Evolutionary Design of Nearest Prototype Classifiers

FERNANDO FERNÁNDEZ*
PEDRO ISASI
Universidad Carlos III de Madrid, Avda/de la Universidad 30, 28911-Leganés, Madrid, Spain
email: ffernand@inf.uc3m.es

Submitted in October 2003 and accepted by Zbigniew Michalewicz in April 2004

Abstract

In pattern classification problems, many works have been carried out with the aim of designing good classifiers from different perspectives. These works achieve very good results in many domains. However, in general they are very dependent on some crucial parameters involved in the design. These parameters have to be found by a trial and error process or by some automatic methods, like heuristic search and genetic algorithms, that strongly decrease the performance of the method. For instance, in nearest prototype approaches, main parameters are the number of prototypes to use, the initial set, and a smoothing parameter. In this work, an evolutionary approach based on Nearest Prototype Classifier (ENPC) is introduced where no parameters are involved, thus overcoming all the problems that classical methods have in tuning and searching for the appropiate values. The algorithm is based on the evolution of a set of prototypes that can execute several operators in order to increase their quality in a local sense, and with a high classification accuracy emerging for the whole classifier. This new approach has been tested using four different classical domains, including such artificial distributions as spiral and uniform distibuted data sets, the Iris Data Set and an application domain about diabetes. In all the cases, the experiments show successfull results, not only in the classification accuracy, but also in the number and distribution of the prototypes achieved.

Key Words: classifier design, nearest prototype classifiers, evolutionary learning

1. Introduction

Nearest Neighbour Classifiers are defined as the sort of classifiers that assign to each new unlabelled example, v, the label of the nearest prototype, r_i, from a set, C, of N different prototypes previously classified (Duda and Hart, 1973). When the set C is very reduced, as is the goal of this work, these kinds of classifiers can be called Nearest Prototype Classifiers (Bezdek and Kuncheva, 2001) (NPC).

These classifiers are very related to vector quantization techniques (Gersho and Gray, 1992) since the nearest neighbour rule is the cornerstone of its design, and similar techniques can be used for both. The design of these classifiers is difficult, and relies on the way of defining the number of prototypes needed to achieve a good accuracy, as well as the initial set of prototypes used. Furthermore, most learning algorithms introduce additional different parameters, that are often summarized in a unique smoothing parameter. This learning parameter defines whether the updates on the classifier are higher (typically

*Author to whom all correspondence should be addressed.

at the beginning of the learning phase) or lower (typically at the end of the learning phase).

Many discussions about what the right technique to use is can be found in the literature (Kuncheva and Bezdek, 1998). Some approaches based on clustering techniques (Patanè and Russo, 2001; Pal et al., 1993; Bermejo and Cabestany, 2000) can be performed in two main steps. The first one is to cluster a set of unlabelled input data to obtain a reduced set of prototypes, for instance, with the LBG algorithm (Linde et al., 1980). The second step is to label these prototypes based on labelled examples and the nearest neighbour rule. Although this approach produces good results, it is obvious that introducing information about the classification performance in the location of the prototypes seems to be needed to achieve a higher performance.

Neural networks approaches are also very common in the literature, such as the LVQ algorithm (Kohonen, 1984) and the works with radial basis functions (Fritzke, 1994). To find the right number of neurons of the net, two basic approaches can be found. On the one hand, some techniques try to introduce or to eliminate prototypes (or neurons) while designing the classifier following different heuristics, such as the average quantization distortion (Russo and Patanè, 2000) or the accuracy in the classification (Pérez and Vidal, 1993). On the other hand, other approaches try to define first the optimal size of the classifier and then learn it using the previous value. Genetic algorithm approaches are typically used to find an initial set of prototypes, as well as its right size, in addition to another technique to achieve local optimization (Merelo et al., 1998). In Zhao and Higuchi (1996), an evolutionary approach is presented based on the R^4 rule (recognition, remembrance, reduction and review) to evolve nearest neighbour multi-layer perceptrons.

In this work, an evolutionary approach called Evolutionary Nearest Neighbour Classifier (ENPC), is introduced to dynamically find the optimal number of prototypes of the classifier as well as the location of these prototypes. The main difference with previous works is that this approach is a full integrated algorithm. Most algorithms that solve initialization problems take advantage of a previously known technique, which is modified to introduce the new capabilities. For instance, they introduce some heuristics for including or eliminating prototypes, or they use genetic algorithms for optimizing the initialization, but typically in a batch mode. However, in this work, both the operations to modify the size of the classifier and the learning algorithm are fully integrated and can not be used separately from the other part.

In this sense, the method is able to obtain a very good map of prototypes, without having to indicate any initial configuration, number of prototypes that would be good to obtain, nor learning parameters.

The method allows the prototypes to execute several operators, like introducing new prototypes, changing their related class, etc. in order to improve the global accuracy of the classifier. Furthermore, the execution of these operators is controlled by the prototypes themselves, taking into account their relationship with the rest of the prototypes of the classifier. In previous works (Fernández and Isasi, 2001, 2002), the biological inspirations of this algorithm were presented, following an ecosystem metaphor.

In the next section, the operators are explained, as well as the main concepts that are used. Section 3 shows principal experiments performed and comparative results with previous works, while Section 4 shows main conclusions achieved and further research.

2. The ENPC algorithm

2.1. Concepts

In the following, the main concepts used in ENPC are described:

Labeled Prototype, r_i. Defines each prototype of the classifier system. The prototype is composed by the location of the prototype in the space (p) and the class of the prototype (s). So a prototype r will be denoted by $\langle p, s \rangle$. The region of a prototype is the space, within which, all the examples will be labeled as the prototype class. This region is computed using the nearest neighbour rule.
Classifier, C. A set of N prototypes $C = \{r_1, \ldots, r_N\}$.
Pattern, v_r. Is each of the examples that will be used for training or testing the system. They all compose a set $V = \{v1, \ldots, v_M\}$, and, as well as the prototypes, they are composed by their position (p) in the space and by their class (s), so a prototype v will be denoted by $\langle p, s \rangle$.
Class, s_j. Both prototypes and patterns belong to a class from the set $S = \{s_1, \ldots, s_L\}$. The goal of a prototype, r_i, of class, s_j, is to contain in its region as many patterns of class s_j as possible and not to contain any pattern of other classes $s_k \neq s_j$.
Quality of a prototype, *quality*(r_i). Is a measure of the goodness of the prototype, taking into account the number of patterns in its region, and whether those patterns belong to the same class as the prototype or not.

In order to compute the quality value for each region, additional features must be taken into account and computed, and they are defined next.

2.2. Structure

The system can be represented by a bidimensional matrix, where each row is associated to a prototype of the whole classifier $r_i \in C$, and each column is associated to a class $s_j \in S$. Each position (i, j) of the matrix is a structure that contains features about the set of training examples (V_{ij}) that are in region r_i and belong to class s_j.

Figure 1 shows the structure used to keep all this information. This structure is dynamic, in the sense that if any set is empty, the node is not created, solving some memory requirements in complex domains.

