IPADE: Iterative Prototype Adjustment for Nearest Neighbor Classification

Isaac Triguero, Salvador García and Francisco Herrera

***Abstract*—Nearest prototype methods are a successful trend of many pattern classification tasks. However, they present several shortcomings such as time response, noise sensitivity and storage requirements. Data reduction techniques are suitable to alleviate these drawbacks. Prototype generation is an appropriate process for data reduction that allows the fitting of a data set for nearest neighbor classification.**

This concise paper presents a methodology to learn iteratively the positioning of prototypes using real parameters' optimization procedures. Concretely, we propose an iterative prototype adjustment technique based on differential evolution (IPADE). The results obtained are contrasted with non-parametrical statistical tests and show that our proposal consistently outperforms previously proposed methods, thus becoming a suitable tool in the task of enhancing the performance of the nearest neighbor classifier.

***Index Terms*—Prototype generation, differential evolution, nearest neighbor, classification.**

I. Introduction

Classification is one of the most important tasks in machine learning and data mining [1], [2]. Most machine learning methods build a model during the learning process, these are known as eager learning methods [3], but there are some approaches where the algorithm does not need to obtain a model. These algorithms are known as lazy learning methods [4].

The Nearest Neighbor (NN) algorithm [5] and its derivatives belong to the family of lazy learning. It has shown itself to perform well for classification problems in many domains [2], [6] and it is considered one of the top ten methods in data mining [7]. NN is a non-parametric classifier, which requires the storage of the entire training set and the classification of unseen cases, finding the class labels of the closest instances to them. In order to determine how close two instances are, several distances or similarity measures have been proposed [8], [9], [10]. The effectiveness and simplicity of the NN may be affected with several weaknesses such as high computational cost, high storage requirement and sensitivity to noise. Furthermore, NN makes predictions over existing data and it assumes that input data perfectly delimits the decision boundaries among classes.

Several approaches have been suggested and studied in order to tackle the drawbacks mentioned above, for instance, weighting schemes [11], [12] have been widely used to improve the results of the NN classifier.

I. Triguero and F. Herrera are with the Department of Computer Science and Artificial Intelligence of the University of Granada, CITIC-UGR, Granada, Spain, 18071.
E-mails: triguero@decsai.ugr.es, herrera@decsai.ugr.es
S. García is with the Department of Computer Science of the University of Jaén, Jaén. Spain, 23071. E-mail: sglopez@ujaen.es

A succesful technique which simultaneously tackles the computational complexity, storage requeriments and sensitivity to noise of NN is based on data reduction. These techniques aim to obtain a representative training set with a lower size compared to the original one and with similar or even higher classification accuracy for new incoming data. Apart from feature selection [13], data reduction can be divided into two different approaches, known as prototype selection [14], [15] and Prototype Generation (PG) or abstraction [16], [17]. The former process consists of choosing a subset of the original training data, while PG can also build new artificial prototypes to better adjust the decision boundaries between classes in NN classification.

In the specialized literature, a great number of PG techniques have been proposed. Since the first approach PNN based on merging prototypes [18] and divide-and-conquer based schemes [19], many other proposals of PG can be found. For instance, MixtGauss [20], ICPL [17] and RSP [21].

Positioning adjustment of prototypes is another perspective within the PG methodology. It aims to correct the position of a subset of prototypes from the initial set by using an optimization procedure. Many proposals belong to this family, such as Learning Vector Quantization (LVQ) [22] and its successive improvements [23], [24], genetic algorithms [25] and Particle Swarm Optimization (PSO) [26], [27].

Many existing positioning adjustment of prototypes techniques start with an initial set of prototypes and try to improve the classification accuracy by adjusting it. Two schemes of initialization are commonly used:

- The number of representative instances for each class is proportional to the number of them in the input data.
- All the classes are represented by the same number of prototypes.

This initialization process becomes their main drawback due to the fact that this parameter can be very dependent on the problem tackled. Some PG approaches [25], [23] compute the number of needed prototypes to be retained automatically, but in complex domains, they require to retain many prototypes. We propose a novel procedure to automatically find the smallest reduced set, which achieves a suitable classification accuracy over different types of problems. This method follows an iterative prototype adjustment scheme with an incremental approach. At each step, an optimization procedure is used to adjust the position of the prototypes, and the method adds new prototypes if needed. As a second contribution of this work, we will adopt the Differential Evolution (DE) [28], [29] technique as optimizer. Our proposal will be denoted by Iterative Prototype Adjustment based on Differential Evolution (IPADE).

