DISCOVERING HIERARCHICAL DECISION RULES WITH EVOLUTIVE ALGORITHMS IN SUPERVISED LEARNING

José C. Riquelme, Jesús S. Aguilar and Miguel Toro

Departamento de Lenguajes y Sistemas Informáticos.
Facultad de Informática y Estadística.
Avenida Reina Mercedes s/n
41012 Sevilla
Spain
e-mail: riquelme@lsi.us.es, aguilar@lsi.us.es, miguel.toro@lsi.us.es

Accepted in the final form: 15 November, 2000

Abstract. This paper describes a new approach, HIDER (**HI**erarchical **DE**cision **R**ules), for learning rules in continuous and discrete domains based on evolutive algorithms. The algorithm produces a hierarchical set of rules, that is, the rules must be applied in a specific order. With this policy, the number of rules may be reduced because the rules could be one inside of another. The evolutive algorithm uses both real and binary codification for the individuals of the population and introduces several new genetic operators. In addition, this paper discusses the capability of learning systems based on an evolutive algorithm to reduce both the number of rules and the number of attributes involved in the rule set. 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 an important improvement.

Keywords: Evolutive Algorithms, Supervised learning, Decision Lists.

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 data structure. 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. This sort of learning is called "lazy learning" because the algorithm does not generate a model of knowledge for 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.

The algorithm is ran once to produce the set of rules that classify many new instances. Other techniques as nearest neighbour classifiers need one execution every time we want to predict the class of an unseen new example. Therefore, the difference between the techniques that can produce knowledge and those that cannot is very important from the point of view of the number of executions.

Supervised learning algorithms tend to emulate the human behaviour, since from input conditions try to predict the action to be taken, based upon experience with similar situations. Such experience

is collected in a database and knowledge is inferred by means of heuristics or algorithms.

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 [1], 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 trees may be called axis-parallel, because the tests at each node are equivalent to axis-parallel hyperplanes in the space. On the contrary, other techniques build oblique decision trees, as $OC1$ [2], that tests a linear combination of the internal attributes at each node, for 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 [3]. Both methods use hill-climbing, that is, the algorithm never backtracks; therefore, it could be converging to locally optimal solutions that are not globally optimal.

Simpson [4] introduced the idea of using hyperrectangles to cluster or classify spatial data. Each hyperrectangle is viewed as a fuzzy cluster, a fuzzy set in which all of the elements within the hyperrectangle have membership 1, and examples outside the hyperrectangle can have a positive membership in the set depending on a fuzzy membership function. Simpson used a deterministic procedure to place and appropriately size hyperrectangles to describe data.

Genetic Algorithms (GA) are a family of computational models inspired by evolution. These algorithms employ a randomized search method to find solutions to a particular problem [5]. 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 [6]. 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 [7, 8] including numerical optimization and combinatorial optimization problems, although the range of problems to which GAs have been applied is quite broad. The main tasks in applying GAs to any problem are 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 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 is within the population in each generation.

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. In previous works, we presented a system to classify databases using binary coding [9]; afterwards, we adopted real codification to handle efficiently continuous domains and axis-parallel representations. (In addition, we explored other representations such as rotated hyperrectangles and hyperellipses). Then, new genetic operators were introduced for real codification. This method is conceptually nearer to evolutionary algorithms (EA) than GA. This new system used an EA to search the best solutions and produced a hierarchical set of rules. The hierarchy follows that an example will be classified by the i^{th}-rule if it does not match the conditions of the $(i-1)^{th}$ precedent rules. The rules are sequentially obtained until the space is totally covered.The behavior is similar to a *decision list* [10]. A decision list DL is a list of pairs $(f_1, v_1), \ldots, (f_r, v_r)$ where each f_j is a term in C_k^n, each v_i is a value in $\{0, 1\}$, and the last function f_r is the constant function *true*. (C_k^n denotes the set of all terms (conjunctions) of size at most k with literals drawn from $L_n = \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$, therefore,

$$|C_k^n| = \sum_{i=0}^{k} \left(\begin{array}{c} 2n \\ i \end{array} \right) \tag{1}$$

If conditions Then class
Else If conditions Then class
Else If conditions Then class
.....
Else "unknown class"

Figure 1: Hierarchical set of rules.

A decision list DL defines a boolean function as follows: for any assignment $x \in X_n$, $DL(x)$ is defined to be equal to v_j where j is the least index such that $f_j(x) = 1$ (such an item always exists, since the last function is always true).

