Discovery of Decision Rules from Databases: An Evolutionary Approach

Wojciech Kwedlo and Marek Krętowski

Institute of Computer Science
Technical University of Białystok
Wiejska 45a, 15-351 Białystok, Poland
e-mail: {wkwedlo,mkret}@ii.pb.bialystok.pl

Abstract. Decision rules are a natural form of representing knowledge. Their extraction from databases requires the capability for effective search large solution spaces. This paper shows, how we can deal with this problem using evolutionary algorithms (EAs). We propose an EA-based system called EDRL, which for each class label sequentially generates a disjunctive set of decision rules in propositional form. EDRL uses an EA to search for one rule at a time; then, all the positive examples covered by the rule are removed from the learning set and the search is repeated on the remaining examples. Our version of EA differs from standard genetic algorithm. In addition to the well-known uniform crossover it employs two non-standard genetic operators, which we call changing condition and insertion. Currently EDRL requires prior discretization of all continuous-valued attributes. A discretization technique based on the minimization of class entropy is used. The performance of EDRL is evaluated by comparing its classification accuracy with that of C4.5 learning algorithm on six datasets from UCI repository.

1 Introduction

Knowledge Discovery in Databases (KDD) is the process of identifying valid, potentially useful and understandable regularities in data [5]. The two main goals of KDD are *prediction* i.e. the use of available data to predict unknown values of some variables and *description* i.e. the search for some interesting patterns and their presentation in easy to understand way.

One of the most well-known data mining techniques used in KDD process is extraction of decision rules. During the last two decades many methods e.g. AQ-family [12], CN2 [2] or C4.5 [15] were proposed. The advantages of the rule-based approach include natural representation and ease of integration of learned rules with background knowledge.

In the paper we present a new system called EDRL (EDRL, for Evolutionary Decision Rule Learner), which searches for decision rules using an evolutionary algorithm (EA). EAs [11] are stochastic search techniques, which have been inspired by the process of biological evolution. They have been applied to many optimization problems. The success of EAs is attributed to their ability to avoid local optima, which is their main advantage over greedy search methods. Several

EA-based systems, which learn decision rules in either propositional (e.g. GABIL [3], GIL [10], GA-MINER [7]) or first order (e.g. REGAL [8, 14], SIAO1 [1]) form have been proposed.

There are two key issues in our approach. The first one is the use of two non-standard genetic operators, which we call changing condition operator and insertion operator. The second issue is the application of entropy-based discretization [6, 4], which allows us to effectively deal with continuous-valued features.

The reminder of the paper is organized as follows. In the next section we present basic definitions and outline the rule induction scheme used by EDRL. Section 3 describes a heuristic based on entropy minimization, which is used to discretize the continuous-valued attributes. Section 4 presents the details of our EA including representation of rules, the fitness function and genetic operators. Preliminary experimental results are given in Section 5. The last section contains our conclusions and possible directions of future research.

2 Learning decision rules

Let us assume that we have a learning set $E = \{e_1, e_2, \ldots, e_M\}$ consisting of M examples. Each example $e \in E$ is described by N attributes (features) $\{A_1, A_2, \ldots, A_N\}$ 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 $v \in V(A_{k_j})$.

EDRL builds separately 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 terminology of EAs called the *fitness function*) prefers rules consisting of few conditions, which cover many positive examples and very few negative ones. The sequential covering is stopped when either all the positive examples are covered or the EA is unable (after three consecutive trials) to find a decision rule covering more then τ percent of all the positive examples from $E^+(c_k)$, where τ is a user-supplied parameter called *rule sensitivity threshold*.

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 Discretization of continuous-valued attributes

As it was mentioned before, each continuous-valued attribute A_j requires prior *discretization*. In this section we briefly explain the method we use (for a more detailed description the reader is referred to [6]) The aim of discretization is to find 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.

EDRL employs a supervised top-down greedy heuristic based on entropy reduction. Given a subset of examples $S \subseteq E$ its *class information entropy* H(S) is defined by:

$$H(S) = -\sum_{c_k \in C} p(S, c_k) \log p(S, c_k), \tag{1}$$

where $0 \leq p(S, c_k) \leq 1$ is the proportion of examples with class c_k in S. The partitioning of the domain of A_j is performed as follows : First the initial interval $I = [l_j, u_j)$ is divided into two subintervals $I_1 = [l_j, a)$ and $I_2 = [a, u_j)$ in such way that this partition maximizes the information gain [15]:

$$Gain(A_j, I, a) = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2), \tag{2}$$

where $S, S_1, S_2 \subseteq E$ denote sets of examples for which the value of A_j belongs to the intervals I, I_1 and I_2 respectively. This procedure is then recursively applied to both subintervals I_1 and I_2. The recursive partitioning is performed only when the condition proposed by Fayyad and Irani [6] based on the Minimal Description Length Principle is met:

$$Gain(A_j, I, a) \geq \frac{log_2(|S| - 1)}{|S|} + \frac{\Delta(A_j, I, a)}{|S|}, \tag{3}$$

where $\Delta(A_j, I, a) = \log_2(3^n - 1) - [nH(S) - n_1 H(S_1) - n_2 H(S_2)]$, and n, n_1, n_2 denote the number of class labels presented in S, S_1, S_2 respectively.

