27

Evolutionary-Based Classification Techniques

Rasha Shaker Abdul-Wahab
Ahlia University,
Bahrain

1. Introduction

Recently data mining has attracted a great attention in the information industry due to the need for turning data into useful information [Freitas]. In fact, Data Mining comes with two different directions depending mainly on the application domain and the user interest. The first direction attempts to classify some target filed whereas the second direction attempts to find similarities among groups of data. However, one of the most common Data Mining tasks is Classification. Classification belongs to the first flavor of Data Mining directions which is used to assign objects to one of several predefined categories. The input data for classification task is a collection of records. Each record, also known an instance or example, is characterized by a typle (x,y), where x is the attributes set and y is a special attributes, designated the class label [Falco, Dasgupta]. In the literature several methods have been proposed to solve classification problem which are [Falco, Dasgupta] statistical methods, trees, neural-networks, rule induction method, evolutionary algorithms methods, etc. The choice of a particular method depends, however, on factors like the kind of problem to be solved, the resources available, etc. Classification is applied with different major types of EAs algorithms which are: Genetic Algorithms (GAs), Genetic Programming (GPs), Evolutions Strategies (ES), and Evolutionary Programming (EP).). All of them share the same basic concepts, but differ in the way they encode the solutions and in the operators they use. However, this work aims to use another type of EAs to solve classification problems.

EAs are very flexible search techniques, they can be used to solve many different kinds of problems and have been used in a wide variety of fields and applications [Langdon]. EAs are randomized search procedures inspired by the mechanics of genetics and natural selection. Most of data mining applications seek to reach optimality in their solutions which is considering the goal of most EAs algorithms. EAs work on a population of individuals that represent possible solutions to a problem in their chromosomes. Each individual can be as simple as a string of zeroes and ones, or as complex as a computer program [Langdon, Norman].

The aim of this chapter is to investigate the effect of using an alternative form of EAs to solve classification problem, for which a new System MRule is developed. This technique is an aggregation of different proposal; the first is based on GAPBNF [Abdul-Wahab] (Genetic Algorithm for developing Program using Backus Naur form) to discover comprehensible If Then rules. The second proposal is to integrate the syntax of Structured Query Language (SQL) language with GAPBNF.

The subsequent sections are organized as follows. Section 2 gives the definition of the classification problem. The objective of this chapter presented in section 3. MRule algorithm is described in section 4 with its major characteristic. The main units of MRule are illustrated in section 5. Section 6 contains the performance of the proposed system compared with that achieved by other methods. In last section, final remarks and future work are outline.

2. Problem definition

Given a dataset $X = \{ x_1, x_2, \ldots, x_n \}$ of tuples (items, records, variable) and a set of classes $C = \{C_1, \ldots\ldots, C_m\}$, the classification problem is to define a mapping $f : X \to C$ where each x_i is assigned to one class. A class C_j contains precisely those tuples mapped to it. That is

$C_j = \{ x_i \mid f(x_i) = C_j, \ 1 \le i \le n, \ \text{and} \ x_i \in X \}$ [Dunham].

3. Objective

This aims of this chapter is to investigate the effect of using an alternative form of EAs to solve the problem of one of the DM tasks, classification, for which a new technique to mine set of rules (MRule) is developed. The second objective tries to design new form which avoiding the time consuming through the frequent evaluation of candidates (rules) against the dataset. In addition try on focusing to deliver understandable rules and easy expressed them later in a database access language such as SQL to retrieve raw data in a particular category.

4. MRule system

MRule is a technique based on GAPBNF (Genetic Algorithm for Developing Program using Backus Naur Form) to perform the task of classification that has the ability to discover comprehensible If-Then rules. The reason for performing classification task with GAPBNF is that GAPBNF uses the GA engine and works on simpler structure, thus it expect some changes in the performance of the proposed technique for developing classification rules.
MRule has the ability to learn one disjunct at a time, and then all the discovered disjuncts together form the target concept description. It follows the standard strategy (also called covering strategy) in separate-and-conquer rule learning algorithms: learn a rule that covers part of the training set, remove the covered examples from training set and then recursively learn the remaining examples until all are covered [Pang_Ning]. Figure (1) describes the components of MRule system.