It is very important to note that decision rules are not the same thing as decision lists. The concept of decision rules is more general than decision lists, because a decision list is a linearly ordered set of decision rules [10].

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).

We let $\mathcal{C}$ denote the set of classes (labels) of a database. For each continuous (real) attribute a_i we obtain the boundary values, called l_i and u_i (lower and upper boundaries, respectively) which define the space R_i (range of the attribute i). Let H_i be an interval $[l, u]$ where $l < u$ and $l, u \in R_i$, for the attribute i. Then, an *extended decision list* EDL is a list of pairs

$$(f_1, v_1), \ldots, (f_r, v_r) \tag{2}$$

where each f_i is a function in $H_1 \times H_2 \times \cdots H_m$, v_i is a value in $\mathcal{C}$ and m is the number of attributes.

Our EDL does not have the last constant function true as DL has. 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. The structure of the set of rules will be as shown in figure 1.

Decision list policy is applied in order to reduce the number of rules. For an artificial two-dimensional database, figure 2 shows the classification that C4.5 gives. Nevertheless, as illustrated in figure 3, rules inside of another one could improve the quality of the rule set.

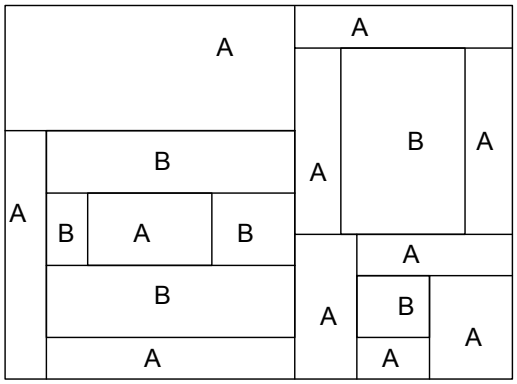

Figure 2: C4.5

The most evident feature, graphically observed in figure 3, is the reduction of the number of rules because of the rules overlapping.

As mentioned in [11] one of the primary motivations for using real-coded EAs is the precision to represent attributes values and another is the ability to exploit the gradualness of functions of

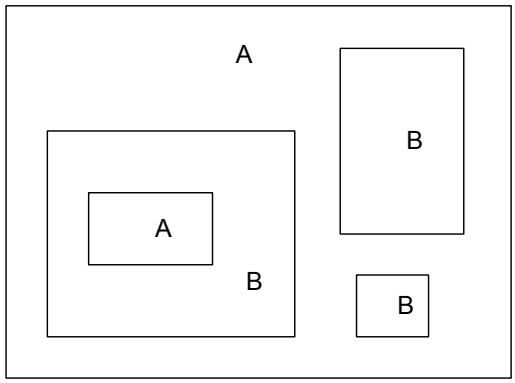

Figure 3: HIDER

continuous attributes. We implemented our first versions with binary-coded GAs, but we could prove that real-coded EAs are more efficient (time and quality of results). In addition, HIDER reduces the number of attributes involved in a rule, making its comprehension by humans easier.

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 of 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 [5].

In first approaches, we studied several GA-based classifier [12, 13] with binary coding. These are generally used as concept learners, which coding assigns a bit to each value of the attribute, i.e., every attribute is symbolic ($GABIL$ and GIL are two very 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 infinity.

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. To apply the evaluation function is necessary to convert the binary to real encoding. This process increases the computation time by a factor of m (number of attributes).

Moreover, when we convert binary to real, we are loosing precision. For that reason, 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 least significant bit of an attribute will not include or exclude more than one example from the training set in a single step. Let l_i and u_i be the lower and upper bounds of

an attribute. Let m_i be the least absolute difference of any two values of the attribute i. The allowed *error* for this attribute must be less than m_i. Then, the length of an attribute would be as follows:

$$L_i = \left\lceil \log_2 \left(1 + \frac{u_i - l_i}{error_i} \right) \right\rceil \tag{3}$$

However, in this paper, to eliminate the problem we use the real codification. This implies to redefine the genetic operators for it as we see below.

The representation for continuous and discrete attributes is best explained by referring to figure 4, 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. A last value (omitted) is for the class.

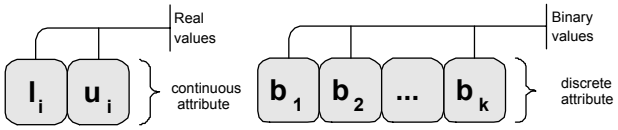

Figure 4: Continuous (left) and discrete (right) attributes.

