# An Adaptive Michigan Approach PSO for Nearest Prototype Classification

Alejandro Cervantes, Inés Galván, and Pedro Isasi

Department of Computer Science, University Carlos III de Madrid
Avda. Universidad, 30. 28911 Leganés, Madrid, Spain
alejandro.cervantes@uc3m.es, inesmaria.galvan@uc3m.es, pedro.isasi@uc3m.es

**Abstract.** Nearest Prototype methods can be quite successful on many pattern classification problems. In these methods, a collection of prototypes has to be found that accurately represents the input patterns. The classifier then assigns classes based on the nearest prototype in this collection. In this paper we develop a new algorithm (called AMPSO), based on the Particle Swarm Optimization (PSO) algorithm, that can be used to find those prototypes. Each particle in a swarm represents a single prototype in the solution; the swarm evolves using modified PSO equations with both particle competition and cooperation. Experimentation includes an artificial problem and six common application problems from the UCI data sets. The results show that the AMPSO algorithm is able to find solutions with a reduced number of prototypes that classify data with comparable or better accuracy than the 1-NN classifier. The algorithm can also be compared or improves the results of many classical algorithms in each of those problems; and the results show that AMPSO also performs significantly better than any tested algorithm in one of the problems.

**Keywords:** Classification, Data Mining, Nearest Neighbor, Particle Swarm, Swarm Intelligence.

## 1 Introduction

This paper introduces a new technique to perform prototype selection to be applied in Nearest Neighbor classification. Nearest Neighbor (NN) is a lazy learning method where the class assigned to a pattern is the class of the nearest pattern known to the system, measured in terms of a distance defined on the feature (attribute) space where patterns are represented as points.

To perform NN classification, data is split in a training and a test set. Each pattern in the test set is classified with the class of the nearest pattern in the training set. This means that each pattern in the training set defines a region of the feature space where it determines the expected class of the test patterns. Those regions are called Voronoi regions. When Euclidean distance is used, the Voronoi regions are delimited by linear borders. This may or may not provide an optimum solution, as the actual solution of the problem may require non-linearly delimited Voronoi regions, which may be accomplished using the k-NN algorithm or non-Euclidean distances.

Both computational reasons and presence of noise in data lead to the development of techniques that reduce the number of patterns evaluated without increasing the classification error. These methods calculate a reduced set of prototypes, that are the only ones used for classification. In instance selection methods [1] prototypes are a subset of the pattern set, but this is not true for all the prototype selection methods [2], where prototypes may be selected in any position in the attribute space.

Our method selects those prototypes by first allocating a population of solutions that represent the prototypes and then moving them in the attribute space in a way inspired by the Particle Swarm Optimization (PSO) algorithm [3].

The PSO algorithm is a biologically-inspired algorithm motivated by a social analogy. The algorithm is based on a set of potential solutions which evolves to find the global optimum of a real-valued function (fitness function) defined in a given space (search space). Particles represent the complete solution to the problem and move in the search space using both local information (the particle memory) and neighbor information (the knowledge of neighbor particles). In the standard approach of PSO, a potential solution is encoded in each particle and the swarm finally converges to a single solution.

However, our method, called AMPSO (Adaptive Michigan PSO) has important differences as it uses a Michigan Approach; this term is borrowed from the area of genetic classifier systems ([4][5]). In this approach a member of the population does not encode the whole solution to the problem, but only part of the solution. To implement this behavior, movement and neighborhood rules of the standard PSO are changed. A related approach was used in [6], where a Michigan binary version of PSO was able to discover a set of induction rules for discrete classification problems.

The advantages of the Michigan approach versus the conventional PSO approach are: a) scalability and computational cost, as particles have much lower dimension; and b) flexibility and reduced assumptions, as the number of prototypes in the solution is not fixed.

Moreover, our algorithm does not use a fixed population of particles; given certain conditions, we allow particles to reproduce to adapt to some situations, and also we destroy useless particles to reduce the computational cost and the complexity (number of final prototypes) of the solution.