5. GAPBNF with classification problem

This section describes how to incorporate GAPBNF with classification problem, namely the phenotype language, ontogentic mapping, phenotype and genotype generator, genetic operators, and fitness function are explained in this section.

5.1 The phenotype language

The phenotype language is the language in which the phenotypes produced by GAPBNF are written. The Phenotype language in this work is the syntax of SQL language. Integrating the syntax of SQL in MRule system avoids the drawbacks of evaluation of rules against data.

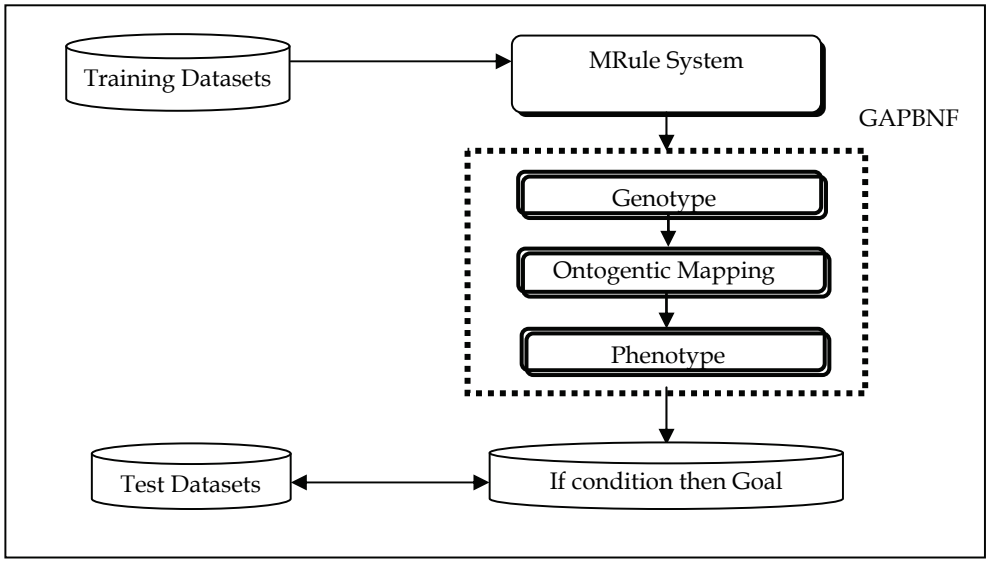

Fig. 1. General Description of MRule System

5.2 The ontogenic mapping

Genotype in GAPBNF is distinct from the phenotype. However, to convert the genotype to phenotype, the ontogenic mapping is needed. Ontogenic mapping uses the BNF definition of SQL language. The diagram to represent the engine of ontogentic mapping is illustrated in figure (2).

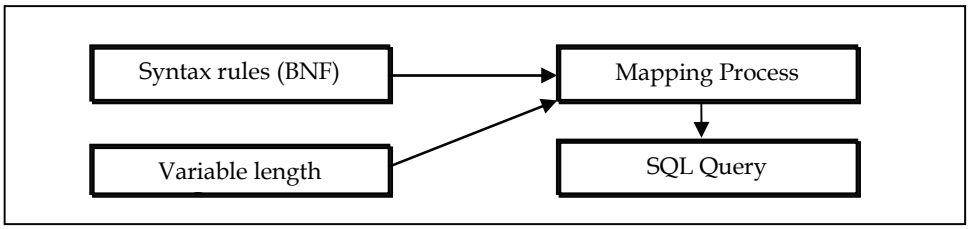

Fig. 2. The Engine of Ontogentic Mapping in MRule Sysyem

5.3 Genotype generator

The GAPBNF genotype is a list of genes encoded as an integer values. The generator used to generate variable length genotypes employs a controlled randomization based on some constraints which are represented as a context free grammar. These constraints help this generator to generate a valuable individual. These constraints are a set of rules that put some restriction on how the gene of GAPBNF should be arranged.