2.2.1. Class sets. The set S_j is defined as the set of patterns that belong to class s_j. Membership function to this set is the equivalence between the class of the pattern and the class associated to the set (i.e. s_j), as defined in Eq. (1).

$$\forall v \in V, v = \langle p, s_j \rangle, s_j \in S, \quad \text{then } v \in S_j \tag{1}$$

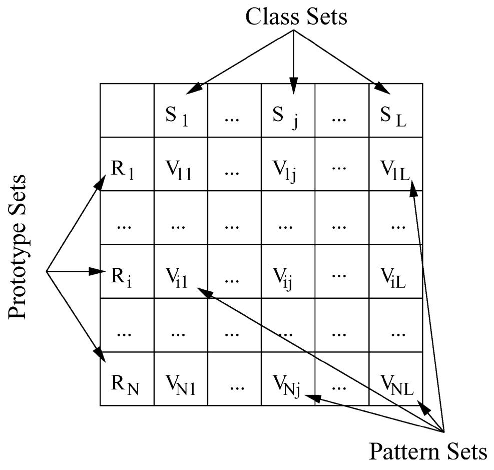

Figure 1. ENPC architecture.

From this set, some interesting features can be obtained:

– *regions*(s_j) is the number of regions whose prototype class is s_j. This function can be computed from Eq. (2).

$$\textit{regions}(s_j) = \sum_{i=1}^{N} \delta(r_i, s_j), \quad \text{where } \delta = \begin{cases} 1, & \text{iff } r_i = \langle p, s_j \rangle \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

– *expectation*(s_j) is the number of patterns that any prototype r_i of the class s_j is expected to correctly classify. This number is computed in a relative way, by the relationship among the number of patterns belonging to class s_j ($\|S_j\|$) and the number of regions of each class s_j. This value gives an idea about how many patterns must classify each prototype so that they all classify a similar number, and it is computed using Eq. (3).

$$\textit{expectation}(s_j) = \frac{\|S_j\|}{\textit{regions}(s_j)} \tag{3}$$

2.2.2. Pattern sets. The set V_{ij} is defined as the set of patterns that are located in region r_i and belong to the class s_j. Each set V_{ij} is the intersection among the sets R_i (see Section 2.2.3) and S_j, and its membership function is defined in Eq. (4).

$$\begin{aligned} &\forall\, v = \langle p_v, s_j \rangle \in V, s_j \in S, r_i = \langle p_i, s' \rangle \in C, \quad \text{then} \\ &v \in V_{ij} \quad \text{iff } \forall\, r'_i = \langle p_{i'}, s'' \rangle \in C, d(p_v, p_i) \le d(p_v, p_{i'}) \end{aligned} \tag{4}$$

Distance metric used in this work is mean square error, as defined in Eq. (5).

$$d(x, y) = \sum_{i=0}^{i<K} (x[i] - y[i])^2 \tag{5}$$

where K is the dimension of the data.

2.2.3. Prototype sets. The set R_i is defined as the set of patterns that are located in the region r_i. Membership function to this set is the nearest neighbour rule, and it is defined in Eq. (6).

$$\begin{aligned} &\forall\, v = \langle p_v, s_j \rangle \in V, r_i = \langle p_i, s' \rangle \in C, \quad \text{then} \\ &\quad v \in R_i \quad \text{iff } \forall\, r_{i'} = \langle p_{i'}, s'' \rangle \in C, d(p_v, p_i) \leq d(p_v, p_{i'}) \end{aligned} \tag{6}$$

Note that in the case that several regions satisfy the condition, only one of them will be chosen. Distance metric used here is again the mean square error defined in Eq. (5).

For each prototype $r_i = \langle p, s_j \rangle \in R$, its own classification accuracy can be computed following the Eq. (7).

$$accuracy(r_i) = \frac{\| V_{ij} \|}{\| R_i \|} \tag{7}$$

where $\| V_{ij} \|$ is the number of prototypes located in the region r_i that belong to the same class as the prototype r_i (see Section 2.2.2).

From this accuracy, the final value of the prototype quality is computed. The quality is a relation between the accuracy of the prototype, that is, the capability of classification, and the size, in number of patterns gathered, of the prototypes. For computing the size, not only the patterns gathered by the prototype have been taken into account, but also the expectation of the class to which this prototype belongs. This relation has been called apportation of the prototype to the whole classification task, and it is defined for each prototype $r_i = \langle p, s_j \rangle \in R$ as shown in Eq. (8).

$$apportation(r_i) = \frac{\| V_{ij} \|}{\frac{\text{expectation } (s_j)}{2}} \tag{8}$$

The motivation of dividing the expectation value by 2 is to give the prototype a less rigid measure of its apportation, given that the distribution of the patterns of each class may not be uniform.

From this, the computation of the final measure of prototype quality is as defined in Eq. (9).

$$quality(r_i) = min(1, accuracy(r_i) * apportation(r_i)) \tag{9}$$

The main idea is that the quality of a prototype is high only if it classifies correctly, and if it classifies a sufficient amount of patterns. The value is limited to the range [0, 1].

Table 1. Example of the ENPC classifier.

	S1	S2
$R_1 = \langle p_1, s_2 \rangle$	$\|V_{11}\| = 7$	$\|V_{12}\| = 9$
$R_2 = \langle p_2, s_1 \rangle$	$\|V_{21}\| = 10$	$\|V_{22}\| = 2$
$R_3 = \langle p_3, s_2 \rangle$	$\|V_{31}\| = 3$	$\|V_{32}\| = 9$

An example of how computing previous values is defined in Table 1. The table shows a situation of the classifier, where the number of prototypes is 3 and the number of different classes is 2. Furthermore, a set of instances has been introduced in the different sets V_{ij} (giving only the sizes of these sets), representing the instances that are in region r_i and belonging to class s_j. If we begin to compute characteristics of the Class Set, for instance, $\|S_1\| = \sum_{i=1}^{N=3} V_{i1} = 20$, so there is a total of 20 instances belonging to class s_1. *Regions*$(s_1) = 1$, given that there is only a prototype (r_2), that belongs to class s_1, so *expectation*$(s_1) = 20$ (the only prototype belonging to class s_1 is expected to classify all the instances belonging to that class). On the other hand, *accuracy*$(r_2) = 10/12 = 0.83$, while *apprtation*$(r_1) = 10/(20/2) = 1$, given that it correctly classifies the 10 instances that it is supposed to classify. Last, *quality*$(r_2) = \min(1, 0.83 * 1) = 0.83$.

2.3. *The algorithm*

The learning phase is an iterative process where the prototypes can execute several operators, once the features defined in Section 2.2 have been computed. These operators consist of heuristics that allow the prototypes to change their location, to introduce new prototypes, etc. The algorithm is summarized in the flow shown in figure 2. Firstly, there is a simple initialization phase where the classifier is composed by only one prototype. After that, the classifier evolves by executing, in a loop, all the operators, described below. These operators take advantage of some information gathered at the beginning of the loop.

Notice that most of the operators/heuristics defined here are taken from the literature. This approach uses them from an evolutionary point of view, giving the prototype capabilities to decide when to execute each of the operators and introducing a high level of randomness in those decisions.

