12

Learning Decision Rules by Means of Hybrid-Encoded Evolutionary Algorithms

J.C. Riquelme and J.S. Aguilar-Ruiz

Summary. This paper describes an approach based on evolutionary algorithms, HIDER (**HI**erarchical **DE**cision **R**ules), for learning rules in continuous and discrete domains. The algorithm produces a hierarchical set of rules, that is, the rules are sequentially obtained and must be therefore tried in order until one is found whose conditions are satisfied. In addition, the algorithm tries to obtain more understandable rules by minimizing the number of attributes involved. The evolutionary algorithm uses binary coding for discrete attributes and integer coding for continuous attributes. The integer coding consists in defining indexes to the values that have greater probability of being used as boundaries in the conditions of the rules. Thus, the individuals handles these indexes instead of the real values. We have tested our system on real data from the UCI Repository, and the results of a 10-fold cross-validation are compared to C4.5s and C4.5Rules. The experiments show that HIDER works well in practice.

12.1 Introduction

Evolutionary Algorithms (EA) 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 [25]. This search is quite different from the other learning methods mentioned above. An EA is any population-based model that uses selection and recombination operators to generate new sample examples in a search space [22]. EAs have been used in a wide variety of optimization tasks [13] including numerical optimization and combinatorial optimization problems, although the range of problems to which EAs have been applied is quite broad. The main task in applying EAs to any problem consists of selecting an appropriate representation (coding) and an adequate evaluation function (fitness).

Genetic-based searching algorithms for supervised learning, as GABIL [7] or GIL [11], do not handle easily numeric attributes because the method of encoding all possible values would lead to very long rules in the case or

real-valued attributes. Concretely, GABIL and GIL are so-called "concept learners" because they are designed for discrete domains. Other approaches, as SIA [19], have been motivated by a real-world data analysis task in a complex domain (continuous and discrete attributes).

The aim of our research was to obtain a set of rules by means of an evolutionary algorithm to classify new examples in the context of supervised learning. With our approach, HIDER, we try to handle efficiently continuous and discrete attributes.

The justification of this method will be discussed in Section 12.2. The characteristics of our approach are presented in section 12.3, where the coding, the algorithm, the selected fitness function, and a particular aspect named *generalization*, are detailed. Section 12.4 shows the experiments, the results and the analysis of them. In Section 12.5 the conclusions are summarized, some of which motivates the future works presented in Section 15.7.

12.2 Motivation

Two artificial two–dimensional databases will be used to clarify the motivation of our approach. The way in which C4.5 splits the space is depicted in Figure 12.1. The figures within the circles describe the level on the tree where the tests (nodes) over these attributes are placed. See the region labeled as B on the bottom left corner of Figure 12.1. C4.5 divides the region into two parts, however, we thought that the region should be completely covered by only one rule. This fact motivates us to design an algorithm able to discover such rule.

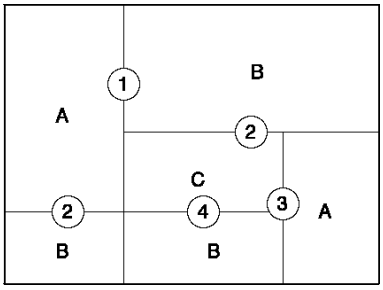

Figure 12.1. C4.5.

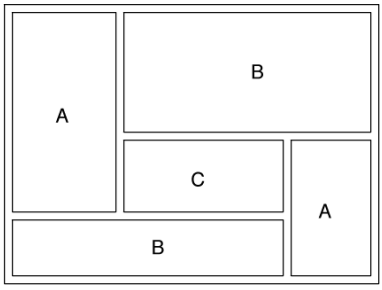

Figure 12.2. HIDER.

HIDER is quite different because it does not divide the space by an attribute, but it extracts sequentially regions from the space. This permits us to obtain *pure* regions, i.e., all examples belong to the same category. As illustrated in Figure 12.2, the region labeled as B on the bottom left corner is discovered by HIDER.

For another artificial two–dimensional database, Figure 12.3 shows the classification that C4.5 gives. Nevertheless, the quality of the rule set would be improved if the algorithm finds rules within others. The most evident feature, graphically observed in Figure 12.4, 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 (unordered) decision rules.