This paper is organized as follows: section 2 shows how the solution to the classification problem is encoded in the particles of the swarm; section 3 details the proposed AMPSO algorithm; section 4 describes the experimental setting and results of experimentation; finally, section 5 discusses our conclusions and future work related to the present study.

## 2 Encoding Prototypes in Particles

The PSO algorithm uses a population of particles each encoding a complete solution to an optimization problem. The position of each particle in the search space changes depending on the particle's fitness and the fitness of its neighbors.

In the Michigan-approach PSO we propose, each particle represents a potential prototype to be used to classify the patterns using the nearest neighbor rule. The particle position is interpreted as the position of a single prototype. Each particle has also a class; this class does not evolve following the PSO rules, but remains fixed for each particle since its creation.

The dimension of the particles is equal to the number of attributes of the problem. The attributes of the problem are transformed in continuous values in the $[0, 1]$ range; this means that maximum and minimum values for the particles' positions are fixed in all the dimensions.

A particle classifies a pattern when it is the closer particle (in terms of Euclidean distance) to that pattern.

## 3 The Adaptive Michigan PSO Algorithm

### 3.1 Algorithm Pseudo Code and Movement

Our algorithm is based in the PSO algorithm but performs some extra calculations and has an extra cleaning phase. The overall procedure follows. Our additions are explained in the following sections.

1. Load training patterns
2. Initialize swarm; dimension of particles equals number of attributes.
3. Insert N particles of each class in the training patterns.
4. Until max. number of iterations reached or success rate is 100%:
   (a) Check for particle reproduction and deletion. (see 3.6)
   (b) Calculate which particles are in the competing and non-competing sets of particles for every class (see 3.4).
   (c) For each particle,
      i. Calculate Local Fitness. (see 3.3)
      ii. Calculate Social Adaptability Factor. (see 3.5)
      iii. Find the closest particle in the non-competing set for the particle class (attraction center).(see 3.4)
      iv. Find the closest particle in the competing set for the particle class (repulsion center).(see 3.4)
      v. Calculate the particle's next position. (see Eq. 1 and Eq. 2 in 3.2)
   (d) Move the particles
   (e) Assign classes to the patterns in the training set using the nearest particle.
   (f) Evaluate the swarm classification success
   (g) If the swarm gives the best success so far, record the particles' current positions as "current best swarm".
5. Delete, from the best swarm found so far, the particles that can be removed without a reduction in the classification success value.
6. Evaluate the swarm classification success over the validation set and report result.

In step 5 of the previous procedure a reduction algorithm is applied after the swarm reaches its maximum number of iterations. Its purpose is to delete unused particles from the solution. Particles are removed one at a time, starting with the one with the worst local fitness value, only if this action does not reduce the swarm classification success rating over the training set. The "clean" solution is then evaluated using the validation set.

### 3.2 Movement Equations

In order to take into account the concepts previously described, the equation that determines the velocity at each iteration becomes the following (1).

$$v_{id}^{t+1} = \chi(w \cdot v_{id}^{t} + c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t}) +$$

$$c_2 \cdot \psi_2 \cdot sign(a_{id}^{t} - x_{id}^{t}) \cdot Sf_i + c_3 \cdot \psi_3 \cdot sign(x_{id}^{t} - r_{id}^{t}) \cdot Sf_i) \; . \tag{1}$$

where the meanings of symbols are: $v_{id}^{t}$, component in dimension d of the $i^{th}$ particle velocity in iteration $t$; $x_{id}^{t}$, same for the particle position; $c_1$ ,$c_2$, $c_3$, constant weight factors; $p_i$, best position achieved so far by particle $i$; $\psi_1$ ,$\psi_2$, $\psi_3$, random factors in the [0,1] interval; $w$, inertia weight; $\chi$, constriction factor; $a_i$, attraction center for particle $i$ ; $r_i$, repulsion center for particle $i$ ; $Sf_i$, Social Adaptability Factor ;

