Looking for Prototypes by Genetic Programming

L. P. Cordella[1], C. De Stefano[2], F. Fontanella[1], and A. Marcelli[3]

[1] Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II,
Via Claudio, 21 80125 Napoli – Italy
{cordel, frfontan}@unina.it
[2] Dipartimento di Automazione, Elettromagnetismo, Ingegneria dell'Informazione e Matematica Industriale
Università di Cassino
Via G. Di Biasio, 43 02043 Cassino (FR) – Italy
destefano@unicas.it
[3] Dipartimento di Ingegneria dell'Informazione e Ingegneria Elettrica
Università di Salerno
84084 Fisciano (SA) – Italy
amarcelli@unisa.it

Abstract. In this paper we propose a new genetic programming based approach for prototype generation in Pattern Recognition problems. Prototypes consist of mathematical expressions and are encoded as derivation trees. The devised system is able to cope with classification problems in which the number of prototypes is not a priori known. The approach has been tested on several problems and the results compared with those obtained by other genetic programming based approaches previously proposed.

1 Introduction

Several modern computational techniques have been introduced in the last years in order to cope with classification problems [1,2,3]. Among others, evolutionary computation (EC) techniques have been also employed. In this field, genetic algorithms [4,5] and genetic programming [6,7] 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 [7] and has already been successfully used in many different applications [8,9], demonstrating its ability to discovering underlying data relationships and to representing them by expressions. Only recently, classification problems have been faced by using GP. In [10], GP has been used to evolve equations (encoded as derivation trees) involving simple arithmetic operators and feature variables. The method was tested on different type of data, including images. In [11], GP has also been employed for image classification, adding exponential functions, conditional functions and constants to the simple arithmetic operators. In both the above quoted approaches, the data set is divided in a number c of *clusters* equal to the number of predefined classes. Thus, these approaches do not take into account the existence of subclasses within one or more of the classes in the analyzed data set.

N. Zheng, X. Jiang, and X. Lan (Eds.): IWICPAS 2006, LNCS 4153, pp. 152–159, 2006.

We present a GP based method for determining a set of prototypes describing the data in a classification problem. In the devised approach, each prototype is representative of a cluster of samples in the training set, and consists of a mathematical expression involving arithmetic operators and variables representing features. The devised method is able to generate a variable number of expressions, allowing us to cope with those classification problems in which single classes may contain not a priori identifiable subclasses. Hence, a fixed number of expressions (prototypes) may not be able to effectively classify all the data samples, since a single expression might be inadequate to express the characteristics of all the subclasses present in a class. The proposed approach, instead, is able to automatically find the number of expressions needed to represent all the possible subclasses present in the data set.

According to our method, 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 list of trees, called *multitree*. 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.

A preliminary version of this method was presented in [12], where prototypes consisted of simple logical expressions.

The method presented here has been tested on three publicly available databases and the classification results have been compared with those obtained by the preliminary version of the method and with another GP based method presented in the literature [10].

2 Description of the Approach

In the approach proposed here, a prototype representing a class or subclass consists of a mathematical expression, namely an inequality, that may contain a variable number of variables connected by the four arithmetic operators (+,-,*,/). Each variable $x_i, (i = 1, \ldots, n)$ represents a particular feature. Note that an inequality characterizes a region of the feature space delimited by an hypersurface. Given an expression E and a sample represented by a feature vector $\mathbf{x}$, we say that E *matches* the sample $\mathbf{x}$ if the values in $\mathbf{x}$ satisfy the inequality E. Training the classifier is accomplished by the EC paradigm described in Section 3 and provides a set of labeled expressions to be used as prototypes. 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 class or subclass) 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 variables: it is assigned to the expression with the smallest number of variables.
3. The sample is matched by more than one expression with the same number of variables and different labels: the sample is rejected.
4. The sample is matched by no expression: the sample is rejected.

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*.