2.3.1. Initialization. One relevant feature of this method is the absolute lack of initial conditions. These initials conditions are usually summarized in three: the number of prototypes, the initial set of prototypes and a smoothing parameter. The ENPC algorithm allows learning without those parameters, given that:

– The initial number of prototypes is always one. The method is able to generate new prototypes stabilizing in the most appropriate number, in terms of the above defined “quality” measure.
– The initial location of the only prototype is not relevant (it clusters all the domain, wherever it is located).

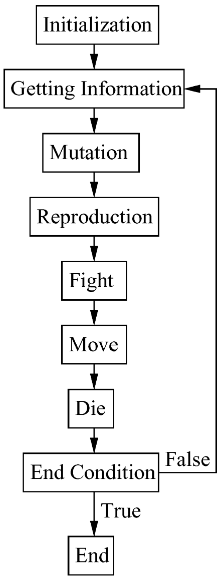

Figure 2. ENPC algorithm flow.

– There are no learning parameters. The method automatically adjust the intensity of change in prototypes taking into account their qualities in each iteration.

2.3.2. Getting information. At the beginning of each iteration, the algorithm must compute the information required to execute the operators. This information was presented in previous section, and it refers to the prototypes, class, and pattern sets.

Getting information is a phase where the patterns are "inserted" into their class, prototype and instance sets defined in previous section, using the membership functions defined in Eqs. (1), (4) and (6) respectively. At the end of this phase, all the patterns will have been "introduced" in the sets, and the required information about them will have been computed. We must point out that previos sets are not really stored, so instead of "introducing" instances in the sets, the information about these set is updated as if the instances would have been introduced.

2.3.3. Mutation operator. The goal of this operator is to label each prototype with the most populate class in each region. Once the features are obtained, each prototype knows the number of patterns of each class located in its region. Then, the prototype changes, if needed, and becomes the same class as the most abundant class of patterns in its region. Figure 3 shows an example of this operator. In the example, a prototype of class 2, changes to class 1, given that it has 19 patterns of class 1 and only 7 of class 2.

This way of getting the main class is typically used when unsupervised learning is applied to supervised classification (Bermejo and Cabestany, 2000; Pal et al., 1993). In these works,

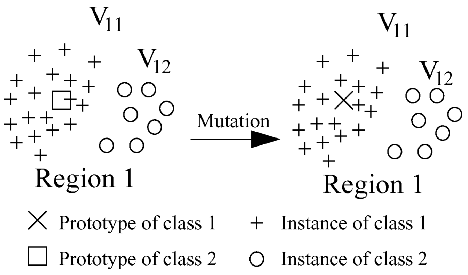

Figure 3. Example of mutation operator execution.

the algorithms typically generate a set of clusters that takes into account only the distribution of the data. In a second phase, the clusters are labelled as the most populate class in the cluster. However, in our work, the supervision is included in each iteration of the algorithm, and not only in "a posteriori" phase. Remember that the quality of each prototype depends on the relationship among the number of patterns in its region, and the number of patterns that belongs to the same class as the prototype, so the way to improve this value is becoming to the most populate class. The formulation of this operator is defined in Eq. (10).

$$\forall r_i = \langle p, s \rangle \in C, s = \arg\max_j \| V_{ij} \| \tag{10}$$

where $\| V_{ij} \|$ is the size of the set V_{ij}.

2.3.4. Reproduction operator. The goal of this operator is to introduce new prototypes in the classifier. The insertion of new prototypes is a decision that is taken by each prototype, in the sense that each prototype has the opportunity of introducing a new prototype in order to increase its own quality. The reason for providing the prototypes with this capacity is to achieve each one of them only containing patterns that belong to the same class. If we remember the structure of the approach shown in figure 1, we see that this goal corresponds with a situation where there is only a non empty set V_{ij} for all r_i, i.e. there is only a non empty set V_{ij} in each row. Figure 4 shows a region r_1 that has two non-empty sets V_{11} and V_{12}, being 1 the class of the prototype. A single way to achieve the desired situation is by

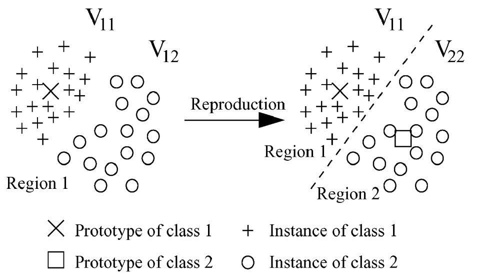

Figure 4. Example of reproduction operator execution.

introducing another region r_2 of class 2 that contains the set V_{12}, that should be renamed as V_{22}. Summarizing, the regions with patterns belonging to different classes can create new regions containing the patterns of a different class from the class of the prototype.

There is only one decision to take: in which situations are new prototypes introduced? To solve this, each prototype, r_i, of the class s_j, executes a roulette. Each slice of the roulette represents a set $V_{ij'}$, for all $j' \in S$. The size of each slice is proportional to the number of elements of the set $V_{ij'}$ which it represents. The possible results of the roulette are two. On the one hand, if the resulting set $V_{ij'}$ is the set V_{ij}, i.e. $j = j'$, no reproduction is executed. On the other hand, if the resulting set $V_{ij'}$ is not the set V_{ij}, i.e. $j \neq j'$, the reproduction is executed, and a new region $r_{i'}$ is created to contain the patterns in $V_{ij'}$, that is renamed to be $V_{i'j'}$.

We can see that if the goal situation of having only one non-empty set V_{ij} for each region r_i is kept, no reproduction will be executed. But if the number of non-empty sets grows, as well as their size, the probability of reproduction will grows too.

2.3.5. Fight operator. This operator provides the capability of getting patterns from other regions to the prototypes. Formally, this operator allows a prototype, r_i, to modify its sets V_{ij} from the sets $V_{i'j}$ of another prototype $r_{i'}$, for $i \neq i'$. This operator takes place in several steps:

1. Choose the prototype $r_{i'}$ to fight against. Prototypes are chosen from the set *neighbours*(r_i), that is defined as the set of regions that have a common border with the region r_i. To decide which prototype to choose from the neighbours' set, a roulette is used to assign to each region $r_j \in$ *neighbours*(r_i) a slice of size proportional to the difference between its quality and the quality of r_i.
2. Decide whether to fight or not. Probability of fighting between prototypes r_i and $r_{i'}$ is proportional to the distance of their qualities, as shown in Eq. (11).

$$P_{\text{fight}}(r_i, r_{i'}) = |quality(r_i) - quality(r_{i'})| \tag{11}$$

3. If prototype $r_i = \langle p_i, s_i \rangle$ decides to fight with prototype $r_{i'} = \langle p_{i'}.s_{i'} \rangle$, there are two possibilities:

   – If $s_i \neq s_{i'}$ (cooperation). Both prototypes belong to different classes. In this case, the prototype r_i' will give to the prototype r_i the patterns of the class s_i. This is done by inserting the patterns of the set $V_{i's_i}$ into the set V_{is_i} and setting $V_{i's_i}$ to empty. Figure 5 shows an execution of this operator, where the prototype 1, which owns patterns of the classes 1 and 2, gives the patterns of class 2 to the prototype 2, by introducing the patterns into the set V_{12} in the set V_{22}.
   – If $s_i = s_{i'}$ (competition). In this case, patterns can be transfered from set $V_{i's_i}$ to the set V_{is_i}, or vice versa, depending on who wins the fight. Figure 6 shows an execution of this operator, where prototype 1 steals some patterns from prototype 2. The winner is decided again using a roulette with only two slices, each slice belonging to each prototype, and sizes proportional to the qualities of each prototype. Furthermore, the amount of patterns that are transferred depends on a probability proportional to the qualities of both prototypes.

