Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Information Sciences 156 (2003) 173–188

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Supervised learning by means of accuracy-aware evolutionary algorithms ☆

José C. Riquelme, Jesús S. Aguilar-Ruiz *, Carmelo Del Valle

Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, Avda, Reina Mercedes s/n, 41012 Sevilla, Spain

Received 15 March 2002; accepted 1 January 2003

Abstract

This paper describes a new approach, HIerarchical DEcision Rules (HIDER), for learning generalizable rules in continuous and discrete domains based on evolutionary algorithms. The main contributions of our approach are the integration of both binary and real evolutionary coding; the use of specific operators; the relaxing coefficient to construct more flexible classifiers by indicating how general, with respect to the errors, decision rules must be; the *coverage* factor in the fitness function, which makes possible a quick expansion of the rule size; and the implicit hierarchy when rules are being obtained. HIDER is accuracy-aware since it can control the maximum allowed error for each decision rule. We have tested our system on real data from the UCI Repository. The results of a 10-fold cross-validation are compared to C4.5's and they show a significant improvement with respect to the number of rules and the error rate.

© 2003 Elsevier Inc. All rights reserved.

Keywords: Evolutionary algorithms; Supervised learning; Decision trees

☆ The research was supported by the Spanish Research Agency CICYT under grant TIC2001–1143–C03–02.

* Corresponding author. Tel.: +34954553871; fax: +34954557139.

E-mail addresses: riquelme@lsi.us.es (J.C. Riquelme), aguilar@lsi.us.es (J.S. Aguilar-Ruiz), carmelo@lsi.us.es (C.D. Valle).

0020-0255/$ - see front matter © 2003 Elsevier Inc. All rights reserved.

doi:10.1016/S0020-0255(03)00175-0

1. Introduction

Supervised learning is used when the user knows the outcomes of the data samples and wants to predict the outcome of a new unseen instance. An algorithm carries out the prediction (classification) and it can produce knowledge by using a suitable and understandable representation. Some techniques, like nearest neighbour searching or neural networks, can classify an instance, but cannot obtain the knowledge from the information stored in the database. However, other techniques produce sets of rules with a specific structure: decision trees, decision lists, or simply set of rules. In general, when a rule-based framework is used to express the acquired knowledge, this is often called decision rules. Such rules can subsequently be used both to infer properties of the corresponding categories and to classify other, previously unseen, examples from the original space.

Decision trees are a particularly useful tool in the context of supervised learning because they perform classification by a sequence of tests whose semantics is intuitively clear and easy to understand. Some techniques, like C4.5 [14], construct decision trees selecting the best attribute by using a statistical test to determine how well it alone classifies the training examples. This sort of decision tree may be called axis-parallel, because the tests at each node are equivalent to axis-parallel hyperplanes in space. On the other hand, other techniques build oblique decision trees, such as *OC*1 [13], that tests a linear combination of the internal attributes at each node, so that, these tests are equivalent to hyperplanes at an oblique orientation to the coordinate axes. To find out the smallest decision tree (axis-parallel or oblique) is a NP-hard problem [5].

Genetic Algorithms (GA) are a family of computational models inspired by the concept of evolution. These algorithms employ a randomized search method to find solutions to a particular problem [9]. This search is quite different from the other learning methods mentioned above. A GA is any population-based model that uses selection and recombination operators to generate new sample examples in a search space [20]. The GA search can move much more abruptly, replacing a parent individual with an offspring less likely to fall into the same kind of local minima that can happen with the other methods. GAs have been used in a wide variety of optimization tasks [8,11] including numerical optimization and combinatorial optimization problems, although the range of problems to which GAs have been applied is m broad. The main tasks in applying GAs to any problem consists in selecting an appropriate representation (coding) and an adequate evaluation function (fitness).

In classical GAs the members of the population (typically maintaining a constant-sized) are represented as fixed-length strings of binary digits. The length of the strings and the population size P are completely dependent on the problem. The population simulates the nature's behavior, since the relatively "good" solutions produce offspring which replace the relatively "worse" ones, retaining many of the features of their parents. The estimate of the quality of a solution is based on a fitness function, which determines how good an individual within the population in each generation is. New individuals (offspring) for the next generation are formed by using (normally) two genetic operators: crossover and mutation. Crossover combines the features of two individuals to create several (commonly two) individuals. Mutation operates by randomly changing several components of a selected individual.

The aim of our research was to obtain a set of rules to classify a database in the context of supervised learning. First approaches searched for rules without errors, so the number of rules was very high. Later we controlled the error a rule could make, introducing the relaxing coefficient, i.e. an allowed error for the ruleset. This parameter provides accuracy-awareness to the system, allowing some experimentation to select an appropriate number of rules for the task in consideration. As we will see below, this parameter has no great influence on the average error rate.