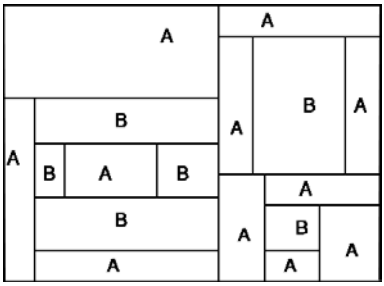

Figure 12.3. C4.5

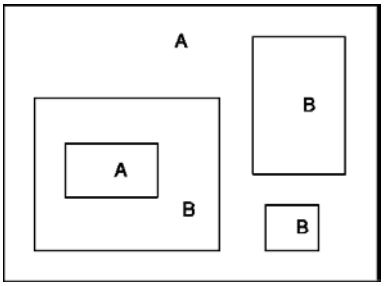

Figure 12.4. HIDER

In short, the obtaining of larger regions (without damaging the prediction accuracy) and the discovery of regions within others are the two main goals that have motivated the development of HIDER.

12.3 HIDER

HIDER (HIerarchical DEcision Rules) uses an EA to search for the best solutions and produces a hierarchical set of rules. According to the hierarchy, an example will 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 the space is totally covered. The behavior is similar to a *decision list* [17]. As mentioned in [6], the meaning of any single rule is dependent on all the other rules that precede it in the rule list, so that it might be a problem for the expert in understanding the rules (if there are many).

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 many classes instead of the 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 the 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" instead of making an erroneous decision. From the viewpoint of the experiments, when no induced rules are satisfied, "unknown class" will be considered as an error.

The structure of the set of rules will be as shown in Figure 12.5.

if conditions **then** class
  else if conditions **then** class
    else if conditions **then** class
      .....
        else "unknown class"

Figure 12.5. Hierarchical set of rules.

As mentioned in [8] one of the primary motivation for using real-coded EAs is the precision with which attribute values can be represented and another is the ability to exploit the continuous nature of functions of continuous attributes. We implemented our first versions with binary-coded GAs, but we realized that real-coded EAs were more efficient (time and quality of results).

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.

12.3.1 Coding

In order to apply EAs to a learning problem, we need to select an internal representation of the space to be searched. These components are critical for 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. To find out an appropriate coding for the problem is very difficult, but it is almost impossible to get the perfect one. There exist two basic principles for choosing the coding: the principle of meaningful building blocks and the principle of minimal alphabets [25].

In our first approaches, we studied other EA-based classifiers [7, 11] 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 so that several bits could be active for the same attribute. This coding is appropriate for symbolic

domains. However, it is very difficult to use it in continuous domains, because the number of elements in the alphabet is very large, prohibiting a complete search.

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 into real, the precision is being lost, so that we have to find the exact number of bits 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. Let l_i and u_i be the lower and upper bounds of an attribute. Let δ_i be the least absolute difference between any two values of the attribute i. The allowed *error* for this attribute must be less than δ_i. Thus, the length of an attribute will depend on that *error*.

Nevertheless, the real coding is more appropriate with real domains, simply because it is more natural to the domain. A number of authors have investigated nonbinary evolutionary algorithms theoretically [3, 4, 12, 20, 21].

The representation for continuous and discrete attributes is best explained by referring to Figure 12.6, where l_i and u_i are values representing an interval for the continuous attribute; b_k are binary values indicating that the value of the discrete attribute is active or not. A last value (omitted in the figure) is for the class.

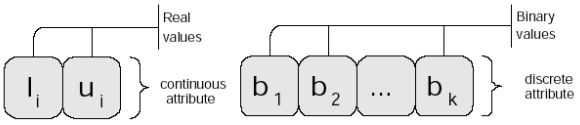

Figure 12.6. Continuous (left) and discrete (right) attributes.

For example, for a database with two attributes, one continuous and one discrete, an individual of the population could be as that depicted in Figure 12.7.

