Intelligent Information Systems VII
Proceedings of the Workshop held in
Malbork, Poland, June 15-19 , 1998

Learning Decision Rules Using an Evolutionary Algorithm and Entropy-Based Discretization

Wojciech Kwedlo and Marek Krętowski

Institute of Computer Science, Technical University of Białystok
Wiejska 45a, 15-357 Białystok, Poland
e-mail: {wkwedlo,mkret}@ii.pb.bialystok.pl

Keywords: learning decision rules, evolutionary algorithms, discretization

1 Introduction

One of the most promising directions in Knowledge Discovery in Databases (KDD) [4] is induction of decision rules. During the last two decades many methods e.g. AQ-family [9], CN2 [1] or C4.5 [10] were proposed. The advantages of the rule-based approach to KDD include natural representation and ease of integration of learned rules with background knowledge.

In the paper we propose a new method of learning decision rules, which uses an evolutionary algorithm (EA). EAs are stochastic search techniques, which have been inspired by the process of biological evolution. They have been applied to many search and optimization problems including learning decision rules [2, 6, 8]. There are two key issues in our approach. The first one is the use of two non-standard recombination operators, which we call changing condition operator and insertion operator. The second issue is the application of entropy-based discretization [5, 3], which allows us to effectively deal with continuous-valued features.

2 Learning decision rules

Let us assume that we have a learning set $E = \{e_1, e_2, \dots, e_M\}$ consisting of M examples. Each example $e \in E$ is described by N attributes (features) $\{A_1, A_2, \dots, A_N\}$ and labelled by a class $c(e) \in C$. The domain of a nominal (discrete-valued) attribute A_i is a finite set $V(A_i)$ while the domain of a continuous-valued attribute A_j is an interval

$V(A_j) = [l_j, u_j]$. For each class $c_k \in C$ by $E^+(c_k) = \{e \in E : c(e) = c_k\}$ we denote the set of *positive examples* and by $E^-(c_k) = E - E^+(c_k)$ the set of *negative examples*. A *classification rule* R takes the form $t_1 \wedge t_2 \wedge \ldots \wedge t_r \rightarrow c_k$, where $c_k \in C$ and the left-hand side is a conjunction of $r(r \leq N)$ conditions $t_1, t_2, \ldots, t_r$. Each condition t_j concerns one attribute A_{k_j}. It is assumed that $k_j \neq k_i$ for $j \neq i$. If A_{k_j} is a continuous-valued attribute than t_j takes one of three forms: $A_{k_j} > a$, $A_{k_j} \leq b$ or $a < A_{k_j} \leq b$, where $a, b \in V(A_{k_j})$. Otherwise (A_{k_j} is nominal) the condition takes the form $A_{k_j} = v$, where $v \in V(A_{k_j})$. The rule induction algorithm builds for each class $c_k \in C$ the set of disjunctive decision rules $RS(c_k)$ covering all (or near all) positive examples from $E^+(c_k)$. This aim is achieved by repeating for each c_k the following procedure (also called *sequential covering*): First "the best" or "almost best" classification rule is found using some global search method (an EA in our case). Next all the positive examples covered by the rule are removed and the search process is iterated on the remaining learning examples. The criterion expressing the performance of a rule (in terms of EAs called the *fitness function*) prefers rules consisting of few conditions which cover many positive examples and very few negative ones.

Currently our EA can work with nominal attributes only. For that reason each continuous-valued attribute A_j requires prior *discretization*. The discretization algorithm searches for a partition of the domain $V(A_j) = [l_j, u_j]$ into d_j subintervals $[a_j^0, a_j^1), [a_j^1, a_j^2), \ldots, [a_j^{d_j-1}, a_j^{d_j}]$. Any original value of the attribute A_j is then replaced by the number of the interval to which it belongs. The partitioning of the domain of A_j is performed as follows [5, 3]: First the initial interval $[l_j, u_j)$ is divided into two subintervals $[l_j, s)$ and $[s, u_j)$ in such way that this partition maximizes Quinlan's information gain measure [10]. This procedure is then recursively applied to both subintervals $[l_j, s)$ and $[s, u_j)$. The recursive partitioning stops when the criterion defined by Fayyad and Irani [5] based on the Minimal Description Length Principle is met. It is important to notice that when learning decision rules for a class c_k it is not necessary to distinguish between all the classes $c_1, c_2, \ldots, c_K$. Instead we merge all the classes different from c_k creating a class c_k^- . Then we run discretization algorithm and finally we generate decision rules.