This expression allows the particle velocity to be updated depending on four different influences: the current velocity of the particle is retained from iteration to iteration (inertia term, $w \cdot v_{id}^{t}$); the particle's memory (individual term, $c_1 \cdot \psi_1 \cdot (p_{id}^{t} - x_{id}^{t})$); the current position of a particle to which the particle is attracted (attractive term, $c_2 \cdot \psi_2 \cdot sign(a_{id}^{t} - x_{id}^{t}) \cdot Sf_i$); and the current position of a particle from which the particle is repelled (repulsive term, $c_3 \cdot \psi_3 \cdot sign(x_{id}^{t} - r_{id}^{t}) \cdot Sf_i$). Also, if $a_i$ or $r_i$ does not't exist the respective term (attractive term or repulsive term) is ignored.

After velocity update, the particle position is calculated using (2).

$$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1} \; . \tag{2}$$

### 3.3 Local Fitness Function

In our approach, each particle has a Local Fitness value that measures its performance independently of the other particles. For this purpose, we use (3).

$$\text{Local Fitness} = \begin{cases} 0 & \text{if } \{g\} = \{b\} = \emptyset \\ \frac{G_f}{Total} + 2.0 & \text{if } \{b\} = \emptyset \\ \frac{G_f - B_f}{G_f + B_f} + 1.0 & \text{otherwise} \end{cases} \text{, where} \quad \begin{array}{l} G_f = \sum_{\{g\}} \frac{1}{d_{g,i}+1.0} \\ B_f = \sum_{\{b\}} \frac{1}{d_{b,i}+1.0} \; . \end{array} \tag{3}$$

where $\{g\}$ is the set of patterns of the same class classified by the particle; $\{b\}$ is the set of patterns of different class classified by the particle; $d_{g,i}, d_{b,i}$ are

the Euclidean distances between the patterns and the particle; and $Total$ is the number of patterns in the training set.

This fitness function gives higher values (greater than +2.0) to the particles that only classify patterns whose class matches the class of the particle, and assigns values in the range $[0.0, +2.0]$ to particles that classify some patterns of a wrong class.

In the lowest range, the particles only take into account local information (the proportion of good to bad classifications made by itself). In the highest range, the particle fitness uses some global information (the total number of patterns to be classified), to be able to rank the fitness of particles with a 100% accuracy (particles for which $\{b\} = \emptyset$).

By including in the formula the distance between patterns and particles we give higher fitness to particles close to the patterns they classify correctly, so they can move closer to the area where the patterns are located. This tendency may be compensated by the repulsion term.

### 3.4 Neighborhood for the Michigan PSO

Our algorithm uses a dynamic neighborhood, which means that on each iteration, movement of each particle may be influenced by different particles in the swarm:

- For each particle of class $C_i$, non-competing particles are all the particles of classes $C_j \neq C_i$ that currently classify at least one pattern of class $C_i$.
- For each particle of class $C_i$, competing particles are all the particles of class $C_i$ that currently classify at least one pattern of that class ($C_i$).

When the movement for each particle is calculated, that particle is both:

1. Attracted by the closest (in terms of Euclidean distance) non-competing particle in the swarm, which becomes the "attraction center" ($a_i$) for the movement. In this way, non-competing particles guide the search for patterns of a different class.
2. Repelled by the closest competing particle in the swarm, which becomes the "repulsion center" ($r_i$) for the movement. In this way, competing particles retain diversity and push each other to find new patterns of their class in different areas of the search space.

Other authors have already used the idea of repulsion in PSO in different ways. For instance, in [7] and [8], repulsion is used to increase population diversity in the standard PSO and to allow the swarm to dynamically adapt to change. In [9] a repulsive force is introduced to achieve particle diversification in a binary PSO.