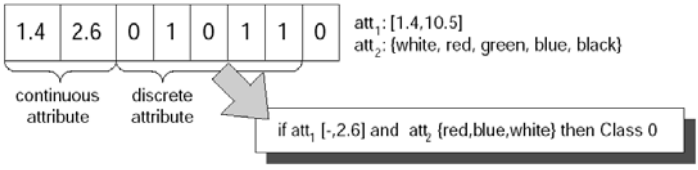

Figure 12.7. Example of coding.

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 $\{0, 1, 2, 3, 4\}$. Each rule will be obtained from this representation, but when $l_i = \min(a_i)$ or $u_i = \max(a_i)$, where a_i is an attribute, the rule will not have that value. For example, in the first case the rule would be $[-, v]$ and in the second case $[v, -]$, v being any value within the range of the attribute (see Figure 12.7). If both values are equal to the boundaries, then the rule $[-, -]$ arises for that attribute, which means that it is not relevant because either of the attribute's values will be covered by the whole range of that attribute ($[-, -]$). Under these assumptions, some attributes might not appear in the set of rules. In the same way, when every discrete value is active, that attribute does not appear in the rule.

12.3.2 Algorithm

The algorithm is a typical sequential covering EA [14]. It chooses the best individual of the evolutionary process, transforming it into a rule used to eliminate data from the training file [19]. In this way, the training file is reduced for the following iteration. HIDER searches for only one rule among the possible solutions, that compared to the algorithms based on the Michigan and Pittsburgh approaches, reduces the search space, even if several searches must be performed if several rules are to be learned.

An overview of HIDER is shown in Figure 12.8. The algorithm is divided in two parts: the procedure HIDER, which constructs the hierarchical set of rules, and the function EvoAlg, which obtains one rule every time is called. Initially, the set of rules R is empty, but in each iteration a rule is included (operator $\oplus$) in R; E is the training file, and n is the number of remainder examples that have not been covered yet (exactly $|E|$ at the begining). In each iteration the training file E is reduced (operator $-$), eliminating those examples that have been covered by the description of the rule r (Δ_r), i.e., the left-hand side of the rule, independently of its class. A parameter epf, called *examples pruning factor*, controls the number of examples that will not be covered during the process (ranging from 0% to 5%). This factor ensures that rules covering few examples are not generated. Some authors have pointed out that these rules are undesirable, especially with noise in the domain [6, 10]. The termination criterion is reached when more examples to cover do not exist, depending on epf. For the trials, we have set epf to 0.

The evolutionary algorithm is run each time to discover one rule. The method of generating the initial population (Initialize) consists of randomly selecting an example from the training file for each individual of the population. Afterwards, an interval to which the example belongs is obtained. For example, in one dimension, let l_i and u_i be the lower and upper bounds of the attribute i; then, the range of the attribute is $u_i - l_i$; next, we randomly choose an example $(a_1, \ldots, a_i, \ldots, a_m, class)$ from the training file; $\left(\ldots, a_i - \left(\frac{u_i - l_i}{N}\right)\alpha_1, a_i + \left(\frac{u_i - l_i}{N}\right)\alpha_2, \ldots, class\right)$ could be an individual of the

```
Procedure HIDER(E, R)
  R := ∅
  n := |E|
  while |E| > n × epf
    r :=EvoAlg(E)
    R := R ⊕ {r}
    E := E − {e ∈ E | e ⊆ Δ_r}
  end while
end HIDER
Function EvoAlg(E)
  i := 0
  P_0 :=Initialize()
  Evaluation(P_0, i)
  while i < num_generations
    i := i + 1
    for j ∈ {1, ..., |P_{i−1}|}
      x̄ :=Selection(P_{i−1}, i, j)
      P_i := P_i+Recombination(x̄, P_{i−1}, i, j)
    end for
    Evaluation(P_i, i)
  end while
  return best_of(P_i)
end EvoAlg
```

Figure 12.8. Pseudocode of HIDER.

population where α_1 and α_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 the example). For discrete attributes, we ensure that the individual has the same active value as the example. The remainder binary values are randomly set to 0 or 1.

For example, let the database be the one used in the Figure 12.7. A possible individual for the initial population is obtained from a randomly selected example $e = (7.6, blue, 0)$. The individual could be $ind = (5.8, 8.2, 1, 0, 0, 1, 0, 0)$. The interval $[5.8, 8.2]$ is for the continuous attribute and the values $(1, 0, 0, 1, 0)$ is for the discrete one. Notice that the value *blue* is active and other value (*white*) has been randomly set to 1. The individual keeps the same class that of the example.