In previous works, we presented a system to classify databases using binary coding [2]; afterwards, we adopted real coding to handle continuous domains and axis-parallel representation efficiently. (In addition, we explored other representations such as rotated hyperrectangles and hyperellipses [3]). Then, new genetic operators were introduced for real coding. In order to indicate that a GA is being implemented with real coding and genetic operators based on this coding, several authors use the term Evolutionary Algorithms (EA) instead of GA. HIerarchical DEcision Rules (HIDER) uses an EA to search for the best solutions and produces a hierarchical set of rules. According to the hierarchy, an example will try to be classified by the ith rule if it does not match the conditions of the $(i-1)$th preceding rules. The rules are obtained sequentially until all the examples from the dataset are covered. The behavior is similar to a *decision list* [17].

We extend the concept of decision list to continuous domains. Decision lists work well with objects that are described as concepts, so it can represent boolean attributes (positives or negatives examples). However, when we want to learn rules in the context of continuous attributes, we need to extend the concept of decision list in two ways: first, for adapting the boolean functions to interval functions; and second, for representing classes instead of true and false values (positives and negatives examples). For each continuous (real) attribute a_i we obtain the boundaries values, called L_i and U_i (lower and upper bounds, respectively) which define the space R_i (range of the attribute i). These intervals allow us to include continuous attributes in a decision list. Our decision list does not have the last constant function true. However, we could interpret last function as an unknown function, that is, we do not know which class the example belongs to. Therefore, it may be advisable to say "unknown class"

```
If conditions Then class
  Else If conditions Then class
       Else If conditions Then class
            ..........................................
                Else “unknown class”
```

Fig. 1. Hierarchical set of rules.

instead of making an erroneous decision. From the point of view of the experiments, “unknown class” will be considered as an error. The structure of the set of rules will be as shown in Fig. 1. A more descriptive example of rule will appear in Fig. 6, where the different sorts of conditions are shown.

The way in which C4.5 splits the space is depicted in Fig. 2. The numbers within the circles describe the level on the tree where these attributes are placed. HIDER is quite different because it does not divide the space by an attribute,

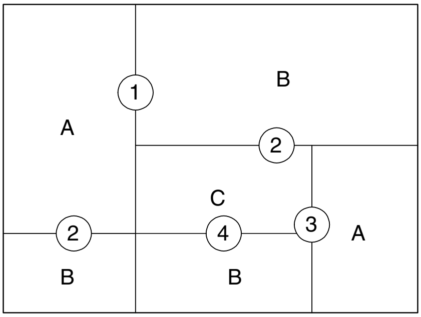

Fig. 2. Splitting the search space using C4.5. Numbers within circles indicate the level of the condition on the decision tree.

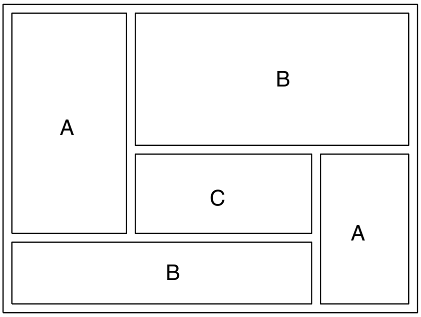

Fig. 3. Splitting the search space using HIDER.

but it extracts sequentially a region from the space. This permits obtaining entire regions with the same class, as illustrated in Fig. 3. See the region labelled as B on the bottom-left corner of the Figs. 2 and 3. In Fig. 2, C4.5 divides the region into two parts. However, HIDER will obtain the complete region. For another artificial two-dimensional database, Fig. 4 shows the classification that C4.5 gives. Nevertheless, as illustrated in Fig. 5, rules inside of another one could improve the quality of the rule set. The most evident feature, graphically observed in Fig. 5, is the reduction of the number of rules because of the rules overlapping. This characteristic motivates us to use hierarchical decision rules instead of independent decision rules.

As mentioned in [7] one of the primary motivation for using real-coded EAs is the precision to represent attributes values and the other is the ability to exploit the gradualness of functions of continuous attributes. We implemented our first versions with binary-coded GAs, but we could show that real-coded EAs are more efficient in time and quality of results.

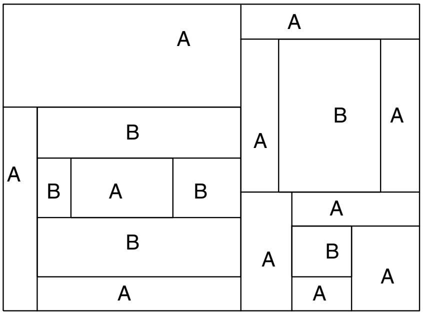

Fig. 4. Division of the search space done by C4.5.