In experiments on 50 real-world benchmark data sets, the classification accuracy and reduction rate of our approach are investigated and its performance will be compared with classical and recent PG models.

In order to organize this paper, Section II describes the background of PG and DE. Section III explains the proposed algorithm IPADE. Section IV discusses the experimental framework and presents the analysis of results. Finally, in Section V we summarize our conclusions.

II. Background

This section covers the background information necessary to define and describe our proposal. Subsection II-A presents the background on PG. Next, Subsection II-B shows the main characteristics of DE.

A. Prototype Generation

PG is an important technique in data reduction. It has been widely applied to instance-based classifiers and can be defined as the application of instance construction algorithms over a data set to improve the classification accuracy of a nearest neighbor classifier.

More specifically, PG can be defined as follows: Let $\mathbf{x}_p$ be an instance where $\mathbf{x}_p = (\mathbf{x}_{p1}, \mathbf{x}_{p2}, ..., \mathbf{x}_{pm}, \mathbf{x}_{p\omega})$, with $\mathbf{x}_p$ belonging to a class ω of Ω possible classes given by $\mathbf{x}_{p\omega}$ and a m-dimensional space in which $\mathbf{x}_{pi}$ is the value of the i-th feature of the p-th sample. Furthermore, let $\mathbf{x}_t$ be an instance where $\mathbf{x}_t = (\mathbf{x}_{t1}, \mathbf{x}_{t2}, ..., \mathbf{x}_{tm}, \mathbf{x}_{t\psi})$, with $\mathbf{x}_t$ belonging to a class ψ, that it is unknown, of Ω possible classes. Then, let us assume that there is a training set TR which consists of n instances $\mathbf{x}_p$ and a test set TS composed by s instances $\mathbf{x}_t$. The purpose of PG is to obtain a prototype generated set GS, which consists of r, $r < n$, prototypes $\mathbf{p}_u$ where $\mathbf{p}_u = (\mathbf{p}_{u1}, \mathbf{p}_{u2}, ..., \mathbf{p}_{um}, \mathbf{p}_{u\omega})$, which are generated from the examples of TR. The prototypes of the generated set are determined to represent efficiently the distributions of the classes and to discriminate well when used to classify the training objects. Their cardinality should be sufficiently small to reduce both the storage and evaluation time spent by an NN classifier.

The PG approaches can be divided into several families depending on the main heuristic operation followed. The first approach that we can find in the literature, called PNN [18] belongs to the family of methods that carry out a merging of prototypes of the same class in successive iterations, generating centroids. Other well-known methods are those based on a divide-and-conquer scheme, by separating the m-dimensional into two or more subspaces with the purpose of simplifying the problem at each step [19]. Recent advances that follow a similar operation include: MixtGauss [20], an adaptive PG algorithm considered in the framework of mixture modeling by Gaussian distributions, while assuming a statistical independence of features, and the RSP3 technique [21] which tries to avoid drastic changes in the form of decision boundaries associated with TR which is the main shortcoming observed in the classical approach [19].

One of the most important families of methods is based on adjusting the position of the prototypes which can be viewed as an optimization process. The main algorithm belonging to this family is LVQ [22]. LVQ can be understood as an artificial neural network in which a neuron corresponds to a prototype and a competition weight based is carried out in order to locate each neuron in a concrete place of the *m*-dimensional space to increase the classification accuracy. The third version of this algorithm, LVQ3 reported the best results. Several approaches have been proposed that modify the basic LVQ, for instance LVQPRU [23], which extends LVQ by using a pruning step to remove noisy instances, or the HYB algorithm [24] that constitutes a hybridization of several prototype reduction techniques. Specifically, HYB combines support vector machines with LVQ3 and executes a search in order to find the most promising parameters of LVQ3.

As a positioning adjustment of prototypes technique, a genetic algorithm called ENPC was proposed for PG in [25]. This algorithm executes different operators in order to find the most suitable position of the prototypes. PSO was proposed for PG in [26], [27] and they also belong to the positioning adjustment of prototypes category of methods. The main difference between them is the type of codification of the particles. The PSO approach proposed in [26] codifies a complete solution GS per particle. However, AMPSO [27] encodes each prototype of GS in a single particle. AMPSO has been shown to be more effective than PSO [26].