Sometimes, the examples very near to 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%), for continuous attributes.

The evolutionary module incorporates elitism: the best individual of every generation is replicated to the next one ($j = 1$, see in Figure 12.8 the loop

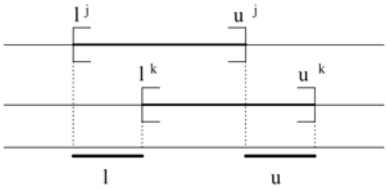

Figure 12.9. Crossover situation 1.

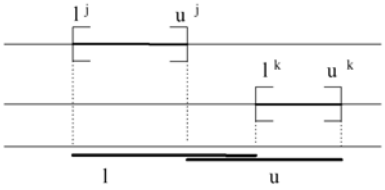

Figure 12.10. Crossover situation 2.

controlled by the variable j). A set of children (from $j = 2$ to $j = \frac{|P_{i-1}|}{2}$) is obtained from copies of randomly selected parents, generated by their fitness values and using the roulette wheel selection method. The remainder individuals (from $j = \frac{|P_{i-1}|}{2} + 1$ to $j = |P_{i-1}|$) are formed by means of crossovers (recombination). As half of the new population is created by applying the crossover operator, the probability of selecting an individual for crossing depends on its fitness value. These individuals could be mutated (recombination) later (only the individual from the elite will not be mutated). The evaluation function (evaluation) assigns a value of merit to each individual, which will be further used in the next generation.

Crossover

Wright's linear crossover operator [24] 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 [16] chooses values for an offspring by uniformly picking values between (inclusively) the two parents values. Eshelman and Schaffer [8] use a crossover operator that is a generalization of Radcliffe's which is called the blend crossover (BLX-α). It uniformly picks values that lie between two points that contain the two parents, but it 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 an extension of Radcliffes's to parents coded as intervals. Let $[l_i^j, u_i^j]$ and $[l_i^k, u_i^k]$ be the intervals of two parents j and k for the same attribute i. From these parents one children $[l, u]$ is generated by selecting values that satisfy the expression: $l \in [min(l_i^j, l_i^k), max(l_i^j, l_i^k)]$ and $u \in [min(u_i^j, u_i^k), max(u_i^j, u_i^k)]$. This type of crossover could produce two situations, which are illustrated in Figures 12.9 and 12.10. When the intersection of two intervals is not empty, as it is shown in Figure 12.9, the new interval $[l, u]$ is clearly obtained. However, a different situation is produced when the intersection is empty, because l could be greater than u. In this case, the offspring is rejected.

When the attribute is discrete, the crossover operator is like uniform crossover [18].

Mutation

Mutation is applied to continuous attributes as follows: if the randomly selected location (gen) is l_i or u_i, then a quantity is subtracted or added, depending on whether it is the lower or the upper bound, respectively (the quantity is currently the smaller Heterogeneous Overlap-Euclidean Metric (HOEM, [23]) between any two examples). In 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 nearly all values are 1. In this case, the attribute does not appear in the rule.

Mutation is always applied with probabilities 0.1 (individual) and $\frac{1}{na}$ (attribute), where na is the number of attributes. If the attribute is discrete, the probability of mutating a value is $\frac{1}{ndv}$, where ndv is the number of discrete values of that attribute.

12.3.3 Generalization

Databases used as training files do not have clearly differentiated areas, so that to obtain a totally coherent rule system (without error from the training file) involves a high number of rules. In [1] a system capable of producing a rule set exempt from error (with respect to the training file)is shown; however, sometimes it is interesting to reduce the number of rules in order to get a rule set that may be used like a comprehensible linguistic model. In this way, it could be better to have a system with fewer rules despite some errors than too many rules and no errors. When databases present a distribution of examples very hard to classify, it may be interesting to introduce the relaxing coefficient (RC) for understanding the behavior of databases by decreasing the number of rules [2]. RC indicates what percentage of examples within 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 its data. For example, if a database 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 might be very similar.

