Grouping Character Shapes by Means of Genetic Programming

Claudio De Stefano[1], A. Della Cioppa[2], A. Marcelli[2], and F. Matarazzo[2]

[1] Facoltà di Ingegneria
Università del Sannio
Piazza Roma, Palazzo Bosco Lucarelli
Benevento, Italy
Ph. +39 0824 305839
destefan@unina.it,

[2] Dipartimento di Ingegneria dell'Informazione e Ingegneria Elettrica
Università di Salerno
Via Ponte Don Melillo
I–84084 Fisciano (SA), Italy
Ph. +39 089 964254, +39 089 964274
dean@unina.it, marcelli@diiie.unisa.it

Abstract. In the framework of an evolutionary approach to machine learning, this paper presents the preliminary version of a learning system that uses Genetic Programming as a tool for automatically inferring the set of classification rules to be used by a hierarchical handwritten character recognition system. In this context, the aim of the learning system is that of producing a set of rules able to group character shapes, described by using structural features, into super–classes, each corresponding to one or more actual classes. In particular, the paper illustrates the structure of the classification rules and the grammar used to generate them, the genetic operators devised to manipulate the set of rules and the fitness function used to match the current set of rules against the sample of the training set. The experimental results obtained by using a set of 5,000 digits randomly extracted from the NIST database are eventually reported and discussed.

1 Introduction

The recognition of handwritten character involves identifying a correspondence between the pixels of the image representing the samples to be recognized and the abstract definitions of characters (models or prototypes). The prevalent approach to solve the problem is that of implementing a sequential process, each stage of which progressively reduces the information passed to subsequent ones. This process, however, requires that decisions about the relevance of a piece of information have to be taken since the early stages, with the unpleasant consequence that a mistake may prevent the correct operation of the whole system. For this reason, the last stage of the process is usually by far the most complex one, in that it must collect and combine in rather complex ways many piece of information that, all together, should be able to recover, at least partially, the loss of information introduced in the previous ones. Moreover, due to the extreme variability exhibited by samples produced by a large population of writers, pursuing such an approach often requires the use of a large number of prototypes for each class, in order to capture the distinctive features of different writing styles. The combination of complex classification methods with large set of prototypes has a dramatic effect on the classifier: larger number of prototypes requires a finer disciminating power, which, in turn, requires more sophisticated methods and algorithms to perform the classification. While such complex methods are certainly useful for dealing with difficult cases, they are useless, and even dangerous, for simple ones; if the input sample happens to be almost noiseless, and its shape corresponds to the most "natural" human perception of a certain character, invoking complex classification schemes on large sets of prototypes results in overloading the classifier without improving the accuracy.

For this reason, we have investigated the possibility of using a preclassification technique whose main purpose is that of reducing the number of prototypes to be matched against a given sample while dealing with simple cases. The general problem of reducing a classifier computational cost has been faced since the 70's in the framework of statistical pattern recognition [1] and more recently within shape–based methods for character recognition [2,3]. The large majority of the preclassification methods for character recognition proposed in the literature belongs to one of two categories, depending on whether they differ in the set of features used to describe the samples or in the strategy adopted for labeling them [4]. Although they exhibit interesting performance, their main drawback is that a slight improvement in the performance results in a considerable increase of the computational costs.

In the framework of an evolutionary approach to character recognition we are developing [5], this paper reports a preliminary version of a learning system that uses Genetic Programming [6,7] as a tool for producing a set of prototypes able to group character shapes, described by using structural features, into super–classes, each corresponding to one or more actual classes. The proposed tool works in two different modes. During an off–line unsupervised training phase, rather rough descriptions of the shape of the samples belonging to the training set are computed from the feature set, and then allowed to evolve according to the Genetic Programming paradigm in order to achieve the maximum coverage of the training set. Then, each prototype is matched and labeled with the classes whose samples are matched by that prototype. At run time, after the feature extraction, the same shape description is computed for a given sample. The labels of the simplest prototype matching the sample represent the classes the sample most likely belongs to.