Dougherty *et al.* in a large experimental study [4], compared the above method with three others. The results indicated that an entropy-based discretization outperformed its competitors, namely equal interval binning, equal frequency binning, and 1R discretizer.

4 Searching for decision rules with EA

Our version of evolutionary algorithm follows the general description presented in [11]. In this section we present 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.

4.1 Representation

Given the class label c_k any decision rule 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. The flag f_i is set if and only if the condition $A_i = w_i$ is present in conjunction on the left-hand side of rule. The rule represented by string S can be expressed as follows:

$$(A_{j_1} = \omega_{j_1}) \wedge (A_{j_2} = \omega_{j_2}) \wedge \cdots \wedge (A_{j_L} = \omega_{j_L}) \rightarrow c_k \tag{4}$$

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_i is not set the value of ω_i is irrelevant.

4.2 The Fitness function and the infeasibility criterion

Consider a string S encoding a decision rule, which covers POS positive examples and NEG negative ones. Its fitness is defined by the equation:

$$f(S) = \frac{POS^{\alpha}}{L+1} PE(x), \tag{5}$$

where

$$x = \frac{NEG}{POS + NEG)} - \beta,$$

L is the number of conditions constituting the left-hand side of the rule, α and β $(0 \leq \beta \leq 1)$ are two users-supplied parameters, $PE(x)$ is the function which significantly degrades the fitness of the rule when the proportion of the number of covered negative examples to the total number of covered examples is greater than β. In all the experiments $PE(x)$ was given by

$$PE(x) = \frac{1}{1 + \exp(\gamma_1 (x - \gamma_2))}, \tag{6}$$

where $\gamma_1 = 30$ and $\gamma_2 = 0.05$ (see Fig. 1), although other forms (e.g. threshold function) might also be used.

The value of β should be chosen carefully. β excessively close to 0 allows the generated rules to cover very few negative examples. Such rules are likely be too specialized and *overfit* the data. They will classify perfectly the examples from the learning set but their accuracy will by very poor when tested on previously unseen examples. On the other hand, excessively high value of β will increase the classification error by making the rules cover many negative examples.

When $PE(x)$ is too small (we have chosen $PE(x) < 0.05$) the rule is regarded as the *infeasible* one. It is rejected and the string S is re-initialized, as described in the next subsection.

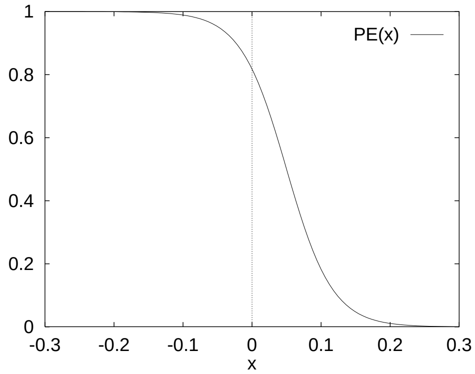

Fig. 1. The plot of $PE(x)$ ($\gamma_1 = 30$ and $\gamma_2 = 0.05$).

4.3 Initialization, termination condition and selection

Each string in the population 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 features 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 decision 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 string in the 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 [9].

4.4 Genetic operators

The search in EAs is performed by genetic operators. They alter the population changing some individuals. Early implementations of EAs used the same operators and representation for every problem they were applied to. For instance a Standard Genetic Algorithm (SGA) [9, 11] represents individuals as binary strings and uses two genetic operators: crossover and mutation. The SGA was successfully applied to many problems, however many researchers e.g. Michalewicz in [11] report that it can be outperformed by the algorithm with carefully designed problem-dependent genetic operators and representation. Michalewicz and others argue that the use of problem-aware operators allows the EA to exploit the knowledge about the problem, which improves the performance. The weakness of domain-specific EA is that it can be applied only to the task it was designed for, while SGA can be used to solve any optimization problem.

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 $1 \le i \le N$. 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 [9].

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

$$f'_i = \begin{cases} 0 \ if \ \ f_i = 1 \text{ and the condition } A_i = \omega_i \text{ is not satisfied by example } e \\ f_i \quad \text{otherwise} \end{cases} \tag{7}$$

Because the string modified by this operator will cover at least one new positive example there is a chance that its fitness (5) 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 [11]. 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.

5 Preliminary experimental results

In this section some initial experimental results are presented. We have tested EDRL on 6 datasets from UCI Repository [13]. Table 1 describes these datasets. Table 2 shows the classification accuracies obtained by EDRL and C4.5 (Rel 8) learning algorithm. The results concerning C4.5 were taken from [16]. In both cases the accuracies were estimated by running ten times the complete ten-fold crossvalidation. The mean of ten runs and the standard error of this mean are presented. As a reference we show the accuracy of the majority classifier[1]

[1] The majority classifier assigns an unknown example to the most frequent class in the learning set.