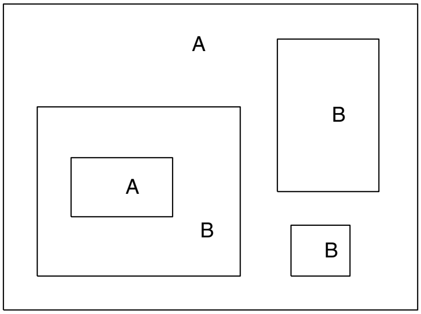

Fig. 5. Hierarchical division of the search space done by HIDER.

2. Principles

Before an EA can be run, a suitable *coding* for the problem must be devised. We also require a *fitness function*, which assigns a figure of merit to each coded solution. During the run, parents are *selected* for reproduction, and *recombined* to generate *offspring*. These aspects are described below.

2.1. Coding

In order to apply EAs to a learning problem, we need to select an internal representation of the space to be searched and define an external function that assigns fitness to candidate solutions. Both components are critical to the successful application of the EAs to the problem of interest.

Information on the environment comes from a data file, where each example has a class and a number of attributes. We have to codify that information to define the search space, which normally will be dimensionally greater. Each attribute will be formed by several components in the search space, depending on the specific representation. Two basic principles exist for choosing the coding: the principle of meaningful building blocks and the principle of minimal alphabets [9].

In first approaches, we studied GA-based classifier [6,10] with binary coding. These are generally used as concept learners, where coding assigns a bit to each value of the attribute, i.e., every attribute is symbolic (GABIL and GIL are two well-known systems). For example, an attribute with three possible values would be represented by three bits. A value of one in a bit indicates that the value of the attribute is present. Several bits could be active. This coding is appropriate for symbolic domains. However, it is very difficult to use in continuous domains, because the number of possible values of an attribute is infinite.

The length of an individual is determined by the sum of the number of values of each attribute. Using binary encoding in continuous domains requires transformations from binary to real for every attribute in order to apply the evaluation function. Moreover, when we convert binary to real, we are loosing precision. For that reason, we have to find the exact number of bits in order to eliminate the difference between any two values of an attribute. This ensures that a mutation of the less significant bit of an attribute should include or exclude at least one example from the training set [16].

The representation for continuous and discrete attributes is best explained by referring to Fig. 6, where l_i and u_i are values representing an interval for the continuous attribute; b_i are binary values indicating that the value of the discrete attribute is active or not (internal disjunction). A last omitted value is for the class.

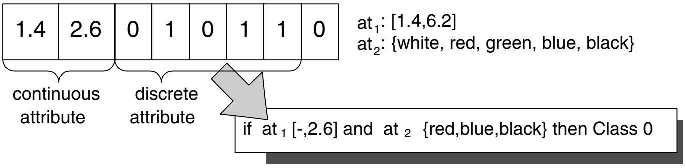
1.4
2.6
0
1
0
1
1
0
at_1: [1.4,6.2]
at_2: {white, red, green, blue, black}
continuous attribute
discrete attribute
if at_1 [-,2.6] and at_2 {red,blue,black} then Class 0

Fig. 6. Continuous (left) and discrete (right) attributes.

The number of classes determines the set of values to which it belongs, i.e., if there are five classes, the value will belong to the set 0, 1, 2, 3 and 4. Every rule will be obtained from this representation, but when $l_i = \min(a_i)$ or $u_i = \max(a_i)$ the rule will not have that value. For example, in the first case the rule would be $[-, v]$ and in the second one $[v, -]$. If both values are equal to the boundaries then the rule appears $[-, -]$ for that attribute, which means it is not relevant because whichever of the attribute's values will be covered by the whole range of that attribute ($[-, -]$). Under these assumptions, some attributes could not appear in the rule set. In the same way, when every discrete value is active, that attribute does not appear in the rule.

