Computing and Informatics, Vol. 21, 2002, 1–16, V 7. 11.

AUTOMATIC FINDING OF GOOD CLASSIFIERS FOLLOWING A BIOLOGICALLY INSPIRED METAPHOR

Fernando Fernández, Pedro Isasi

Universidad Carlos III de Madrid
Avda de la Universidad 30
28911 Leganés
Madrid, Spain
e-mail: ffernand@inf.uc3m.es, isasi@ia.uc3m.es

Manuscript received 29 April 2002; revised 10 October 2002
Communicated by Gheorghe Păun

Abstract. The design of nearest neighbour prototypes can be seen as the partitioning of the whole domain in different regions that can be directly mapped to a class. The definition of the limits of these regions is the goal of any nearest neighbour based algorithm. These limits can be described by the location and class of a reduced set of prototypes and the nearest neighbour rule. The nearest neighbour rule can be defined by any distance metric, while the set of prototypes is the matter of design. To compute this set of prototypes, most of the algorithms in the literature require some crucial parameters as the number of prototypes to use, and a smoothing parameter. In this work, an evolutionary approach based on Nearest Neighbour Classifiers (ENNC) is introduced where no parameters are involved, thus overcoming all the problems derived from the use of the above mentioned parameters. The algorithm follows a biological metaphor where each prototype is identified with an animal, and the regions of the prototypes with the territory of the animals. These animals evolve in a competitive environment with a limited set of resources, emerging a population of animals able to survive in the environment, i.e. emerging a right set of prototypes for the above classification objectives. The approach has been tested using different domains, showing successful results, both in the classification accuracy and the distribution and number of the prototypes achieved.

Keywords: Classifier design, nearest neighbour classifiers, evolutionary learning, biologically inspired algorithms

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 [3]. When the set C is very reduced, this kind of classifiers can be called Nearest Prototype Classifiers [2] (NPC), but, given that the limits among them are not defined in detail, we will keep using the first nomenclature.

These classifiers are very much related to vector quantization techniques [7] 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 in 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 several different parameters, that are often summarized in a unique learning parameter. This learning parameter defines whether the updates over the classifier are higher (typically at the beginning of the learning phase) or lower (typically at the end of the learning phase).

Many discussions about what is the right technique to use can be found in the literature [11]. Some approaches based on clustering techniques [16, 15, 1] are based on 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 [12]. The second step is to classify these prototypes on the basis of previously labelled examples and the nearest neighbour rule. Although this approach produces good results, it is obvious that to introduce information about the classification performance in the location of the prototypes it seems to be needed to achieve a higher performance.

Neural networks approaches are also very common in the literature, like the LVQ algorithm [10] and the works with radial basis functions [6]. 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, as the average quantization distortion [18] or the accuracy in the classification [17]. On the other hand, other approaches try to define the optimal size of the classifier first, and then to learn it using the previous value. Genetic algorithms 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 [14]. Following this idea, in [19], an evolutionary approach can be found based on the R^4 rule (recognition, remembrance, reduction and review) to evolve the nearest neighbour multi-layer perceptrons.

In this work, an evolutionary approach called Evolutionary Nearest Neighbour Classifier (ENNC) [4] is introduced to dynamically define the number of prototypes of the classifier as well as the location of these prototypes. The main difference from the previous works is that this approach is a fully integrated algorithm. Most algorithms that solve initialization problems take advantage of a previous known technique and modify it 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 used to modify the size of the classifier and the learning algorithm are fully integrated and cannot be used separately from the other part.

The algorithm is summarized as follows. The classifier is defined as a population of animals (prototypes) that must fight to eat vegetables (training examples) that allows them to survive and to find an equilibrium in the environment (optimum number of prototypes). The method allows the animals to execute several operators, like to introduce new animals (reproduction), to change their specie (mutation), etc. in order to improve their adaptation to the environment (the global accuracy of the classifier). Furthermore, the execution of these operators is controlled by the animals themselves, taking into account their relationship with the rest of the animals in the environment. So, the evolution will allow the individuals to locate themselves in the right position, and to be labelled in the right way, achieving the equilibrium only when the right number of prototypes is achieved.