### 3.5 Social Adaptability Factor

The social part of the algorithm (influence from neighbors) determines that particles are constantly moving towards their non-competing neighbor and far from their competing neighbor. However, particles that are already located in

the proximity of a good position for a prototype should rather try to improve their position and should possibly avoid the influence of neighbors.

To implement this effect we have generalized the influence of fitness in the sociality terms by introducing a new term in the PSO equations, called "Social Adaptability Factor" ($S_f$), that depends inversely on the "Best Local Fitness" of the particle. In particular, we have chosen plainly the expression in (4); value for the exponent $s_{exp}$ was set to 1.0 after some experimentation.

$$Sf_i = 1/(\text{Best Local Fitness}_i + 1.0)^{s_{exp}} \tag{4}$$

### 3.6 Reproduction and Deletion of Particles

Though the swarm starts with a fixed number of particles, we introduce two features that adjust the number of particles and their classes to the problem's requirements. The rules for reproduction and deletion of particles are:

- Particles that classify a set of patterns of several classes have a probability to give birth to one particle for each of classes in that set. The chance of "reproduction" is inversely proportional to the particle fitness (so the worst particles are more likely to reproduce) and proportional to a parameter ($P_r$). The particles are placed in the position of the "parent" particle and their velocities are randomized.
- Particles that do not classify any pattern have a chance to be deleted that grows linearly from 0 to the value of a parameter ($P_d$) in the last iteration.

## 4 Experimentation

### 4.1 Global Swarm Evaluation

The local fitness function defined above, is used for particles' movement. However, to evaluate the goodness of the swarm as a classifier system, we use the classification success rate ( 5).

$$\text{Swarm Evaluation} = \frac{Good\ classifications}{Total\ patterns} \cdot 100 \tag{5}$$

Given the NN criterion for classification, unclassified patterns are not possible, as every pattern is assigned the class of the nearest prototype.

The system stores the best swarm evaluation obtained when performing classification on the training set. This function is also used to evaluate the final best swarm success rate over the validation set.

### 4.2 Problems' Description

We perform experimentation on the problems summarized in Table 1. The first (diagonal) is an artificial bi-dimensional problem that is very simple for lineal classifiers, and it is used to understand the new algorithm's properties. We generate 500

random training patterns and 1500 validation patterns with coordinates in the $[0, 1]$ range. Patterns where $x > y$ are assigned class 1, and the rest are assigned class 0.

The rest are well-known real problems taken from the UCI collection, used for comparison with other classification algorithms. All the problems have real-valued attributes, so no transformation was done on data besides normalization. Success rates from several classification algorithms over some of these problems can be found in [2].

**Table 1.** Problems used in the experiments

| Name | Instances | Attrbs. | Classes | Class Distribution | Validation |
|---|---|---|---|---|---|
| Diagonal | 2000 | 2 | 2 | 1000 / 1000 | Train & Test |
| Diabetes | 768 | 8 | 2 | 500 / 268 | 10-fold CV |
| Bupa | 345 | 6 | 2 | 200 / 145 | 10-fold CV |
| Glass | 214 | 9 | 6 | 70 / 76 / 17 / 13 / 9 / 29 | 10-fold CV |
| Thyroid (new) | 215 | 5 | 3 | 150 / 35 / 30 | 10-fold CV |
| Wisconsin | 699 | 6 | 2 | 458 / 241 | 10-fold CV |

### 4.3 Parameter Selection

The values of the swarm parameters were selected after some preliminary experimentation.

In all cases we found that it was better to use a small value for the inertia coefficient ($w = 0.1$); the number of iterations was set to 300 after checking that number was roughly equal to double the average iteration in which the best result was achieved.

For the PSO coefficients, we used low values for the parameters in the diagonal problem ($c_1 = 0.5$, $c_2 = 0.15$, $c_3 = 0.05$), which means that movement is performed in small steps. In the other problems we used the same set of values ($c_1 = c_2 = 1.0$, $c_3 = 0.5$). Velocity was clamped to the interval $[-1.0, +1.0]$ and $\chi = 0.1$ in all the experiments.

