2L} \tag{2}$$

4) From these central values each range would have a value of $\leq(M-m)/2L$, that is, the first term would come defined by the range $[m,m+(M-m)/L]$, the second term would give value to the numbers of the range $[m+(M-m)/L, m+2(M-m)/L]$, and so on until $[m+(L-1)(M-m)/L,M]$.

3.2 Linguistic terms assignment to an interval

Let the interval $[c,C]$ that have been determined by COGITO for a given parameter. For the assignment of a rule to each interval the following methodology is proposed:

Notation:

C : if it exists superior bound.
c : if it exists inferior bound.
L : number of linguistic terms.
v_i $1\leq i\leq L$: central value of the ith term.
t_i $1\leq i\leq L$: ith linguistic term.
TERM: linguistic term assigned.

Method:

If $\exists\ c$, let r with $2\leq r\leq L$ / $v_{r-1} < c$ & $v_r > c$ else $r=1$.
If $\exists\ C$, let s with $1\leq s\leq L-1$ / $v_s < C$ & $v_{s+1} > C$
else $s=L$

TERM is calculated through the equation:

$$TERM = \bigcup_{i=r}^{s} t_i \tag{3}$$

where the union symbol indicates the conjunction of the terms by the logical operator or. If *TERM* is formed by the union of L-1 linguistic term, it is substituted for the expression *not TERM'* where *TERM'* is the linguistic term that is not in *TERM*. Finally, if a parameter have an interval that coincide with the range of parameter, then the correspondent premise is unnecessary.

4. Application

First, it has been applied to two files specifically created to demonstrate the weakness of C4.5 to find good regions in same cases. Afterwards, it has been applied to the standard databases for learning and classification of the UCI Repository [8]. We have compared results with C4.5, being presented in both cases the number of rules and error rate for every training file.

4.1 Bricks and boxes

The figure below shows a database (that we have called bricks) with two parameters and three labels, so that if the first rule collects the information of the central square, five hierarchic rules are sufficient to cover the whole database (left). However, C4.5 would need eight rules at least (right).

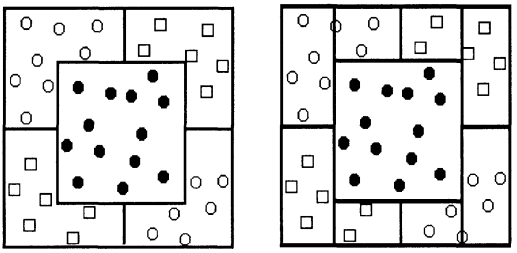


Fig. 2. Bricks database

Next figure shows a database (that we have called boxes), yet more extreme, since white or black circles are found distributed in squares one inside of the other. A hierarchical rules set only needs five rules while C4.5 produces not less than seventeen rules.

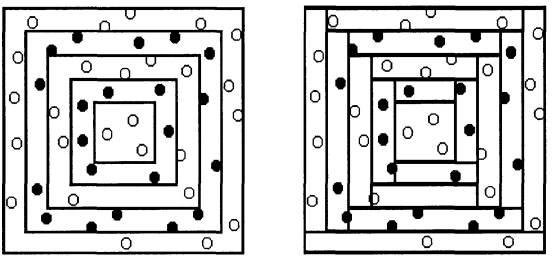


Fig. 3. Boxes database

Results, for both C4.5 and COGITO are shown in the table 1.

DATABASE	C4.5		COGITO	
	ER	NR	ER	NR
BRICKS	0	9	0	5
BOXES	0.8	27	0	5

Table 1. ER means error rate and NR means number of rules

This new version of COGITO improves the execution time on 25%. It can be asserted that a classifier system based on genetic algorithm with real codification is more powerful than other based on binary codification, since for a number of generations and a size of population fixed, the results are better in the first one than in the second one.

4.2 Databases of the UCI Repository

Below, they are presented the obtained results which are compared to those that C4.5 offers. In the two cases are indicated the number of rules and the error rate.

DATABASES	C4.5		COGITO	
	ER	NR	ER	NR
IRIS	1.3	6	0	6
WINES	1.1	5	0	3
PIMA	3.6 8.5	115 63	11.46	13
BUPA	6.1 9.6	55 35	13.91	7
BREAST CANCER	0.7	23	2.64	2

Table 2. Results for UCI Databases

It is notorious the difference between results offered by C4.5 and COGITO, above all, with respect to the number of rules. Though in the last three cases the error rate of C4.5 is lower, the number of rules is much greater, which makes complex the linguistic model. Furthermore, with all certainty, COGITO will obtain lower error rate than C4.5 for the same number of rules. Wine and Breast Cancer databases are surprising since they needs only one rule for each class -in the first case with unsurpassable efficiency-, what demonstrates the quality of the provided solutions.

In the following table it can be appreciated the range of error rate varying the number of rules for Pima database.

