Genetic Programming for Generating Prototypes in Classification Problems

L. P. Cordella
DIS, Università di Napoli
Via Claudio, 21 80125
Napoli – Italy
cordel@unina.it

C. De Stefano
DEIMI, Università di Cassino
Via G. Di Biasio, 43 02043
Cassino (FR) – Italy
destefano@unicas.it

F. Fontanella
DIS, Università di Napoli
Via Claudio, 21 80125
Napoli – Italy
frfontan@unina.it

A. Marcelli
DIIIE, Università di Salerno
Via Ponte don Melillo, 1
84084
Fisciano (SA) – Italy
amarcelli@unisa.it

Abstract- We propose a genetic programming based approach for generating prototypes in a classification problem. In this context, the set of prototypes to which the samples of a data set can be traced back is coded by a multitree, i.e. a set of trees, which represents the chromosome. Differently from other approaches, our chromosomes are of variable length. This allows coping with those classification problems in which one or more classes consist of subclasses. The devised approach has been tested on several problems and the results compared with those obtained by a different genetic programming based approach recently proposed in the literature.

1 Introduction

Classification is a very important task in the machine learning context. In the most common case, the problem implies a supervised training phase in which a set of data samples, each labeled with the name of the class it belongs to, is provided to the system. From such data, classification rules, decision trees, or mathematical functions can be inferred and afterwards employed to classify unknown data. Many methods have been proposed for solving classification problems [1] and many of them are based on mathematical theories. The probability-based methods, for instance, have been founded on Bayesian decision theory [2].

In the last years several modern computational techniques have been introduced for developing new classifiers [3, 4]. Among others, evolutionary computation techniques have been also employed. In this field, genetic algorithms [5, 6] and genetic programming [7, 8] have mostly been used. The former approach encodes a set of classification rules as a sequence of bit strings. In the latter approach instead, such rules, or even classification functions, can be learned. The technique of Genetic Programming (GP) was introduced by Koza [8] in 1987 and has been applied to several problems like symbolic regression, robot control programming, classification, etc. GP based methodologies have demonstrated the ability to discover underlying data relationships and to represent these relationships by expressions. GP has already been successfully used in many different applications [9, 10, 11]. Although genetic algorithms have often been used for dealing with classification problems, only recently some attempts have been made to solve such problems using GP [12, 13, 14, 15]. In [13], GP has been used to evolve equations (encoded as derivation trees) involving simple arithmetic operators and feature variables, for hyper-spectral image classification. In [12], GP has also been employed for image classification problems, adding exponential functions, conditional functions and constants to the simple arithmetic operators. In [15], an interesting method which considers a c-class problem as a set of c two-class problems has been introduced. In all the above quoted approaches, the number c of classes to be dealt with is used to divide the data set at hand in exactly c *clusters*. Thus, these approaches do not take into account the existence of subclasses within one or more of the classes in the analyzed data set.

We present a new GP based method for determining the prototypes in a c-class problem. In the devised approach, the prototypes describing samples belonging to c different classes, with $c \geq 2$, consist of logical expressions. Each prototype is representative of a cluster of samples in the training set and consists of a set of assertions (i.e. logical predicates) connected by Boolean operators. Each assertion establishes a condition on the value of a particular feature of the samples in the data set to be analyzed. The number of expressions is variable and may be greater or equal to the number of classes of the problem at hand. In fact, in many classification problems a single class may contain a variable number of subclasses. Hence, c expressions may not be able to effectively classify all the samples, since a single expression might be inadequate to express the characteristics of all the subclasses present in a class. The devised approach, instead, is able to automatically finding all the subclasses present in the data set, since a class can be represented by a variable number of logical expressions. The length of a single expression, i.e. the number of predicates contained in it, is also variable. Each expression may represent either a class or a subclass of the problem. The proposed method works according to the evolutionary computation paradigm.

