A Decision Queue based on Genetic Algorithms: Axis-Parallel Classifier versus Rotated Hyperboxes

JOSÉ RIQUELME, JESÚS AGUILAR Y MIGUEL TORO
Departamento de Lenguajes y Sistemas Informáticos.
Facultad de Informática y Estadística. Universidad de Sevilla.
Avenida Reina Mercedes s/n. Sevilla 41012
SPAIN

Abstract : This paper describes a system for learning rules using axis-parallel or rotated hyperboxes as individuals of a genetic algorithm (GA). Our method attempts to find out hyperboxes: aligned with the coordinate axes, such C4.5; or at any orientation by combining deterministic hill-climbing with GA. The system uses a decision queue (DQ) as method of representing the rule set. It means that the obtained rules must be applied in specific order, that is, an example will be classified by the i-rule only if it doesn't satisfy the condition part of the i-1 previous rules. With this policy, the number of rules may be reduced because the rules could be one inside of another one. We have tested our system on real data from UCI repository. We have also designed some two-dimensional artificial databases to graphically represent the experiments. The results are summarized in the last section.

Key-Words :-data mining, supervised learning, genetic algorithms. *CSCC'99* Proceedings, Pages:4821-4827

1 Introduction

Supervised learning is used when the data samples have known outcomes that the user wants to predict. This type of learning is the more common form because data are usually collected with some outcome in mind. Human problem solving is normally an exercise in studying input conditions to predict a result based upon previous experience with similar situations. SL algorithms tend to emulate that sort of human behavior.

Decision trees (DT) are a particularly useful tool in the context of machine learning techniques because they perform classification by a sequence of simple, easy-to-understand tests whose semantics is intuitively clear to domain experts. Some techniques, like C4.5, construct decision trees selecting the best attribute by using a statistical test to determine how well it alone classifies the training examples [10]. This class of DTs may be called axis-parallel, because the tests at each node are equivalent to axis-parallel hyperplanes in the space. Others techniques build oblique decision trees (ODT), as OC1[9], 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 axes.

To find out the smallest DT (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 [13] introduced the idea of using hyperboxes to cluster or classify spatial data. Each hyperbox is viewed as a fuzzy cluster, a fuzzy set in which all of the elements within the hyperbox have membership 1.0 for being in that set, and elements outside the hyperbox can have a positive membership in the set depending on a fuzzy membership rule for that set. Simpson used a deterministic procedure to place and appropriately size hyperboxes to describe data. Hyperboxes were created and sized by considering the data in an ordered sequence. A hyperbox was placed around preliminary data. As subsequent data was added, either the present hyperbox was expanded to include the new data, or a new hyperbox was added and the process continued. This procedure was of limited efficacy because it required trial-and-error setting of operator parameters and the final solution depended on the order of presentation of the data, even when the data possessed only spatial and not sequential characteristics.

Genetic algorithms (GA) employ a randomized search method to seed a maximally fit hypothesis [4, 5].

In previous works [1,2,11], we presented a system to classify databases by using hyperboxes (axis-parallel). This system used a GA to search the best solutions and produced a hierarchical set of rules. The hierarchy follows that an example will be classified by the i-rule if it does not satisfy the conditions of the i-1 precedent rules. The rules are sequentially obtained until the space is totally covered. The behavior is similar to a queue, for that reason we have given the name decision queue (DQ) to the produced rule set. This concept is based on the k-DL, the set of decision lists with conjunctive clauses of size at most k at each decision [12]. A decision list is a list L of pairs

$$(f_1, v_1),...,(f_r, v_r)$$

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.

A decision list L defines a boolean function as follows: for any assignment $x \in X_n$, L(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).

Furthermore, DQ does not have the last constant function true. However, we could interpret that last function as an unknown function, that is, we do not know to which class the example belongs to. Therefore, it may be advisable to say "unknown class" instead of making an erroneous decision.

DQ is based on DL. Really, DQ is a DL-generalization because it permits codifying functions f_i of continuous attributes and the values v_i can belong to any set.

In the sense mentioned above, our system has a measure, called unknowledge, to indicate how many test examples have no associated class. The domain expert can choose the number of rules or the allowed error rate (relaxing coefficient), thus avoiding some unnecessary mistakes if the rule does not assign to the test example a class. Incrementing the relaxing coefficient the unknowledge will be less, but the number of misclassified examples will be higher. The expert, based on experimentation, must determine such parameter.

DQ presents the following structure:

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

We show in figure 1 an example, in which rotated hyperboxes can find out better solutions than axis-parallel hyperboxes can. Decision queue policy is applied in order to reduce the number of rules. With this DQ-method, there is no problem if the regions are overlapped. An extreme case is presented in the next figure.