B. Differential Evolution

Differential evolution follows the general procedure of an evolutionary algorithm. DE starts with a population of NP candidate solutions, so-called individuals. The generations in DE are denoted by $G = 0, 1, \ldots, G_{max}$. It is usually to denote each individual as a D-dimensional vector $X_{i,G}$ = $\{x^1_{i,G}$,..., $x^D_{i,G}\}$, called a "target vector".

After initialization, DE applies the mutation operator to generate a mutant vector $V_{i,G}$, with respect to each individual $X_{i,G}$, in the current population. For each target $X_{i,G}$, at the generation G, its associated mutant vector $V_{i,G}$= $\{V^1_{i,G}$,..., $V^D_{i,G}\}$. The method of creating this mutant vector is that which differentiates one DE scheme from another. We focus on the *DE/Rand/1* which generate the mutant vector as follows:

$$V_{i,G} = X_{r_1,G} + F \cdot (X_{r_2,G} - X_{r_3,G}) \tag{1}$$

After the mutation phase, the crossover operation is applied to each pair of the target vector $X_{i,G}$ and its corresponding mutant vector $V_{i,G}$ to generate a new trial vector that we denote $U_{i,G}$. There are three kinds of crossover operators known as 'Binomial', 'Exponential' and 'Arithmetic' crossovers.

Specifically, we will focus on the well-known *DE/CurrentToRand/1* strategy [30], which generates the trial vector $U_{i,G}$ by linearly combining the target vector $X_{i,G}$ and the corresponding mutant vector $V_{i,G}$ likeas follows:

$$U_{i,G} = X_{i,G} + K \cdot (V_{i,G} - X_{i,G}) \tag{2}$$

Now incorporating equation (1) in (2) and simplifying, we obtain

$$U_{i,G} = X_{i,G} + K \cdot (X_{r_1,G} - X_{i,G}) + F \cdot (X_{r_2,G} - X_{r_3,G}) \quad (3)$$

The indices r_1^i, r_2^i, r_3^i are mutually exclusive integers randomly generated within the range $[1, NP]$, which are also different from the base index i. The scaling factor F is a positive control parameter for scaling the different vectors. K is a random number from [0,1].

When the trial vector has been generated, we must decide which individual between $X_{i,G}$ and $U_{i,G}$ should survive in the population of the next generation $G+1$. If the new trial vector yields an equal or better solution than the target vector, it replaces the corresponding target vector in the next generation; otherwise the target is retained in the population.

The success of the DE algorithm in solving a specific problem crucially depends on the appropriately choice of their associated control parameter values that determine the convergence speed. Hence, a fixed selection of these parameters can produce a slow and/or premature convergence depending on the problem. Thus, researchers have investigated the parameter adaptation mechanisms to improve the performance of the basic DE algorithm. One of the most successful adaptive DE algorithms is SFLSDE [31]. It uses two local search algorithms in the scale factor space to find the appropriate parameters for a given $X_{i,G}$.

III. Iterative Prototype Adjustment Based On Differential Evolution

In this section, we present and describe the IPADE approach in depth. IPADE follows an iterative scheme, in which it determines the most appropriate number of prototypes per class and their best positioning. Concretely, IPADE is divided into three different stages: initialization (Subsection III-A), optimization (Subsection III-B) and addition of prototypes (Subsection III-C). Figure 1 shows the pseudocode of the model proposed. In the following we describe the most significant instructions enumerated from 1 to 26.

```
1: $GS$ = Initialization($TR$)
2: DE_Optimization($GS$, $TR$)
3: $Accuracy_{Global}$ = Evaluate($GS$, $TR$)
4: registerClass[0..$\Omega$] = $optimizable$
5: while $Accuracy_{Global}$ <> 1.0 or all classes are $non-optimizables$ do
6:    lessAccuracy = $\infty$
7:    for $i = 1$ to $\Omega$ do
8:       if registerClass[$i$] == $optimizable$ then
9:          $AccuracyClass[i]$ = Evaluate ($GS$, Examples of class $i$ in $TR$)
10:         if $AccuracyClass[i]$ < lessAccuracy then
11:            lessAccuracy = $AccuracyClass[i]$
12:            $targetClass$ = $i$
13:         end if
14:      end if
15:   end for
16:   $GS_{test}$ = $GS$ $\cup$ RandomExampleForClass($TR$, $targetClass$)
17:   DE_Optimization($GS_{test}$, $TR$)
18:   $accuracy_{Test}$ = Evaluate($GS_{test}$, $TR$)
19:   if $accuracy_{Test} > Accuracy_{Global}$ then
20:      $Accuracy_{Global}$ = $accuracy_{Test}$
21:      $GS$ = $GS_{test}$
22:   else
23:      registerClass[$targetClass$] = $non-optimizable$
24:   end if
25: end while
26: return $GS$
```