The set of prototypes describing the classes makes up a *single* individual of the evolving population. Each prototype is encoded as a derivation tree, thus an individual is a multi-tree (i.e a list of trees). Given an individual and a sample, classification consists in attributing the sample to one of the classes (i.e. in associating the sample to one of the prototypes). The recognition rate obtained on the training set when using an individual is assigned as fitness value to that individual. At any step of the evolution process, individuals are selected according to their fitness value. At the end of the process, the best individual obtained, constitutes the set of prototypes to be used for the considered application. Our method for automatic prototyping has been tested on three publicly available databases and the classification results have been compared with those obtained by another GP based approach [16]. In this method individuals are also represented by lists of trees, but expressions involve simple arithmetic operators and constants. Differently from our approach, the number of expressions making up an individual is a priori fixed.

The paper is organized as follows: Section 2 reports a formalization of data classification; Section 3 illustrates the approach used to classify the data; in Section 4 the implementation of the evolutionary approach is presented, while Section 5 reports the experimental results. Finally, Section 6 is devoted to the conclusions.

2 Data Classification

In the data classification context a set of objects to be analyzed is called *data set*, and each object is called *sample* and represented by $\mathbf{X} = (x_1, \ldots, x_\ell)$ with $\mathbf{X} \in \mathbf{S}$, where $\mathbf{S}$ is the universe of all possible elements characterized by ℓ features and x_i denotes the i–th feature of the sample. A data set with cardinality N_D is denoted by $\mathcal{D} = \{\mathbf{X}_1, \ldots, \mathbf{X}_{N_D}\}$ with $\mathcal{D} \subseteq \mathbf{S}$. The set $\mathcal{D}$ is said *labeled* if it exists a set of integers:

$$\Lambda = \{\lambda_1, \ldots, \lambda_{N_D}\} : \lambda_i \in [1, c]$$

The i–th element λ_i of Λ is said the *label* of the i–th sample $\mathbf{X}_i$ of $\mathcal{D}$. We will say that the samples of $\mathcal{D}$ can be grouped into c different classes. Moreover, given the sample $\mathbf{X}_i$ and the label $\lambda_i = j$, we will say that $\mathbf{X}_i$ belongs to the j–th class.

Given a data set $\mathcal{D} = \{\mathbf{X}_1, \ldots, \mathbf{X}_{N_D}\}$ containing c classes, a classifier Γ is defined as a function

$$\Gamma : \mathcal{D} \longrightarrow [0, c]$$

In other words, a classifier assigns a label $\gamma_i \in [0, c]$ to each input sample $\mathbf{X}_i$.
If $\gamma_i = 0$, the corresponding sample $\mathbf{X}_i$ is said *rejected*. This fact means that the classifier is unable to trace the sample back to any class.

The sample $\mathbf{X}_i$ is *recognized* by Γ if and only if:

$$\gamma_i = \lambda_i$$

otherwise the sample is said *misclassified*. If N_{corr} is the number of samples of $\mathcal{D}$ recognized by Γ the ratio N_{corr}/N_D is defined as the *recognition rate* of the classifier Γ obtained on the data set $\mathcal{D}$.

3 Description of the Approach

In our approach a set of prototypes, each characterizing a different class or subclass, consists of a set of logical expressions. Each expression may contain a variable number of predicates holding for the samples belonging to one class in the training set taken into account. A predicate establishes a condition on the value of a particular feature. If all the predicates of an expression are satisfied by the values in the feature vector describing a sample, we say that the expression *matches* the sample. Training the classifier is accomplished by means of the evolutionary computation paradigm described in the following Section 4 and provides a set of labeled expressions (i.e. of labeled prototypes). Note that different expressions may have the same label in case they represent subclasses of a class. Given a data set and a set of labeled expressions, the classification task is performed in the following way: each sample of the data set is matched against the set of expressions and *assigned* to one of them (i.e. to a subclasses) or rejected. Different cases may occur:

1. The sample is matched by just one expression: it is assigned to that expression.
2. The sample is matched by more than one expression with different number of predicates: it is assigned to the expression with the smallest number of predicates.
3. The sample is matched by more than one expression with the same number of predicates and different labels: the sample is rejected.
4. The sample is matched by no expression: the sample is rejected.
5. The sample is matched by more than one expression with equal label: the sample is assigned to the class the expressions belong to.

Hereinafter, this process will be referred to as *assignment* process, and the set of samples assigned to the same expression will be referred to as *cluster*.