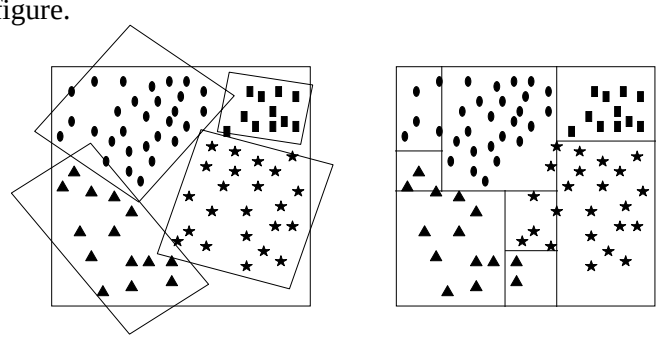


Fig. 1. Rotated versus axis-parallel hyperboxes.

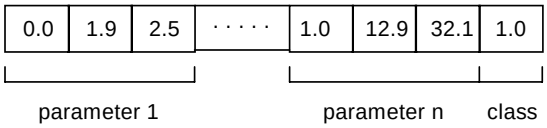
In the other hand, if we use axis-parallel techniques, the number of rules is very high. When does one technique apply over the other? In principle, it is not possible to know, but it could be a good idea to begin with axis-parallel technique and, increasingly, to rotate the best solutions found.

2 Description

In order to apply GAs 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 GAs 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.

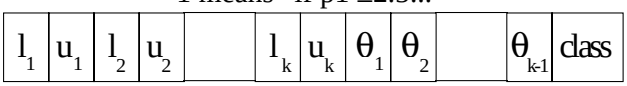
2.1. Environment for axis-parallel hyperboxes

The representation of an individual takes the following form:



Each parameter is defined by,

* A value belonging to the set {0, 1, 2} identifies the type of operation:
  - 0 means "if p1 ≤ 1.9..."
  - 1 means "if p1 ≥2.5..."



  - 2 means "if 1.9 ≤ p1 ≤ 2.5..."

* The two following values are the limits for each

parameter. If the operator is ≤ only makes sense the first value; if the operator is ≥, the second value is used; and, in the third case the two values indicates lower and upper boundaries.

* The last value identifies the class. The number of classes determines the set of values to which it belongs. That is to say, if there are five classes, the value will belong to the set {0,1,2,3,4}.

2.2 Environment for rotated hyperboxes

The GA uses real codification; that is, an individual is formed by an n-tuple of real. If k is the dimension, an individual has exactly $3 \times k$ values: $2 \times k$ for the boundaries of each dimension; $k-1$ for the angles of rotations, in radians, anti-clockwise around the hyperbox center; and one for the class. The next figure shows a n-tuple:

Fig. 2. Representation of an individual

where l_i and u_i represent the lower and upper bounds of the individual, respectively, for every dimension; θ_i is the rotation angle; and class. In two dimensions is possible to put a hyperbox at any orientation by using only one rotation; in k-dimension it is necessary k-1 rotations.

We consider that an example belongs to the area determined for an individual if it satisfies its condition part. Thus, let an example be given by $P_j=(p_1, p_2, ..., p_k, c)$ that fall within the region defined by the individual (or equivalently, an example will be covered by the rule) $ind_h = (l_1, u_1, l_2, u_2, ..., l_k, u_k, \theta_1, \theta_2, ..., \theta_{k-1}, class)$ if rotating the example $P=(p_1, p_2, ..., p_k)$ with the angles $-\theta_1, -\theta_2, ..., -\theta_{k-1}$ with relation to the center of the hyperbox defined by $(l_1, u_1, l_2, u_2, ..., l_k, u_k)$, the resulting P' belongs to this hyperbox.

Let $(m_1, m_2, m_3) = ((l_1 + u_1)/2, (l_2 + u_2)/2, (l_3 + u_3)/2)$ be the center of the hyperbox defined by the rule, then the coordinates of P' are:

$$(p'_1, p'_2, p'_3) = (p_1 - m_1, p_2 - m_2, p_3 - m_3) \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & -\operatorname{sen}\theta_1 \\ 0 & \operatorname{sen}\theta_1 & \cos\theta_1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & -\operatorname{sen}\theta_2 & 0 \\ \operatorname{sen}\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} + (m_1, m_2, m_3) \quad (1)$$

The example (p_1, p_2, p_3, c) will be covered by the rule $(l_1, u_1, l_2, u_2, l_3, u_3, \theta_1, \theta_2, class)$ if P' satisfies

$$l_1 \le p'_1 \le u_1 \wedge l_2 \le p'_2 \le u_2 \wedge l_3 \le p'_3 \le u_3 \quad (2)$$

where P' is obtained as above. And it will be correctly classified if its class is equal to c.