Fig. 1: IPADE algorithm basic structure

A. Initialization

A random selection (stratified or not) of examples from TR may not be the most adequate procedure to initialize the GS. Instead, IPADE iteratively learns prototypes in order to find the most appropriate structure of GS. Instruction 1 generates the initial solution GS. In this step, GS must represent each class with one prototype and should cover the entire search space as much as possible. For this reason, each class distribution is represented with its respective centroid. This initialization was satisfactorily used by the approaches proposed in [16], [20]. The centroid of the class does not completely cover the region of each class and it does not avoid misclassifications. Thus, instruction 2 applies the first optimization stage using the initial GS composed of centroids for each class. The optimization stage must modify the prototypes of GS using the movement idea in the *m*-dimensional space, adding or subtracting some quantities to the attribute values of the prototypes. It is important to point out that we normalize all attributes of the data set to the $[0, 1]$ range.

B. Differential Evolution Optimization for IPADE

In this subsection we explain the proposal to apply the underlying idea of the DE algorithm to the PG problem as a position adjusting of prototypes scheme.

First of all, it is necessary to define the solution codification. In the proposed DE algorithm, each individual in the population encodes a single prototype without the class label and, as such, the dimension of the individuals is equal to the number of attributes of the specific problem. An individual classifies an example of TR when it is the closest particle (in terms of Euclidean distance) to that example.

The DE algorithm uses each prototype $\mathbf{p}_u$ of GS, provided by the IPADE algorithm, as an initial population. Next, mutation and crossover operators guide the optimization of the positioning of each $\mathbf{p}_u$ in the *m*-dimensional space. It is important to point out that these operators only produce modifications in the attributes of the prototypes of GS. Hence, the class value remains unchangeable throughout the evolutionary cycle. We will focus on the well-known *DE/CurrentToRand/1* strategy [30] to generate the trial prototypes $\mathbf{p}_u^{'}$ because it has reported the best behavior. It can be viewed as:

$$\mathbf{p}_u^{'} = \mathbf{p}_u + K \cdot (\mathbf{p}_{r_1} - \mathbf{p}_u) + F \cdot (\mathbf{p}_{r_2} - \mathbf{p}_{r_3}) \qquad (4)$$

The examples $\mathbf{p}_{r_1}$, $\mathbf{p}_{r_2}$, $\mathbf{p}_{r_3}$ are randomly extracted from TR and they belong to the same class as $\mathbf{p}_u$. In the hypothetical case that TR does not contain enough prototypes of the $\mathbf{p}_u$ class, that is, there is not at least three prototypes of this class in TR, we artificially generate the necessary number of new prototypes $\mathbf{p}_{r_j}$, $1 \leq j \leq 3$, with the same class label as $\mathbf{p}_u$, using little random perturbations such as $\mathbf{p}_{r_j} = (\mathbf{p}_{u1} + rand[-0.1, 0.1], \mathbf{p}_{u2} + rand[-0.1, 0.1], ..., \mathbf{p}_{um} + rand[-0.1, 0.1], \mathbf{p}_{u\omega})$.

After applying this operator, we check if there have been values out of range of $[0, 1]$. If a computed value is greater than 1, we truncate it to 1, and if is lower than 0, we establish it at 0.

After the mutation process over all the prototypes of GS, we obtain a trial solution $GS^{'}$, which is constituted for each $\mathbf{p}_u^{'}$. The selection operator decides which solution $GS^{'}$ or GS should survive for the next iteration. The 1-NN rule guides this operator to obtain the corresponding fitness value. We try to maximize this value, so the selection operator can be viewed as follows:

$$GS = \begin{cases} GS^{'} & \text{if } accuracy(GS^{'}) >= accuracy(GS) \\ GS & \text{Otherwise} \end{cases} \qquad (5)$$

In order to guarantee a high quality solution, we use the ideas established in [31] to obtain a self adaptive algorithm. Instruction 3 evaluates the accuracy of the initial solution, measured by classifying the examples of TR with the prototypes of GS by using the NN rule.