4 Learning Classification Rules

As mentioned in the introduction, the prototypes to be used for classification are given in terms of logical expressions. Since logical expressions may be thought of as computer programs, a natural way for introducing them in our learning system is that of adopting the GP paradigm. Our GP based system starts by randomly generating a population of p individuals. The individual phenotype is a string containing a set of logical expressions, each one encoded as a derivation tree. Hence, the chromosome of the individual is a multitree (i.e. a list of trees), and the number of trees in its chromosome will be referred in the following as the *length* of the individual. The length of the individuals in the initial population ranges from 2 to L_{max}. Afterwards, the fitness of such individuals is evaluated. In order to generate a new population, first the best e individuals are selected and copied in the new population so as to implement an elitist strategy. Then $(p-e)/2$ couples of individuals are selected using the tournament method, so as to control loss of diversity and selection intensity [17]. The crossover operator is applied to each of the selected couples, according to a chosen probability factor p_c. Then, the mutation is applied to the individuals according to a probability factor p_m. Finally, these individuals are added to the new population. The process just described is repeated for N_G generations. The following steps must be executed before implementing the above paradigm:

- definition of the structure to be evolved;
- choice of the fitness function;
- definition of the genetic operators;

4.1 Structure Definition

The implementation requires a program generator, providing syntactically correct programs, and an interpreter for executing them. The program generator is based on a grammar written for S-expressions. A grammar $\mathcal{G}$ is defined as a quadruple $\mathcal{G} = (\mathcal{T}, \mathcal{N}, S, \mathcal{P})$, where $\mathcal{T}$ and $\mathcal{N}$ are disjoint finite alphabets. $\mathcal{T}$ is said the *terminal alphabet*, whereas $\mathcal{N}$ is said the *nonterminal alphabet*. S, is *the starting symbol* and $\mathcal{P}$ is the set of *production rules* used to define the strings belonging to the language, usually indicated by $v \longrightarrow w$ where v is a string on $(\mathcal{N} \cup \mathcal{T})$ containing at least one nonterminal symbol, and w is an element of $(\mathcal{N} \cup \mathcal{T})^*$. The grammar employed is given in Table 1.

As mentioned above, the chromosome of each individual consists of a variable number of derivation trees. The root of every tree is the symbol S that, according to the related production rule, can be replaced only by the string "A$". The symbol A can be replaced by any recursive combination of logical predicates, each one establishing the constraints on the value of a feature. The terminal symbol $ has been introduced to delimit different logical expressions within the phenotype of an individual. Summarizing, the genotype of each individual is seen as a list of derivation trees whose leaves are the terminal symbols of the grammar defined for constructing the set of logical expressions, i.e. the prototypes. The set of logical expressions making up the phenotype of an individual is obtained by visiting each derivation tree in depth first order and copying into a string the symbols contained in the leaves. In such string, the first logical expression derives from the first tree in the list, the second one derives from the second tree in the list and so on. Since the grammar is non-deterministic, to reduce the probability of generating too long expressions (i.e. too deep trees) the action carried out by a production rule is chosen on the basis of fixed probability values (shown in the last column of Table 1). Moreover, an upper limit has been imposed on the total number of nodes contained in an individual, i.e. the sum of nodes contained in each tree. Examples of chromosomes are shown in Fig. 1.

Number	Rule	Probability
1	$S \longrightarrow A\$$	1.0
2	$A \longrightarrow ABA \mid D$	0.2, 0.8
3	$B \longrightarrow \vee \mid \wedge$	equiprobable
4	$D \longrightarrow (P > V) \mid (P < V)$	equiprobable
5	$P \longrightarrow a_0 \mid a_1 \mid \ldots \mid a_N$	equiprobable
6	$V \longrightarrow +0.XX \mid -0.XX$	equiprobable
7	$X \longrightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$	equiprobable

Table 1: The grammar for randomly generating logical expressions. N is the dimension of the feature space. Nonterminal symbols are denoted by capital letters.

The interpreter is implemented by an automaton which computes Boolean functions. Such an automaton accepts as input an expression and a sample and returns as output the value true or false depending on the fact that the expression matches or not the sample.