When an individual tries to expand and it always reaches examples of a different Class, its fitness value cannot be higher, unless a few errors were allowed. In this case, depending on the fitness function, such a value might increase. In Figure 12.11 (right) the individual cannot get bigger, unless one error is allowed, in which case the individual will have four new examples (left), increasing its fitness value.

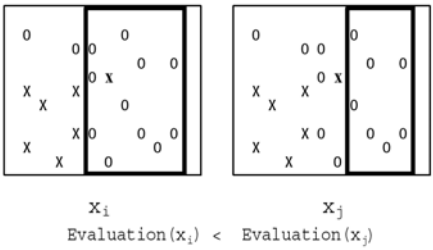

Figure 12.11. Relaxing coefficient.

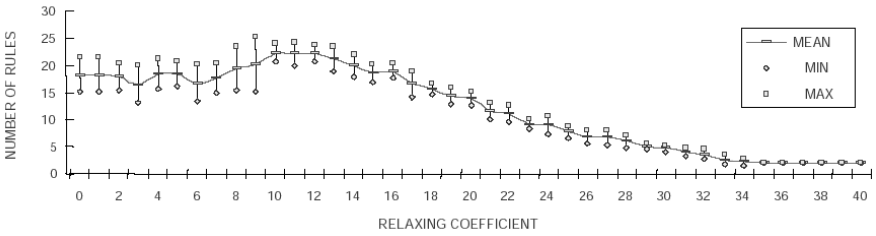

Figure 12.12. Number or rules varying *RC* for Pima database.

We have tested that the concept of the coefficient relaxation turns out to be useful when the number of rules is greater than expected to understand the information stored in the database.

We used Pima Indians Diabetes database to analyze the influence of the relaxing coefficient on both the error rate and the number of rules. This example showed that the error rate ranges from 18% to 34% depending on the relaxing coefficient (from 0 to 40) and therefore depending on the number of rules (from 26 to 2; see Figure 12.12). When $RC = 36$, HIDER produced only two rules for the Pima database and the error rate mean was about 30%. The lower error rate was achieved when $RC = 16$. All the experiments reported in the next tables were obtained using $RC = 0$.

12.3.4 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 databases.

The evolutionary algorithm maximizes the fitness function f for each individual. It is given by Eq. (12.1):

$$f(i) = 2(N - CE(i)) + G(i) + coverage(i) \tag{12.1}$$

where N is the number of examples being processed; $CE(i)$ is the class error, which is produced when the example i belongs to the region defined by the

rule but it does not have the same class; $G(i)$ 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. The reason why $f(i)$ is not $N - CE(i) + G(i) + coverage(i)$ is as follows: for example, when $CE(i) = 7$ and $G(i) = 9$, we will have the same fitness value as when $CE(i) = 15$ and $G(i) = 17$ (the difference is 2; assuming the same coverage for both). Therefore, we decided to penalty the second case ($\frac{9}{7}$ is greater than $\frac{17}{15}$) since fewer errors are preferred.

The coverage of a rule is calculated by dividing the volume of the region defined by the rule by the whole volume of the search space. Let $[l_i, u_i]$ be the interval associated with an attribute i of the rule; k_i the number of active discrete values of an attribute i; $[L_i, U_i]$ the range of a continuous attribute i, and $|A_i|$ the number of different values of a discrete attribute i. Then, the coverage of a rule is given by

$$coverage(x) = \prod_{i=1}^{m} \frac{coverage(x,i)}{range(x,i)},$$

where

$$coverage(x,i) = \begin{cases} u_i - l_i & \text{if the attribute } i \text{ is continuous,} \\ k_i & \text{if it is discrete,} \end{cases}$$

and

$$range(x,i) = \begin{cases} U_i - L_i & \text{if the attribute } i \text{ is continuous,} \\ |A_i| & \text{if it is discrete.} \end{cases}$$

12.4 Results

The experiments described in this section are from the UCI Repository [5]. The results obtained by HIDER have been compared to that of C4.5 Release 8 and C4.5Rules. To measure the performance of each method, a 10-fold cross-validation was achieved with each dataset (18 databases that involve continuous and/or discrete attributes). The algorithms were all run on the same training data and their induced knowledge structures tested using the same test data, so that the 10 resulting performance numbers for C4.5Rules, C4.5, and HIDER are comparable. It is very important to note that the experiments were run with the same default settings for all parameters of the EA: a population size of as little as 100 individuals and 300 generations. In cases of small data sets, like Iris, the results would have been the same using a smaller number of generations (it had been enough around 50). There are very small numbers considering the number of examples and the dimensionality of some databases. HIDER needed about 8 hours to complete the 10-fold cross-validation for the 18 databases in a Pentium 400Mhz with 64 Mb of RAM.