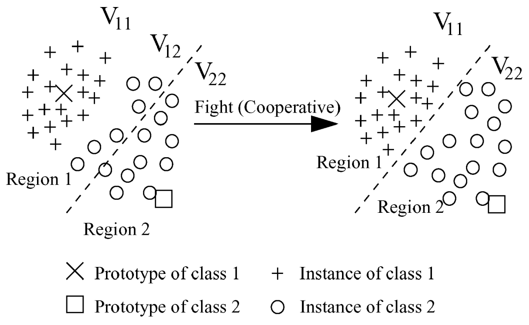

Figure 5. Example of fight operator execution with cooperation.

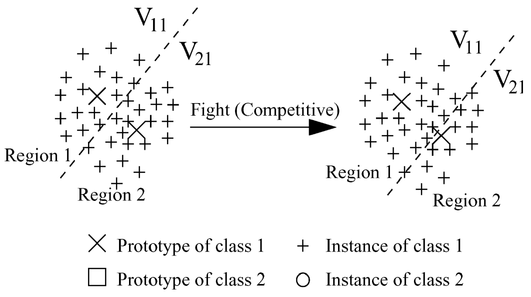

Figure 6. Example of fight operator execution with competition.

Last, notice that in figures 5 and 6, the translation of patterns from one set to another is done only by changing the border between the regions. So, when it is said that some patterns are extracted from one set and inserted into another one, the only thing that it is done is to move that border. In a nearest neighbour approach, this movement can be done only by changing the centroid of the sets, given that these centroids define, with the nearest neighbour rule, the borders among the regions.

2.3.6. Move operator. The move operation implies relocating each prototype in the best expected place. So each prototype, $r_i = \langle p_i, s_j \rangle$, decides to move to the centroid of the set V_{ij}, as shown in Eq. (12).

$$p_i = centroid(V_{ij}) \tag{12}$$

Figure 7 shows the execution of the move operator from the situation achieved in figure 5, and how the prototype 2 changes its position to the centroid of the set V_{22}.

This operation based in the second step of Lloyd iteration (Lloyd, 1982) allows making a local optimization of the classifier, increasing its performance.

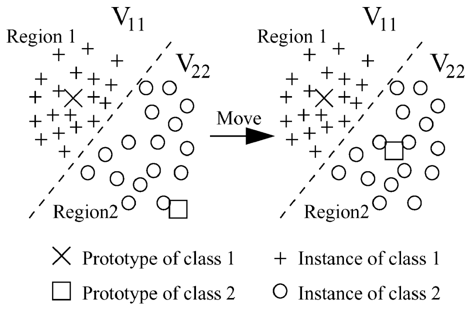

Figure 7. Example of move operator execution.

2.3.7. Die operator. Probability to die is 1 minus the double of the quality, as defined in Eq. (13). Then, successful prototypes will survive with probability of 1, while useless prototypes with quality less than 0.5 might die. A wide range of heuristics about how to reduce the number of prototypes in a classifier can be found in the bibliography (Fritzke, 1994; Russo and Patanè, 2000; Cagnoni and Valli, 1994).

$$P_{\text{die}}(r_i) = \begin{cases} 0, & \text{when } \mathit{quality}(r_i) > 0.5 \\ 1 - 2 * \mathit{quality}(r_i) & \text{when } \mathit{quality}(r_i) \leq 0{,}5 \end{cases} \tag{13}$$

2.3.8. End condition. End condition is the hardest element to define in this approach. It is supposed that the algorithm convergence to an optimal solution is desirable, but: what is an optimal solution? In this area, an optimal solution is said to be that solution that achieves a highest classification success with the smallest number of prototypes. However, what is the heaviest parameter? Some people may think that if increasing the number of prototypes, the accuracy of the classifier can also be increased, it is better to increase this number, but over-fitting problems may occur. On the other side, if we reduce the number of prototypes, we can do it only by decreasing the accuracy. So, what is the best solution? The approach of this work is to let the population evolve, and the user decides when and why to stop. Obviously, a lot of different conditions could be introduced to decide when to stop: convergence in the number of prototypes, convergence in the accuracy, a weighted version of both, or even a learning parameter. Anyway, the experiments will show how the algorithm always converge to a small accuracy range, and in most of the times to a small range in the number of prototypes.

Anyway, several stopping criteria may be be used, and are defined next:

1. Number of iterations. The user defines a maximum number of iterations.
2. Accuracy. The user defines its desired classification success, so the algorithm will work until that value has been reached.
3. Accuracy and Number of Iterations. The user defines its desired accuracy, but it also defines a maximum number of iterations, so even if the algorithm is not able to achieve the expected result in that time, it stops.

4. Convergence to a number of prototypes: This is an automatic way to verify the convergence of the algorithm to a defined number of prototypes. This criteria stops only when the relative frequency of appearance of the size of the classifier is higher than a certain level, after the algorithm has been executed a minimum number of iterations in order to get enough statistics. Thus, if the algorithm converges to a defined number of prototypes, the frequency of this number will increase, and the end condition will become true.
5. Convergence to an accuracy: This is an automatic way to verify the convergence of the algorithm to an accuracy. The algorithm will stop when it is not able to increase the accuracy of the classifier in a large number of iterations.
6. Any combination of previous approaches. For instance, executing the algorithm until user success and convergence to a number of prototypes are achieved, stopping before only if the algorithm is executed a user defined number of iterations.

Once the convergence criteria have been defined, the winner classifier can be obtained in several ways. A single one is to find the classifier with higher accuracy in the training set. However, in order to improve generalization capabilities and to avoid fitting over the training set, it could be useful to select another classifier with fewer prototypes and similar success.

3. Experiments

In this section, several experiments performed with the ENPC algorithm in different domains are shown: in Section 3.1 is a two spiral data set, a domain with two interlaced spirals each one belonging to a different class; in Section 3.2 domain Uniformly Distributed Data; in Section 3.3 the Iris Data Set; and in Section 3.4, the Pima Indians Diabetes Database.

For all the experiments, the end condition is defined by a maximum number of iterations, and the classifier returned is the one with highest accuracy over the training data set. Comparative results are divided in two. On the one hand, the algorithm has been compared with four different classifiers in all the domains. These classifiers are decision trees (C4.5) (Quinlan, 1993), decision rules (PART) (Frank and Witten, 1998), Naive Bayes (Duda and Hart, 1973; John and Langley, 1995), and IBK (Aha and Kibler, 1991) for different values of k. The implementation of these algorithms is provided by WEKA (Witten and Frank, 2000), and they are used with the pre-defined parameters. On the other hand, in the domains where comparative results have been found in the literature, they have been included too.