C. Addition of Prototypes

After the first optimization process, IPADE enters in an iterative loop (Instructions 5-25) to determine which classes need more prototypes to faithfully represent their class distribution. In order to do this, we need to define two types of classes. A class ω is said to be optimizable if it allows the addition of new prototypes to improve its local classification accuracy. The local accuracy of ω is computed by classifying the examples of TR whose class is ω with the prototypes kept in GS (using the NN rule). The *target class* will be the *optimizable* class with the least accuracy registered. From instructions 7 to 15, the algorithm identifies the *target class* in each iteration. Initialy, all classes start as *optimizable* (Instruction 4)

In order to reduce the classification error of the *target class*, IPADE extracts a random example of this class from TR and adds this to the current GS in a new trial set GS_{test} (Instruction 16). This addition forces the re-positioning of the prototypes of GS_{test} by again using the optimization process (Instruction 17) and its corresponding evaluation (Instruction 18) of predictive accuracy.

After this process, we have to ensure that the new positioning of prototypes of GS_{test}, generated with the optimizer, has reported a successful improvement of the accuracy rate in respect to the previous GS. If the global accuracy of the GS_{test} is lesser than the accuracy of GS, IPADE does not add this prototype to GS and this class is registered as *non-optimizable*. Otherwise, $GS = GS_{test}$.

The stopping criterion is satisfied when the accuracy rate is 1.0 or all the classes are registered as *non-optimizable*. The algorithm returns GS as the smallest reduced set which is able to classify the TR appropriately.

TABLE I: Parameter specification for all the methods employed in the experimentation

Algorithm	*Parameters*
IPADE	iterations of Basic DE = 300/500/1000, iterSFGSS = 8, iterSFHC =20, Fl=0.1, Fu=0.9
RSP3	Subset Choice = Diameter
Mixt_Gauss	Reduction Rate = 0.95
ENPC	Iterations = 300/500/1000
AMPSO	Iterations = 300/500/1000, C1 = 1.0, C2 = 1.0, C3 = 0.25, Vmax = 1, W = 0.1, X = 0.5, Pr = 0.1, Pd = 0.1
LVQPRU	Iterations = 300/500/1000, $alpha$ = 0.1 ,WindowWidth = 0.5
HYB	Search_Iter = 300/500/1000, Optimal_Iter = 1000 $alpha$ = 0.1 , I_$epsilon$ = 0, F_$epsilon$ = 0.5 Initial_Window = 0, Final_Window = 0.5 $delta$ = 0.1, delta_Window = 0.1 Initial Selection = SVM
LVQ3	Iterations = 300/500/1000, α = 0.1, WindowWidth=0.2, $epsilon$ = 0.1

IV. Experimental Framework and Analysis of Results

This section presents the experimental framework (Subsection IV-A) and the comparative study between our proposal and other PG techniques (Subsection IV-B).

A. Experimental Framework

In this section we show the issues related to the experimental study. In order to compare the performance of the algorithms, we use four measures: *Accuracy* [1], [32], the *Reduction rate* measured as

$$ReductionRate = 1 - size(GS)/size(TR) \qquad (6)$$

Acc·Red measured as *Accuracy·Reduction rate* and *Execution time*[1].

We use 50 data sets[2] from the KEEL-dataset repository[3][33], [34]. These data sets contain between 100 and 20,000 instances, and the number of attributes ranges from 2 to 60. The data sets considered are partitioned using the ten fold cross-validation (10-fcv) procedure.

Many different configurations are established by the authors of each paper for the different techniques. We focus this experimentation on the recommended parameters proposed by their respective authors, assuming that the choice of the values of the parameters was optimally

[1]Reduction rate and execution time information can be found in the web page

[2]Data sets: abalone, appendicitis, australian, balance, banana, bands, breast, bupa, car, chess, cleveland, coil2000, contraceptive, crx, dermatology, ecoli, flare-solar, german, glass, haberman, hayes-roth, heart, hepatitis, housevotes, iris, led7digit, lymphography, magic, mammographic, marketing, monks, newthyroid, page-blocks, pima, ring, saheart, satimage, segment, sonar, spectheart, splice, tae, thyroid, tic-tac-toe, titanic, twonorm, wine, wisconsin, yeast, zoo.

