Enhancing IPADE Algorithm with a Different Individual Codification

Isaac Triguero[1], Salvador García[2], and Francisco Herrera[1]

[1] Dept. of Computer Science and Artificial Intelligence, CITIC-UGR
University of Granada. 18071, Granada, Spain
triguero@decsai.ugr.es, herrera@decsai.ugr.es
[2] Dept. of Computer Science. University of Jaén. 23071, Jaén, Spain
sglopez@ujaen.es

Abstract. Nearest neighbor is one of the most used techniques for performing classification tasks. However, its simplest version has several drawbacks, such as low efficiency, storage requirements and sensitivity to noise. Prototype generation is an appropriate process to alleviate these drawbacks that allows the fitting of a data set for nearest neighbor classification. In this work, we present an extension of our previous proposal called IPADE, a methodology to learn iteratively the positioning of prototypes using a differential evolution algorithm. In this extension, which we have called IPADECS, a complete solution is codified in each individual. The results are contrasted with non-parametrical statistical tests and show that our proposal outperforms previously proposed methods.

1 Introduction

The nearest neighbor (NN) algorithm [1] and its derivatives have been shown to perform well for classification problems. These algorithms are also known as instance-based learning and belong to the lazy learning family [2]. 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. The effectiveness 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.

A successful technique which simultaneously tackles the computational complexity, storage requirements 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. Data reduction can be divided into two different approaches, known as prototype selection [3],[4] and Prototype Generation (PG) or abstraction [5]. The former process consists of choosing a subset of the original training data, while PG is not only able to select data, but can also build new artificial prototypes.

E. Corchado, M. Kurzyński, M. Woźniak (Eds.): HAIS 2011, Part II, LNAI 6679, pp. 262–270, 2011.

In the specialized literature, a great number of PG techniques have been proposed. Since the first approach PNN based on merging prototypes [6] and divide-and-conquer based schemes [7], many other proposals of PG can be found. For instance, ICPL [5] and RSP [8]. 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 [9], particle swarm optimization [10] and differential evolution (DE) [11].

In [11] we proposed an iterative prototype adjustment procedure based on DE (IPADE) to automatically find the smallest reduced set with the appropriate number of instances for each class, which achieves a suitable classification accuracy over different types of problems. This method follows an incremental approach. At each step, an optimization procedure is used to adjust the positioning of the prototypes, and the method adds new prototypes if needed. We adopted the DE technique as optimizer.

In this work, we use a different individual codification for the DE algorithm to increase the optimization capabilities. Concretely, this proposal follows an IPADE approach with a complete solution per individual (IPADECS). In experiments on 20 real-world benchmark data sets, the classification accuracy and reduction rate of our approach are investigated and its performance will be compared with IPADE, which showed to outperform previously proposed methods.

In order to organize this paper, Section 2 describes the background of PG and DE. Section 3 explains the IPADE algorithm. Section 4 shows the IPADECS approach. Section 5 discusses the experimental framework and presents the analysis of results. Finally, in Section 6 we summarize our conclusions.

2 Background

This section covers the background information necessary to define our proposal. Subsection 2.1 presents the background on PG. Next, Subsection 2.2 shows the main characteristics of DE.

2.1 Prototype Generation

PG can be defined as the application of instance construction algorithms over a data set to improve the classification accuracy of a NN classifier. 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 (PGS), 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. PGS must represent efficiently the distributions of the classes.

2.2 Differential Evolution

DE starts with a population of NP solutions, so-called individuals. The generations 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 $V_{i,G}$= $\{V^1_{i,G}, ..., V^D_{i,G}\}$. In this contribution, we focus on the *RandToBest/1* which generates the mutant vector as follows:

$$V_{i,G} = X_{i,G} + F \cdot (X_{best,G} - X_{i,G}) + F \cdot (X_{r^i_1,G} - X_{r^i_2,G}) \quad (1)$$

The indices r^i_1, r^i_2 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.