PIMA	5 R	6 R	7 R	8 R	11 R	12 R	13 R
TC	18.88	18.49	16.02	14.45	13.02	12.11	11.46

Table 3. Error rate for Pima Database with several NR

The executions have been accomplished in SUN SPARC 1000 E and, logically, execution time depends on the number of data of the training file and on the number of rules sought. For 200 individuals and 200 generations, to find the thirteen rules of Pima can take about 15 minutes, while the two rules of Breast Cancer about two minutes.

It could be a disadvantage the diversity of solutions since COGITO depends on probabilistic search. The tables shows the better of the possible executions, however, after realizing 20 executions to find 13 rules for Pima, the error rate range of found solutions was from 11.5 to 12.9, what from our point of view is not a relevant factor.

Below, it has shown two rule systems, Bupa and Breast Cancer, respectively, and number of goals/errors of each rule with respect to the training file.

Database BUPA:

Quantitative rules:

If p1≤97.0 y 44.0≤p2 y p3≤70.0 y 22.0≤p4 y p5≤297 y p6≤16.0 then class=A (79/15)

Else If p1≤103.0 y 19.5≤p2≤79.0 y p3≤20.0 y 5≤p4 y 7.0≤p5 y 0.0≤p6 then class=A (49/8)

Else If 87.0≤p1≤100.0 y 59.0≤p2 y 15.1≤p3 y p4≤47.8 y p5≤297 y 0.0≤p6 then class=B (81/18)

Else If p1≤94.0 y 23.0≤p2 y 15.1≤p3 y 14.0≤p4≤21.0 y p5≤30.0 y p6≤20.0 then class=B (18/1)

Else If 78.0≤p1 y 35.0≤p2≤119.0 y 11.0≤p3≤33.0 y 5.0≤p4 y 9.0≤p5 y p6≤20.0 then class=A (36/2)

Else If p1≤103.0 y p2≤134.0 y 4.0≤p3 y p4≤55.0 y p5≤48.0 y 0.0≤p6 then class=B (18/1)

Else If p1≤103.0 y 23≤p2 y p3≤155.0 y p4≤82.0 y p5≤297.0 y 0.0≤p6 then class=A (16/3)

Qualitative rules:

If p1 is not big and p2 is not small and p3 is small or medium and p4 is not small and p6 is not big then A

Else If p1 is small or medium and p2 is small or medium and p3 is small then A

Else If p1 is big and p2 is medium or big and p3 is not small and p4 is small or medium then B

Else If p1 is not big and p3 is not small and p4 is small and p5 is small then B

Else If p1 is small and p2 is medium and p3 is small then A

Else If p4 is small or medium and p5 is small then B

Else A

Database CANCER:

Quantitative rules:

If p1≤7.0 y p2≤6.0 y 1.0≤p3 y 1.0≤p4 y1.0≤p5 y p6≤6.3 y1.0≤p7 y p8≤8.3 y 1.0≤p9 then class=A (432/16)

Else If 1.0≤p1 y 1.0≤p2 y 1.0≤p3 y 1.0≤p4 y 1.0≤p5 y 1.0≤p6 y 1.0≤p7 y 1.0≤p8 y 1.0≤p9 then class=B (233/12)

Qualitative rules:

If p1 is not big and p2 is small or medium and p6 is small or medium and p8 is not big then A Else B

5 Conclusions

A supervised learning tool to classify databases in n-orthohedrons is presented. It produces a hierarchic rules set where conditions of each rule indicates the belonging of a example to an orthohedron. Relevant qualities of this system are two: important reduction of number of rules in comparison to C4.5 and flexibility to construct the classifier fixing boundaries to the number of rules or error rate through relaxing coefficient. The intervals defined by this system are easily translated to qualitative rules. The results as for bricks and boxes as UCI Repository databases has been very satisfactory.

References

1. Quinlan, J.R. C4.5: Programs for Machine Learning, Morgan Kaufmann Pub., 1993.
2. Goldberg, D.E. Genetic Algorithms in search, optimization and machine Learning. Addison-Wesley, 1991.
3. Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution programs. Second Edition, Springer-Verlag, 1994.
4. De Jong,K.A. Using Genetic Algorithms for Concept Learning. Machine Learning, 93.
5. Janikow, C.Z. A Knowlege-Intensive Genetic Algorithm for Supervised Learning. Machine Learning, 93.
6. Aguilar, J., Riquelme, J. y Toro, M. COGITO: Un Sistema de Autoaprendizaje Basado en Algoritmos Genéticos. Actas de las III Jornadas de Informática. pp. 79-88., 1997.
7. Mitchell, T. Machine Learning. MacGraw-Hill, 1997.
8. Murphy, P. y Aha, D.W. UCI repository of machine learning databases. Dept. of Information and C.S. University of California, Irvine, 1994 .