To test reproduction and deletion of particles, we performed three series of experiments, the first with no reproduction nor deletion (Series 1, $P_r = P_d = 0.0$), the second with only reproduction (Series 2, $P_r = 0.1$, $P_r = 0.0$) and the third with reproduction and deletion(Series 3, $P_r = P_d = 0.1$).

In each one of the series we performed 100 runs for the Diagonal problem, and 10 runs for the problems that use 10-fold cross validation, which also gives a total of 100 runs over each. In all the experiments, we used 10 particles per class for the initial swarm.

### 4.4 Experimental Results

The results of AMPSO on all the data sets are shown in Table 2; in this table we compare our results with the results of our own tests with other algorithms,

**Table 2.** Experiment results (success rate) for the UCI data sets; the three AMPSO series compared to several commonly-used classifiers

| Domain | AMPSO Series 1 | AMPSO Series 2 | AMPSO Series 3 | IBK (K=1) | IBK (K=3) | J48 | PART | Naive Bayes | SMO |
|---|---|---|---|---|---|---|---|---|---|
| Diagonal | 94.26 | 95.71 | 95.56 | 97.93 | 97.64 | 95.22 | 95.50 | 96.30 | 97.76 |
| Diabetes | 74.14 | 74.45 | 74.25 | 70.44 | 73.88 | 74.48 | 74.18 | 75.69 | 76.63 |
| Bupa | 63.29 | 64.51 | 65.05 | 62.90 | 63.16 | 66.01 | 63.08 | 55.63 | 57.95 |
| Glass | 82.04 | 84.46 | 83.17 | 71.99 | 70.58 | 72.86 | 73.79 | 47.25 | 57.10 |
| Thyroid | 94.03 | 93.81 | 94.88 | 97.21 | 93.46 | 92.06 | 93.92 | 96.75 | 89.74 |
| Wisconsin | 96.46 | 96.66 | 96.48 | 95.14 | 96.42 | 94.70 | 95.14 | 95.99 | 96.85 |

using WEKA with the default parameters for each of the algorithms. In the table, IBK(k=1) means plain NN, and IBK(k=3), 3-NN classification.

The results are very similar though both Series 2 and 3 are slightly better than the AMPSO algorithm with no reproduction nor deletion (Series 1) in all the problems but the Thyroid problem.

The success rate in terms of accuracy for AMPSO is comparable or clearly better than the result of IBK with K=1 in all but the Diagonal and the Thyroid problems. In the rest of the problems, the patterns are correctly represented with the particles in the solution swarm.

For the Diagonal problem, the reason is that the learning bias of the algorithm is not favoring the accuracy of the solution. A very good solution can be found just placing one prototype of each class, located at the central position of each cluster; the solutions found by our method are worse than that trivial solution due to the fact that AMPSO explicitly searches near the classes boundary (the diagonal) where a solution is more difficult to find.

In the thyroid problem our hypothesis is that NN classification requires more prototypes than the particles generated by the algorithm. It may be difficult to beat the performance of 1-NN in this problem as it seems that it significantly improves any other method. Reduction of the prototype number is more likely to decrease accuracy in this case.

The algorithm significantly improved the accuracy of all the rest in the Glass problem. Good performance on this problem is possible due to the fact that non-euclidean distances seem not to improve the results of Euclidean distance in this case (see [2]). Our algorithm seems to significantly improve the results of all the other classifiers, being to our knowledge the best result over this data set.

In Table 3, we compare the results of the three series, to show the impact of the introduction of particle reproduction and removal over the algorithm's performance and the complexity of the solutions found, measured in terms of the average number of prototypes in the solution.

The results in Table 3 show that Series 2 produces a greater number of prototypes, that is, solutions are of greater complexity. Particle deletion reduces this number of prototypes in the final solution as expected. Those results also show that series 2 and 3 have greater computational costs (average number of iterations).