The paper is organized as follows: Section 2 describes the character shape description scheme and how it is reduced to a feature vector, which represent the description the Genetic Programming works on. In Section 3 we present our approach and its implementation, while Section 4 reports some preliminary experimental results and a few concluding remarks.

2 From Character Shape to Feature Vector

In the framework of structural methods for pattern recognition, the most common approach is based on the decomposition of an initial representation into elementary parts, each of which is simply describable. In this way a character is described by means of a structure made up by a set of parts interrelated by more or less complex links. Such a structure is then described in terms of a sentence of a language or of a relational *graph*. Accordingly, the classification, that is the assignment of a specimen to a class, is performed by parsing the sentence, so as to establish its accordance with a given grammar or by some graph matching techniques. The actual procedure we have adopted for decomposing and describing the character shape is articulated into three main steps: *skeletonization*, *decomposition* and *description* [8]. During the first step, the character skeleton is computed by means of a MAT–based algorithm, while in the following one it is decomposed in parts, each one corresponding to an arc of circle which we have selected as our basic elements. Eventually, each arc found within the character is described by the following features:

– *size of the arc*, referred to the size of its bounding box;
– *angle spanned by the arc*;
– *direction of the arc curvature*, represented by the oriented direction of the normal to the chord subtended by the arc.

The spatial relations among the arcs are computed with reference to arc projections along both the horizontal and vertical axis of the character bounding box. In order to further reduce the variability still presents among samples belonging to the same class, the descriptions of both the arcs and the spatial relations are given in discrete form. In particular, we have assumed the following ranges of values:

– *size*: small, medium, large;
– *span*: closed, medium, wide;
– *direction*: N, NE, E, SE, S, SW, W, NW;
– *relation*: over, below, to–the–right, superimposed, included.

Those descriptions are then encoded into a feature vector of 139 elements. The first 63 elements of the vector are used to count the occurrences of the different arcs that can be found within a character, the following 13 elements describe the set of possible relations and the remaining 63 ones count the number of the different relations that originates from each type of arc [9].

3 Learning Explicit Classification Rules

Evolutionary Learning Systems seem to offer an effective prototyping methodology, as they are based on Evolutionary Algorithms (EAs) that represent a powerful tool for finding solutions in complex high dimensional search space, when there is no *a priori* information about the distribution of the sample in the feature space [10]. They perform a parallel and adaptive search by generating new populations of individuals and evaluating their fitness while interacting with the environment. Since EAs work by directly manipulating an encoded representation of the problem, and because such a representation can hide relevant information, thus severely limiting the chance of a successful search, problem representation is a key issue in EAs. In our case, as in many others, the most natural representation for a solution is a set of prototypes or rules, whose genotype's size and shape are not known in advance, rather than a set of fixed–length strings. Since classification rules may be thought as computer programs, the most natural way to introduce them into our learning system is that of adopting the Genetic Programming paradigm [6,7]. Such a paradigm combines GAs and programming languages in order to evolve hierarchical computer programs of dynamically varying complexity (size and shape) according to a given defined behavior. According to this paradigm, populations of computer programs are evolved by using the Darwin's principle that evolution by *natural selection* occurs when a population of replicating entities possesses the *heritability* characteristic and are subject to *genetic variation* and *struggle to survive*.

Typically, Genetic Programming 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 nature also depends on the problem. 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. Then, the whole process is repeated until a termination criterion is satisfied.

In order to implement the Genetic Programming paradigm, the following steps has to be executed:

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

3.1 Structure Definition

In order to define the individual structures that undergo to adaptation in Genetic Programming, one needs a program generator, providing syntactically correct programs, and an interpreter for the structured computer programs, in order to execute them.