In the next section, the main concepts used are presented, showing the ecosystem metaphor; Section 3 describes the algorithm in depth. Section 4 shows principal experiments performed and a comparison with previous works, while Section 5 presents some conclusions and suggests topics for further research.

2 BIOLOGICALLY INSPIRED DESIGN OF NEAREST NEIGHBOUR CLASSIFIERS (ENNC)

The ENNC algorithm offers an evolutionary point of view to the design of nearest neighbour classifiers. The main advantage of this method is that neither the number of prototypes used, nor an initial set of prototypes are required. The first difference among this algorithm and previous evolutionary approaches is the way of representing the population: in this case, and following the Michigan approach, each chromosome represents only one prototype, and not a whole classifier, so the classifier is represented by the whole population. The main concepts can be defined as follows:

Prototype/Animal, r_i**.** Each prototype/animal is composed by its localization in the environment and its class/specie.

Classifier/Population, C. A set of N prototypes or animals $C = \{r_1, \ldots, r_N\}$.

Region, r_i**.** The environment is divided into a set of N regions defined by the localization of the animals and the nearest neighbour rule. In this sense, there is a direct relationship among the location of the animals and the regions (regions are calculated from prototype localization), so in the rest of this work, we may talk about regions, prototypes and animals indistinctly. Each animal only eats vegetables in its own region.

Pattern/Vegetable, v_r**.** It is each of the examples that will be used for training or testing the system. They all compose a set $V = \{v_1, \ldots, v_M\}$, and, as well as

the prototypes, they are composed by its location and by their class. They are considered as vegetables of the biological system.

Class/Specie, s_j**.** Both animals and vegetables belong to a class or specie from the set $S = \{s_1, \ldots, s_L\}$. The goal of an animal r_i of specie s_j is to eat as many vegetables of class s_j as possible and not to eat vegetables of other classes $s_k \neq s_j$.

Quality/Health of a prototype/animal. This is a measure of the goodness of the prototype, taking into account the number of patterns into its region, $apportation_{r_i}$, and whether those patterns belong to the same class than the prototype or not, $accuracy_{r_i}$. The final value is computed as follows:

$$quality_{r_i} = min(1, accuracy_{r_i} * apportation_{r_i}), \tag{1}$$

where r_i is the prototype we are computing its health, and a maximum value of 1 is included in order to normalize the measure.

The second main difference of this algorithm with previous evolutionary approaches comes from the operators that are used to evolve. In this case, most of the operators are based on heuristics of previous works [1, 15, 6, 13, 18], and new ones have been incorporated. So the learning phase is an iterative process that execute several operators over each individual. Each of this iteration is called a year in the animals life, and the year is divided into four seasons: spring, summer, fall and winter. In each season, different operators are executed, and are summarized in Table 1.

Season	Operators	Description
Spring	Mutation	Each animal changes its own specie to the majority specie of vegetables in its region
Summer	Reproduction	The animals reproduce to create animals that eat what they do not want to eat
Fall	Fight and Move	The animals fight against other animals and move to a different position to get more food
Winter	Die	Weak animals die

Table 1. Phases of the algorithm and operators used in each phase

3 THE ALGORITHM

The algorithm follows the flow defined in Figure 1. The algorithm starts with a single initialization, followed of an iterative process of evolution, where the different

operations are executed. In this section, all these operations are defined from the biological point of view. A formal description of all these operations can be found in [5].