The number of classes determines the set of values to which an example belongs, i.e., if there are five classes, the value of a class will belong to the set $\{0, 1, 2, 3, 4\}$. 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 defined that value in the interval. For example, in the first case the rule would be $[-, v]$ (the left value of the interval is equal to the minimum value of the range of the attribute) and in the second one $[v, -]$ (the right value of the interval is equal to the maximum value of the range of the attribute). If both values are equal to the boundaries then the rule appears $[-, -]$ for that attribute, which means that it is not relevant. Under these assumptions, some continuous attributes may not appear in the rule set. In addition, when every discrete value is active, that attribute does not appear in the rule.

2.2 Algorithm

The algorithm is a typical sequential covering GA [14]. It chooses the best individual of the evolutionary process, transforming the individual into a rule, which is used to eliminate data from the training file [15]. In this way, the training file is reduced for the following iteration. A termination criterion could be reached when more examples to cover do not exist. The method of generating the initial population consists of randomly selecting an example from the training file for every individual of the population. Then, an interval to which the example belongs is obtained by adding and subtracting a random quantity from the values of the example.

Sometimes, the examples very near to the boundaries are hard to cover during the evolutionary process. To resolve this problem, the search space is increased (actually, the lower bound is decreased by 5%, and the upper bound is increased by 5%). For example in one dimension, let a and b be the lower and upper bounds of the attribute; then, the range of the attribute is $b - a$; next, we randomly choose an example $(x_1, class)$ from the training file; for last, a possible individual of the population could thus be $(x_1 - range \times k_1, x_1 + range \times k_2, class)$ where k_1 and k_2 are random values belonging to $[0, \frac{range}{N}]$ (N is the size of the training data, and class is the same of that of the example). For discrete attributes, this is not a problem because the individual has the same active values as the example. 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. These individuals could be mutated later (only the individual from the elite will not be mutated). The remainder is formed through crossovers. Afterwards, mutation is applied depending on a probability.

An overview of the EA-based classifier is shown in the figure 5.

```
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
    Step 2.3. Replication
    Step 2.4. Crossover and Mutation
  Step 3. Put the best one in Decision List
  Step 4. Eliminate examples covered by best
```

Figure 5: Pseudocode.

Wright's linear crossover operator [16] 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 [17] chooses values for an offspring by uniformly picking values between (inclusively) the two parents values. Eshelman and Schaffer 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, and sometimes the value is perturbed to approximate it to the boundary. Let $[l_1, u_1]$ and $[l_2, u_2]$ be the intervals of two parents for the same attribute. In the figure 6 the parents are in the first two segments. 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 to two parents and let L and U be the boundaries of the attribute being treated. We have four possibilities (the percentages of application are to the right):

$l \in [l_1, l_2]$	$u \in [u_1, u_2]$	85%
$l \in [L, min(l_1, l_2)]$	$u \in [max(u_1, u_2), U]$	5%
$l \in [l_1, l_2]$	$u \in [max(u_1, u_2), U]$	5%
$l \in [L, min(l_1, l_2)]$	$u \in [u_1, u_2]$	5%

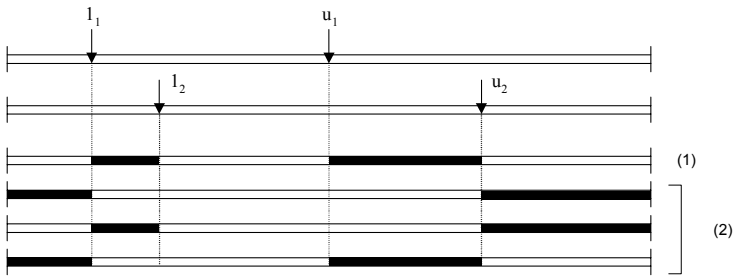

Figure 6: Axis-Parallel crossover operator.

Mutation is applied in two different ways: if the randomly chosen location corresponds to a value of the interval, then a quantity is subtracted or added, depending on whether it is the lower or the upper bound, respectively (the quantity actually is the lower Euclidean distance between any two examples); if the location corresponds to the class, a new value is randomly generated.

When the attribute is discrete, the crossover operator is like uniform crossover [18] and mutation 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 figure 10, the attribute *sex* is not in the rule $R1$.