The program generator is based on a grammar written for S–expressions. A grammar $\mathcal{G}$ is a quadruple $\mathcal{G} = (\mathcal{T}, \mathcal{V}, S, \mathcal{P})$, where $\mathcal{T}$ and $\mathcal{V}$ are disjoint finite alphabets. $\mathcal{T}$ is the *terminal alphabet*, $\mathcal{V}$ is the *non–terminal alphabet*, S is the *start symbol*, and $\mathcal{P}$ is the set of *production rules* used to define the strings belonging to our language, usually written as $v \to w$ where v is a string on $(\mathcal{T} \cup \mathcal{V})$ containing at least one non–terminal symbol, while w is an element of $(\mathcal{T} \cup \mathcal{V})^*$. For the problem at hand, the set of terminals is the following:

Table 1. The grammar for the random rules generator.

Production Rule No.	Production Rule	Probability
1	$S \longrightarrow A$	1.0
2	$A \longrightarrow CBC \mid (CBC) \mid (IMX)$	0.25 | 0.25 | 0.5
3	$I \longrightarrow a_1 \mid \ldots \mid a_{139}$	uniform
4	$X \longrightarrow 0 \mid 1 \mid \ldots \mid 9$	uniform
5	$M \longrightarrow < \mid \leq \mid = \mid \geq \mid >$	uniform
6	$C \longrightarrow A \mid \neg A$	uniform
7	$B \longrightarrow \vee \mid \wedge$	uniform

$$\mathcal{T} = \{a_1, a_2, \ldots, a_{139}, 0, 1, \ldots, 9\},$$

and the set $\mathcal{V}$ is composed as follows:

$$\mathcal{V} = \{\wedge, \vee, \neg, <, \leq, =, >, \geq, A, X, I, M, C, B\},$$

where a_i is a variable atom denoting the i–th element in the feature vector, and the digits $0, 1, \ldots, 9$ are constant atoms used to represent the value of each element in the feature vector. It should be noted that the above sets satisfy the requirements of closure and sufficiency [6]. The adopted set of production rules is reported in Table 1.

Each individual in the initial random population is generated starting with the symbol S that, according to the above grammar, can be replaced only by the symbol A. This last, on the contrary, can be replaced by itself, by its opposite, or by any recursive combination of logical operators whose arguments are the occurrences of the elements in the feature vector. It is worth noticing that, in order to avoid the generation of very long individual, the clause IMX has a higher probability of being selected to replace the symbol A with respect to the other ones that appear in the second production rule listed in Table 1. As it is obvious, the set of all the possible structures is the set of all possible compositions of functions that can be obtained recursively from the set of predefined functions.

Finally, the interpreter is a simple model of a computer and is constituted by an automaton that computes Boolean functions, i.e. an acceptor. Such an automaton computes the truth value of the rules in the population with respect to a set of samples.

3.2 Fitness Function

The definitions reported in the previous subsection allow for the generation of the genotypes of the individuals. The next step to accomplish is the definition of a fitness function to measure the performance of the individuals. For this purpose, it should be noted that EAs suffer of a serious drawback while dealing with multi–peaked (multi–modal) fitness landscape, in that they are not able to deal with cases where the global solution is represented by different optima, i.e.

species, rather than by a single one. In fact, they do not allow the evolution of stable multiple species within the same population, because the fitness of each individual is evaluated independently on the composition of the population at any time [11]. As a result, the task of the evolutionary process is reduced to that of optimizing the average value of the fitness in the population. Consequently, the solution provided by such a process consists entirely of genetic variations of the best individual, i.e. of a stable single species distribution. This behaviour is very appealing whenever it is desirable the uniform convergence to the global optimum exhibited by the canonical EAs. On the other hand, there exist many applications of interest that require the discovering and the maintenance of multiple optima, such as multi–modal optimization, inference induction, classification, machine learning, and biological, ecological, social and economic systems [12].

In order to evolve several stable distributions of solutions, each representing a given species, we need a mechanism that prevents the distribution of the species with highest selective value to replace the competing species inducing some kind of restorative pressure in order to balance the convergence pressure of selection. To this end, several strategies for the competition and cooperation among the individuals have been introduced. The natural mechanism to handle such cooperation and competition is *sharing* the environmental resources, i.e. similar individuals share such resources thus inducing *niching* or *speciation* [11, 12,13,14,15].