3 Learning Classification Rules

As already said, the prototypes to be used for classification are given in terms of inequalities, thus they may be thought of as computer programs and can be generated by adopting the GP paradigm. Our GP based system starts by randomly generating a population of p individuals. An individual is made by a set of prototypes each encoded as a derivation tree, so that it is a *multitree* (i.e. a list of trees). The number of trees making up an individual will be called *length* of the individual: in the initial population, it ranges from 2 to L_{max}. Afterwards, the fitness of the initial 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 and manipulated by using two genetic operators: crossover and mutation. 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 obtained 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. In order to implement the above system the following steps must be executed:

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

In the following each of these steps is detailed.

3.1 Structure Definition

In order to generate syntactically correct expressions (i.e., prototypes), a nondeterministic grammar is defined. A grammar $\mathcal{G}$ is a quadruple $\mathcal{G} = (\mathcal{T}, \mathcal{N}, S, \mathcal{P})$, where $\mathcal{T}$ and $\mathcal{N}$ are disjoint finite alphabets. $\mathcal{T}$ is the *terminal alphabet*, whereas $\mathcal{N}$ is 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. The grammar employed is given in Table 1.

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 "C". The symbol C can be replaced by any mathematical expression obtained by recursively combining variables, representing features, and operators.

Table 1. The context free grammar used for generating the expressions employed as prototypes. In the right column, the probability of being chosen for each of the right side clause is shown.

Number	Rule	Probability
1	$S \longrightarrow C$	1.0
2	$C \longrightarrow [E > V] \mid [E < V]$	equiprobable
3	$E \longrightarrow PFD \mid P$	0.4, 0.6
4	$D \longrightarrow PFD \mid P \mid V$	0.5, 0.25, 0.25
5	$F \longrightarrow * \mid + \mid / \mid -$	equiprobable
5	$P \longrightarrow x_0 \mid x_1 \mid \ldots \mid x_N$	equiprobable
6	$V \longrightarrow +0.XX \mid -0.XX$	equiprobable
7	$X \longrightarrow 0\|1\|2\|3\|4\|5\|6\|7\|8\|9$	equiprobable

Summarizing, each individual is a list of derivation trees whose leaves are the terminal symbols of the grammar defined for constructing the set of inequalities. The set of inequalities making up 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, each inequality derives from the corresponding tree in the list. 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 of each tree. Examples of individuals are shown in Fig. 1.

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

3.2 Training Phase and Fitness Function

The aim of the training phase is that of generating the prototypes. The system is trained with a set containing N_{tr} samples. During training, the fitness of each individual in the population has to be evaluated. This process implies the following steps:

1. The assignment of the training set samples to the expressions belonging to the individual is performed. After this step, n_i ($n_i \geq 0$) samples will have been assigned to the i-th expression. The expressions for which $n_i > 0$ will be referred to as *valid*, whereas the ones for which $n_i = 0$ will be ignored in the following steps.
2. Each valid expression is labeled with the label most widely represented in the corresponding cluster.
3. The recognition rate (on the training set) of the individual is evaluated and assigned as fitness value to that 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 k/N_e, where N_e is the number of expressions in the individual and k is a constant.