2.2. Algorithm

The algorithm is a typical sequential covering EA [12]. An overview of the EA-based classifier is shown in the Fig. 7. It chooses the best individual in the evolutionary process, transforming it into a rule which is used to eliminate data from the training file [19]. In this way, the training file is reduced for the following iteration. A termination criterion might be reached when more examples to cover do not exist. The method of generating the initial population consists in randomly selecting an example from the training file for each individual of the population, and afterwards, an interval to which the example belongs is obtained. For example, let L_i and U_i be the lower and upper bounds

```
While exists examples in training file
  Step 1. Initialize population
  Step 2. Repeat num generations times
    Step 2.1. Evaluation
    Step 2.2. Select the best one
    Step 2.3. Replication
    Step 2.4. Crossover and Mutation
  Step 3. Put the best one in the Decision List
  Step 4. Eliminate examples covered by the best one
```

Fig. 7. Pseudocode.

of the attribute i; then, the range of the attribute is $U_i - L_i$; next, we randomly choose an example $(v_{a_1}, \ldots, v_{a_i}, \ldots, v_{a_m}, class)$ from the training file, where m is the number of attributes; for the last, a possible individual of the population could thus be $(\ldots, v_{a_i} - (\frac{U_i - L_i}{N})k_1, v_{a_i} + (\frac{U_i - L_i}{N})k_2, \ldots, class)$, where v_{a_i} is a value for the attribute i; k_1 and k_2 are random values belonging to $[0, \frac{N}{C}]$ (N is the size of the training data; C is the number of different classes; and $class$ is the same of that of the example). For discrete attributes, the individual has as many positions as different values for the attribute, although we assure that at least the same active value of the example will remain active in the individual.

For instance, let the dataset be the one used in Fig. 6. A possible individual for the initial population is obtained from a randomly selected example $e = (1.8, blue, 0)$. The individual could be $ind = (1.4, 2.6, 0, 1, 0, 1, 1, 0)$. The interval [1.4, 2.6] is for the continuous attribute and the values (0,1,0,1,1) is for the discrete one. Notice that the value *blue* is active and other values (*red* and *black*) have also been randomly set to 1. The individual keeps the same class that of the example.

Sometimes, the examples very near the boundaries are hard to cover during the evolutionary process. To solve this problem, the search space is increased (currently, the lower bound is decreased by 5%, and the upper bound is increased by 5%). This value is calculated from experimentation and when invalid offspring is generated, those values are adjusted to the boundaries of the attribute.

The evolution module includes elitism: the best individual of every generation is replicated to the next one. A set of children (50%) is obtained from copies of randomly selected parents, generated by their fitness values and using the roulette wheel selection. The remainder is formed through crossovers. Since half of the new population is created by applying the crossover operator, the probability of crossover is 0.5 and the probability of selecting an individual for crossing depends on its fitness value. These individuals could be mutated later (only the individual from the elite will not be mutated).

Wright's linear crossover operator [22] creates three offspring: treating two parents as two points p_1 and p_2, one child is the midpoint of both, and the other two lie on a line determined by $\frac{3}{2}p_1 - \frac{1}{2}p_2$ and $-\frac{1}{2}p_1 + \frac{3}{2}p_2$. Radcliffe's flat crossover [15] chooses values for an offspring by uniformly picking values between the two parents values inclusively. Eshelman and Schaffer [7] use a crossover operator that is a generalization of Radcliffe's which is called blend crossover (BLX–α). It uniformly picks values that lie between two points that contain the two parents, but may extend equally on either side determined by a user specified EA-parameter α. For example, BLX–0.1 picks values from points that lie on an interval that extends $0.1I$ on either side of the interval I between the parents. Logically, BLX–0.0 is the Radcliffe's flat crossover.

Our crossover operator is like Radcliffes's most of the time, but sometimes the value is slightly varied to approximate it to the boundary. Let $[l_i^j, u_i^j]$ and

$$\begin{array}{lll}
l \in [min(l_i^j, l_i^k), max(l_i^j, l_i^k)] & u \in [min(u_i^j, u_i^k), max(u_i^j, u_i^k)] & (0.85) \\
l \in [L, min(l_i^j, l_i^k)] & u \in [max(u_i^j, u_i^k), U] & (0.05) \\
l \in [min(l_i^j, l_i^k), max(l_i^j, l_i^k)] & u \in [max(u_i^j, u_i^k), U] & (0.05) \\
l \in [L, min(l_i^j, l_i^k)] & u \in [min(u_i^j, u_i^k), max(u_i^j, u_i^k)] & (0.05)
\end{array}$$

Fig. 8. Crossover operator: four alternatives.

$[l_i^k, u_i^k]$ be the intervals of two parents j and k for the same attribute i. From these parents we can generate four possible children selecting values as follows: let $[l, u]$ be the interval we want to obtain after applying the crossover operator to the two parents j and k, and let L and U be the boundaries of the attribute i being treated, where L and U define the range of the attribute. Once the crossover operator is selected, one of the four alternatives is applied depending on the probability denoted in brackets in Fig. 8. When the attribute is discrete, the crossover operator is like a uniform crossover [18].