However, C4.5 only needed about 8 minutes in the same machine. C4.5 is an extremely robust algorithm that performs well on many domains. It is very difficult to consistently outperform C4.5 on a variety of data sets.

Table 12.1 gives the values of the parameters involved in the evolutionary process. The results of the trials appear in Tables 12.2, 12.3, 12.4, and 12.5.

Table 12.1. Parameters of HIDER

Parameter	Value
Population size	100
Generations	300
Crossover probability	0.5
Individual mutation probability	0.2
Gen mutation probability	0.1

Table 12.2. Comparing Error Rates

Database	C4.5Rules		C4.5		HIDER
Bupa	34.5	−	34.7	−	35.7
Breast-C (Wisc)	5.2	+	6.28	+	4.3
Cleveland	25.8	+	26.8	+	20.5
German^{5de10}	28.8	−	32.1	+	29.1
Glass^{2de10}	18.5	−	32.7	+	29.4
Heart	20.7	−	21.8	−	22.3
Hepatitis^{8de10}	16.9	−	21.4	+	19.4
Horse Colic	17.5	−	19.0	+	17.6
Iris	4.0	+	4.77	+	3.3
Lenses	16.7	−	29.9	+	25.0
Mushroom	0.0	−	0.0	−	0.8
Pima	26.2	+	32.1	+	25.9
Sonar^{9de10}	29.3	−	30.3	−	43.1
Tic-Tac-Toe	18.8	+	14.2	+	3.8
Vehicle^{8de10}	57.6	+	30.6	+	30.6
Vote	5.3	−	6.2	−	6.4
Wine	6.7	+	6.7	+	3.9
Zoo	29.8	+	7.0	−	8.0
Average	20.1		19.8		18.3

Table 12.2 gives the error rates (numbers of misclassified examples expressed as a percentage) for the C4.5Rules, C4.5, and HIDER algorithms on the selected domains. HIDER outperforms C4.5 and C4.5Rules in 12 out of 18 and 8 out 18 datasets, respectively. If C4.5 produces bad trees, the results from C4.5Rules will not be very good. We can observe that there are four databases whose results generated by C4.5 are about 40% worse than those obtained by

Table 12.3. Comparing Number of Rules

Database	C4.5Rules	C4.5	HIDER
Bupa	14.0 +	28.6 +	11.3
Breast-C (Wisc)	8.1 +	21.9 +	2.6
Cleveland	11.3 +	35.2 +	7.9
German	5.2 −	181.5 +	13.3
Glass	14.0 −	29.0 +	19.0
Heart	10.5 +	29.2 +	9.2
Hepatitis	5.4 +	13.8 +	4.5
Horse Colic	4.1 −	39.3 +	6.0
Iris	4.0 −	5.5 +	4.8
Lenses	3.1 −	4.1 −	6.5
Mushroom	17.2 +	15.7 +	3.1
Pima	9.8 −	93.6 +	16.6
Sonar	5.1 +	16.8 +	2.8
Tic-Tac-Toe	10.7 −	93.9 +	11.9
Vehicle	3.3 −	102.3 +	36.2
Vote	6.6 +	14.7 +	4.0
Wine	4.6 +	5.4 +	3.3
Zoo	5.3 −	9.9 +	7.2
Average	7.9	41.1	9.5

Table 12.4. Comparing Global Results (C4.5/HIDER)