3.1. Spiral data

This domain consists of two squared spirals of data, as shown in figure 8. The examples in the same spiral belong to the same class and there are 500 examples for each spiral. Thus, a total of 900 examples were used for training and 100 for testing.

For all the experiments, 20 executions have been performed; in this case, of a length of 300 iterations per execution. Results of these experiments for the spiral data set are shown in Table 2. The table shows, for each of the executions, some information about the classifier that obtains better classification success on training (iteration when it was obtained, training

Table 2. Results of different executions of the ENPC algorithm over the spiral data set.

	Iteration	Prototypes	Learning (%)	Test (%)
	289	82	99.33	97.0
	158	75	99.0	96.0
	174	81	99.11	95.0
	267	81	99.33	96.0
	282	81	99.44	97.0
	260	78	99.11	97.0
	170	79	99.56	96.0
	182	82	99.33	97.0
	120	79	98.67	95.0
	281	73	98.9	96.0
	300	84	99.44	97.0
	123	84	99.56	96.0
	81	79	99.33	94.0
	260	82	99.44	96.0
	131	77	98.67	95.0
	81	**81**	**99.33**	**98.0**
	97	79	99.33	96.0
	299	78	99.56	96.0
	291	84	99.78	97.0
	179	75	99.0	97.0
Average	201.25	79.7	99.26	96.2
Average deviation	71.77	2.5	0.24	0.74

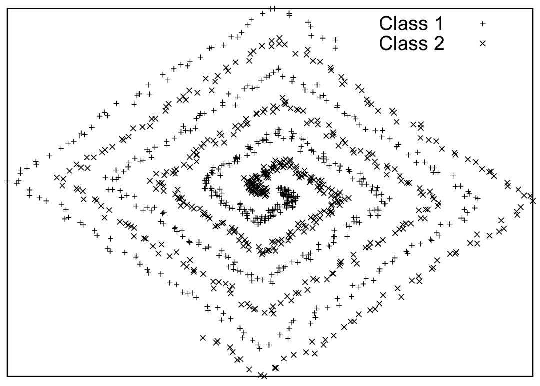

Figure 8. Spiral data.

Table 3. Comparative results of spiral data set.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENPC (best)	ENPC (worst)	ENPC (average)
62	56	50	100	100	98	94	96.2

success, test success, and number of prototypes). This classifier is selected for executing the test (which result is shown in the table too). The table shows that classification success obtained in learning is around 99%, while for test set this value ranges from 94 to 98%, obtaining an average value of 96.2%, and an average deviation of 0.74 (less that one test instance). The average number of prototypes achieved is 79.7, with and average deviation of 2.5 prototypes over that value.

Table 3 shows a comparative of previous results with other classifiers, using the average value obtained by ENPC in the previous 20 executions as comparative result. The table shows that results for decision trees (C4.5), decision rules (PART), and Naive Bayes results are poor, and they are far from the results obtained by ENPC. IBK is the one that achieves best results, obtaining a 100% success rate both for $k = 1$ and for $k = 3$, that is expected in domains where instances from different classes are very separated, as shown in figure 8.

The evolution of an execution of the algorithm is shown in figure 9, where the number of prototypes, and the accuracy obtained over the training and test sets are shown. In only 25 iterations of the algorithm, the number of prototypes has been increased up to 64, providing an accuracy over the training set of 91.34% and of 85.00% for the test set. However, the number of prototypes is still increased until the range 70–80, where it continues the search

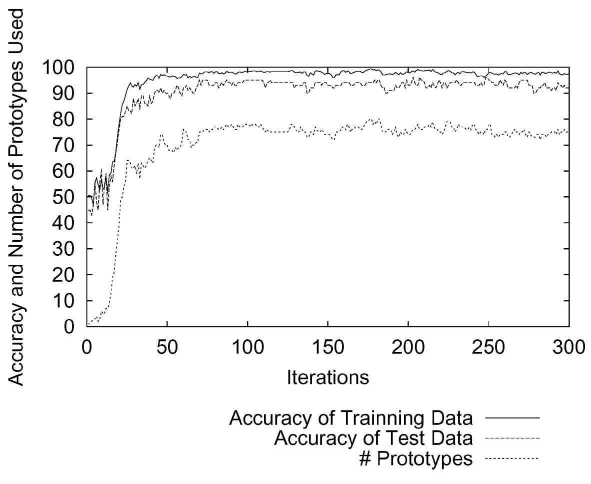

Figure 9. Evolution of the ENPC algorithm over spiral data set.

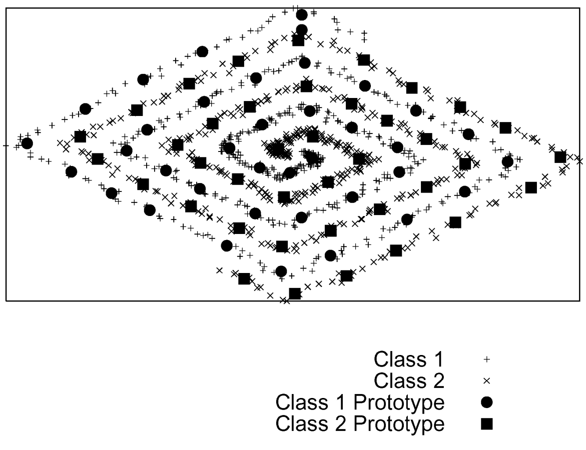

Figure 10. Classifier of 76 prototypes obtained with ENPC in spiral data set.

and achieves the best result in iteration 203 of 98.56% for the training set and 96.00% for the test set, with 76 prototypes. Maximum accuracy obtained for the training set is 99.34% in iteration 176, where the accuracy of the test set is 94.00%. The classifier obtained in this execution is shown in figure 10.

3.2. *Uniformly distributed data*

In this experiment, the algorithm has been tested in a domain that follows a uniform distribution with 2 different classes shown in figure 11, as defined in Burrascano (1991). As in

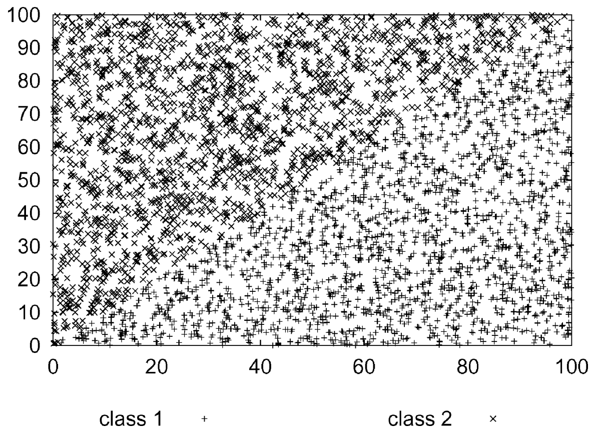

Figure 11. Data with uniform distributions.

Table 4. Results of different executions of the ENPC algorithm over uniform distributed data.