[3]http://sci2s.ugr.es/keel/datasets

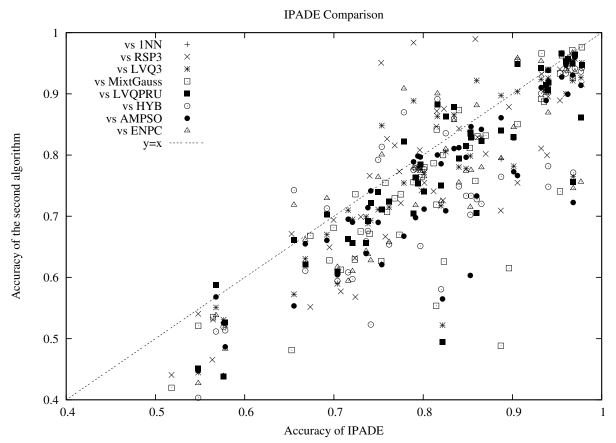

Fig. 2: Accuracy Results over 50 data sets

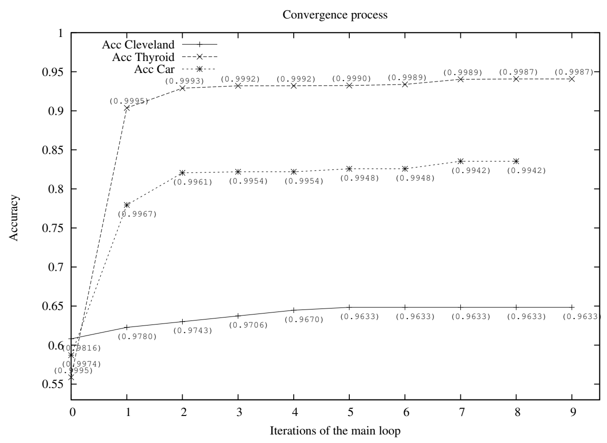

Fig. 4: Map of convergence over 3 different data sets

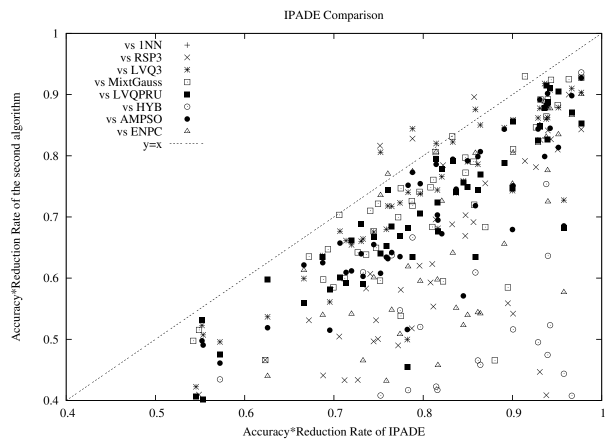

Fig. 3: *Acc·Red* Results over 50 data sets

chosen. However, we have done a previous study for each method that depends on the number of iterations performed, with 300, 500 and 1000 iterations in all the data sets. This parameter can be very sensitive to the problem tackled. An excesive number of iterations may produce overfitting for some problems, and a lower number of iterations may not be enough to tackle other data sets. For this reason, we present the results of the best performing number of iterations in each method and data set. The complete set of results can be found in the associated web site (http://sci2s.ugr.es/ipade/). The configuration parameters of IPADE and the methods used in the comparison are shown in Table I. In this table, the values of the parameters F_l, F_u, $iterSFGSS$ and $iterSFHC$ of the IPADE algorithm are the recommended values established in [31]. Furthermore, euclidean distance is used as a similarity function and those which are stochastic methods have been run three times per partition.

Implementations of the algorithms can be found in the web site associated or in the KEEL software tool [33].

B. Analysis of Results

In this section, we analyze the results obtained. Specifically, we check the performance of the IPADE model and another 7 PG techniques.

In the scatterplot of Figure 2, each point compares IPADE to some second algorithm on a single dataset. The x-axis position of the point is the accuracy of IPADE, and the y-axis position is the accuracy of the comparison algorithm. Therefore, points below the $y = x$ line correspond to datasets for which IPADE performs better than some second algorithm.

In order to test the reduction capabilities of PG methods in comparison with IPADE, Figure 3 shows at each point the *Acc·Red* obtained on a single data set.

Figure 4 shows a graphical representation of the convergence of the IPADE model over 3 different data sets. The graphic shows a line representing the accuracy rate in each step and its corresponding reduction rate (in brackets). The x-axis represents the number of iterations of the main loop of IPADE, and the y-axis represents the accuracy rate currently achieved.