Database	ϵ_{er}	ϵ_{nr}
Bupa	.97	2.53
Breast-C (Wisc)	1.46	8.42
Cleveland	1.31	4.46
German	1.10	13.65
Glass	1.11	1.53
Heart	.98	3.17
Hepatitis	1.10	3.07
Horse Colic	1.08	6.55
Iris	1.40	1.15
Lenses	1.20	.63
Mushroom	.01	5.00
Pima	1.24	5.64
Sonar	.70	6.00
Tic-Tac-Toe	3.69	7.89
Vehicle	1.00	2.83
Vote	.96	3.68
Wine	1.70	1.64
Zoo	.88	1.38
Average	1.22	4.40

Table 12.5. Comparing Global Results (C4.5Rules/HIDER)

Database	ϵ_{er}	ϵ_{nr}
Bupa	.97	1.24
Breast-C (Wisc)	1.21	3.12
Cleveland	1.26	1.43
German	.99	.39
Glass	.63	.74
Heart	.93	1.14
Hepatitis	.87	1.20
Horse Colic	.99	.68
Iris	1.21	.83
Lenses	.67	.48
Mushroom	.01	5.55
Pima	1.01	.59
Sonar	.68	1.82
Tic-Tac-Toe	4.95	.90
Vehicle	1.88	.09
Vote	.83	1.65
Wine	1.72	1.39
Zoo	3.72	.74
Average	1.36	1.33

HIDER (Breast Cancer, Iris, Tic-Tac-Toe and Wine). It is especially worthy the error rate of the Tic-Tac-Toe database. C4.5Rules improves the results of C4.5 for nearly all databases, except three of them (Tic-Tac-Toe, Vehicle and Zoo). C4.5Rules did not achieve to improve those results generated by C4.5, quite the opposite, made results worse, particularly for Tic–Tac–Toe and Zoo databases. As catalogued in the last row of Table 12.2, HIDER is on average better than the others. In Table 12.4 these results will be analyzed by means of the measure (*ratio*) used in the Quinlan's works [15].

Table 12.3 compares the number of rules generated by the three 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. C4.5Rules improves C4.5 in all databases, except Mushrooms. These results are very similar to those generated by HIDER. Nevertheless, although the result for German database is very interesting (5.2 rules), for others databases C4.5Rules reduces the number of rules too much (3.3 rules for Vehicle and 5.3 rules for Zoo), leading to a high error rate (57.6% for Vehicle and 29.8% for Zoo). Due to that reason, although C4.5Rules on average generated fewer rules (7.9) than HIDER (9.5), the error rate increased: C4.5Rules (20.1%) and HIDER (18.3%).

Table 12.4 shows a measure of improvement (ϵ) for the error rate [second and fourth columns: (ϵ_{er})] and the number of rules [third and fifth columns: (ϵ_{nr})]. To calculate those coefficients (ϵ_{er} and ϵ_{nr}, respectively) the error rate (number of rules) for C4.5 (or C4.5Rules) has been divided by the corre-

sponding error rate (number of rules) for HIDER. 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. It is worth noting that applying HIDER, more than two thirds of the databases produce less than half the rules. C4.5 only was better with the Lenses database. C4.5 made the error rate better for six databases, although only three of them improved significantly (Mushrooms, Sonar and Zoo). In summary, the averaged error rate generated by C4.5 is 22% greater and the averaged number of rules 340%. This reason leads to us to make a comparison with C4.5Rules, mainly in regard to the number of rules. The average ratio of the error rate of C4.5 to that of HIDER is 1.22, while the ratio of the number of rules is 4.40. Although the results in Table 12.3 indicated that C4.5Rules improved on average (7.9 rules) to HIDER (9.5 rules), analyzing the relative increase of the number of rules, we can observe that those numbers can be deceptive. C4.5Rules generates an averaged number of rules 33% greater (fourth column), as well as an averaged error rate 36% higher (fifth column), as it is shown in the last row of Table 12.5.

As the overall averages at the bottom of the tables indicate, HIDER is more accurate than C4.5, and C4.5 is more accurate than C4.5Rules. HIDER produces fewer rules than C4.5Rules, which also generates fewer than C4.5.