The values of the parameter β of the fitness function are also given; in all the experiments we used $\alpha = 2.0$ and the rule sensitivity threshold $\tau = 3\%$. Table 3 shows the number of rules and the total number of conditions obtained when the rules were extracted from the complete datasets.

Dataset	Features	Examples	Classes
australian	15 (9 nominal)	690	2
diabetes	8	768	2
german	20 (13 nominal)	1000	2
glass	9	214	7
hepatitis	19 (13 nominal)	155	2
iris	4	150	3

Table 1. Description of the datasets used in the experiments.

Dataset	Majority	C4.5	EDRL	β
australian	55.5	85.3 ± 0.2	86.1 ± 0.4	0.05
diabetes	65.1	74.6 ± 0.3	77.9 ± 0.3	0.2
german	70.0	71.6 ± 0.3	70.1 ± 0.8	0.2
glass	35.5	67.5 ± 0.8	66.7 ± 1.0	0.3
hepatitis	79.4	79.6 ± 0.6	81.2 ± 1.8	0.1
iris	33.3	95.2 ± 0.2	96.0 ± 0.0	0.05

Table 2. Classification accuracies of our method and C4.5 (Rel 8) algorithm.

6 Conclusions and future work

In the paper we proposed an rule learning EA-based system EDRL and conducted its experimental evaluation. The preliminary experimental results merely entitle us to conclude, that the classification accuracy of the current version of EDRL is comparable to that of C4.5. A real improvement was observed only for diabetes dataset. However we believe that the performance of our system can be further improved.

Several directions of future research exist. Currently all the continuous-valued features are discretized *globally* [4] prior to extraction of rules. Each feature is discretized independently from the others. We are working on the modification of EDRL, which will enable it to perform simultaneous search for all attribute thresholds during the induction of rules.

Dataset	Number of rules	Number of conditions
australian	5	15
diabetes	8	21
german	8	36
glass	9	19
hepatitis	12	50
iris	5	7

Table 3. Number of rules and conditions obtained.

Another idea is the extension of the rule representation form to VL_1 [12] language, in which a test can contain comparison to multiple values (internal disjunction). This could be especially beneficial for datasets with nominal attributes with large domains.

We also intend to replace the current strategy of dealing with infeasible rules by a more sophisticated method e.g. repair algorithm [11].

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), Poland and by the grant W/II/1/97 from Technical University of Białystok.

References

1. Augier, S., Venturini, G., Kodratoff, Y.: Learning first order logic rules with a genetic algorithm. In: *Proc. of The First International Conference on Knowledge Discovery and Data Mining KDD-95*, AAAI Press (1995) 21-26.
2. Clark, P., Niblett, T.: The CN2 induction algorithm. *Machine Learning* 3 (1989) 261-283.
3. De Jong, K.A., Spears, W.M., Gordon, D.F.: Using genetic algorithm for concept learning. *Machine Learning* 13 (1993) 168-182.
4. Dougherty, J., Kohavi, R., Sahami, M.: Supervised and unsupervised discretization of continuous features. In: *Machine Learning: Proc of 12th International Conference*. Morgan Kaufmann (1995) 194-202.
5. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (eds.): *Advances in Knowledge Discovery and Data Mining*. AAAI Press (1996).
6. Fayyad, U.M., Irani, K.B.: Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of 13th Int. Joint Conference om Artificial Intelligence*. Morgan Kaufmann (1993) 1022-1027.
7. Flockhart, I.W., Radcliffe, N.J.: A Genetic Algorithm-Based Approach to Data Mining. In: *Proc. of The Second International Conference on Knowledge Discovery and Data Mining KDD-96*, AAAI Press (1996) 299-302.
8. Giordana, A., Neri, F.: Search-intensive concept induction. *Evolutionary Computation* 3(4) (1995) 375-416.

9. Goldberg, D.: *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley (1989).
10. Janikow, C.: A knowledge intensive genetic algorithm for supervised learning. *Machine Learning* 13 (1993) 192-228.
11. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. 3rd edn. Springer-Verlag (1996).
12. Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N., : The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: *Proc. of the 5th National Conference on Artificial Intelligence* (1986) 1041-1045.
13. Murphy, P.M., Aha, D.W.: *UCI repository of machine-learning databases*, available on-line: http://www.ics.uci.edu/pub/machine-learning-databases.
14. Neri, F., Saitta, L.: Exploring the power of genetic search in learning symbolic classifiers. *IEEE Transaction on Pattern Analysis and Machine Intelligence* 18 (1996) 1135-1142.
15. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann (1993).
16. Quinlan, J.R.: Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research* 4 (1996) 77-90.