Tables II and III present the statistical analysis conducted by nonparametric multiple comparison procedures for *Accuracy* and *Acc·Red* respectively. More specifically, we have used the Friedman Aligned (FA) procedure [35], [36] to compute the set of rankings that represent the effectiveness associated with each algorithm (second column). Both tables are ordered from the best to the worst ranking. In addition, the third column shows the adjusted p-value with the Holm's test (*HAPV*) [35]. Note that IPADE is established as the control algorithm because it has obtained the best FA ranking. By using a level of significance of α = 0.01, IPADE is significantly better than the rest of methods, considering both *Accuracy* and *Acc·Red* measures. More information about these tests and other statistical procedures can be found at http://sci2s.ugr.es/sicidm/.

For the sake of simplicity, we only include the graphical and statistical results achieved, whereas the complete results can be found at the web page associated with this paper.

Looking at Tables II and III and Figures 2, 3, 4, we want to outline some interesting comments:

- Figure 2 shows that the proposed IPADE outperforms on average the rest of the PG techniques with the parameter setting established. The most competitive

TABLE II: Average Rankings of the algorithms (FA + HAPV) for the Accuracy measure

Algorithm	Accuracy FA	Accuracy HAPV
IPADE	109.63	–
LVQPRU	199.66	$6.4064{\cdot}10^{-4}$
LVQ3	203.22	$6.4064{\cdot}10^{-4}$
RSP3	231.80	$7.9160{\cdot}10^{-6}$
1-NN	236.53	$4.2657{\cdot}10^{-6}$
AMPSO	248.11	$5.0703{\cdot}10^{-7}$
ENPC	259.13	$5.4223{\cdot}10^{-8}$
HYB	268.14	$5.6781{\cdot}10^{-9}$
Mixt_Gauss	272.02	$3.4239{\cdot}10^{-6}$

TABLE III: Average Rankings of the algorithms (FA + HAPV) for the *Acc·Red* measure

Algorithm	*Acc·Red* FA	*Acc·Red* HAPV
IPADE	53.83	–
LVQ3	125.92	0.0055
Mixt_Gauss	169.18	$2.2324{\cdot}10^{-5}$
LVQPRU	170.38	$2.2324{\cdot}10^{-5}$
AMPSO	182.54	$2.9965{\cdot}10^{-6}$
ENPC	267.91	$9.3280{\cdot}10^{-16}$
RSP3	275.32	$9.9684{\cdot}10^{-17}$
HYB	362.57	$1.5321{\cdot}10^{-31}$
1NN	421.83	$1.5321{\cdot}10^{-44}$

algorithms for IPADE, in terms of the accuracy measure, are the LVQ3 and LVQPRU algorithms. In this figure, most of the LVQ3 and LVQPRU points are close to the $y = x$ line. However, the statistical test confirms that IPADE significantly outperforms these methods.

- The trade-off between accuracy and reduction rate is an important factor because the efficiency of the NN classifier depends on the resulting number of prototypes of the GS. Figure 3 shows that achieving this balance between accuracy and reduction rate is a difficult task. IPADE is the best performing method considering the balance between accuracy and reduction rates. In Figure 3, there are more point under the $y = x$ line in comparison with Figure 2. Furthermore, Table III also supports this statement, showing smaller p-values when the reduction rate is considered.
- Observing the map of convergence of Figure 4, we can highlight the DE algorithm as a promising optimizer because it is able to reach highly accurate results very fast. This implies that the IPADE scheme needs a small number of iterations.

V. Conclusions

In this brief paper, we have presented a new data reduction technique called IPADE which iteratively learns the most adequate number of prototypes per class and their respective positioning for the nearest neighbor algorithm, acting as a prototype generation method. This technique uses a real parameter optimization procedure based on differential evolution in order to adjust the positioning of the prototypes at each step. The large experimental study performed allows us to show that IPADE is a suitable method for prototype generation in nearest neighbor classification. Furthermore, due to the fact that IPADE is an heuristic optimization approach, as future work, this technique could be used for building an ensemble of classifiers.

Acknowledgment

This work was supported by TIN2008-06681-C06-01.

References

[1] E. Alpaydin, *Introduction to Machine Learning, 2nd Edition*. The MIT Press, 2010.

[2] I. Kononenko and M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007.

[3] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[4] E. K. Garcia, S. Feldman, M. R. Gupta, and S. Srivastava, "Completely lazy learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 9, pp. 1274–1285, 2010.