For instance, if a_1, a_2,…a_L is the genotype, the selection of a_2 is not generated randomly, instead the selection is done depending on a_1 and a_L is dependent on L-1, where L is the maximum length of the genotype which was generated randomly. In addition, using these constraints leads to reduce the search space. Consider an individual made up of the

following genes: 11 30 10 21 41 50 10 22 40 60 51 52. These numbers represent the production rules number of the syntax language which is represented as table. The syntax which is consider the base to generate genotypes is presented in figure (3). To generate genotypes, two production numbers are fixed for all individuals which are (depending on Table 1) 11 and 30. These numbers represent the first and the second genes of individuals. The remaining genes will be generated depending on the identified constraints for the corresponding problem.

Rule	Production rule number		Number for each choice
Condition → <input> | <application>	1	0 1	10 11
Input → <att > = <var> | <att> < = <var> | <att> between <exp>	2	0 1 2	20 21 22
Application → <condition> and <condition>	3	0	30
Att → <A_1> | <A_2> | <A_3>	4	0 1 2	40 41 42
Var → <r>	5		5 [index]
Exp → <var> and <var>	6	0	60

Fig. 3. The required syntax rules of MRule System

To build these constraints, a number will be assign to each production rule as follows:

$$S_1 \to 10, \mathrm{A} \to 11, \mathrm{B}_1 \to 20, \mathrm{B}_2 \to 21, \mathrm{B}_3 \to 22, \mathrm{C} \to 30, \mathrm{D}_1 = 40, \mathrm{D}_2 = 41, \mathrm{D}_3 = 42,$$
$$E \to 5[INDEX], \mathrm{F} \to 60$$

thus the constraints depending on figure (3) are shown in figure (4).

$$\begin{aligned}
S &\to A_1 \mid \lambda \\
A_1 &\to AC_1 \mid \lambda \\
C_1 &\to \mathrm{CS}_1 \mid \lambda \\
S_1 &\to \mathrm{BD} \\
D &\to D_1 I \mid D_{2I} \mid D_3 I \mid \mathrm{D}_1 F \mid D_2 F \mid \mathrm{D}_3 F \\
I &\to ES \mid EI \\
B &\to B_1 \mid B_2 \mid B_3 \\
F &\to I
\end{aligned}$$

Fig. 4. The CFG of fig. 3.

As shown in figure (3), for production rule 5 there is no number associated with this rule, but through the creation process of the genotype, this rule is presented as 50, 51, 52 or 53 and so on. These associated numbers represent the index of the local memory which contains the intermediate values for the corresponding genotype. GAPBNF genotype needs these local values to give its corresponding phenotype the data upon which to operate.

Generating local memory values for each genotype is done as follows: grouping all examples that belong to the same class, then for each attribute in these examples compute its

maximum and minimum values. These values are considered the ranges from which the corresponding value of the selected attribute in the genotype will be chosen. This technique tries to generate rules which have high consistency and coverage. Through the creation process the uniqueness of the attributes selection must be considered, i.e., it means genotype creation must be stopped when all attributes are chosen. The pseudo-code for creating the initial population is presented in algorithm (1)

Algorithm 1 Initial population Processer

Input : Popsize which represents the number of individuals in a population, Maxlenrule which represents the maximum length of the generated genotype.

Output : Set of individual (genotypes)

```
1: While P < = Popsize do
2      L = uniform (0, Maxlenrule)
3:    For i = 1 to L do
4:    Create geni using the identify constraints of the corresponding    problem
5:      If ( geni is equal to the number of attribute rule ) then
6:         Check the selected geni is not chosen before
7:      Else
8:          Exit if there is no more attributes that construct the genotype
9:      End if
10:     If ( geni is equal to the number of l.m rule ) then
11:         Stored the selected value in its local memory
12:     End if
13:     rule(genotype) = rule(genotype) ∪ {geni}
14:   End for
15:   Individual-set= Individual-set ∪ rule(genotype)
16: End while
```

5.4 Rule generator

At the beginning of the process of converting the genotype to the corresponding phenotype, an initial value is needed. The initial value of the phenotype is usually the start symbol of the syntax of the problem language. In this work, "Condition" symbol is identified as an initial value for the converter process.