For the problem at hand, it is obvious that we deal with different species. In fact, our aim is to perform an unsupervised learning in such a way that the result of the evolution is the emergence of different rules (species) each of which covers different sets of samples. Thus, the global solution will be represented by the disjunctive–normal–form of the discovered species in the final population. Moreover, in case of handwritten characters the situation gets an additional twist. In fact, it is indisputable that different writers may refer to different prototypes when drawing samples belonging to a given class. A classical example is that of the digit '7' that is written with an horizontal stroke crossing the vertical one in many European countries, while such a stroke is usually not used in North American countries.

Therefore, some kind of niching must be incorporated at different levels into the fitness function. In our case, we have adopted a niching mechanism based on *resource sharing* [14,15]. According to resource sharing the cooperation and competition among the niches is obtained as follows: for each finite resource s_i a subset P of prototypes from the current population is selected. They are let to compete for the resource s_i in the training set. Only the best prototype receives the payoff. In the case of a tie, the resource s_i is assigned randomly among all deserving individuals. The winner is detected by matching each individual in the sample P against the sample of the training set. In our case, a prototype p matches a sample s *if and only if* p *covers* s, i.e. the features present in the sample are represented also in the prototype and their occurrences satisfy the constraints expressed in the prototype. At the end of the cycle, the fitness of each individual is computed by adding all rewards earned. In formula:

$$\phi(p) = \sum_{i=1}^{m} c \cdot \mu(p, s_i),$$

where $\phi(p)$ is the the fitness of the prototype p, m is the number of samples in the training set, $\mu(p, s_i)$ is a function that takes into account if the prototype p is the winner for the sample s_i (it is 1 if p is the winner for the sample s_i and 0 otherwise) and c is a scaling factor.

3.3 Selection and Genetic Operators

The selection mechanism is responsible for choosing among the individuals in the current population the ones that are replicated, without alterations, in the new population. To this aim many different selection mechanisms have been proposed. Nevertheless, we have to choose a mechanism that helps the maintenance of the discovered niches by reducing the selection pressure and noise [16]. So, we have used in our Genetic Programming the well known Stochastic Universal Selection mechanism [16], according to which we have exactly the same expectation as Roulette Wheel selection, but lower variance in the expected number of individuals generated by the selection process.

As regards the variations operators, we have actually used only the mutation operator, performing both *micro–* and *macro–mutation.* The macro–mutation is activated when the point to be mutated in the genotype is a node, and it substitutes the relative subtree with another one randomly generated according to the grammar described in subsection 3.1. It follows from the above that the macro–mutation is responsible for modifying the structure of the decision tree corresponding to each prototype in the same general way with respect to that implemented by the classical tree–crossover operator. For this reason we have chosen not to implement the tree-crossover operator.

Eventually, the micro–mutation operator is applied whenever the symbol selected for mutation is either a terminal or a function, and it is responsible for changing both the type of the features and their occurrences in the prototype. Therefore, it resembles closely the classical point–mutation operator.

Finally, we have allowed also *insertions*, i.e. a new random node is inserted in a random point, along with the relative subtree if it is necessary, and *deletions*, i.e. a node in the tree is selected and deleted. Obviously, such kind of mutations are effected in a way that ensures the syntactic correctness of the newly generated programs.

4 Experimental Results and Conclusions

A set of experiments has been performed in order to evaluate the ability of Genetic Programming to generate classification rules in very complex cases, like the one at hand, and the efficiency of the preclassifier. The experiments were performed on a data set of 10,000 digits extracted from the NIST database and equally distributed among the 10 classes. This data set was randomly subdivided into a training and a test set, both including 5,000 samples. Each character was decomposed, described and eventually coded into a feature vector of 139 elements, as reported in Section 2. Starting from those feature vectors, we have considered only 76 features, namely the 63 features describing the type of the arcs found in the specimen and the 13 used for coding the type of the spatial relations among the arcs, and used that as descriptions for the preclassifier to work with.