2.3 Relaxing coefficient

Databases used as training files do not have clearly differentiated areas. Therefore, to obtain a rule system totally coherent (without error from the training file) involves a high number of rules. We showed in previous papers [19] a system capable of producing a rule set exempt from error; 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 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 [20]. RC indicates what percentage of examples inside of a rule can have a different class than what 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 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 might be the same as before.

2.4 Fitness function

The fitness function must be able to discriminate between correct and incorrect classification of examples. Finding an appropriate function is not a trivial task, due to the noisy nature of most databases.

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

$$f(i) = 2(N - CE(i)) + G(i) + Coverage(i)$$

where the rule *coverage* is the value of the side of a k-dimensional hypercube which volume is equivalent to the volume of the k-dimensional region covered by the rule; $CE(i)$ is the class error, which are produced when the example i belongs to the region defined by the rule, but does not have the same class; $G(i)$ is the number of goals of the rule. Every rule can be quickly expanded for finding more examples due to the rule 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 $\frac{17}{15}$).

3 Application

The experiments described in this section are from the UCI repository [21]. To measure the performance of the method, a 10-fold cross validation was realized 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. These are very low numbers considering the number the examples and the dimensionality of some databases. HIDER needed about 8 hours to complete the 10-fold cross validation for the 18 databases on a Pentium 400Mhz with 64Mb of RAM. However, C4.5 only needed about 8 minutes on 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 datasets. Thus, improving C4.5 should yield an interesting learning algorithm.

The results of these trials appear in tables 1, 2 and 3. Table 1 gives a 10-fold cross validation of the error rates for the C4.5 and HIDER algorithms on the selected domains.

Table 2 compares the number of rules generated by the two approaches. 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 very much lower.

Table 1: Comparing error rates.

Database	C4.5R8	HIDER
Bupa	34.73	35.71
Breast Cancer	6.28	4.29
Cleveland	26.77	20.49
German	32.1	29.1
Glass	32.73	29.41
Heart	21.83	22.32
Hepatitis	21.42	19.41
Horse Colic	19.0	17.64
Iris	4.67	3.33
Lenses	29.99	25.0
Mushroom	0.01	0.76
Pima	32.06	25.9
Sonar	30.31	43.07
Tic-Tac-Toe	14.2	3.85
Vehicle	30.6	30.6
Vote	6.19	6.42
Wine	6.71	3.95
Zoo	7.0	8.0
Average	19.81	18.29

The set of rules generated for the Wine database is presented in figures 7 and 8. The exact same folds were used for both algorithms so that the ten resulting performance numbers for HIDER and C4.5 are pairwaise comparable. HIDER produced an error rate of 0%, however, that of C4.5 was 22.2%.

```
c12 <= 2.15 :
|   c4 <= 17.5 : 2 (6.0)
|   c4 > 17.5 : 3 (45.0/2.0)
c12 > 2.15 :
|   c13 <= 725 : 2 (54.0/1.0)
|   c13 > 725 :
|   |c10 <= 3.4 : 2 (4.0)
|   |   c10 > 3.4 : 1 (51.0)
```

Figure 7: Decision Tree generated by C4.5 for Wine database.

Table 3 shows a measure of improvement (ϵ) for the error rate (first column: (ϵ_{er})) and the number of rules (second column: (ϵ_{nr})). To calculate those coefficient (ϵ_{er} and ϵ_{nr}, respectively) the error rate (number of rules) of C4.5 has been divided by the error rate (number of rules) of HIDER. The last row contains the average of every column. On average, HIDER found solutions that had less than half 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.

Figures 9 and 10 illustrate an example a little more complex (Hepatitis database), which shows that when the number of rules is large, the number of attributes involved in the rule set is also reduced. Thus, in the example, C4.5 uses 63 conditions and HIDER uses 30 conditions.

Morover, the error rate was 31.2% for C4.5, in contrast with 12.5% for HIDER (using the same fold).

Table 2: Comparing number of rules.

Database	C4.5R8	HIDER
Bupa	28.6	11.3
Breast Cancer	21.9	2.6
Cleveland	35.2	7.9
German	181.5	13.3
Glass	29.0	19.0
Heart	29.2	9.2
Hepatitis	13.8	4.5
Horse Colic	39.3	6.0
Iris	5.5	4.8
Lenses	4.1	6.5
Mushroom	15.5	3.1
Pima	93.6	16.6
Sonar	16.8	2.8
Tic-Tac-Toe	93.9	11.9
Vehicle	102.3	36.2
Vote	14.7	4.0
Wine	5.4	3.3
Zoo	9.9	7.2
Average	41.12	9.46

```
IF R1: c7 [1.08,-] and
       c10 [3.82,-] and
       c13 [741.80,-] : 1(51|0)
ELSE IF R2: c7 [1.14,-] and
            c11 [0.68,-] and
            c12 [1.61,-] : 2(64|1)
ELSE IF R3: c4 [11.57,-] : 3(43|0) ELSE unknown
```

Figure 8: Hierarchical Decision List generated by HIDER for Wine database.