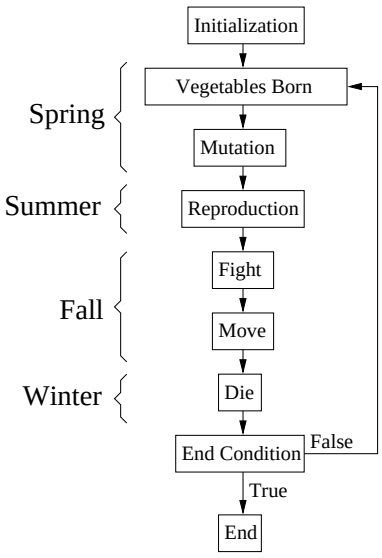


Fig. 1. ENNC algorithm

3.1 Initializing

One relevant feature of our method is the absolute elimination of initial conditions. These initial conditions are usually summarized in three ones: the number of prototypes, the initial set of prototypes and a smoothing parameter. The ENNC algorithm allows to learn 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 mentioned "quality" measure.
- The initial location of the only one prototype is not relevant (it clusters all the domain, wherever it is located).
- There are no learning parameters. The method automatically adjusts the intensity of change in prototypes taking into account their qualities in each iteration.

3.2 Spring

The spring season is the time when the **vegetables are born**. All the animals are placed in their own region, and will recollect all the vegetables in its region. The

way to define whether a vegetable belongs to one animal or to another is based on the nearest neighbour rule.

At the end of spring, each animal knows the quantity of vegetables of each specie that it can eat, so it will become (modify its state) to the specie of the most abundant specie of vegetables. This operator corresponds with the labelling phase of the unsupervised learning approaches [1, 15], but in this case, the supervision is included in each iteration and not only in a posterior phase. This operator is called **mutation operator** and it is shown in Figure 2. In the figure, an animal of specie 1 becomes to specie 2, given that vegetables of specie 2 are the most populated in its region.

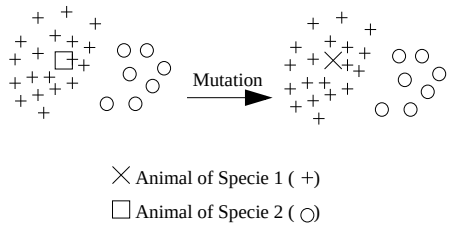


Fig. 2. Example of mutation operator execution

3.3 Summer

Summer is the season where animals reproduce (second operator). In this case the reproduction is not sexual, and an animal only reproduce if it needs another animal that eats what it does not want to eat, so there is a selfish motivation. In a neural network domain, reproduction is equivalent to the insertion of new neurons in the net based on the accuracy of the classifier [6].

So an animal only reproduces if, vegetables of different classes are found in its region. The probability of reproduction is proportional to the difference among the number of vegetables of each class in its region. Newborn animal is located in order to increase the ancestor performance, as shown in Figure 3.

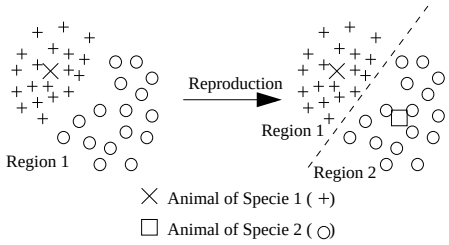


Fig. 3. Example of reproduction operator execution

3.4 Fall

Fall is the time where food starts to scarce, and the animals decide to look for more food. In this sense, fall have two phases. In the first one, animals can fight, in order to steal territory from other animals and to get more food. In the second phase, animals locate themselves in an optimum place to spend the winter and to wait for the next spring.

1. Fights: An animal can decide to fight with other animals in order to get more food. Fight operator is executed for each animal, and has the following phases:

   (a) To choose a rival by assigning all the animals in its neighbourhood a probability proportional to the quality of that animal and using a roulette as the selection method.

   (b) Once the rival is selected, the animal has to decide whether to fight or not. The probability of fighting is proportional to the difference in health of both rivals.

   (c) Once the rival has been selected and the animal decides to fight, there are two possibilities:

      i The animals do not belong to the same specie. In this case, there is no sense to fight, and both animals make an agreement that the second one gives the vegetables required to the first one, as shown in Figure 4.