3 Searching for decision rules with EA

Our version of evolutionary algorithm follows the general description presented in [8]. In this section we describe the following application-specific issues: representation, the evolutionary operators, the termination condition and the fitness function. We assume that all continuous-valued features have already been discretized.

Representation Given the class label c_k any decision rule $\Phi \rightarrow c_k$ can be represented as a fixed-length string $S = \langle f_1, f_2, \ldots f_N, \omega_1, \omega_2, \ldots, \omega_N \rangle$ where f_i is a binary flag and $\omega_i \in V(A_i)$ is the value of attribute A_i encoded as an integer number. If the flag f_i is set than a condition $A_i = w_i$ is present in conjunction on the left-hand side of the classification rule. The rule represented by string S can be expressed as follows:

$$(A_{j_1} = \omega_{j_1}) \wedge (A_{j_2} = \omega_{j_2}) \wedge \cdots \wedge (A_{j_L} = \omega_{j_L}) \rightarrow c_k \quad (1)$$

where L is the length of the rule and $j_1, j_2, \ldots, j_L \in \{j : f_j = 1\}$. One can see that if the flag f_j is not set the value of ω_j is irrelevant.

The Fitness function Consider any string S encoding classification rule, which covers POS positive examples and NEG negative ones. Its fitness is defined by the equation:

$$f(S) = \begin{cases} POS + \frac{N-L-NEG}{N} & if \quad \frac{NEG}{POS+NEG} < T_{NEG} \; and \; POS \neq 0 \\ 0 & otherwise \end{cases} \tag{2}$$

where N is the number of attributes, L is the number of conditions constituting the left-hand side of the rule, $T_{NEG} \in [0,1]$ is a user-defined threshold which limits the maximal percentage of negative examples covered by the rule encoded by S. The value of T_{NEG} has a big effect on classification accuracy. $T_{NEG} = 0$ may cause the induced rules to be too specialized and *overfit* the data. Such rules will classify perfectly the examples from the learning set but they will work very poor with an independent testing set. $T_{NEG} > 0$ prevents overfitting by allowing the learned rules to cover some negative examples. Such rules can be more general i.e. consist of fewer tests and cover additional positive examples.

Initialization, termination condition and selection Each string is initialized using a randomly chosen positive example e from $E^+(c_k)$. Let us assume that $i_1, i_2, ..., i_r$ denote the numbers of non-missing attributes describing e and $\omega_{i_1}, \omega_{i_2}, ..., \omega_{i_r}$ denote the values of these attributes. A new string S is created in such way that it represents the classification rule $(A_{i_1} = \omega_{i_1}) \wedge (A_{i_2} = \omega_{i_2}) \wedge \ldots \wedge (A_{i_r} = \omega_{i_r}) \rightarrow c_k$. This method assures that the rule represented by S covers at least one positive example and if the learning set is consistent it does not cover any negative ones. The algorithm terminates if the fitness of the best individual in population does not improve during N_{TERM} generations where N_{TERM} is the user-supplied parameter. As a selection operator we use proportional elitist selection with linear scaling [7].