4 Conclusions

A supervised learning tool to classify databases is presented in this paper. It produces a hierarchical set of decision rules where the conditions of each rule indicate if an example belongs to a region. The number of rules is reduced with regard to other systems, like C4.5, and improves the flexibility to construct a classifier varying the relaxing coefficient. We also have explored several types of crossover and mutation operators. Finally, real-coded genetic algorithms are more efficient finding rule sets than binary-coded ones on supervised learning with continuous domains. The tables show how effective HIDER is, particularly, with respect to the number of rules. 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 produces. In addition, the number of attributes involves in the rule set is reduced too. Therefore, HIDER would be considered an approach of great quality.

Acknowledgements

The research was supported by the Spanish research agency CICYT under grant TIC99-0351.

Table 3: Comparing global results.

Database	ϵ_{er}	ϵ_{nr}
Bupa	0.97	2.53
Breast Cancer	1.46	8.42
Cleveland	1.31	4.46
German	1.10	13.65
Glass	1.11	1.53
Heart	0.98	3.17
Hepatitis	1.10	3.07
Horse Colic	1.08	6.55
Iris	1.4	1.15
Lenses	1.20	0.63
Mushroom	0.01	5.00
Pima	1.24	5.64
Sonar	0.70	6.00
Tic-Tac-Toe	3.69	7.89
Vehicle	1.00	2.83
Vote	0.96	3.68
Wine	1.70	1.64
Zoo	0.88	1.38
Average	1.22	4.40

References

[1] J. R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, San Mateo, California, 1993.

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

[3] A. Blum and 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.

[4] P. K. Simpson, "Fuzzy min-max neural network," *IEEE Transactions on Neural Networks*, vol. 3, pp. 776–786, 1992.

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

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

[7] S. Forrest, "Genetic algorithms," *ACM Computer Surveys*, vol. 28, no. 1, pp. 77–80, 1996.

[8] Z. Michalewicz "Genetic Algorithms + Data Structures = Evolution Programs (Third edition)," Springer-Verlag, 1996.

[9] J. Aguilar, J. Riquelme, and M. Toro, "A tool to obtain a hierarchical qualitative set of rules from quantitative data," in *Lectures Notes in Artificial Intelligence. Springer-Verlag*, 1998, pp. 336–346.

[10] R. L. Rivest, "Learning decision lists," *Machine Learning*, vol. 1, no. 2, pp. 229–246, 1987.

```
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)
```

Figure 9: Decision Tree generated by C4.5 for Hepatitis database.

[11] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," *Foundations of Genetic Algorithms-2*, pp. 187–202, 1993.

[12] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Machine Learning*, vol. 1, no. 13, pp. 169–228, 1993.

[13] K. A. De Jong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Machine Learning*, vol. 1, no. 13, pp. 161–188, 1993.

[14] T. Mitchell, *Machine Learning*, McGraw Hill, 1997.

[15] 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.

[16] A. H. Wright, "Genetic algorithms for real parameter optimization," *Foundations of Genetic Algorithms-1*, pp. 205–218, 1991.

[17] N. J. Radcliffe, *Genetic Neural Networks on MIMD Computers*, Ph.d., University of Edinburgh, 1990.

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

[19] J. Aguilar, J. Riquelme, and M. Toro, "Decision queue classifier for supervised learning using rotated hyperboxes," in *Progress in Artificial Intelligence IBERAMIA'98. Lectures Notes in Artificial Intelligence 1484. Springer-Verlag*, 1998, pp. 326–336.

```
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
```

Figure 10: Hierarchical Decision List generated by HIDER for Hepatitis database.

[20] J. Riquelme and J. Aguilar, "A ga-based tool to obtain a hierarchical classifier for supervised learning (in spanish)," *Revista Iberoamericana de Inteligencia Artificial*, vol. 1, no. 5, pp. 38–43, 1998.

[21] C. L. Blake and C. J. Merz, "UCI repository of machine learning databases," "www.ics.uci.edu/mlearn/MLRepository.html, 1998."