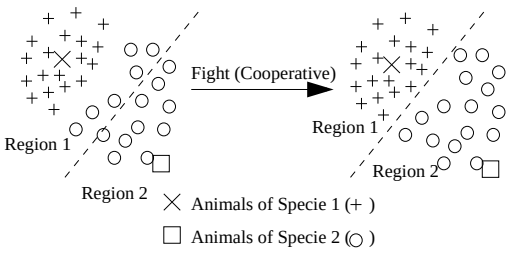


Fig. 4. Example of fight operator execution with cooperation

      ii Both animals belong to the same specie. Animals fight, with a probability of victory proportional to the animals health. The winner steals food from the loser, as shown in Figure 5. If the winner is allowed to steal all the food from the loser, the loser dies.

2. Move: The move operator implies to relocate each animal in the best expected place to spend the winter and wait until next spring. So each animal decides to move to the centroid of the vegetables of the same class, as shown in Figure 6 for animal 2. This operator is based on the Lloyd iteration of the GLA algorithm [13].

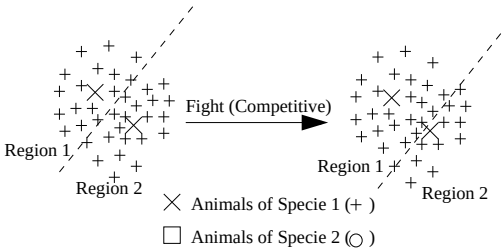


Fig. 5. Example of fight operator execution with competition

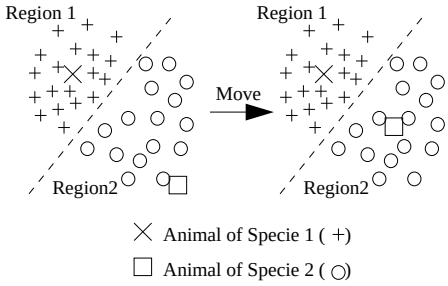


Fig. 6. Example of move operator execution

3.5 Winter

In winter, weak animals (those which have low quality values) increase their dying probability. This probability is 1 minus the double of the health. Then, healthy animals will survive, while weak animals with health of less than 0.5 might die. In the neural network bibliography, a deeper documentation about which neurons to select in order to simplify the network structure can be found [6, 18]. At the end of this season, all vegetables disappear.

3.6 End Condition

End condition is the hardest element to define in this approach. It is supposed that the algorithm conversion to an optimal solution is desirable, but: what is an optimal solution? In this area, an optimal solution is said to be the solution that achieves the highest classification accuracy with the smallest number of prototypes. However, what is the heaviest parameter? Some people can think that if increasing the number of prototypes, we can increase the accuracy of the classifier; it is better to increase this number, but over-fitting problems may occur and the generalization capabilities may be reduced. On the other hand, 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 to evolve, to store a set of paradigmatic classifiers

and let the user to choose the most appropriate ones. Obviously, a lot of different approaches could be introduced to decide when to stop. Several of them can be found in [5].

4 EXPERIMENTS

In this section, two experiments performed with the ENNC algorithm are shown. The first one is a data set of Gaussian-distributed examples. It is a single experiment that will show how the algorithm works. The second experiment in Section 4.2 allows to compare this approach with previous ones in the literature. For both experiments, the end condition is defined by a maximum number of iterations enough to achieve good solutions in all the cases.

4.1 Gaussian-Distributed Data

In this experiment, two different classes are defined following the distributions shown in Figure 7.