Iteration	Prototypes	Learning (%)	Test (%)
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
266	19	98.8	98.3
1	2	98.6	98.6
279	30	98.8	98.7
1	2	98.6	98.6
1	2	98.6	98.6
1	2	98.6	98.6
179	14	99.1	98.73
37	**2**	**98.7**	**98.93**

that case, the data contains 2000 instances of each class, using 500 for training and 1500 for testing.

Given that evolution introduces random behaviours, different executions of the algorithm may achieve different solutions. Table 4 shows the resulting classifier for 20 different executions with a maximum length of 300 iterations. The same information as in previous experiments is given, i.e. the iteration, accuracy over training and test set and the number of prototypes of the best classifier achieved in each execution of the algorithm.

In the table it is shown that in 16 of the 20 executions, the algorithm returns the solution of only 2 prototypes and 98.6% of accuracy in classification over the test set, a value that is achieved in the first iterations of the algorithm. However, given that the algorithm is allowed to continue searching for better solutions, in some occasions it is able to improve this value up to 98.93 with the same number of prototypes, introducing an average deviation of only 0.05 from the average success of 98.61. It is very interesting that the algorithm achieves very good results with 2 prototypes in only one or two iterations. This surprising result is due to the fact that the initialization of the algorithm is very adequate for this domain. In the initialization, the only operators that could be executed are reproduction and move. The latter moves one prototype to the center of the first distribution and the other one to the

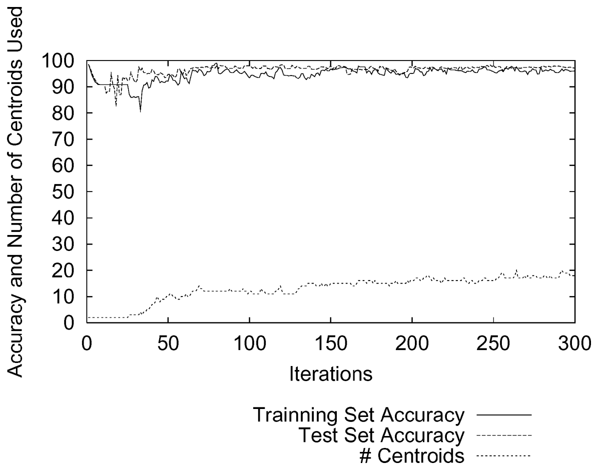

Figure 12. Evolution of the ENPC algorithm over uniformly distributed data.

center of the second distribution, which becomes one of the best solutions. From this point, improvement is really difficult.

Figure 12 shows the evolution of the algorithm for the last of previous executions. As in the rest of the experiments, the x-axis shows the iteration, and the y-axis shows both the number of prototypes used and the accuracy of the classifier for the training and test data sets.

In figure 12, how the classifier with only two prototypes is achieved at the beginning of the evolution is shown. Later, the number of prototypes is increased to achieve better accuracies. In this case, in iteration 79, a classifier of 12 prototypes achieves an accuracy of 98.967%.

Table 5 shows the comparison of these results with previous approaches, such as the LVQ algorithm (Kohonen, 1984), and two implementations of the Probabilistic Neural Networks (PNN) (Specht, 1990) that allow reducing the structure of the network. LVQ-PNN

Table 5. Comparative results for uniform distributed data classification.

Algorithm	Number of prototypes/neurons	Accuracy
LVQ	10	96.13
LVQ	100	97.77
LVQ-PNN	10	96.99
LVQ-PNN	100	98.17
Improved PNN	8	98.85
ENPC (average)	2	98.6
ENPC (best)	2	98.93

Table 6. Additional comparative results over the uniformly distributed data.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENPC average	ENPC best
96.97	96.6	96.77	98.7	98.47	98.6	98.93

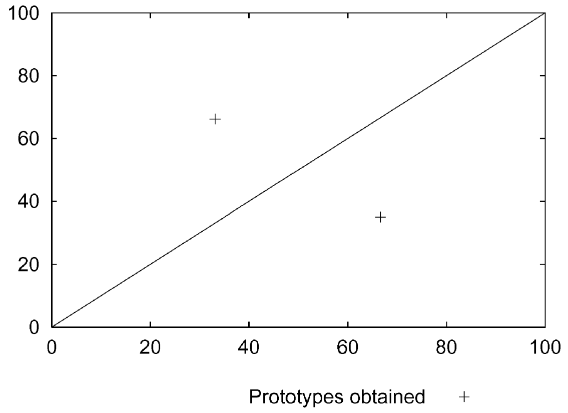

Figure 13. Classifier obtained with ENPC over uniformly distributed data domain.

(Burrascano, 1991) does this by using LVQ to find a reduced set of neurons in its second layer, instead of using as many neurons as input patterns. For LVQ and LVQ-PNN, two different executions are reported in Burrascano (1991), one with 10 neurons and another with 100 neurons, giving the best results for LVQ-PNN with 98.17% accuracy throushout the test. The second one (Mao et al., 2000) is an algorithm that permits designing PNN by determining the smoothing parameter by Genetic Algorithms and the structure of the network by an orthogonal algorithm that selects important neurons, so the number of neurons to use is automatically computed by the algorithm, as is the case of ENPC. In the table it is shown that the typical solution of 2 prototypes obtained with the ENPC algorithm (98.6% accuracy) is very close to the best solution of the improved PNN algorithm (98.85%), while other classifiers obtained may improve that value (98.93% with 2 prototypes). The classifier obtained by ENPC of only two prototypes is shown in figure 13.

Lastly, Table 6 shows some comparative results with other algorithms, showing that ENPC results are always close to the best results obtained by other algorithms, and achieving the best results in some cases.

3.3. Iris data set

Iris Data Set from UCI Machine Learning Repository[1] (Blake and Merz, 1998) is used for this experiment. The dataset consists of 150 samples of three classes, where each class has

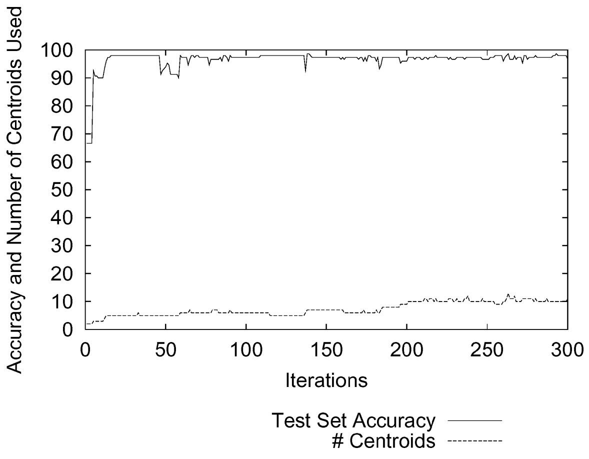

Figure 14. Evolution of the ENPC algorithm over the Iris data set.

50 examples. The dimension of the feature space is 4. In this case, and for comparison reasons, the whole data set was used for training and for testing.