4.2 Training Phase and Fitness Function

The system is trained with a set containing N_{tr} patterns. The training set is used for evaluating the fitness of the individuals in the population. This process implies the following steps:

1. The assignment of the training set samples to the expressions belonging to each individual is performed. After this step, n_i ($n_i \geq 0$) samples have been assigned to the i-th expression. Note that each expression for which $n_i > 0$ is associated with a cluster. In the following these expressions will be referred to as *valid*. The expressions for which $n_i = 0$ will be

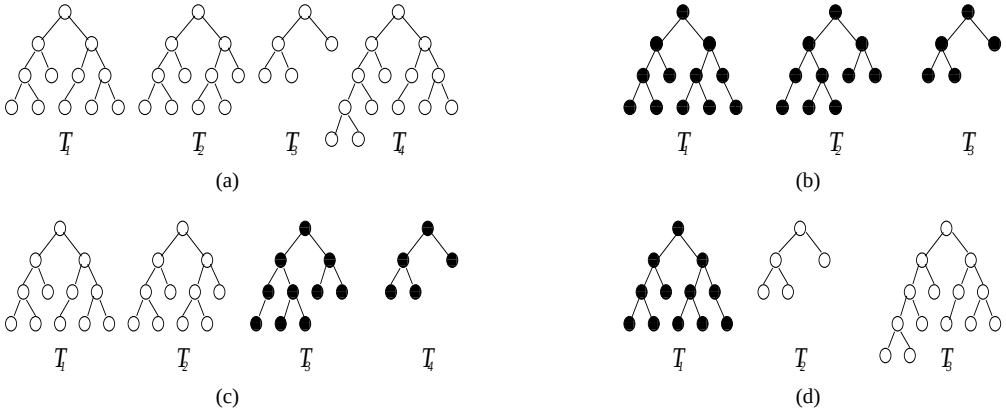


Figure 1: (a) and (b) Chromosomes C_1 and C_2 of length respectively equal to 4 and 3. (c) and (d) The chromosomes obtained after applying the crossover operation with $t_1 = 2$ and $t_2 = 1$.

ignored in the following steps.

2. Each valid expression of an individual is labeled with the label most widely represented in the corresponding cluster.

3. For every individual (i.e. a set of prototypes) a classifier is built up and its recognition rate is evaluated. Such rate is assigned as fitness value to the individual.

In order to favor those individuals able to obtain good performances with a lesser number of expressions, the fitness of each individual is increased by $0.1/N_c$, where N_c is the number of expressions in an individual.

4.3 Genetic Operators

The genetic operators are applied to the lists of derivation trees encoding the individuals defined in Section 4.1. This allows us to implement the genetic operators in a simple way acting either on the lists or on the trees. Two operators are defined for modifying the individuals of the population: *crossover* and *mutation*. Both the operators preserve the syntactic correctness of the expressions representing the new individuals generated.

4.3.1 Crossover

The crossover operator is applied to two chromosomes C_1 and C_2 and yields two new chromosomes by swapping parts of the lists of the initial chromosomes. Assuming that the length of C_1 and C_2 are respectively L_1 and L_2, the crossover is applied in the following way: the first chromosome is split in two parts by randomly choosing an integer t_1 in the interval $[1, L_1]$. The obtained multitrees $C_1^{'}$ and $C_1^{''}$ will have length t_1 and $L_1 - t_1$ respectively. Analogously, by randomly choosing an integer t_2 in the interval $[1, L_2]$, two multitrees $C_2^{'}$ and $C_2^{''}$, respectively of length t_2 and $L_2 - t_2$, are obtained from C_2. At this stage, in order to obtain a new chromosome, the lists $C_1^{'}$ and $C_2^{''}$ are merged. This operation yields a new chromosome of length $t_1 + L_2 - t_2$. The same operation is applied to the remaining lists $C_2^{'}$ and $C_1^{''}$ and a new chromosome of length $t_2 + L_1 - t_1$ is obtained.

From the phenotype perspective, the result of applying the crossover is swapping substrings containing entire logical expressions. The number of the swapped expressions depends on the integers t_1 and t_2. It is worth noting that the implemented crossover operator allows us to obtain chromosomes of variable length. Hence, during the evolution process, individuals made of a variable number of prototypes can be evolved.