Mutation is applied for continuous attributes as follows: if the location (gen) corresponds to a value of the interval (l_i or u_i), then a small value is subtracted or added, depending on whether it is the lower or the upper boundary, respectively. The small value is currently the smaller heterogeneous overlap-Euclidean metric (HOEM, [21]) between any two examples. In the case of discrete attributes, mutation changes the value from 0 to 1 or viceversa and it is applied with low probability. We introduce a specific mutation operator to generalize the attribute when almost all values are 1. In this case, the attribute does not appear in the rule. For example in Fig. 12b, the attribute *sex* is not in the rule $R1$. For both kinds of attributes, if the location (gen) corresponds to the class, the mutation generates a new value from C, the set of classes, randomly.

2.3. Generalization

Databases used as training files do not have clearly differentiated areas, so that obtaining a rule system without errors from the training file involves a high number of rules. We showed in previous papers [1] a system capable of producing a rule set exempt from error with respect to the training file. However, sometimes it is interesting to reduce the number of rules for having a rule set which may be used like a comprehensible linguistic model. In this way, it may be advisable to have a system with fewer rules despite some errors, than too many rules and no errors. In fact, the generalization can produce a decrease in the number of rules, although this has a slight effect on the accuracy of the rule set. Due to the allowance some errors during the obtaining of the first rules, the number of rules becomes lower because a further generation of rules which cover a small number of examples (the last ones) is not necessary. However, these "allowed errors" in the first rules have no influence in the cross-validation phase. When databases present a distribution of examples very hard to classify, it can be interesting to introduce the relaxing coefficient (RC) for understanding the behavior of databases by decreasing the number of rules [16]. RC indicates what percentage of examples inside of a rule can have a different class than the rule has. RC behaves like the upper bound of the error with respect to the training file, that is, as an allowed error rate. To deal efficiently with noise and find a good value for RC, the expert should have an estimate of the noise percentage in his data. For example, if database X produces too many rules when RC is 0, we could set RC to 5 to decrease the number of rules and, possibly, the error rate is the same as before. All the experiments in this paper were run using $\mathrm{RC} = 0$ and 10. We have verified that some complex databases improve on average in both the error rate and the number of rules when RC is greater than 0 (Pima is an example).

When an individual tries to expand and always reaches examples of a different class, its fitness value cannot become higher, unless a few errors were allowed. In this case, depending on the characteristics of the fitness function, such value might increase. In Fig. 9 (right) the individual cannot get bigger, unless one error is allowed, in which case the individual will have four new examples (left), which could increase its fitness value.

2.4. Fitness function

The fitness function must be able to discriminate between correct and incorrect classifications of examples. Finding an appropriate function is not a trivial task, due to the noisy nature of most datasets. In our case, we try to both minimize the number of errors and maximize the number of correctly classified examples. A simple solution to this two-objective optimization problem is considering both variables within fitness function f as is shown in Eq. (1), where f is maximized for each individual φ from the population.

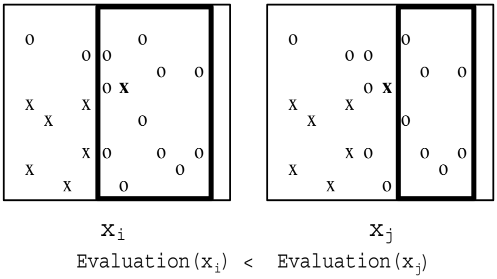

Fig. 9. Effect of RC on fitness function values.

$$f(\varphi) = 2(N - \mathrm{CE}(\varphi)) + G(\varphi) + \mathrm{coverage}(\varphi) \tag{1}$$

where N is the number of examples being processed; $\mathrm{CE}(\varphi)$ is the class error, which is produced when an example belongs to the region defined by the rule but it does not have the same class; $G(\varphi)$ is the number of examples correctly classified by the rule; and the *coverage* of a rule is the proportion of the search space covered by such rule. Each rule can be quickly expanded to find more examples thanks to the coverage in the fitness function.

3. Application

The experiments described in this section are from the UCI Repository [4]. The results obtained by HIDER have been compared to that of C4.5 Release 8 (C4.5R8). C4.5 was run using the default parameters (`c4.5 --f file --u`). To measure the performance of the method, a 10-fold cross-validation was achieved with each dataset. It is very important to note that every execution has been carried out with a population size of as little as 100 individuals and 300 generations for the EA (in cases of small datasets, like Iris, the results would have been the same using a smaller number of generations: about 50 is enough). There are very small numbers considering the number the examples and the dimensionality of some databases. Mutation is always applied with a probability of 0.1 (individual) and 0.2 (gen: inside the individual). HIDER needed about 8 h to complete the 10-fold cross-validation for the 18 databases in a Pentium 400 MHz with 64 Mb of RAM. C4.5 only needed about 8 min in the same machine. C4.5 is an extremely robust algorithm that performs well on many domains. However, C4.5 is not flexible in generating decision rules (or trees), because it is not possible to control the error rate (it only provides two decision trees: unpruned and pruned). Table 1 gives a 10-fold cross-validation of the error rates and number of rules for the C4.5 and HIDER algorithms on the selected domains. HIDER outperforms C4.5 in 12 out of 18 datasets.