The ENPC algorithm was executed 20 times as in previous experiments, and results are described in Table 7. The table shows that the best solutions achieved are for 2 and 3 misclassifications (98.667% and 98.000% accuracy respectively) with classifiers of 5 prototypes for 3 misclassifications, and a range of 7–12 prototypes for 2 misclassification results. In figure 14 one of the executions is shown. As in previous experiments, the figure represents the iteration in the x-axis, while the accuracy and the number of prototypes are shown in y-axis. Given that all the data set is used for training and testing, only one accuracy evolution is shown. The figure shows that the result of 5 prototypes and 98.000% success is found in only 15 iterations, after keeping with 3 prototypes for several iterations, and around 90% success. After achieving 98.000%, the algorithm evolves to new classifier sizes, finding better accuracy (98.667%) for 7 prototypes in iteration 138.

These results are compared with the ones presented in Kuncheva and Bezdek (1998) and Mao et al. (2000), and are summarized in Table 8, where the number of prototypes and the misclassifications are shown. The algorithms compared are LVQ (Kohonen, 1984), GLVQ-F and DR (Bezdek et al., 1998), and the improved PNN algorithm (Mao et al., 2000) introduced in Section 3.2.

We can see how ENPC algorithm improves the results of MFCM3, LVQ and GLVQ-F, but cannot achieve the results of the improved PNN, which has one misclassification with only 3 prototypes.

Lastly, the results are also compared with other algorithms, as is shown in Table 9. The table shows that C4.5 produces only 3 mistakes, while PART produces 4 and NAIVE BAYES 6. Obviously, IBK with $k = 1$ produces 0 mistakes, given that test set is the as the training set, but if k is increased up to 3, the number of errors increases up to 5.

Table 7. Results of different executions of ENPC algorithm over the Iris data set.

	Iteration	Accuracy	Prototypes
	86	98.667	7
	39	98.000	6
	138	98.667	7
	107	98.667	7
	152	98.667	11
	236	98.667	8
	24	**98.667**	**7**
	24	98.000	5
	43	98.667	12
	19	98.000	5
	25	98.000	5
	252	98.667	10
	133	98.667	8
	134	98.667	10
	145	98.667	9
	10	98.000	5
	12	98.000	5
	143	98.667	7
	255	98.667	11
	95	98.000	6
Average	103.6	98.43	7.55
Average Deviation	65.9	0.3	1.86

Table 8. Comparative results for Iris data set.

Algorithm	MFCM-3	LVQ	LVQ	GLVQ-F	DR	DR	I. PNN	ENPC	ENPC
Prototypes	7	7	3	8	5	3	3	5	7
MisClass.	11	3	17	3	3	10	1	3	2

Table 9. Additional comparative results over the Iris data set.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	ENPC (best)	ENPC (worst)
3	4	6	0	5	2	3

3.4. Pima indians diabetes database

This domain, obtained from the UCI Machine Learning Repository (Blake and Merz, 1998) includes medical information for diabetes diagnosis. The dataset consists of information of eight continue-valued attributes plus the associated class, that in this case is binary. The dataset is composed of 768 examples, and validation is done by a 10 fold cross validation.

When executing ENPC algorithm with the only parameter defined up to now, i.e. the number of iterations, set to 300, the results are not good. On training data, the algorithm obtains 94.21% success, but on test data it obtains only 65.72%, a similar result that the obtained by ZeroR classifier. This means that there is an overfitting of the algorithm on the training set, so the test set receives very bad results. These kinds of results are typical of nearest neighbour approaches when useless attributes are used for classification. For instance, IBK, for $k = 1$ obtains 70.18% success.

However, ENPC gives a single way to reduce this overfitting, only by modifying the classifier selection method. Section 2.3 showed that typically, after the execution of a defined number of iterations of the ENPC algorithm, the selected classifier from all the ones generated in previous iterations is the one with higher classification accuracy over the training data. A single way to improve generalization capabilities is to separate training set into two sets: the first one, called the evolution set, will be used for training, as is typical. But the second one will be used for classifier selection. So, in this case, the classifier selection will be done using different data to the one used for prototypes evolution. Figure 15 shows the results of executing the ENPC algorithm following this approach, and where the percentage of data used for evolution and for classifier selection ranges from 10 to 90.

The figure shows results for training and for testing when the 10 fold cross validation is executed, where the training result is the one obtained after classifier selection. The figure shows how when using a different set for classifier selection, even when the percentage

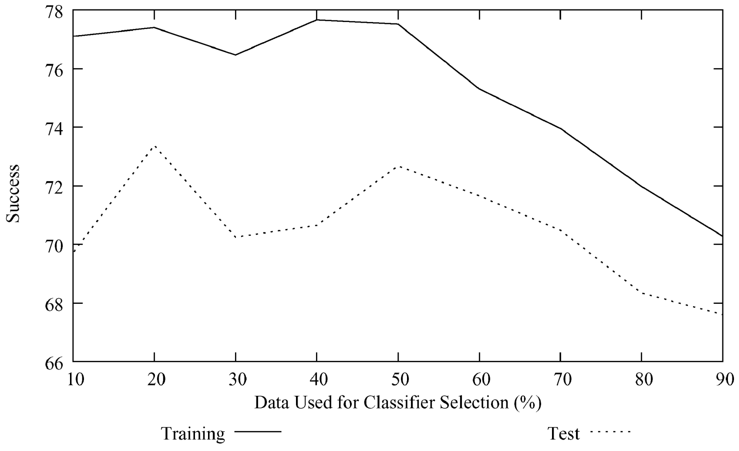

Figure 15. Results of the ENPC algorithm for different sizes of the evolution and classifier selection sets.

Table 10. Comparative results over the Pima indians diabetes database.

C4.5	PART	Naive Bayes	IBK ($k = 1$)	IBK ($k = 3$)	CNN	DEL	DROP 5	SNN	ENPC
73.83	75.26	76.3	70.18	72.65	69.78	71.61	73.05	67.97	73.38

of data used for this set is low, the differences among training and test sets are reduced from almost 30 (value obtained when the same data is used for prototypes evolution and classifier selection) down to a range of 10 points. The best result obtained is when data used for classifier selection is 20% of the training data, achieving 73.38% classification success.

Last, Table 10 shows different classification results obtained by other different algorithms. Added to the ones included previously (C4.5, PART, Naive Bayes, and IBK), we have also introduced comparative results with other approaches based on editing and condensing the training set, as is the case of CNN (Condensed Nearest Neighbour rule) (Hart, 1968), SNN (Selective Nearest Neighbour rule) (Ritter et al., 1975), DEL (Decremental Encoding Length) and DROP (Decremental Reduction OPtimization) (Wilson and Martinez, 2000). All the results are obtained from Wilson and Martinez (2000).

4. Conclusions and future work

The algorithm presented in this work is an evolutionary approach to solve the problem of finding a set of prototypes that are able to correctly classify the examples of a domain, following a 1-nearest prototype approach. The main advantages of this method are, on the one hand, that it is able to achieve high accuracy in all the domains where it has been tested, even compared with other techniques from the literature. On the other hand, the achievement of these good results is done without the user defining any initial conditions nor learning parameter.

