A Novel Genetic Programming Based Approach for Classification Problems

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. A new genetic programming based approach to classification problems is proposed. Differently from other approaches, the number of prototypes in the classifier is not a priori fixed, but automatically found by the system. In fact, in many problems a single class may contain a variable number of subclasses. Hence, a single prototype, may be inadequate to represent all the members of the class. 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

In the last years several modern computational techniques have been introduced for developing new classifiers [1]. Among others, evolutionary computation techniques have been also employed. In this field, genetic algorithms [2] and genetic programming [3] 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 [3] in 1987 and has been applied to several problems like symbolic regression, robot control programming, classification, etc. GP based methodologies have demonstrated to be able to discover underlying data relationships and to represent these relationships by expressions.

GP has already been successfully used in many different applications [4,5]. 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 [6,7,8]. In [7], GP has been used to evolve equations (encoded as derivation trees) involving simple arithmetic operators and feature variables, for hyper-spectral image classification. In [6], GP has also been employed for image classification problems. In [8], 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 is encoded 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 all the classes make up an individual of the evolving population. 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 [9]. In this method individuals are also represented by 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.

2 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 Section 3 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*.

3 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. Such paradigm combines genetic algorithms and programming languages in order to evolve computer programs of different complexity for a given task. According to this paradigm, populations of computer programs are evolved by using the Darwin's principle that evolution by natural selection occurs when the replicating entities in the population possess the *heritability* characteristic and are subject to *genetic variation* and *struggle to survive* [2].

Typically, GP starts with an initial population of randomly generated programs composed of functionals and terminals especially tailored to deal with the problem at hand. The performance of each program in the population is measured by means of a *fitness* function, whose form also depends on the problem faced. After the fitness of each program has been evaluated, a new population is generated by selection, recombination and mutation of the current programs, and replaces the old one. This process is repeated until a termination criterion is satisfied. In order to implement such paradigm, the following steps have to be executed:

Table 1. The grammar for the random program generator. N is the dimension of the feature space. Nonterminal symbols are denoted by capital letters.

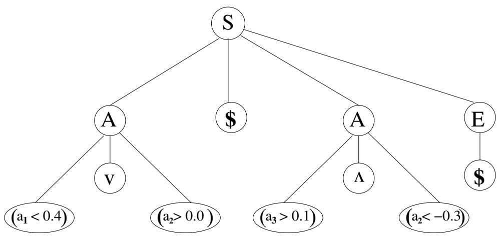
Rule number	**Rule**	**Probability**
1	$S \longrightarrow A\$AE$	1.0
2	$E \longrightarrow A\$E \mid \$$	0.2, 0.8
3	$A \longrightarrow ABA \mid C$	0.2, 0.8
4	$B \longrightarrow \vee \mid \wedge$	equiprobable
5	$D \longrightarrow (P > V) \mid (P < V)$	equiprobable
6	$P \longrightarrow a_0 \mid a_1 \mid \dots \mid a_N$	equiprobable
7	$V \longrightarrow +0.XX \mid -0.XX$	equiprobable
8	$X \longrightarrow 0\mid1\mid2\mid3\mid4\mid5\mid6\mid7\mid8\mid9$	equiprobable

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

3.1 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.

Each individual in the initial population is generated by starting with the symbol S that, according to the related production rule can be replaced only by the string "A$AE". The symbol A can be replaced by any recursive combination of logical predicates whose arguments are the occurrences of the elements in the feature vector. It is worth noting that the grammar has been defined so as to generate individuals containing at least two logical expressions. The role of the nonterminal symbol E and the corresponding production rule is that of adding new expressions to an individual. The terminal symbol $ has been introduced to delimit different logical expressions within an individual. Summarizing, each individual is seen as a unique derivation tree where the leaves are the terminal symbols of the grammar that has been defined for constructing the set of logical expressions to be used as prototypes. Usually, in the literature, in analogy with the phenomena of the natural evolution, a derivation tree is denoted as *genotype* or *chromosome*. Visiting a derivation tree in depth first order and copying into a string the symbols contained in the leaves, we obtain the desired set of logical expressions separated by the symbol $. This string is usually called *phenotype*. 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



$$(a_1 < 0.4) \vee (a_2 > 0.0)\$(a_3 > 0.1) \wedge (a_2 < -0.3)\$$$

Fig. 1. Example of an individual consisting of two expressions: its tree (the genotype or chromosome) and the corresponding string (the phenotype).

rule is chosen on the basis of fixed probability values (see Table 1). Moreover, an upper limit has been imposed on the total number of nodes contained in a tree. An example of chromosome of an individual is shown in Fig. 1.

The interpreter is implemented by an automaton which computes Boolean functions. Such an automaton accepts in 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.

3.2 Training and Fitness Function

The system is trained with a set containing N_{tr} samples. 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, p_i ($p_i \geq 0$) samples have been assigned to the i-th expression. Note that each expression for which $p_i > 0$ is associated with a cluster. In the following these expressions will be referred to as *valid.* The expressions for which $p_i = 0$ will be 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.

3.3 Selection Mechanism and Genetic Operators

The selection mechanism is responsible for choosing, in the current population, the individuals that will undergo genetic manipulation for producing the new population. The tournament method has been chosen as selection mechanism in order to control loss of diversity selection intensity [10].