4.3.2 Mutation

The mutation operator is independently applied to every tree of the chromosome C with probability p_m. More specifically, given a tree T_i, the mutation operator is applied by randomly choosing a *single* nonterminal node in T_i and then activating the corresponding production rule in order to substitute the subtree rooted under the chosen node. It is important to note that if the chromosome C contains n nonterminal nodes and p_m is the mutation probability, the probability of mutating each single node of C is equal to p_m/n.

The effect of the mutation depends on the nonterminal symbol chosen. In fact, this operation can result either in the substitution of the related subtree, causing a macromutation, or in a simple substitution of a leaf node (micromutation). For instance, considering the grammar of Table 1, if a node containing the symbol D is chosen, then the whole corresponding subtree is substituted. In the phenotype, this operation causes the substitution of the predicates encoded by the old subtree with those encoded by the new

generated subtree. If, instead, the symbol X is chosen, only a leaf of the tree is substituted, causing, in the phenotype, the substitution of a single digit with one of those in the right side of the rule.

5 Experimental Results

Three data sets have been used for training and testing the previously described approach. The sets are made of real data and are available at UCI site [18] with the names IRIS, BUPA and Vehicle (see Table 2).

IRIS This is the well known Anderson's Iris data set [19]. It consists of 150 samples characterized by four features representing measures taken on iris flowers of three different classes, equally distributed in the set. The four features are sepal length, sepal width, petal length and petal width.

BUPA The BUPA liver disorders data set consists of 345 samples distributed in two classes of liver disorders. Six features characterize each sample.

Vehicle The samples of this data set are images of 3D objects (vehicles). The data set is made of 846 samples distributed in four classes. Each sample is described by a vector of 18 features.

In order to use the grammar shown in Table 1 the features of the data sets taken into account have been normalized in the range $[-1.0, 1.0]$. Given a not normalized sample $\mathbf{X} = (x_1, \ldots, x_N)$, every feature x_i is normalized using the formula:

$$x_i = \frac{x_i - \overline{x}_i}{2\sigma_i}$$

where $\overline{x}_i$ and σ_i, respectively represent the mean and the standard deviation of the i-th feature computed over the whole data set.

Each dataset has been divided in two parts, a training set and a test set. These sets have been randomly extracted from the data sets and are disjoint and statistically independent. The first one has been used during the training phase to evaluate, at each generation, the fitness of the individuals in the population. The second one has been used at the end of the evolution process to evaluate the performance of our method. In particular, the recognition rate over the test set has been computed using a classifier implemented by choosing the best individual generated during the training phase. For the sake of comparison, at each generation, the best individual in the population has been considered and the obtainable recognition rates on both training set and test set, have been evaluated.

Name	**Classes**	**Features**	**Size**
IRIS	3	4	150(50+50+50)
BUPA	2	6	345(145+200)
Vehicle	4	18	846(212+217+218+199)

Table 2: The data sets used in the experiments. The class distribution is shown between brackets in the last column.