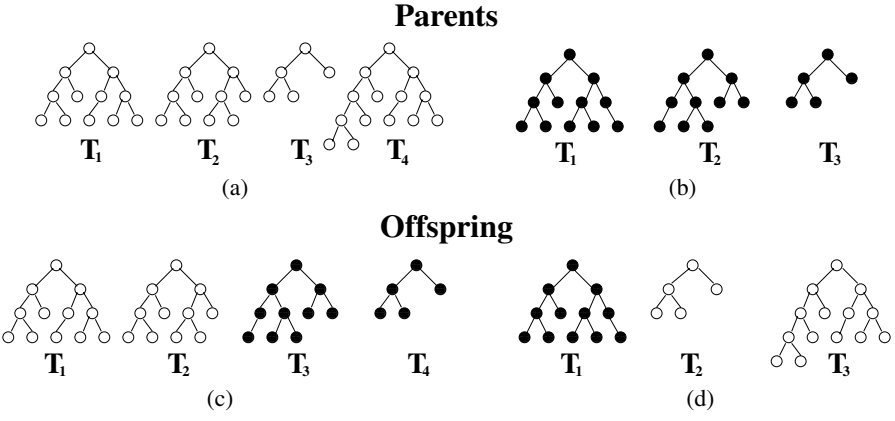

Fig. 1. An example of application of the crossover operator. The top figures (a and b)show a couple of individuals involved as parents of the crossover operator. The bottom figures (c and d) show the offspring obtained after the application of the operator. In this case case, t_1 and t_2 ha been chosen respectively equal to 2 and 1.

3.3 Genetic Operators

The choice of encoding the individuals as lists of derivation trees (see Section 3.1) allows us to implementing the genetic operators in a simple way.

The crossover operator is applied to two individuals I_1 and I_2 and yields two new individuals by swapping parts of the lists of the initial individuals (see Figure 1). Assuming that the lengths of I_1 and I_2 are respectively L_1 and L_2, the crossover is applied in the following way: the first individual is split in two parts by randomly choosing an integer t_1 in the interval $[1, L_1]$, so generating two multitrees $I_1^{'}$ and $I_1^{''}$, respectively of length t_1 and $L_1 - t_1$. Analogously, by randomly choosing an integer t_2 in the interval $[1, L_2]$, two multitrees $I_2^{'}$ and $I_2^{''}$ are obtained from I_2. Two new individuals are obtained: the first, by merging $I_1^{'}$ and $I_2^{''}$ and the second by merging $I_2^{'}$ and $I_1^{''}$.

It is worth noting that the implemented crossover operator allows us to obtain individuals of variable length. Hence, during the evolution process, individuals made of a variable number of prototypes can be evolved.

The mutation operator is independently applied to every tree of an individual I 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.

4 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 (http://www.ics.uci.edu/∼mlearn/MLSummary.html) with the names IRIS, BUPA and Vehicle.

IRIS is made of 150 samples of iris flowers of three different classes, equally distributed in the dataset. Four features, namely sepal length, sepal width, petal length and

petal width, are used for describing the samples. BUPA is made of 345 samples representing liver disorder using six features. Two classes are defined. The samples of the data set Vehicle are feature vectors representing 3D vehicle images. The data set has 846 samples distributed in four classes: 18 features characterize each sample.

In order to use the grammar shown in Table 1 the feature values 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: $x_i = (x_i - \overline{x}_i)/2\sigma_i$ 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 for classification the best individual generated during the training phase.

The values of the evolutionary parameters, used in all the performed experiments, have been heuristically determined and are: Population size = 500; Tournament size = 6; Elithism size = 5; Crossover probability = 0.5; Mutation probability = 0.3; Number of Generations = 300; Maximum number of nodes in an individual = 1000; maximum length of an individual = 20. The value of the constant k (see Subsection 3.2) has been set to 0.1.

In order to investigate the generalization power of our system, i.e. a measure of its performance on new data, the recognition rates both on training and test sets 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, are displayed for BUPA and Vehicle data sets. It can be seen that the recognition rate increases with the number of generations both for the training set and for the test set. The best recognition rates occur in both cases nearby generation 250 and then remain stationary.

The proposed approach has been compared with another GP based approach previously proposed in [10]. Furthermore, the results obtained by the preliminary version of the method [12] are also shown for comparison. The substantial difference between the new and the old version of the method consists in the form of the encoded expressions: in [12] each expression contains a variable number of logical predicates connected by Boolean operators. Each predicate represents an assertion establishing a condition on the value of a particular feature of the samples. This implies that the hypersurfaces

Table 2. The recognition rates R_{new}, R_{old} and R_{Muni} obtained respectively by the method presented here, its preliminary version and the method presented in [10]

Data sets	$\mathbf{R}_{\text{new}}$	R_{old}	R_{Muni}
IRIS	**99.6**	99.4	98.67
BUPA	**78.6**	74.3	69.87
Vehicle	**70.2**	66.5	61.75