Table 1 also compares the number of rules generated by the two approaches. In order to count the number of rules generated by C4.5, we could sum the leaves on the tree or apply the expression $\frac{s+1}{2}$, where s is the size of the tree. From the point of view of the comprehension of the knowledge inside the database, HIDER is much better than C4.5 since the number of rules is much smaller. Some databases present a dramatic reduction (for example, Breast Cancer or German). Results have been generated using $\mathrm{RC} = 0$ and 10. As we can see, the error rate is approximately the same for $\mathrm{RC} = 0$ and 10, although the number of rules has decreased by 15% in the last case. Breast Cancer dataset is a good example in which RC has considerable influence on the number of rules, conserving the initial error rate. The most important feature of this coefficient is that the user can control the number of rules by varying this factor

Table 1
Comparing error rates

Dataset	Error rate			Number of rules		
	C4.5R8	HIDER		C4.5R8	HIDER	
		RC = 0	RC = 10		RC = 0	RC = 10
Breast cancer	6.28	4.29	4.5	21.9	11.3	2.4
Bupa liver disorder	34.73	35.71	36.4	28.6	11.3	5.7
Cleveland	26.77	20.49	21.7	35.2	7.9	7.3
German credit	32.1	29.1	30.1	181.5	13.3	12.4
Glass	32.73	29.41	29.6	29.0	19.0	17.0
Heart disease	21.83	22.32	22.1	29.2	9.2	9.2
Hepatitis	21.42	19.41	19.3	13.8	4.5	4.5
Horse colic	19.0	17.64	18.1	39.3	6.0	6.0
Iris	4.67	3.33	4.3	5.5	4.8	3.3
Lenses	29.99	25.0	25.0	4.1	6.5	5.1
Mushrooms	0.01	0.76	0.2	15.5	3.1	3.1
Pima Indian	32.06	25.9	24.9	93.6	16.6	12.1
Sonar	30.31	43.07	43.07	16.8	2.8	2.7
Tic-Tac-Toe	14.2	3.85	3.9	93.9	11.9	10.6
Vehicle	30.6	30.6	30.6	102.3	36.2	30.4
Vote	6.19	6.42	6.2	14.7	4.0	3.8
Wine	6.71	3.95	3.9	5.4	3.3	3.1
Zoo	7.0	8.0	8.0	9.9	7.2	6.5
Average	19.81	18.29	18.43	41.1	9.5	8.06

without having much influence on the error rate. Logically, after a determined value for RC, the number of rules is very low and the accuracy begins to decrease. That value can be obtained by experimentation, so different sets of rules can be provided in order to select the one which best represents the knowledge within the dataset, i.e. normally that with a lesser number of rules or attributes involved in the rule set.

Fig. 10 shows a measure of improvement for the error rate and the number of rules. To calculate those coefficients the error rate (number of rules) of C4.5 has been divided by the error rate (number of rules) of HIDER (with RC = 0). On average, HIDER found solutions that had less than one fourth of the rules output by C4.5. Surprisingly, C4.5 generated a number of rules five times greater than HIDER for one third of the databases. When the bar is to the left of 1, C4.5 does better than HIDER, and worse to the right. The ratio values indicate the percentage of improvement with respect to C4.5. For example, 1.5 (2, 2.5, etc.) means an improvement on 50% (100%, 150%, etc). It is worth noting that in applying HIDER, more than two thirds of the databases produce less than half of the rules. C4.5 only was better with Lenses database (see in Fig. 10 the sole bar to the left). C4.5 made the error rate better for six databases, although only three of them improved significatively (mushrooms,