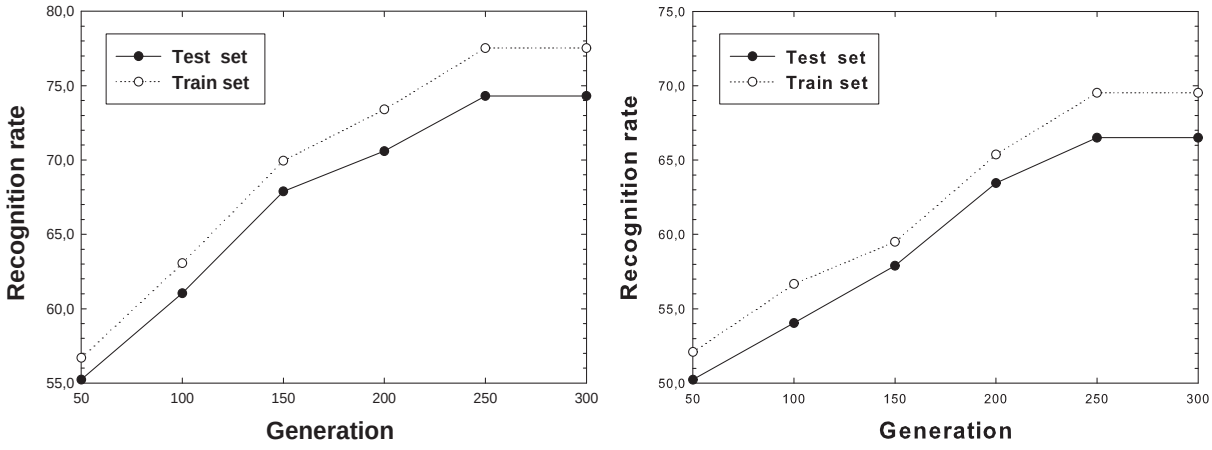
Parameter	symbol	value
Population size	p	200
Tournament size	$\mathcal{T}$	10
elithism size	e	5
Crossover probability	p_c	0.4
Mutation probability	p_m	0.8
Number of Generations	N_G	300
Maximum number of nodes	N_{max}	1000
Maximum length of an individual	L_{max}	20

Table 3: Values of the basic evolutionary parameters used in the experiments.

The values of the evolutionary parameters, used in all the performed experiments, have been heuristically determined and are summarized in Table 3. The performance of the proposed classification scheme has been evaluated by a 10-fold cross validation procedure: the data set has been divided in ten parts, alternatively used as test set. For each given test set ten runs have been performed with different initial population, but keeping unchanged all the other parameters. Hence, 100 runs have been performed for each data set. Such protocol has been used for comparing our results with those reported in the literature.

In a learning process, in most cases, when the maximum recognition rate is achieved, the generalization power, i.e. the ability of obtaining the same rate on a different data set (the test set), may not be the best. In order to investigate such ability for our system, the recognition rates on training and test set have been taken into account for the different considered data sets. In Figure 2 such recognition rates, evaluated every 50 generations, in a typical run for the BUPA and Vehicle data sets, are displayed. It can be observed from the figure that, in the experiments carried out, the recognition rate increases with the number of generations both for the training set and for the test set. The best recognition rates occurring in both cases nearby generation 250. Moreover, the fact that the difference between the two recognition rates tend to increase when that on the training set reaches its maximum, suggests that the use of a validation set could further improve the classifier performances.

The proposed approach has been compared with another GP based approach [16] in which an individual consists of a set of expressions (i.e. prototypes) involving simple arithmetic operators, constants and feature variables. Each ex-



(a) (b)

Figure 2: Recognition rate on training and test sets for the BUPA (a) and Vehicle (b) data sets during a typical run.

pression establishes conditions on the values of a variable number of features characterizing the data to be analyzed. The samples are matched against the expressions and assigned to the one satisfying the constraints on feature values.

Similarly to our approach, each individual is encoded as a multitree, but the number of trees (i.e. expressions) for each individual is constant and a priori fixed equal to the number of classes of the problem at hand. Moreover, each tree is a priori labeled: the first tree with the label of the first class, the second tree with that of the second class and so on. As a consequence, a sample belonging to the i–class is correctly classified only if it is assigned to the i–tree of the individual.

In Table 4 the recognition rates obtained on the test set by the two methods are shown. Since the GP approach is a stochastic algorithm, the standard deviations are also shown. Moreover, the average number of clusters found by our method (represented by valid expressions in the considered individual) and the related standard deviation are reported.

For the IRIS data set, 99.4% of the test set has been correctly recognized by our classification system, against 98.67% obtained by the method considered for comparison. Instead, for the BUPA data set our method is able of correctly recognizing 74.3% of the test set, a rate significantly better than that obtained in [16] (69.87%). Also for the Vehicle data set, our method performs significantly better than the comparison method, achieving on the test set the recognition rate of 66.5%, against 61.75% obtained by the other method. Summarizing, the experimental results show that the proposed method outperforms the method used for comparison on all the data sets taken into account. Thus, the comparison carried out confirms the validity and the effectiveness of the proposed approach.