2.3 Algorithm

The algorithm is a typical sequential covering GA[7]. It chooses the best individual of the evolutionary process, transforming it into a rule, which is used to eliminate data from the training file[14]. 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 for every individual of the population an example from the training file. After, an interval to which the example belongs is obtained by adding and subtracting a random quantity from the values of the example.

The angles are randomly generated between zero and $\pi/2$. 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 *(x1, class)* from the training file; for last, a possible individual of the population could thus be:

$$(x_1 - range * k_1, x_1 + range * k_2, class)$$

where k_1 and k_2 are random values belonging to [0,1.05], and *class* is the same of that of the example.

The evolution module includes elitism: the best individual of every generation is replicated to the next one. A set of children is obtained from copies of randomly selected parents, generated by their fitness values. The remainder is formed through crossovers. Afterwards, mutation is applied depending on a probability.

Crossovers are specifically designed, choosing a value among one of the three segments formed inside the interval of the attribute by putting the two values of the individual as cross points. That is, for every attribute, an individual has two values, so these values are partitioning the interval in three segments. We select randomly a value inside of a segment also randomly chosen.

Mutation is applied in two different ways: if the 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).

In the case of rotated hyperboxes, the crossover operator calculates the mean of the angles; and mutation randomly generates another one.

To improve the best individual is a difficult task. If the fitness value is better, then the new angle replaces the old, and one value of one attribute is modified as mentioned above. This method allows the rotation of a hyperbox using only one dimension.

A possible criterion for implementing the mutation operator consists of distinguishing between mutation of values and mutation of angles as two independent operators. Mutation of values could have a higher probability of application than mutation of angles. Thus, we can penalize the rules with wrong angles; that is, the new individual is moving away from the closest example of the same class.

2.4 Fitness function

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

$$\text{if } G(i)*RC<=CE(i) \text{ then } CE(i)=0 \qquad (2)$$
$$f(i) = \frac{V}{T} + \frac{G(i)}{1+CE(i)}$$

where T is the cardinality of the training file, V is a new factor called coverage (the rule coverage is 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 errors, which are produced when the *i* example 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; *RC* is the relaxing coefficient. Every rule can be quickly expanded for finding more examples due to V in the fitness function.

2.5 Relaxing coefficient

Databases used as training files do not have clearly differentiated areas, for that, to obtain a rule system totally coherent involves a high number of rules. We show in previous paper [1] a system capable of producing a rule set exempt from error rate; 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. When databases present a distribution of examples very hard to classify, then it is advisable to use a relaxing coefficient [11]. Many times, we are more interested in understanding the structure of the databases than in the error rate. In this way, it could be better to have a system with fewer rules and despite some errors than too many rules and no errors.

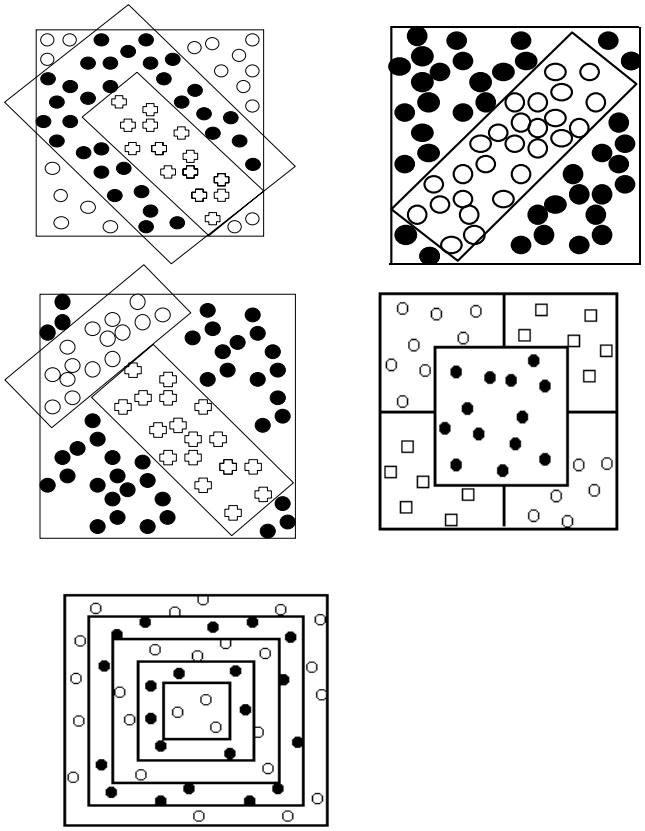


Fig. 3. Artificial databases named DB1, DB2, DB3, DB4 and DB5.