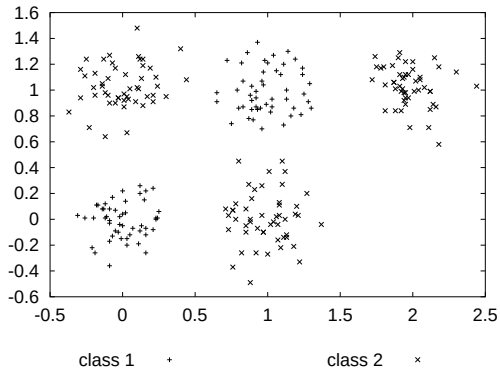


Fig. 7. Gaussian-distributed data

The data set consists of 250 examples, where 200 were used for training and 50 for test. The ENNC algorithm starts with a population of 1 prototype that is supposed to evolve to find a right set of prototypes. The ENNC algorithm is a stochastic method, so different executions may achieve different results. To verify whether the solutions achieved are similar, the algorithm is executed 20 times, each of them of a length of 100 iterations. The results are given in Table 2, where the information of the best classifier obtained in each execution is shown: iteration where it was evolved, accuracy over the test set and the number of prototypes.

These results show that most of the executions (95 % of the cases) achieve the optimal solution (5 prototypes and a 100 % of success) over the test. One execution achieved the same classification accuracy but needs one more prototype.

Iteration	Accuracy (%)	Prototypes
3	100.000	5
13	100.000	5
8	100.000	5
10	100.000	5
6	100.000	6
6	100.000	5
6	100.000	5
11	100.000	5
11	100.000	5
10	100.000	5
9	100.000	5
11	100.000	5
10	100.000	5
8	100.000	5
16	100.000	5
6	100.000	5
6	100.000	5
6	100.000	5
6	100.000	5
14	100.000	5

Table 2. Results of different executions of ENNC algorithm over Gaussian-distributed data

Figure 8 shows the evolution of one of these executions. The x-axis shows the iterations of the algorithm, while the y-axis shows the accuracy of the classifier in that iteration for the training and the test sets, as well as the number of prototypes used. Figure 9 allows to understand this evolution by showing the state of the classifier at the end of four of the iterations of the algorithm. Figure 9(a) shows the result of the first iteration, where a new prototype has been introduced to the initial one. Figures (b), (c) and (d) show the results of iterations 4, 6, and 14, respectively, each one with one more prototype than in the previous figure.

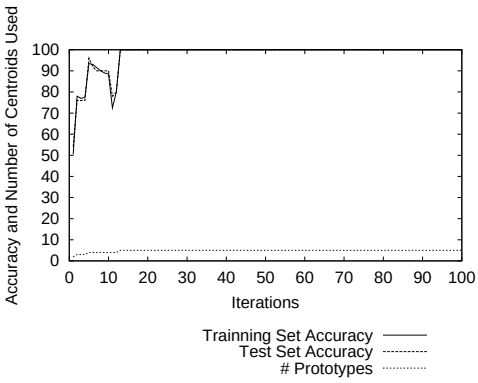
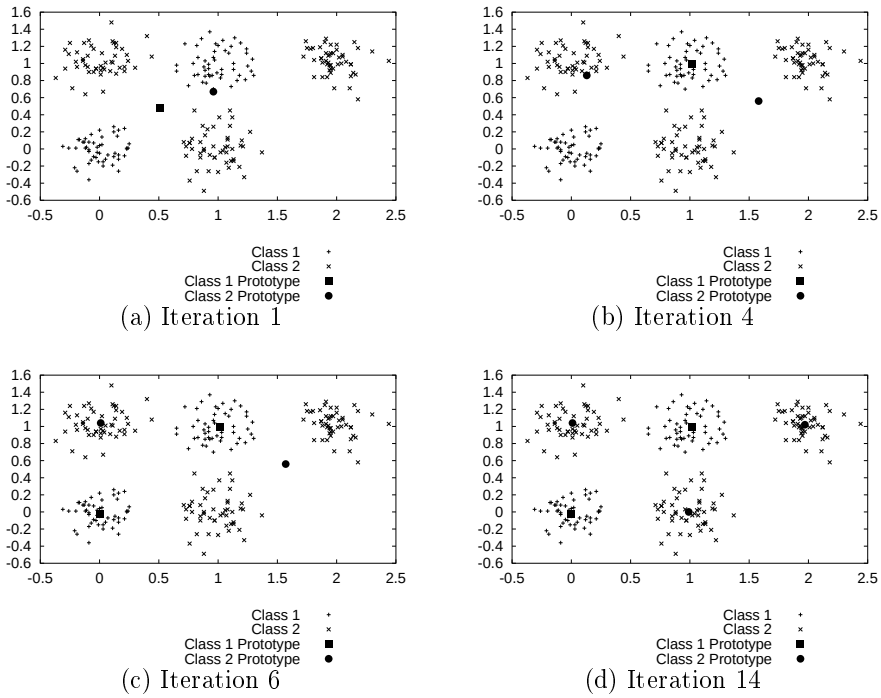