Data sets	R_2	$\mathbf{R_1}$	σ_{R_1}	N_{C_1}	$\sigma_{N_{C_1}}$
IRIS	98.67	**99.4**	0.5	3.03	0.2
BUPA	69.87	**74.3**	3.0	2.36	0.5
Vehicle	61.75	**66.5**	2.0	4.8	0.6

Table 4: The average recognition rates (%) R_1 and R_2 for our classifier C_1 and the comparison classifier C_2. The standard deviation in case of C_1 is given. The average number N_c of data clusters found by C_1 and the related standard deviation are also shown.

6 Conclusions

A new genetic programming based approach to prototype generation and classification has been proposed. According to the approach, a population is evolved where each individual consists of a set of possible prototypes of the classes present in the data set to be analyzed. A prototype consists of a set of logical expressions establishing conditions on feature values and thus describing classes of data samples. The recognition rate obtained using each given set of prototypes is used as fitness function for controlling the evolution. The method is able to automatically clustering the data, without forcing the system to find a predefined number of clusters. This means that a class is neither necessarily represented by one single prototype nor by a fixed number of prototypes. On the contrary, other methods, namely the one used for comparison, a priori set the number of possible clusters. The greater flexibility of our method depends on the dynamic labeling mechanism of logical expressions, which allows us to reduce the constraints imposed during the prototyping process.

The proposed approach has been tested on three publicly

available data sets and the obtained results have been compared with those obtained by another GP based approach reported in the literature. The comparison has shown that the results obtained by our system on the data sets taken into account are significantly better than those obtained by the other approach.

Bibliography

[1] Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Classification*, John Wiley & sons, Inc., 2001.

[2] David Heckerman and Michael P. Wellman, "Bayesian Networks.", *Communications of the ACM*, vol. 38, n. 3, pp. 27–30, 1995.

[3] Guoqiang Peter Zhang, "Neural networks for classification: a survey.", *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 30, n. 4, pp. 451–462, 2000.

[4] J. Ross Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., 1993.

[5] John H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*, MIT Press, 1992.

[6] David E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., 1989.

[7] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, USA, 1992.

[8] John R. Koza, *Genetic programming II: automatic discovery of reusable programs*, MIT Press, Cambridge, MA, USA, 1994.

[9] Riccardo Poli, "Genetic Programming for Image Analysis", Technical Report CSRP-96-1, University of Birmingham, UK, January 1996.

[10] Andreas Bastian, "Identifying fuzzy models utilizing genetic programming", *Fuzzy Sets and Systems*, vol. 113, n. 3, pp. 333–350, 2000.

[11] Mario Koppen and Bertram Nickolay, "Genetic programming based texture filtering framework", *Pattern recognition in soft computing paradigm*, pp. 275–304, 2001.

[12] Davide Agnelli, Alessandro Bollini and Luca Lombardi, "Image classification: an evolutionary approach.", *Pattern Recognition Letters*, vol. 23, n. 1-3, pp. 303–309, 2002.

[13] Patrick J. Rauss, Jason M. Daida and Shahbaz A. Chaudhary, "Classification of Spectral Image Using Genetic Programming.", in *GECCO*, pp. 726–733, 2000.

[14] Ivanoe De Falco, Antonio Della Cioppa and Ernesto Tarantino, "Discovering interesting classification rules with genetic programming.", *Appl. Soft Comput.*, vol. 1, n. 4, pp. 257–269, 2002.

[15] J. K. Kishore, L. M. Patnaik, V. Mani and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification", *IEEE Transactions on Evolutionary Computation*, vol. 4, n. 3, pp. 242–258, September 2000.

[16] Durga Prasad Muni, Nikhil R. Pal and Jyotirmoy Das, "A novel approach to design classifiers using genetic programming.", *IEEE Trans. Evolutionary Computation*, vol. 8, n. 2, pp. 183–196, 2004.

[17] Tobias Blickle and Lothar Thiele, "A Comparison of Selection Schemes Used in Genetic Algorithms", Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland, 1995.

[18] C.L. Blake and C.J. Merz, "UCI Repository of machine learning databases", 1998.

[19] E. Anderson, "The Irises of the gaspe peninsula", *Bull. Amer. IRIS Soc.*, vol. 59, pp. 2–5, 1935.