The Phenotype is generated as Abstract Syntax Tree (AST). Consider the following individual (genotype): 11 30 10 21 41 50 10 22 40 60 51 52. Each gene in the above genotype is used to map into the corresponding production rule, this is done by reading each one alone and an appropriate production rule will be used to build one node of the AST (phenotype).

To build the AST of the above genotype, the first gene is read and will be used to generate the root node of this tree which is 11 in this case. This number matches production rule 1 (based on figure (1)). So to create the root node, the right hand side of this rule will be taken which has number 1 in this case. The created root node is presented in figure(5).

Null	Application	Nu11

Fig. 5. The root of the Abstract Syntax Tree.

The process is continuing by reading the second gene, which is 30 in this case. The left hand side matches the above root node, and then the right hand side of this rule will be taken and will be linked to the root node. The created AST at this step is presented in Figure (6).

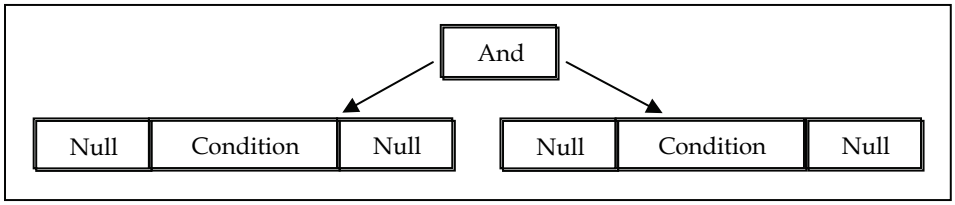

Fig. 6. The Abstract Syntax Tree (AST)

Then, the generator continues to process other genes in that genotype until the AST creation is complete. The AST of the above genotype after processing all its genes is presented in Figure (7), which represents the expression (A1 ≤ 128 and A2 between 3 and 7). The generated AST represents the condition part of the discovered rule which will be converted as a SQL statement through the evaluation process and will be discarded after computing the fitness value of this individual.

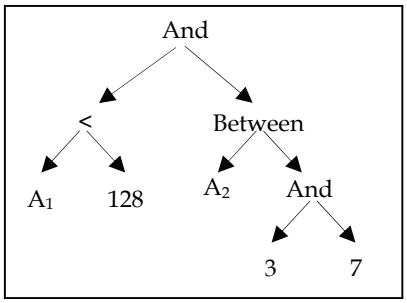

Fig. 7. The completed AST

Example: if we have the following genotype: 11 30 10 22 47 60 50 51 10 21 42 52 10 20 45 53 11 30 10 20 40 54 11 30. Based on the BNF of figure (3), the AST which represents the condition part of the rule is presented in figure (8).

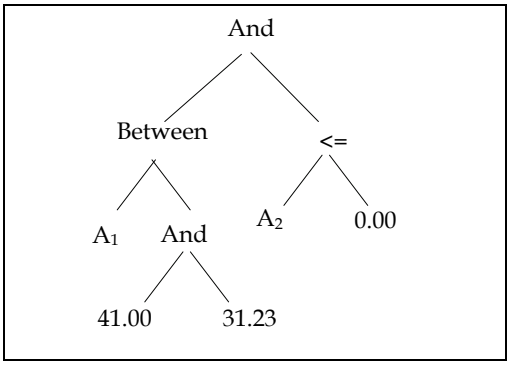

Fig. 8. The completed AST of the example

The corresponding expression of the above tree is:

A_1 between 41.00 and 31.23 and A_2 <= 0.00

the SQL statement to evaluate the above condition is:

Select * from Tabledataset where A_1 between 41.00 and 31,23 and A_2 <= 0.00

5.5 The modification operators

According to the fitness value, individuals are selected to reproduce with modification, creating the necessary genetic diversification that allows evolution in the long run. The following subsections proceed with the detailed description of the necessary GAPBNF operators.