Fig. 8. Evolution of the ENNC algorithm over Gaussian-distributed data



(a) Iteration 1 (b) Iteration 4

(c) Iteration 6 (d) Iteration 14

Fig. 9. Evolution of the prototypes

4.2 Straight Line Class Boundaries

This classification problem was first defined by [9] and used in [8] to show the performance of the DSM classifier. The domain consists on two different classes defined as shown in Figure 10, with 6400 samples for training and 6400 for test. DSM algorithm belongs to the family of LVQ algorithms and works in the following way: it selects a small set of training samples and uses them as seed of the learning algorithm. This algorithm gradually adapts their location to correctly classify the whole data set.

In Figure 10 a solution of the problem with only 10 prototypes is shown. As in the previous experiment, we have executed the ENNC algorithm 20 times to verify its behaviour, each run of 300 iterations. For each run, the best classifier over the test set is chosen. Results are summarized in Table 3 and show that the algorithm converges to solutions in the range of 30–40 prototypes, with approx. 98 % accuracy. The average success of the 20 results is 98.14 and is the value used for comparisons with the results reported in [8] and shown in Table 4. The table shows how this value improves the results of LVQ1, and it is very close to the results of the net trained with the backpropagation algorithm, while it is not able to achieve the results of

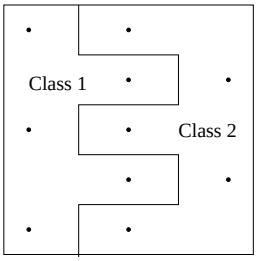


Fig. 10. Straight line class boundaries domain and a solution

DSM. Anyway, note that our approach automatically defines the right number of prototypes.

Figure 11 shows the evolution of one of the executions. In the initial iterations, the number of prototypes used is below 10, and the accuracy of the classifier keeps below 90 %. Once the number of prototypes is higher than 10, the classifier starts

Iteration	Accuracy	Prototypes
248	98.672	30
238	98.203	29
225	98.484	31
276	97.922	32
258	98.234	29
283	98.078	29
196	97.969	30
181	98.047	32
252	98.375	37
206	97.953	31
300	98.109	34
100	97.859	30
213	98.375	35
294	97.984	31
177	97.922	30
253	97.781	36
40	97.766	11
249	98.547	30
139	98.297	28
237	98.094	33

Table 3. Results of different executions of ENNC algorithm over straight line class boundaries domain

Prototypes	DSM	LVQ1	Backpropagation
6	92.86	81.00	90.58
8	96.18	80.45	98.47
9	98.14	85.36	98.73
10	99.57	87.66	98.34
20	99.55	95.66	98.47
24	99.59	96.94	98.62
50	99.51	97.49	98.44
250	99.21	98.16	98.45

Table 4. Comparative results over straight line class boundaries domain

to achieve results of approximately 98 %. The number of prototypes is successively increased in order to improve the accuracy. This number of prototypes does not stabilize in a specific value, given that more than one similar solution can be found. In this sense, the system is able to oscillate between close solutions in order to leave the last decision to the user. Last, note that both training and test sets keep similar values while learning.

Figure 12 shows the prototypes of the classifier obtained in the first execution of the ENNC algorithm. There are 30 prototypes, achieving 98.672 % accuracy.