As previously seen in Section 3.1, the individuals are encoded as derivation trees and represent the chromosomes to which the genetic operators are applied. This encoding allows to implement the actions performed by the genetic operators as simple operations on the trees. The individuals in the population are modified using two operators: *crossover* and *mutation*. Both these operators preserve the syntactic correctness of the expressions making up the new individuals generated.

The crossover operator works with two chromosomes C_1 and C_2 and yields two new chromosomes. These new chromosomes are obtained by swapping parts of the trees (i.e. subtrees) of the initial chromosomes C_1 and C_2. The crossover operates by randomly selecting a nonterminal node in the chromosome C_1 and a node of C_2 with the same nonterminal symbol. Then, it swaps the derivation subtrees rooted under the selected nodes. From a phenotype perspective, the result of applying the crossover is swapping substrings representing parts of two chromosomes. The sizes of the swapped parts depend on the nonterminal symbol chosen. For instance, with reference to Table 1, if the symbol chosen is A, then the swapped substrings contain at least an entire predicate or even an entire expression. On the contrary, if the symbol chosen is X, then only single digits of the two strings are swapped.

Given a chromosome C, the mutation operator is applied by randomly choosing a nonterminal node in C and then activating the corresponding production rule in order to substitute the subtree rooted under the chosen node. 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 macro-mutation, or in a simple substitution of a leaf node (micro-mutation). 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 B is chosen, only a leaf of the tree is substituted, causing, in the phenotype, the substitution of a boolean operator with one of those in the right side of the rule.

4 Experimental Results

Three publicly available data sets [11] have been used for training and testing the previously described approach (see Table 2). The IRIS data set is very fa-

Table 2. The class distribution is shown within the parentheses of the last column

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)

mous in the literature and consists of 150 samples of iris flowers belonging to three different classes equally distributed in the set. BUPA is a Liver Disorders data set, while the Vehicle data set is made of feature vectors describing vehicle images. 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 data set has been divided in two parts, the first one used as training set and employed in the fitness evaluation of the individuals, the second one used as test set in order to evaluate the classifier performance (i.e. the recognition rate) at the end of the evolution process. The two data sets are disjoint and randomly generated from the original data sets. The values of the evolutionary parameters, used in all the performed experiments, have been heuristically determined and are: Population size = 200; Tournament size = 10; Elithism size = 5; Crossover probability = 0.4; Mutation probability = 0.8; Number of Generations = 300; Maximum number of nodes in an individual = 1000.

The performance of the proposed classification scheme has been evaluated by averaging the recognition rate over 10 runs, using a 3-fold cross validation procedure. The data set has been divided into three parts alternatively used as test set. 10 runs have been performed with different initial population, but keeping unchanged all the other parameters. Hence, 30 runs have been performed for each data set. In Table 3 the results obtained by our method are shown in comparison with those obtained by the GP based approach presented in [9], in which the number of clusters to be found is a priori fixed and set equal to the number of classes of the problem at hand. Since the GP approach is a stochastic algorithm, the standard deviations are also shown. Moreover, the average numbers of found clusters (represented by valid expressions in the considered individual) and the related standard deviations are reported. The experimental results show that the proposed method obtains higher recognition rates than the method used for comparison, on all the data sets.

Table 3. The average recognition rates (%) for the compared classifiers and the average number of data clusters found by the system

Data sets	other GP	our GP	S.D.	N_c	σ_{N_c}
IRIS	98.67	99.4	0.5	3.03	0.2
BUPA	69.87	73.8	3.0	2.36	0.5
Vehicle	61.75	65.5	2.0	4.8	0.6

5 Conclusions

A new genetic programming based approach to classification problems has been proposed. The prototypes of the classes consist of logical expressions establishing conditions on feature values and thus describing clusters of data samples. The proposed method is able to automatically discover the clusters contained in the data, without forcing the system to find a predefined number of clusters. This means that a class is not necessarily represented by one single prototype or 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. As already mentioned, the labels of the expressions of each individual in the new population generated at every iteration, are assigned so as to maximize the recognition rate of the classifier based on that individual. The experimental results obtained on three publicly available data sets show a significant improvement with respect to those reported in the literature.

References

1. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. John Wiley & sons, Inc. (2001)
2. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. MIT Press (1992)
3. Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. Statistics and Computing **4** (1994)
4. Bastian, A.: Identifying fuzzy models utilizing genetic programming. Fuzzy Sets and Systems **113** (2000) 333–350
5. Koppen, M., Nickolay, B.: Genetic programming based texture filtering framework. Pattern recognition in soft computing paradigm (2001) 275–304
6. Agnelli, D., Bollini, A., Lombardi, L.: Image classification: an evolutionary approach. Pattern Recognition Letters **23** (2002) 303–309
7. Rauss, P.J., Daida, J.M., Chaudhary, S.A.: Classification of spectral image using genetic programming. In: GECCO. (2000) 726–733
8. Kishore, J.K., Patnaik, L.M., Mani, V., Agrawal, V.K.: Application of genetic programming for multicategory pattern classification. IEEE Transactions on Evolutionary Computation **4** (2000) 242–258
9. 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
10. Blickle, T., Thiele, L.: A comparison of selection schemes used in genetic algorithms. Technical Report 11, Gloriastrasse 35, 8092 Zurich, Switzerland (1995)
11. Blake, C., Merz, C.: UCI repository of machine learning databases (1998)