a. Crossover operator

The crossover operator used in this work uses two parent genotypes (Parent1, Parent2) producing two offspring (Offspring1, Offspring2). This operator follows the principles of one-point crossover. Two parents (genotype) of different lengths are aligned with each other and two crossover points are chosen at random but one must consider the specified constraint, whereas if the two selected points conform to the CFG, then these two points are taken otherwise select other points until the correct points are found. The steps of doing the conformation process are presented in algorithm (2). The first point (Pc1) is randomly selected according to the length of the first parent; the second point is selected randomly according to the length of the second parent (Pc2). The tails of the second parent from the onward point are switched to create the first Offspring1, while the tails of the first parent from the point Pc1 are switched to create the second child Offspring2.

Through the job of this operator, one must consider the uniqueness of the attribute in Offspring1 and Offspring2, i.e., each attribute can occur only once. This is implemented to avoid the inconsistent genotype which leads to avoiding the inconsistence of the generated rule. The pseudo-code version of this operator is presented in algorithm (3).

Algorithm 2 The validation process

Input : Two point $gene_{pc1}$ and $gene_{pc2}$, and the previous gene of pc1
Output : True or false

1: **If** ($gene_{pc1}$ =30) and ($gene_{pc2}$ =10 or $gene_{pc1}$= 11) return true
2: **Else**
3: **If** ($gene_{pc1}$ represent the number of l.m rule) and ((previous of $gene_{pc1}$= represent the number of l.m rule) and ($gene_{pc2}$ =10 or $gene_{pc2}$ =11) return true
4: **Else**
5: **If** ($gene_{pc1}$ = represent the number of l.m rule) and ((previous of $gene_{pc1}$ = represent the number of attribute rule) and ($gene_{pc2}$ =10 or $gene_{pc2}$ =11)) then return true
6: **Else**
7: **If** ($gene_{pc1}$ =10) and ($gene_{pc2}$ =20 or $gene_{pc2}$ =21 or $gene_{pc1}$ =22) then return true
8: **Else**
9: **If** ($gene_{pc1}$ =11) and ($gene_{pc2}$ =30) then return true
10: **Else**
11: return false
12: **End algorithm**

b. The improvement operator

This operator is applied to modify the values that have been stored in the local memory of the individual. The dummy groups which correspond for each class is created in this operator. The centroids of the dummy groups for classi are denoted by the mean vector of all the data points for this class. Then the distance value is computed between the values that are stored in the local memory which were selected randomly and the corresponding values stored in the dummy groups. For instance, if local memory (l.m) = {lm1, lm3} which represents the values of attribute one and three that have been created during the generation process, and the dummy group of the corresponding class which is in the learning process is equal to = {D1, D2,….., DM}, where M is the number of attributes, this operator works as follows: in the beginning, if the value of the attribute stored in the local memory (let be lm3) is selected randomly, then the distance between lm3 and D3 (the centroid of the corresponding selected attribute) is computed (let it be d3). Then the stored value in l.m will be either increased or decreased depending on the d=(r*d3)/2 (the increased or decreased process is done randomly). After that the competition is done between the new created individual with its new local memory value and the current individual with the old local memory, the best one is chosen and the processing of this operator will be repeated until the specified condition is reached.

Algorithm 3 Crossover operator

Input: Local memory of parent$_1$ (l.m.p_1), local memory of parent$_2$ (l.m.p_2), pc_1= crossover point of parent$_1$, pc_2 =crossover point of parent$_2$, p_1_length, p_2_length which represents the length of parent$_1$ and parent$_2$ respectively.