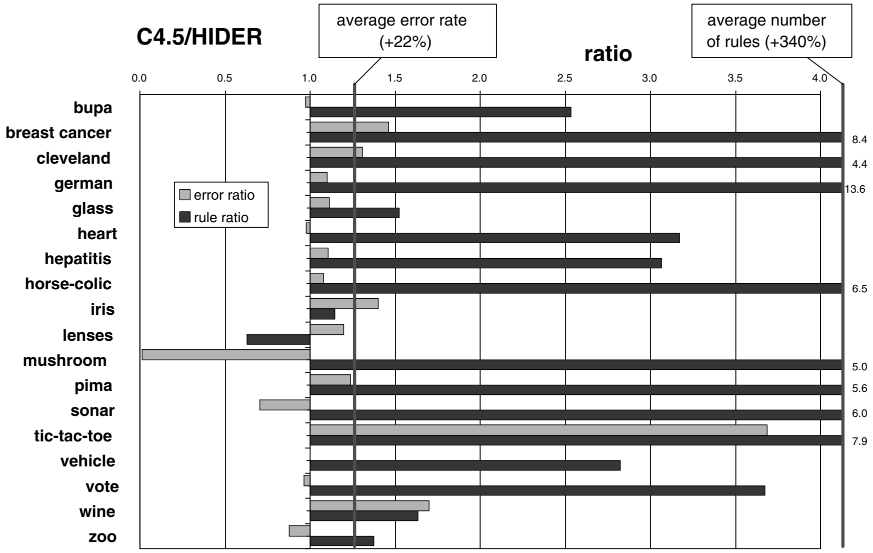

Fig. 10. Comparing the error rate and the number of rules.

sonar and zoo) as we can see in Fig. 10. Figures on the right side of the bars are shown when the ratio value is greater than 4.

The exact same folds were used for both algorithms so that the 10 resulting performance numbers for HIDER and C4.5 are pairwise comparable. The ruleset generated for the Wine database is presented in Fig. 11. HIDER produced an error rate of 0%, however, that of C4.5 was 22.2%, for that fold. Numbers in brackets are not from the test file, but the training file (correctly classified/errors).

Fig. 12 illustrates a more complex example which shows that when the number of rules is large, the number of conditions involved in the rule set is also reduced. Thus in the example, C4.5 uses 63 conditions and HIDER uses 30 conditions. The number of conditions for C4.5 is calculated by counting all the

```
c12 <= 2.15 :                       IF R1: c7 [1.08,-] and
|   c4 <= 17.5 : 2 (6.0)                   c10 [3.82,-] and
|   c4 > 17.5 : 3 (45.0/2.0)               c13 [741.80,-] : 1(51|0)
c12 > 2.15 :                        ELSE IF R2: c7 [1.14,-] and
|   c13 <= 725 : 2 (54.0/1.0)                   c11 [0.68,-] and
|   c13 > 725 :                                 c12 [1.61,-] : 2(64|1)
|   |   c10 <= 3.4 : 2 (4.0)        ELSE IF R3: c4 [11.57,-] : 3(43|0)
|   |   c10 > 3.4 : 1 (51.0)        ELSE unknown
```

(a) C4.5 (b) HIDER

Fig. 11. Ruleset generated by C4.5 (a) and HIDER (b) for the Wine database.

```
ALBUMIN <= 2.8 : 1 (9.0/1.0)
ALBUMIN > 2.8 :
|  BILIRUBIN <= 3.5 :
|  |  PROTIME <= 43 :
|  |  |  LIVER BIG = 0: 2 (3.0)
|  |  |  LIVER BIG = 1:
|  |  |  |  SGOT <= 54 : 2 (3.0/1.0)
|  |  |  |  SGOT > 54 : 1 (5.0)
|  |  PROTIME > 43 :
|  |  |  BILIRUBIN <= 1.9 :
|  |  |  |  AGE <= 58 : 2 (89.0/2.0)
|  |  |  |  AGE > 58 :
|  |  |  |  |  SGOT <= 55 : 2 (7.0)
|  |  |  |  |  SGOT > 55 :
|  |  |  |  |  |  SEX = 0: 1 (2.0)
|  |  |  |  |  |  SEX = 1: 2 (2.0)
|  |  |  BILIRUBIN > 1.9 :
|  |  |  |  SPIDERS = 1: 2 (6.0)
|  |  |  |  SPIDERS = 0:
|  |  |  |  |  AGE <= 45 : 2 (3.0/1.0)
|  |  |  |  |  AGE > 45 : 1 (3.0)
|  BILIRUBIN > 3.5 :
|  |  ALBUMIN <= 3.7 : 1 (5.0)
|  |  ALBUMIN > 3.7 : 2 (2.0)
```

(a) C4.5

```
IF R1: ASCITES {2} and
       VARICES {2} and
       BILIRUBIN [0.77,-] and
       ALBUMIN [2.97,6.3] and
       PROTIME [38.48,-] : 2(100|8)
ELSE IF R2: AGE [26.88,75.1] and
            SEX {1} and
            FATIGUE {1} and
            ALK PHOSPHATE [35.45,178.83] and
            ALBUMIN [2.19,-] and
            PROTIME [-,61.01] and
            HISTOLOGY {2} : 1(17|1)
ELSE IF R3: ALBUMIN [2.77,-] : 2(11|1)
ELSE unknown
```