To achieve this, a fully integrated technique has been developed that includes elements from other works, such as heuristics to introduce new prototypes, to eliminate other ones, labelling phases, etc. Previous works typically introduce modifications over other techniques that provides them the capability of defining the architecture. Typical solutions were genetic algorithms or heuristics to modify the architecture. These techniques can be split in two steps, defining or modifying the architecture and learning the problem with the new architecture, sometimes even in an iterative process. However, in this work, all the mechanisms are fully integrated, so it is not possible to separate the elements that define the architecture from the ones that solve the problem with the architecture.

Comparisons with other techniques have shown that the ENPC algorithm is able to successfully solve most of the problems presented without any additional parameter: the user only has to define the training and test sets, and an additionaly (optionaly) end condition. It is also well known that there is no technique able to behave better for all domains. In this work we achieve one of the best results for all the domains in the experiments. This makes our technique a good candidate when the domain is uncertain, or when it has to be included in automatic processes. This is the case, for instance, in applying ENPC as function approximator in reinforcement learning methods (Fernández and Borrajo, 2002).

The end condition used in this work is the number of iterations that the algorithm must be executed. However, other end conditions could be defined, given that in all the experiments it is shown that, even though the algorithm does not ever converge to a fixed solution, it converges to a small set of very similar solutions. Note that the goal of the algorithm is not to find an "optimal" solution, which in most cases may not exist, but a "useful" solution to the problem. In this sense, we define a "useful" solution as a solution with a high accuracy and a low number of prototypes: the algorithm does not ensure that both values, or the combination of both, are optimal. But it will be very close.

Future work is oriented to the concept of similarity of the data, i.e. the distance metric used. On one hand, the use of weighted distance metrics that learn which attributes are important and which ones are not is an important issue, as well as techniques that are able to automatically normalize the data without losing information. On the other hand, the adaptation of all the ideas presented in this work to other non Euclidean domains, where different distance metrics are defined appears as an interesting goal.

Note

1. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

References

Aha, D. and K. Kibler. (1991). "Instance-Based Learning Algorithms." *Machine Learning* 6, 37–66.

Bermejo, S. and J. Cabestany. (2000). "A Batch Learning Algorithm Vector Quantization Algorithm for Nearest Neighbour Classification." *Neural Processing Letters* 11, 173–184.

Bezdek, J.C. and L.I. Kuncheva. (2001). "Nearest Neighbour Classifier Designs: An Experimental Study." *International Journal of Intelligent Systems* 16, 1445–1473.

Bezdek, J.C., T.R. Rechherzer, G.S. Lim, and Y. Attikiouzel. (1998). "Multiple-Prototype Classifier Design." *IEEE Transactions on Systems, Man and Cybernetics* 28(1), 67–79.

Blake, C.L. and C.J. Merz. (1998). "UCI Repository of Machine Learning Databases."

Burrascano, P. (1991). "Learning Vector Quantization for the Probabilistic Neural Network." *IEEE Transactions on Neural Networks* 2(4), 458–461.

Cagnoni, S. and G. Valli. (1994). "OSLVQ: A Training Strategy for Optimum-Size Learning Vector Quantization Classifiers." In *IEEE International Conference in Neural Networks*, pp. 762–775.

Duda, R.O. and P.E. Hart. (1973). *Pattern Classification and Scene Analysis*. John Wiley And Sons.

Fernández, F. and D. Borrajo. (2002). "On Determinism Handling While Learning Reduced State Space Representations." In *Proceedings of the European Conference on Artificial Intelligence (ECAI 2002)*, Lyon, France.

Fernández, F. and P. Isasi. (2001). "Designing Nearest Neighbour Classifiers by the Evolution of a Population of Prototypes." In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'01)*, pp. 172–180.

Fernández, F. and P. Isasi. (2002). "Automatic Finding of Good Classifiers Following a Biologically Inspired Metaphor." *Computing and Informatics* 21(3), 205–220.

Frank, E. and I.H. Witten. (1998). "Generating Accurate Rule Sets Without Global Optimization." In *Proceedings of the Fifteenth International Conference on Machine Learnin*.

Fritzke, B. (1994). "Growing Cell Structures—A Self-Organizing Network for Unsupervised and Supervised Learning." *Neural Networks* 7(9), 1441–1460.

Gersho, A. and R.M. Gray. (1992). *Vector Quantization and Signal Compression*. Kluwer Academic Publishers.

Hart, P.E. (1968). "The Condensed Nearest Neighbour Rule." *IEEE Transactions on Information Theory*.

John, G.H. and P. Langley. (1995). "Estimating Continuous Distributions in Bayesian Classifiers." In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345.

Kohonen, T. (1984). *Self-Organization and Associative Memory*, 3rd ed. Berlin, Heidelberg: Springer, 1989.

Kuncheva, L.I. and J.C. Bezdek. (1998). "Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search?" *IEEE Transactions on Systems, Man and Cybernetics* 28(1), 160–164.

Linde, Y., A. Buzo, and R.M. Gray. (1980). "An Algorithm for Vector Quantizer Design." In *IEEE Transactions on Communications*, Vol 1, Com-28, No. 1, pp. 84–95.

Lloyd, S.P. (1982). "Least Squares Quantization in PCM." In *IEEE Transactions on Information Theory*, pp. 127–135.

Mao, K.Z., K.-C. Tan, and W. Ser. (2000). "Probabilistic Neural-Network Structure Determination for Pattern Classification." *IEEE Transactions on Neural Networks* 11(4), 1009–1016.

Merelo, J.J., A. Prieto, and F. Morán. (1998). "Optimization of Classifiers using Genetic Algorithms." In Honavar, P. (ed.), *Advances in Evolutionary Synthesis of Neural Systems*. MIT Press.

Pal, N.R., J.C. Bezdek, and E.C.K. Tsao. (1993). "Generalized Clustering Networks and Kohonen's Self-Organizing Scheme." *IEEE Transactions on Neural Networks* 4(4).

Patanè, G. and M. Russo. (2001). "The Enhanced LBG Algorithm." *Neural Networks* 14, 1219–1237.

Pérez, J.C. and E. Vidal. (1993). "Constructive Design of LVQ and DSM Classifiers." In Mira, J., Cabestany, J., and Prieto, A. (eds.), *New Trends in Neural Computation*, Vol. 686 of *Lecture Notes in Computer Science*, Springer Verlag.

Quinlan, J.R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ritter, G.L., H.B. Woodruff, S.R. Lowri, and T.L. Isenhour. (1975). "An Algorithm for a Selective Nearest Neighbour Decision Rule." *IEEE Transactions on Information Theory* 21(6), 665–669.

Russo, M. and G. Patanè. (2000). "ELBG Implementation." *International Journal of Knowledge Based Intelligent Engineering Systems* 2(4), 94–109.

Specht, D.F. (1990). "Probabilistic Neural Networks." *Neural Networks* 3(1), 109–118.

Wilson, D.R. and T.R. Martinez. (2000). "Reduction Techniques for Instance Based Learning Algorithms." *Machine Learning* 38, 257–286.

Witten, I.H. and E. Frank. (2000). *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.

Zhao, Q. and T. Higuchi. (1996). "Evolutionary Learning of Nearest Neighbour MLP." *IEEE Transactions on Neural Networks* 7(3), 762–767.