Output: New individual (offspring)

```
1:  Index=1
2:  For i=1 to pc1 do
3:      If gen_i of parent_1 represent production rule 5 then
4:          l.m.offspring=l.m.offpring    l.m.p_1[index]
5:          Increment the value of index
6:      End if
7:      offspring_genotype=offspring_genotype    geni
8:  End for
9:  For i=pc2 to p_2_length do
10:     If gen_i of parent_2 represents production rule 5 then
11:       l.m.offspring=l.m.offspring    l.m.p_2[index]
12:       increment the value of index
13:     Endif
14:     If gen_i of parent_2 represents the production rule 4 then
15:       check the consistency of attribute
16:     Else
17:       Exit if there is no more attribute that constructs the genotype
18:     End if
19:   offspring_genotype=offspring_genotype    gen_i
20: End for
```

c. Other operators

The selection operator assigns to each chromosome a real number which is the target sampling rate that indicates the expected number of offspring to be generated from that chromosome, and gives the probability offspring to be generated from that chromosome, and gives the probability of the chromosome to be selected in the following sampling process.

The 2-tournament selection is chosen in this work. It works by randomly choosing 2 individual from the population and copying the best one of these 2 individuals to the new population.

An elitist reproduction strategy, where the best individual of each generation passes unaltered to the next generation is used.

The relational operator mutation is used. This operator modifies the relational operator currently being used in condition of the rule by replacing it with another one generated randomly.

Some extracted rules are removed if they are noisy or redundant rules [Han]. Noisy rules mean the rules that cover more examples from the other classes than from its own and redundant rules means the rules cover the same examples that covered by another rules which exist in the rule list.

5.6 The fitness function

The fitness function evaluates the quality of each rule (individual). Let tp, fp, fn denote respectively the number of true positive, false positives, and false negative observed when a rule is used to classify a set of examples. The fitness function combines two indicators, namely the confidence measures and coverage measures which is defined as follows:

$$Confidence = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fp}}, \tag{1}$$

$$\mathrm{Coverage} = \frac{\mathrm{tp}}{\mathrm{tp} + \mathrm{fn}} \tag{2}$$

Computing the value of tp and fp is very simple task because all it needs is to send the query to the training dataset and return the number of examples that match correctly the condition which belongs to classes C (let be Count(c)) and not C. While fn is calculated by using the following formula:

$$fn = Count(C) - tp \tag{3}$$

Finally, the fitness function used by MRule technique is defined as follows:

$$Fitness = 0.5 * Confidenc + 0.5 * Coverage \tag{4}$$

Each run of GAPBNFR solves a two class classification problem. Therefore, the GAPBNFR is run at least one for each class. Hence, the above formulas can be applied to the problem with

any number of classes [Takač]. Therefore, GAPBNFR is not necessary to encode class in the chromosome represenation.

6. Experimental results

In this section, the results on applying the proposed method to a number of benchmark problems are presented. For the comparison reasons, the same datasets that have been used with other based evolutionary approaches: Iris, Hearts, Breast, and Pima are adopted. These datasets form UCI machine learning datasets repository [Blake].
The proposed method is evaluated using 10_fold cross validation where the performance is the average accuracies over 10_fold. Comparison is made between the proposed method and ESIA[June] and clustering GP[Falco]. The results for ESIA and clustering GP reported here are taken directly from the above mentioned papers. Both algorithms also use 10-folds cross validation. This comparison is made in terms of predictive accuracy of the discovered rules. Table 2 shows the results of these three methods; N/A indicates that no results are available. As can be seen in table 2, the proposed method outperforms ESIA and Clustering GP in pima and breast data. The proposed method gives the same accuracy rate in heart data while outperforming ESIA. For iris datasets, the proposed method gives the same accuracy rate. As a result, we consider these results very promising, bearing in mind that, unlike ESIA and Clustering GP algorithms, the structure adopted in the proposed method is much simpler than the ESIA and clustering GP. Like most evolutionary algorithms, this method needs a substantial amount of computational time to run.