Changing condition operator. This unary operator takes as an argument a single string $S = \langle f_1, f_2, \ldots f_N, \omega_1, \omega_2, \ldots, \omega_N \rangle$. It works as follows: First we choose a random number i where $1 \leq i \leq N$. Then the flag f_i is tested. If it is set i.e. the condition concerning attribute A_i is present in the rule represented by S we reset f_i and drop this condition from the rule. If f_i is not set we set f_i and replace ω_i with randomly chosen $\omega_i' \subseteq V(A_i)$ thus introducing the condition $A_i = \omega_i'$ to the rule. This operator is similar to standard mutation operator of genetic algorithms [7].

Insertion operator. The aim of this unary operator is to modify a classification rule R in such manner that it will cover a randomly chosen positive example e currently uncovered by R. This can be achieved by removing from R all logical conditions $A_i = \omega_i$ which return false when the rule is tested on the example e. The removal is done by resetting the corresponding flag f_i. As a result the string $S = \langle f_1, f_2, \ldots f_N, \omega_1, \omega_2, \ldots, \omega_N \rangle$ representing the classification rule R is replaced by $S' = \langle f_1', f_2', \ldots f_N', \omega_1, \omega_2, \ldots, \omega_N \rangle$ where

$$f_i' = \begin{cases} 0 \; if \quad f_i = 1 \text{ and the condition } A_i = \omega_i \text{ is not satisfied by the example } e \\ f_i \quad \text{otherwise} \end{cases} \tag{3}$$

Because the string modified by this operator will cover at least one new positive example there is a chance that its fitness (2) will increase. Of course it may decrease if the new string covers some new negative examples.

Crossover operator. We use a modification of the uniform crossover [8]. This binary operator requires two arguments $s_1 = \langle f_1, f_2, \ldots, f_N, \omega_1, \omega_2, \ldots, \omega_N \rangle$ and $s_2 = \langle g_1, g_2, \ldots, g_N, v_1, v_2, \ldots, v_N \rangle$. For each $i = 1, 2, \ldots, N$ it exchanges f_i with g_i and ω_i with v_i with the probability 0.5.

4 Preliminary experimental results

In this section some initial experimental results are presented. We have tested our approach on 5 datasets from UCI Repository. Table 1 describes these datasets and presents the classification accuracies obtained by our method and by C4.5 induction algorithm.

Dataset	Features	Examples	Classes	C4.5 (see [3])	Our approach	T_{NEG}
australian	14 (8 nominal)	690	2	85.4%	86.0%	0.1
diabetes	8	768	2	70.8%	76.1%	0.2
german	24	1000	2	72.3%	71.2%	0.2
glass	9	214	7	65.9%	65.9%	0.4
iris	4	150	3	94.7%	96.1%	0.05

Table 1. Description of the datasets used in experiments and classification accuracies of our method and C4.5 algorithm. The accuracies were measured by crossvalidation.

Acknowledgments The authors are grateful to Prof. Leon Bobrowski for his support and useful comments. This work was partially supported by the grant 8T11E00811 from State Committee for Scientific Research (KBN) and by the grant W/II/1/97 from Technical University of Białystok.

References

1. P. Clark, T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3: 261-283, 1989.
2. K. De Jong *et al.* Using genetic algorithm for concept learning. *Machine Learning*, 13: 168-182, 1993.
3. J. Dougherty *et al.* Supervised and unsupervised discretization of continuous features. In *Proc. of* 12^{th} *Int. Conf. on Machine Learning*. pp. 194-202, Morgan Kaufmann, 1993.
4. U. Fayyad *et al.* (eds.). *Advances in Knowledge Discovery and Data Mining.* AAAI Press, 1996.
5. U. Fayyad, K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of IJCAI '93.* pp. 1022-1027, Morgan Kaufmann, 1993.
6. A. Giordana *et al.* A fine-grained parallel evolutionary program for concept induction. In Z. Ras , M. Michalewicz (eds.) . *Proc. of the* 9^{th} *Int. Symp. ISMIS '96.* LNAI 1099.
7. D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
8. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 3^{rd} edition, 1996.
9. R. Michalski *et al.* The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. of AAAI-86*. Morgan Kaufmann, 1986.
10. J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.