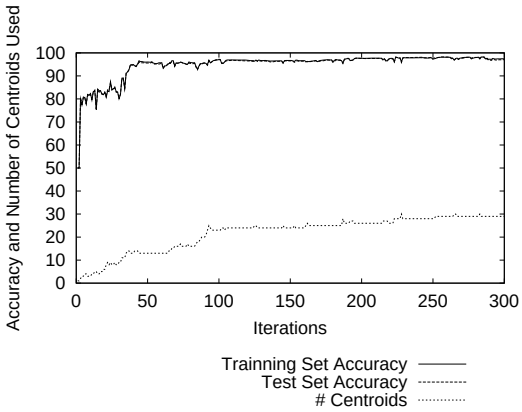


Fig. 11. Execution of the ENNC algorithm over straight line class boundaries domain

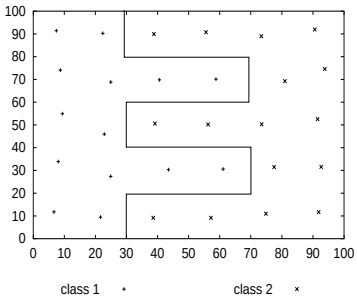


Fig. 12. Classifier of 30 prototypes obtained with ENNC in straight line class boundaries domain

5 CONCLUSIONS AND FUTURE WORK

This work has presented a biologically inspired approach to the problem of nearest neighbour classifiers design. This approach follows an ecosystem metaphor in which the prototypes are defined as animals and the training examples as animal resources

(vegetables). The biological perspective allows to define in a very simple way new concepts such as to steal vegetables, cooperative actions among the different animals, etc., while other typical operations such as introduce new prototypes, or to eliminate useless ones have been redefined.

The main advantages of this method are, on the one hand, that it is able to achieve a high accuracy in 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 definition by the user of any initial conditions or other parameters for learning. Furthermore, it is a fully integrated technique that includes elements from other works, as heuristics to introduce prototypes, to eliminate another ones, labelling phases, etc. Previous works typically introduce modifications over other techniques that provide them with the capability of defining the architecture. This way, other solutions were genetic algorithms or heuristics to modify the architecture. These techniques can be split into 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 those that learn the problem with the architecture.

Comparisons with other techniques have shown that the approach is able to successfully solve the problem presented without any additional parameter: the user only has to define the training and test sets, and one end condition.

Even though the algorithm is a stochastic method, it has shown that similar solutions are achieved when different runs are executed. This property ensures that the solutions achieved in a first run of the algorithm are very close to the optimal, so no more trials are required. Furthermore, given that the algorithm does not introduce more parameters, only one execution of the algorithm is enough to obtain a successful classifier.

Future work is oriented to the concept of similarity on the data, i.e. the distance metric used. On the one hand, the use of weighted distance metrics for learning which attributes are important and which are not, is an important issue, as well as introducing some techniques for automatically normalizing the data without losing information. On the other hand, the adaptation of all the ideas presented in this work to other not Euclidean domains, where different distance metrics to compute square error was used in this work, appears as an interesting task.

REFERENCES

[1] Bermejo, S.—Cabestany, J.: A Batch Learning Algorithm Vector Quantization Algorithm for Nearest Neighbour Classification. Neural Processing Letters, Vol. 11, 2000, pp. 173–184.

[2] Bezdek, J. C.—Kuncheva, L. I.: Nearest Neighbour Classifier Designs: An Experimental Study. International Journal of Intelligent Systems, Vol. 16, 2001, pp. 1445–1473.

[3] Duda, R. O.—Hart, P. E.: Pattern Classification and Scene Analysis. John Wiley and Sons, 1973.

[4] Fernández, F.—Isasi, P.: 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, 2001.

[5] Fernández, F.—Isasi, P.: Evolutionary Design of Nearest Neighbour Classifiers. Technical Report UC3M-TR-CS-2001-08, Universidad Carlos III de Madrid, 2001.