(b) HIDER

Fig. 12. Ruleset generated by C4.5 (a) and HIDER (b) for the Hepatitis database.

necessary conditions to match one class. In Fig. 12, there are 13 leaves on the decision tree so that the total number of conditions is the sum of the conditions that reach each leaf $(1+4+5+5+5+6+7+7+5+6+6+3+3=63)$. The number of conditions is, in the case of HIDER, $5+12+13=30$. Moreover, the error rate was 31.2% for C4.5, in contrast with 12.5% for HIDER (using the same fold). We thought that there would not be equity if we simply counted either the number of conditions, 24 for C4.5 and 13 for HIDER, or the number of different attributes, 8 for C4.5 and 10 for HIDER, involved in the rule set.

4. Conclusions

An EA-based supervised learning tool to classify databases is presented in this paper. Real-coded GA are very efficient finding rule sets in both continuous and discrete domains (more than those based on binary-coding). HIDER produces a hierarchical set of decision rules where the conditions of each rule must be applied in a specific order. The use of hierarchical decision rules lead to an overall improvement in the performance on the 18 databases investigated here, especially with respect to the number of rules. In addition, HIDER improves the flexibility to construct a classifier varying the relaxing coefficient, which allows to the user to search for an appropriate number of rules maintaining approximately the error rate. This accuracy-awareness can be beneficial when we are interested in reduced set of rules. On average, the error rate provided by C4.5 is about 20% greater. Likewise, the number of rules provided by C4.5 is about a factor of four greater than HIDER's. Therefore, HIDER can be considered an approach of great quality.

References

[1] J.S. Aguilar, J.C. Riquelme, M. Toro, Decision queue classifier for supervised learning using rotated hyperboxes, in: Progress in Artificial Intelligence IBERAMIA'98. Lecture Notes in Artificial Intelligence 1484, Springer-Verlag, 1998, pp. 326–336.

[2] J.S. Aguilar, J.C. Riquelme, M. Toro, A tool to obtain a hierarchical qualitative set of rules from quantitative data, in: Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998, pp. 336–346.

[3] J.S. Aguilar, J.C. Riquelme, M. Toro, Three geometric approaches for representing decision rules in a supervised learning system, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '99), Orlando, Florida, USA, 1999.

[4] C. Blake, E.K. Merz, Uci repository of machine learning databases, 1998.

[5] A. Blum, R.L. Rivest, Training a 3-node neural network is np-complete, in: Proceedings of the First ADM Workshop on the Computational Learning Theory, Cambridge, MA, 1988, pp. 9–18.

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

[7] L.J. Eshelman, J.D. Schaffer, Real-coded genetic algorithms and interval-schemata, Foundations of Genetic Algorithms 2 (1993) 187–202.

[8] S. Forrest, Genetic algorithms, ACM Computer Surveys 28 (1) (1996) 77–80.

[9] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

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

[11] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer-Verlag, Berlin, 1996.

[12] T. Mitchell, Machine Learning, McGraw Hill, 1997.

[13] S.K. Murthy, S. Kasif, S. Salzberg, A system for induction of oblique decision trees, Journal of Artificial Intelligence Research (1994) 1–33.

[14] J.R. Quinlan, C4.5: Programs for machine learning, Morgan Kaufmann, San Mateo, CA, 1993.

[15] N.J. Radcliffe, Genetic Neural Networks on MIMD Computers, Ph.D., University of Edinburgh, 1990.

[16] J.C. Riquelme, J.S. Aguilar, M. Toro, A GA-based tool to obtain a hierarchical classifier for supervised learning (in spanish), Revista Iberoamericana de Inteligencia Artificial 1 (5) (1998) 38–43.

[17] R.L. Rivest, Learning decision lists, Machine Learning 1 (2) (1987) 229–246.

[18] G. Syswerda, Uniform crossover in genetic algorithms, in: Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 2–9.

[19] G. Venturini, Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts, in: Proceedings of European Conference on Machine Learning, 1993, pp. 281–296.

[20] D. Whitley, A genetic algorithm tutorial, Tech. Rep. CS-93-103, Colorado State University, Fort Collins, CO 80523, 1993.

[21] D.R. Wilson, T.R. Martinez, Improved heterogeneous distance functions, Journal of Artificial Intelligence Research 6 (1) (1997) 1–34.

[22] A.H. Wright, Genetic algorithms for real parameter optimization, Foundations of Genetic Algorithms 1 (1991) 205–218.