Datasets	GAPBNFR	ESIA	Clustering GP
Breast	**94.5**	**80.5**	**N/A**
Heart	**80.4**	**80.4**	**80.1**
Pima	**80.7**	**78.1**	**73.7**
iris	**94.5**	**95.33**	**N/A**

Table 2. Accuracy rate on test data of GAPBNFR, ESIA, Clustering GP

7. Discussions and conclusion

A new method for rule discovery has been developed for numerical datasets. This technique uses GAPBNF to perform this task. GAPBNF is an evolutionary algorithm that distinguishes between the genotype and the phenotype. The genotype is a list of integers representing productions in a defined syntax while phenotype is based on SQL operations in order to produce an SQL query. The most important points of the proposed method are: it performs global search of solutions space s and copes well with attribute interaction and producing a comprehensible classification rules. Its effectiveness through the evaluation process of individuals is inherent from one of the facilities provided by GAPBNF technique. The type of a representation scheme obtains efficient results and allows a compact representation of complex conditions using liner chromosomes. Our method incorporates innovative ideas with respect to the encoding of individuals which afterwards converted into the

corresponding rules. This facilities the work of crossover and mutation operators in which the GA engine can be used without any modification.
The proposed method is evaluated against four numerical public datasets and compared with two other evolutionary systems ESIA and GP clustering techniques. The results can be considered promising and allow us to conclude that the type of representation that is used in this work to discover a set of rules is efficient. In addition the good predictive accuracy that is obtained by our method stems from its ability to correctly predict the class label of previously unseen data. The performance of GAPBNF in the proposed technique is good enough to confirm its feasibility and is an important direction that further research should follow.

8. References

R. Sh. Abdul-Wahab, "Genetic Algorithm for Developing Program Using Backus Naur Form(GAPBNF), Engineering and Technology Journla, Vol 24, no. 6, 2005.

C. L. Blake and C.J. Merz. UCI Repository of Machine Learning Databases, University of California, Irvine, Dept. of Information and Computer Sciences, http://www.ics.uci.edu/simmlearn/MLRepository. html.1998

C. C. Bojarczuk, H. S. Lopes, A. A. Freitas. Discovering Comprehensible Classification Rules Using Genetic Programming: a case study in a medical domain.

E. Cantu-Paz, C. Kamath. "On the Use of Evolutionary Algorithms in Data Mining,in Data Mining". In Data Mining: A heuristic Approach, 2001.

D. Dasgupta, F. A. González. " Evolving Complex Fuzzy Classifier Rules Using a Linear Genetic Representation". In the proceedings of the International Conference Genetic and Evolutionary Computation (GECCO), San Francisco, California, July 7-11, 2001.

M. H. Dunham. Data Mining Techniques and Algorithms, Prentice Hall.2000.

D. Falco, A. Della Cioppa, A. Lassetta, E. Tarantino, Evolutionary Approach for Automatically Extracting Intelligible Classification Rules, Knowledge and Information System. Springer Verlag London Lid, pp 179-201, 2004.

A. A. Freitas. "Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction". In John R. Koza and Kalyanmoy Deb and Marco Dorigo and David B. Fogel and Max Garzon and Hitoshi Iba and Rick L. Riolo (Eds.) Genetic Programming 1997: Proceedings of the Second Annual Conference. Morgan Kaufmann. pp 96-101.1997

A. A. Freitas. Data Mining and Knowledge Discovery with Evolutionary Algorithms (Natural Computing Series). Springer,2002.

J. Han, M. Kamber. Data Mining: Concepts and Techniques, Academic press. 2001.

J. June, J. T. Kwok. "An Extended Genetic Rule Induction Algorithm". In Proceedings of the 2000 Congress on Evolutionary Computation CECoo,pp 458-463.2000.

W. B. Langdon, R. Poli. Foundation of Genetic Programming, Springers, 2004.

Norman R. Paterson, Mike Livesey, "Distinguishing Genotype and Phenotype in Genetic Programming", in Koza, J.R. (Ed.), Late breaking papers at the genetic programming 1996 Conf. Stanford Univ., July, pp141-150. 1996.

Pang_Ning Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, Pearson Education, 2006.

A. Takač. Genetic Programming in Data Mining: Cellular Approach. Institute of Informatics Faculty of Mathematics, Physics and Informatics Comenius University, Bratislava, Slovakia. MS.cThesis. 2003.