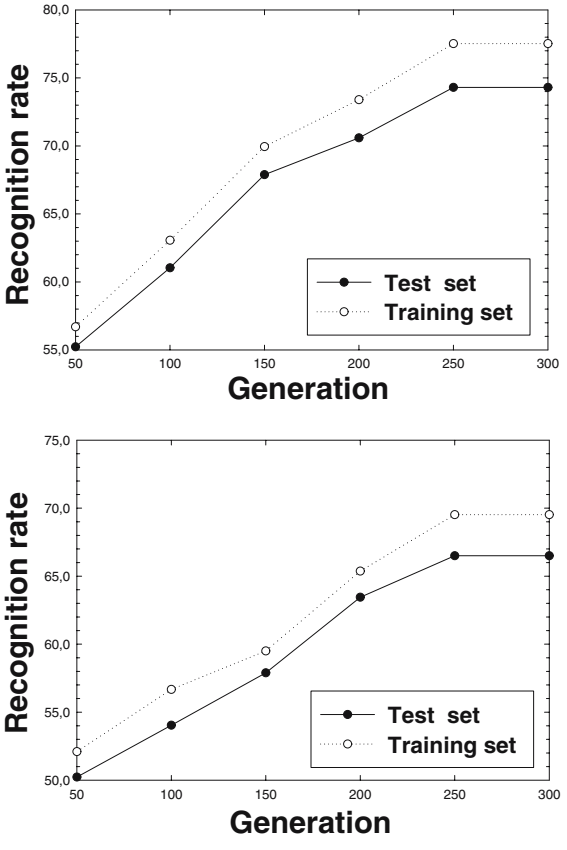

Fig. 2. Typical runs for BUPA (top) and Vehicle (bottom) datasets

separating the regions of the feature space belonging to different classes can only be hyperplanes parallel to the axes. In the new version of the method such hypersurfaces are of polynomial type, thus enabling a more effective separation between classes.

In Table 2 the recognition rates achieved on the test set by the three methods are shown. The results have been obtained by using the 10-fold cross validation procedure. Since the GP approach is a stochastic algorithm, the recognition rates have been averaged over 10 runs. Hence, 100 runs have been performed for each data set. Note that, in [10], the number of prototypes is a priori fixed, while in our method it is automatically found. The results show that the proposed method outperforms those used for comparison on all the data sets taken into account, confirming the validity of the approach.

5 Conclusions

A new GP based approach to prototype generation and classification has been proposed. A prototype consists of a set of mathematical inequalities establishing conditions on

feature values and thus describing classes of data samples. The method is able to automatically find the number of clusters in 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. A remarkable feature of our method is that the hypersurfaces separating the regions of the feature space belonging to different classes are of polynomial type, thus enabling an effective separation between classes. The results show that the proposed method outperforms those used for comparison.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. John Wiley & sons, Inc. (2001)
2. Zhang, G.P.: Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C **30** (2000) 451–462
3. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc. (1993)
4. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press (1992)
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc. (1989)
6. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)
7. Koza, J.R.: Genetic programming II: automatic discovery of reusable programs. MIT Press, Cambridge, MA, USA (1994)
8. Sette, S., Boullart, L.: Genetic programming: principles and applications. Engineering Applications of Artificial Intelligence **14** (2001) 727–736
9. Bastian, A.: Identifying fuzzy models utilizing genetic programming. Fuzzy Sets and Systems **113** (2000) 333–350
10. Muni, D.P., Pal, N.R., Das, J.: A novel approach to design classifiers using genetic programming. IEEE Trans. Evolutionary Computation **8** (2004) 183–196
11. Agnelli, D., Bollini, A., Lombardi, L.: Image classification: an evolutionary approach. Pattern Recognition Letters **23** (2002) 303–309
12. Cordella, L.P., De Stefano, C., Fontanella, F., Marcelli, A.: Genetic programming for generating prototypes in classification problems. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation. Volume 2., IEEE Press (2005) 1149–1155