[5] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[6] A. N. Papadopoulos and Y. Manolopoulos, *Nearest Neighbor Search: A Database Perspective*. Springer, 2004.

[7] X. Wu and V. Kumar, Eds., *The Top Ten Algorithms in Data Mining*. Chapman & Hall/CRC Data Mining and Knowledge Discovery, 2009.

[8] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34, 1997.

[9] F. Fernández and P. Isasi, "Local feature weighting in nearest prototype classification," *IEEE Transactions on Neural Networks*, vol. 19, no. 1, pp. 40–53, 2008.

[10] N. García-Pedrajas, "Constructing ensembles of classifiers by means of weighted instance selection," *IEEE Transactions on Neural Networks*, vol. 20, no. 2, pp. 258–277, 2009.

[11] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, 1997.

[12] R. Parades and E. Vidal, "Learning prototypes and distances: a prototype reduction technique based on nearest neighbor error minimization," *Pattern Recognition*, vol. 39, pp. 180–188, 2006.

[13] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, 2001.

[14] ——, *Instance Selection and Construction for Data Mining*. Kluwer Academic Publishers, 2001.

[15] H. Fayed and A. Atiya, "A novel template reduction approach for the k-nearest neighbor method," *IEEE Transactions on Neural Networks*, vol. 20, no. 5, pp. 890–896, 2009.

[16] H. A. Fayed, S. R. Hashem, and A. F. Atiya, "Self-generating prototypes for pattern classification," *Pattern Recognition*, vol. 40, no. 5, pp. 1498–1509, 2007.

[17] W. Lam, C.-K. Keung, and D. Liu, "Discovering useful concept prototypes for classification based on filtering and abstraction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 8, pp. 1075–1090, 2002.

[18] C.-L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Transactions on Computers*, vol. 23, no. 11, pp. 1179–1184, 1974.

[19] C. H. Chen and A. Jóźwik, "A sample set condensation algorithm for the class sensitive artificial neural network," *Pattern Recognition Letters*, vol. 17, no. 8, pp. 819–823, 1996.

[20] M. Lozano, J. M. Sotoca, J. S. Sánchez, F. Pla, E. Pekalska, and R. P. W. Duin, "Experimental study on prototype optimisation algorithms for prototype-based classification in vector spaces," *Pattern Recognition*, vol. 39, no. 10, pp. 1827–1838, 2006.

[21] J. S. Sánchez, "High training set size reduction by space partitioning and prototype abstraction," *Pattern Recognition*, vol. 37, no. 7, pp. 1561–1564, 2004.

[22] T. Kohonen, "The self organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.

[23] J. Li, M. T. Manry, C. Yu, and D. R. Wilson, "Prototype classifier design with pruning." *International Journal on Artificial Intelligence Tools*, vol. 14, no. 1-2, pp. 261–280, 2005.

[24] S.-W. Kim and B. J. Oommen, "Enhancing prototype reduction schemes with LVQ3-type algorithms," *Pattern Recognition*, vol. 36, no. 5, pp. 1083–1093, 2003.

[25] F. Fernández and P. Isasi, "Evolutionary design of nearest prototype classifiers," *Journal of Heuristics*, vol. 10, no. 4, pp. 431–454, 2004.

[26] L. Nanni and A. Lumini, "Particle swarm optimization for prototype reduction," *Neurocomputing*, vol. 72, no. 4-6, pp. 1092–1097, 2009.

[27] A. Cervantes, I. M. Galván, and P. Isasi, "AMPSO: A new particle swarm method for nearest neighborhood classification," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 39, no. 5, pp. 1082–1091, 2009.

[28] R. Storn and K. V. Price, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 10, pp. 341–359, 1997.

[29] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution A Practical Approach to Global Optimization*, ser. Natural Computing Series, G. Rozenberg, T. Bäck, A. E. Eiben, J. N. Kok, and H. P. Spaink, Eds., 2005.

[30] K. V. Price, *An introduction to differential evolution*. New Ideas Optimization, London, U.K.: McGraw-Hill, 1999.

[31] F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Computing*, vol. 1, no. 2, pp. 153–171, 2009.

[32] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, San Francisco, 2005.

[33] J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera, "KEEL: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009.

[34] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, 2010, in press.

[35] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, pp. 2044–2064, 2010.

[36] S. García, A. Fernández, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics–based machine learning: Accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, pp. 959–977, 2009.