3 Application

We have used five cross validation in all our experiments to estimate classification accuracy. This cross validation experiment consists of the following steps: randomly divide the data into two disjoint partitions (70% and 30%); build a rule set using 70% of data and test the rule set with 30% of the data; for each partition the number of bad classification of the rule set are all counted and divided it by the number of instances of the test file to compute the error rate; the five values are summed and divided by five. We have chosen C4.5 to compare the results, also with five experiments and cross validation.

3.1 Artificial databases

We have designed some databases of varying complexity to graphically represent the experiments.

These databases are shown in the fig. 6. Results are in table 1.

Every database has two hundred examples, two dimensions and two classes.

DATABASE	C4.5		DQ-CLASSIFIER-AP		DQ-CLASSIFIER-RH	
	#RULES	ERROR RATE	#RULES	ERROR RATE	#RULES	ERROR RATE
DB1	17	12.9	10.6	14.56	**3**	**3.3**
DB2	12.5	16.25	12.0	10.54	**2**	**2.7**
DB3	11	12.13	10.0	9.43	**3**	**6.6**
DB4	9	0	**5**	**0**	5.6	2.7
DB5	27	0.8	**5**	**0**	6.2	4.1

Table1. Databases (number of examples, dimension, number of classes)

3.2 Databases from UCI repository.

The experiments described in this section are from UCI Repository [8].

It is very important to note that every execution has been realized with a very small population of 50 individuals and only 50 generations. These are very low numbers considering the number of examples and number of dimensions of the databases.

Decision queue is very relevant in relation to the number of rules.

IRIS (150,4,3)	4.4	6.3	**3.4**	7.47	3.6	**4.83**
PIMA (768,8,2)	77.6	28.4	20.0	27.2	**17.4**	**26.35**
CANCER (683,9,2)	5.2	13.8	**2.2**	**4.24**	2.2	5.38

Table 2. Databases (number of examples, dimension, number of classes).

4 Conclusions

A supervised learning tool to classify databases is presented in this paper. It produces a decision queue where the conditions of each rule indicates if an example belongs to a region. The regions can present two shapes, axis-parallel hyperboxes and rotated hyperboxes. The user chooses what rule representation is better for his problem. 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.

References:

[1] Aguilar, J and Riquelme, J. A Tool to obtain a Hierarchical Qualitative Set of Rules from Quantitative Data. Lectures Notes in Artificial Intelligence 1415. pp 336-346. Springer-Verlag, 1998.

[2] Aguilar, J. and Riquelme, J. Decision Queue Classifier for Supervised Learning using Rotated Hyperboxes. Progress in Artificial Intelligence IBERAMIA'98. Lectures Notes in Artificial Intelligence 1484. pp. 326-336. Springer-Verlag, 1998.

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

[4] Ghozeil, A. and Fogel, D.B. Discovering Patterns in Spatial Data Using Evolutionary Programming. Genetic Program Conference 96.

[5] Goldberg, D. Genetic Algorithms in search, optimization and machine learning. Addison-Wesley Publishing Company, Inc. 1989.

[6] Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution Programs. Second Edition, Springer-Verlag, 1994.

[7] Mitchell, T. Machine Learning. MacGraw-Hill, 1997.

[8] Murphy, P. and Aha, D.W. UCI Repository of

Machine Learning Databases. Dept. of Information and Computer Science. University of California, Irvine, 1994.
[9] Murthy, S. K., Kasif, S. and Salzberg, S. A system for induction of oblique decision trees. Journal of Artificial Intelligence Research, 1994. Morgan Kaufmann Publishers.
[10] Quinlan, J. R. C4.5: programs for Machine Learning. Morgan Kaufmann Pub.,1993.
[11] Riquelme J. and Aguilar, J. y Toro, M. Revista Iberoamericana de Inteligencia Artificial nº 5. A GA-based Tool to obtain a Hierarchical Classifier for Supervised Learning. (in spanish) pp 38-43, 1998.
[12] Rivest, R.L. Learning Decision Lists. Machine Learning, 87. pp. 229-246.
[13] Simpson, P.K. Fuzzy Min-Max Neural Networks. II. Clustering. IEEE Trans. Fuzzy Systems, Vol. 1:1,32-45.
[14] Venturini, G. SIA: a Supervised Inductive Algorithm with Genetic Search for Learning Attributes based Concepts.