It must be noted that the Genetic Programming paradigm is controlled by a set of parameters whose values affect in different ways the operation of the system. Therefore, before starting the experiments, suitable values should be assigned to the system parameters. To this purpose, we have divided this set of parameters into external parameters, that mainly affect the performance of the learning system and internal parameters, that are responsible for the effectiveness and efficiency of the search, As external parameters we have assumed the population size N, the number of generations G and the maximum depth of the trees representing the rules in the initial population D. The internal parameters are the mutation probability p_m, the number of rules n competing for environmental resources in the resourse sharing and the maximum size of the subtree generated by the macro–mutation operator d. The results reported in the sequel has been obtained with $N = 1000$, $G = 350$, $D = 10$, $p_m = 0.6$, $n = N$ and $d = 3$. Eventually, let us recall that, as mentioned in subsection 3.2, the fitness function requires to select the winner among the prototypes that match a given sample. We have adopted the Occam's razor principle of simplicity closely related to Kolmogorov Complexity definition [17,18], i.e. "*if there are alternative explanations for a phenomenon, then, all other things being equal, we should select the simplest one*". In the current system, it is implemented by choosing as winner the prototype whose genotype is the shortest one. With such a criterion we expect the learning to produce the simplest prototypes covering as many samples as possible, in accordance with the main goal of our work: to design a method for reducing the computational cost of the classifier while dealing with simple cases by reducing the number of classes to be searched for by the classifier.

As mentioned before, the first experiment was aimed at evaluating the capability of the Genetic Programming paradigm to deal with complex cases such as the one at hand. For this purpose, we have monitored the covering rate $\mathcal{C}$, i.e. the percentage of the samples belonging to the training set covered by the set of prototypes produced by the system during the learning. Let us recall now that, in our case, a prototype p matches a sample s *if and only if* p *covers* s, i.e. the features present in the sample are represented also in the prototype and their occurrences satisfy the constraints. The learning was implemented by providing the system with the descriptions of the samples without their labels, so as to implement un unsupervised learning, and let it evolve the set of prototypes for achieving the highest covering rate. Despite the loss of the information due

Table 2. The experimental results obtained on the Test Set.

$\mathcal{C}$	$\mathcal{R}$	≤ 3	≤ 5	≤ 7	$\mathcal{E}$	≤ 3	≤ 5	≤ 7
87.00	67.16	74.18	24.72	1.10	19.84	81.65	17.64	0.10

to both the discrete values allowed for the features and the reduced number of features considered in the experiment, the system achieved a covering rate of 91.38% In other words, there were 431 samples in the training set for which the system was unable to generate or to maintain a suitable set of prototypes with the time limit of 350 generations.

In order to measure the efficiency of the classifier, we preliminarily labeled the prototypes obtained at the end of the learning phase. The labeling was such that each time a prototype matched a sample of the training set, the label of the sample, i.e. its class, was added to the list of labels for that prototype. At the end of the labeling, thus, each prototype had a list of labels of the classes it covered, as well as the number of samples matched in each class. A detailed analysis of these lists showed that there were many prototypes covering many samples of very few classes and very few samples of many other classes. This was due to "confusing" samples, i.e. samples belonging to different classes but having the same feature values, thus looking indistinguishable for the system. In order to reduce this effect, the labels corresponding to the classes whose number of samples covered by the prototype was smaller than 10% of the total number of matchings for that prototype were removed from the list. Finally, a classification experiment on the test set was performed, that yielded to the results reported in Table 2 in terms of covering rate $\mathcal{C}$, correct recognition rate $\mathcal{R}$ and error rate $\mathcal{E}$.

Such results were obtained by assuming that a sample was correctly preclassified if its label appeared within the list of labels associated to the prototype covering it. To emphasize the efficiency of the preclassifier, Table 2 reports the experimental results by grouping the prototypes depending on the number of classes they cover. For instance, the third column shows that 74.18% of the total number of samples correctly preclassified were covered by prototypes whose lists have at most 3 classes.