12.5 Conclusions

An EA-based supervised learning tool to classify databases is presented in this paper. HIDER produces a hierarchical set of rules, where each rule is tried in order until one is found whose conditions are satisfied by the example being classified. The use of hierarchical decision rules led to an overall improvement of the performance on the 18 databases investigated here. In addition, HIDER improves the flexibility to construct a classifier varying the relaxing coefficient. In other words, one can trade off accuracy against understanding. HIDER was compared to C4.5 and C4.5Rules and the number of rules as well as the error rate were decreased. To summarize shortly, the experiments show that HIDER works well in practice.

12.6 Future Works

Evolutionary algorithms are very time-consuming. This aspect is being analyzed from the viewpoint of the coding. We are designing a new type of coding that uses natural numbers for both continuous and discrete attributes, so as the specific genetic operators. This encoding method allows us to reduce the dimensionality of the search space so that the algorithm might converge more quickly.

Another aspect being studied is the way in which the evaluation function analyzes each example from the database. Normally, the information is loaded in a vector, and for each individual of the population the whole vector is processed. Research on improvements to the data structure as input of EAs in order to reduce the time complexity is currently being conducted.

12.7 Acknowledgment

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

References

1. J. S. Aguilar, J. C. Riquelme, and M. Toro (1998) Decision queue classifier for supervised learning using rotated hyperboxes. In *Progress in Artificial Intelligence IBERAMIA'98. Lecture Notes in Artificial Intelligence 1484. Springer-Verlag*, pages 326–336.
2. J. S. Aguilar, J. C. Riquelme, and M. Toro (1998) A tool to obtain a hierarchical qualitative set of rules from quantitative data. In *Lecture Notes in Artificial Intelligence 1415. Springer-Verlag*, pages 336–346.
3. J. Antonisse (1989) A new interpretation of schema notation that overturns the binary encoding constraint. In *Third International Conference on Genetic Algorithms*, pages 86–97. Morgan Kaufmann.
4. S. Bhattacharyya and G.J. Koehler (1994) An analysis of non-binary genetic algorithms with cardinality 2^v. *Complex Systems*, 8:227–256.
5. C. Blake and E. K. Merz (1998) UCI repository of machine learning databases, 1998.
6. P. Clark and R. Boswell (1991) Rule induction with cn2: Some recents improvements. In *Machine Learning: Proceedings of the Fifth European Conference (EWSL-91)*, pages 151–163.
7. K. A. DeJong, W. M. Spears, and D. F. Gordon (1993) Using genetic algorithms for concept learning. *Machine Learning*, 1(13):161–188.
8. L. J. Eshelman and J. D. Schaffer (1993) Real-coded genetic algorithms and interval-schemata. *Foundations of Genetic Algorithms-2*, pages 187–202.
9. D. E. Goldberg (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
10. R. C. Holte (1993) Very simple classification rules perform well on most commonly used datasets. *Machine learning*, 11:63–91.
11. C. Z. Janikow (1993) A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 1(13):169–228.
12. G.J. Koehler, S. Bhattacharyya, and M.D. Vose (1998) General cardinality genetic algorithms. *Evolutionary Computation*, 5(4):439–459.
13. Z. Michalewicz (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3 ed..
14. T. Mitchell (1997) *Machine Learning*. McGraw Hill.

15. J. R. Quinlan (1996) Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90.
16. N. J. Radcliffe (1990) *Genetic Neural Networks on MIMD Computers*. Ph. d., University of Edinburgh.
17. R. L. Rivest (1987) Learning decision lists. *Machine Learning*, 1(2):229–246.
18. G. Syswerda (1989) Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9.
19. G. Venturini (1993) Sia: a supervised inductive algorithm with genetic search for learning attributes based concepts. In *Proceedings of European Conference on Machine Learning*, pages 281–296.
20. M.D. Vose and A.H. Wright (1998) The simple genetic algorithm and the walsh transform: Part i, theory. *Evolutionary Computation*, 6(3):253–273.
21. M.D. Vose and A.H. Wright (1998) The simple genetic algorithm and the walsh transform: Part ii, the inverse. *Evolutionary Computation*, 6(3):275–289.
22. D. Whitley (1993) A genetic algorithm tutorial. Technical Report CS-93-103, Colorado State University, Fort Collins, CO 80523.
23. D. R. Wilson and T. R. Martinez (1997) Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6(1):1–34.
24. A. H. Wright (1991) Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms-1*, pages 205–218.