**Table 3.** Comparison of the three series of experiments

| | Series | Diagonal | Diabetes | Bupa | Glass | Thyroid | Wisconsin |
|---|---|---|---|---|---|---|---|
| Prototypes | 1 | 13.37 | 10.47 | 10.89 | 10.30 | 11.37 | 11.12 |
| | 2 | 14.48 | 13.52 | 12.61 | 12.76 | 14.67 | 19.32 |
| | 3 | 13.38 | 13.00 | 12.01 | 11.73 | 13.80 | 19.00 |
| Evaluations | 1 | 3222 | 3114 | 2918 | 10620 | 5235 | 3198 |
| | 2 | 3185 | 4116 | 3550 | 14236 | 7114 | 5959 |
| | 3 | 2909 | 4224 | 3908 | 12760 | 6737 | 8762 |

## 5 Conclusions

The purpose of this paper is to develop an effective algorithm for prototype creation, to be applied to classification problems. Our algorithm is based on a new approach of Particle Swarm Optimization (PSO); it determines a set of prototypes that represent the training patterns and can be used as a classifier using the nearest neighbor (1-NN) rule.

In the standard PSO, a straightforward encoding of a set of prototypes in each particle would produce a search space of high dimension that might hinder the swarm success. For this reason, we propose a Michigan Approach PSO (AMPSO), in which each particle represents a single prototype. In AMPSO the population of particles also changes over time to adapt to the problem.

The AMPSO algorithm introduces a local fitness function to guide the particles' movement and dynamic neighborhoods that are calculated on each iteration to ensure particles are influenced only by others that either compete or cooperate to classify part of the patterns.

We have tested the algorithm in an artificial problem and six well-known benchmark problems and we have found that the results are always close to or better than the standard nearest neighbor classifier (1-NN) with the limited number of prototypes that compose the solutions found by the algorithm. This proves that PSO can be used to produce a small representative set of prototypes.

When the results are compared to other classifiers, AMPSO can produce competitive results in all the problems, specially when used in problems where 1-NN classifier does not perform very well. Finally, AMPSO outperforms significantly all the algorithms on the Glass Identification data set, where it achieves more that 10% improvement on average.

The introduction of particle reproduction and deletion allows the number of particles in the swarm to grow to produce solutions of greater complexity but the extra cost in terms of computational cost has to be evaluated.

It is clear than further work could improve the algorithm performance if it makes AMPSO able to adaptively tune important parameters (such as the reproduction and deletion rate) to the problem. Also, any technique that may improve Nearest Neighbor classifiers' performance could be applied to AMPSO.

## Acknowledgments

This article has been financed by the Spanish founded research MEC project OPLINK::UC3M, Ref: TIN2005-08818-C04-02 and CAM project UC3M-TEC-05-029.

## References

1. Henry Brighton and Chris Mellish. Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172, 2002.
2. Fernando Fernández and Pedro Isasi. Evolutionary design of nearest prototype classifiers. *Journal of Heuristics*, 10(4):431–454, 2004.
3. J. Kennedy, R.C. Eberhart, and Y. Shi. *Swarm intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.
4. J.H. Holland. Adaptation. *Progress in theoretical biology*, pages 263–293, 1976.
5. Steward W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
6. Alejandro Cervantes, Pedro Isasi, and Inés Galván. A comparison between the pittsburgh and michigan approaches for the binary pso algorithm. In *Proceedings of the 2005 IEEE Congress on Evolucionary Computation, CEC 2005*, pages 290–297, 2005.
7. T. M. Blackwell and Peter J. Bentley. Don't push me! collision-avoiding swarms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1691–1696, 2002.
8. T. M. Blackwell and Peter J. Bentley. Dynamic search with charged swarms. In *Proceedings of the Genetic and Evolutionary Computation Conference 2002 (GECCO)*, pages 19–26, 2002.
9. Alejandro Cervantes, Pedro Isasi, and Inés Galván. Binary particle swarm optimization in classification. *Neural Network World*, 15(3):229–241, 2005.