[6] Fritzke, B.: Growing Cell Structures — a Self-Organizing Network for Unsupervised and Supervised Learning. Neural Networks, Vol. 9, 1994, No. 7, pp. 1441–1460.

[7] Gersho, A.—Gray, R. M.: Vector Quantization and Signal Compression. Kluwer Academic Publishers, 1992.

[8] Geva, S.—Sitte, J.: Adaptive Nearest Neighbour Pattern Classification. IEEE Transactions on Neural Networks, Vol. 2, 1991, No. 2, pp. 318–322.

[9] Hart, P. E.: The Condensed Nearest Neighbour Rule. IEEE Transactions on Information Theory, Vol. 14, 1968, pp 515–516.

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

[11] Kuncheva, L. I.—Bezdek, J. C.: Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search? IEEE Transactions on Systems, Man and Cybernetics, Vol. 28, 1998, No. 1, pp. 160–164.

[12] Linde, Y.—Buzo, A.—Gray, R. M.: An Algorithm for Vector Quantizer Design. IEEE Transactions on Communications, Vol. 1, 1980, No. 1, pp. 84–95.

[13] Lloyd, S. P.: Least Squares Quantization in PCM. In IEEE Transactions on Information Theory, 1982, No. 28, pp. 127–135.

[14] Merelo, J. J.—Prieto, A.—Morán, F.: Optimization of Classifiers Using Genetic Algorithms. In Patel Honavar, editor, Advances in Evolutionary Synthesis of Neural Systems, MIT press, 1998.

[15] Pal, N. R.—Bezdek, J. C.—Tsao, E. C. K.: Generalized Clustering Networks and Kohonen's Self-Organizing Scheme. IEEE Transactions on Neural Networks, Vol. 4, 1993.

[16] Patanè, G.—Russo, M.: The Enhanced LBG Algorithm. Neural Networks, Vol. 14, 2001, pp. 1219–1237.

[17] Pérez, J. C.—Vidal, E.: Constructive Design of LVQ and DSM Classifiers. In J. Mira, J. Cabestany, and A. Prieto, editors, New Trends in Neural Computation, Vol. 686 of Lecture Notes in Computer Science. Springer Verlag, 1993.

[18] Russo, M.—Patanè, G.: ELBG Implementation. International Journal of Knowledge Based Intelligent Engineering Systems, Vol. 2, 2000, No. 4, pp. 94–109.

[19] Zhao, Q.—Higuchi, T.: Evolutionary Learning of Nearest Neighbour MLP. IEEE Transactions on Neural Networks, Vol. 7, 1996, No. 3, pp. 762–767.

Fernando Fernández is currently Ph.D. candidate in computer science at Universidad Carlos III de Madrid (UC3M). He received his B.Sc. in 1996 from Universidad Complutense de Madrid and his Masters in 1999 from UC3M, both in Computer Science. His doctoral dissertation concerns finding efficient state space representations in reinforcement learning problems, and their application to robotic domains in which information received from sensors is continuous. In the summer and fall of 2000, he was a visiting student at the Center for Engineering Science Advanced Research at Oak Ridge National Laboratory (Tennessee) and from 1998 to 2000 he participated in a European funded research and development project. His research interests include reinforcement learning, autonomous robotics, neural networks and nearest neighboour approaches for supervised and unsupervised learning.

Pedro Isasi is currently full professor in the Department of Computer Science at Universidad Carlos III de Madrid (UC3M). He received his B.Sc. in 1991 and his PhD. in 1994, both in computer science at Universidad Politecnica de Madrid. He is in charge of the ScaLab laboratory of the Computer Science Department of UC3M from 1993, his main research fields are adaptive complex systems. He is author of many works in machine learning (theoretical — classification methods, coevolution, learning control knowledge, and practical — robotics control, chemical and hydroelectric Al power plant, stock market, etc.). His research interests include evolutionary computation, coevolution, unsupervised neural networks and radial basis functions.