The experimental results reported above allow for two concluding remarks. The first one is that Genetic Programming represents a promising tool for learning classification rules in very complex and structured domains, as the one we have considered in our experiments. In particular, it is very appealing in developing hierarchical handwritten character recognition system, since it may produce prototypes at different level of abstraction, depending on the way the system is trained. The second one is that the learning system developed by us, although makes use of a fraction of the information carried by the character shape, is higly efficient. Since the classification is performed by matching the unknown sample against the whole prototype set to determine the winner, reducing the number of classes reduces the number of prototype to consider, thus resulting in an overall reduction of the classification time. Therefore, roughly speaking, the preclassifier proposed by us is able to reduce the classification time to 50% or less in more than 65% of the cases.

References

1. Fukunaga, K.; Narendra, P. M. 1975. *A Branch and Bound Algorithm for Computing K–Nearest Neighbors. IEEE Trans. on Computers* C–24:750–753.
2. Marcelli, A.; Pavlidis, T. 1994. Using Projections for Preclassification of Character Shape. In *Document Recognition.* L.M. Vincent, T. Pavlidis (eds.), Los Angeles: SPIE, 2181:4–13.
3. Marcelli, A.; Likhareva, N.; Pavlidis, T. 1997. *Structural Indexing for Character Recognition. Computer Vision and Image Understanding*, 67(1): 330–346.
4. Mori, S.; Yamamoto, k.; Yasuda, M. 1984. *Research on Machine Recognition od Handprinted Characters. IEEE Trans. on PAMI*, PAMI–6: 386–405.
5. De Stefano, C.; Della Cioppa, A.; and Marcelli, A. 1999. Handwritten numerals recognition by means of evolutionary algorithms. In *Proc. of the 3th Int. Conf. on Document Analysis and Recognition*, 804–807.
6. Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Natural Selection.* Cambridge, MA: MIT Press.
7. Koza, J. R. 1994. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.
8. Boccignone, G. 1990. Using skeletons for OCR. V. Cantoni et al. eds., *Progress in Image Analysis and Processing*, 235–282.
9. Cordella, L. P.; De Stefano, C.; and Vento, M. 1995. Neural network classifier for OCR using structural descriptions. *Machine Vision and Applications* 8(5):336–342.
10. Goldberg, D. E. 1989. *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison–Wesley.
11. Deb, K., and Goldberg, D. E. 1989. An investigation of niche and species formation in genetic function optimization. In Schaffer, J. D., ed., *Proc. of the 3th Int. Conf. on Genetic Algorithms*, 42–50. San Mateo, CA: Morgan Kaufmann.
12. Mahfoud, S. 1994. Genetic drift in sharing methods. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, 67–72.
13. Booker, L. B.; Goldberg, D. E.; and Holland, J. H. 1989. Classifier Systems and Genetic Algorithms. In *Artificial Intelligence*, volume 40. 235–282.
14. Forrest, S.; Javornik, B.; Smith, R. E.; and Perelson, A. S. 1993. Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation* 1(3):191–211.
15. Horn, J.; Goldberg, D. E.; and Deb, K. 1994. Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation* 2(1):37–66.
16. Baker, J. E. 1987. Reducing bias and inefficiency in the selection algorithm. In Grefenstette, J. J., ed., *Genetic algorithms and their applications : Proc. of the 2th Int. Conf. on Genetic Algorithms*, 14–21. Hillsdale, NJ: Lawrence Erlbaum Assoc.
17. Li, M.; Vitányi, P. 1997. *An Introduction to Kolmogorov Complexity and its Applications.* Series: Graduate text in computer Science. Springeri–Verlag, New York.
18. Conte, M.; De Falco, I.; Della Cioppa, A.; Tarantino, E.; and Trautteur, G. 1997. Genetic Programming Estimates of Kolmogorov Complexity. *Proc. of the Seventh Int. Conf. on Genetic Algorithms* (ICGA'97), 743–750. Morgan–Kaufmann.