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 $U_{i,G}$. We will focus on the binomial crossover scheme, which is performed on each component whenever a randomly picked number between 0 and 1 is less than or equal to the crossover rate (CR), which controls the fraction of parameter values copied from the mutant vector. Then, we must decide which individual should survive in 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 DE in solving a problem crucially depends on the associated control parameter values (F and CR) that determine the convergence speed. One of the most successful adaptive DE algorithms is SFLSDE [12], we use the ideas established in this work to obtain a self adaptive algorithm.

3 Iterative Prototype Adjustment Based on Differential Evolution

In this section, we describe the IPADE approach. 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 3.1), optimization (Subsection 3.2) and addition of prototypes (Subsection 3.3). Figure 1 shows the pseudocode. In the following we describe the most significant instructions enumerated from 1 to 26.

3.1 Initialization

A random selection (stratified or not) of examples from TR may not be the most adequate procedure to initialize the PGS. Instead, IPADE iteratively learns prototypes in order to find the most appropriate structure of PGS. Instruction 1

```
1: PGS = Initialization(TR)
2: DE_Optimization(PGS, TR)
3: Accuracy_Global = Evaluate(PGS, TR)
4: registerClass[0..Ω] = optimizable
5: while Accuracy_Global <> 1.0 or all classes are non − optimizables do
6:    lessAccuracy = ∞
7:    for i = 1 to Ω do
8:       if registerClass[i] == optimizable then
9:          AccuracyClass[i] = Evaluate (PGS, 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:   PGS_test = PGS ∪ RandomExampleForClass(TR, targetClass)
17:   DE_Optimization(PGS_test, TR)
18:   accuracy_Test = Evaluate(PGS_test, TR)
19:   if accuracy_Test > Accuracy_Global then
20:      Accuracy_Global = accuracy_Test
21:      PGS = PGS_test
22:   else
23:      registerClass[targetClass] = non − optimizable
24:   end if
25: end while
26: return PGS
```

Fig. 1. IPADE algorithm basic structure

generates the initial solution PGS. In this step, PGS represents each class distribution with its respective centroid. 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 PGS composed of centroids for each class. The optimization stage must modify the prototypes of PGS using the movement idea in the m-dimensional space. It is important to point out that we normalize all attributes of the data set to the $[0, 1]$ range.

3.2 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. In the proposed IPADE 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 to that example.

The DE algorithm uses each prototype $\mathbf{p}_u$ of PGS, provided by the IPADE algorithm, as an initial population. Next, mutation and crossover operators guide the optimization of the positioning of each $\mathbf{p}_u$. These operators only produce modifications in the attributes of the prototypes of PGS. Hence, the class value remains unchangeable throughout the evolutionary cycle. IPADE focuses on the *DE/CurrentToRand/1* strategy to generate the trial prototypes $\mathbf{p}_u^{'}$.

After this, we obtain a trial solution $PGS^{'}$, which is constituted for each $\mathbf{p}_u^{'}$. The selection operator decides which solution $PGS^{'}$ or PGS should survive for the next iteration. The NN rule guides this operator to obtain the corresponding fitness value. Implementation of these operators was described in depth in [11].

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.

3.3 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 PGS (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. All classes start as *optimizable*.

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 PGS in a new trial set PGS_{test} (Instruction 16). This addition forces the re-positioning of the prototypes of PGS_{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 PGS_{test} has reported a successful improvement of the accuracy rate in respect to the previous PGS. If the global accuracy of the PGS_{test} is lesser than the accuracy of PGS, IPADE does not add this prototype to PGS and this class is registered as *non-optimizable*. Otherwise, $PGS = PGS_{test}$.

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

4 IPADECS: IPADE with a Complete Solution Per Individual

In this work we choose a different codification scheme for the DE optimization. Concretely, a complete solution is codified in each individual. This corresponds to the case where each individual of the population encodes a complete PGS. Following the notation used in Subsection 2.2, $X_{i,G}$ defines the target vector, but in this case, the target vector can be represented as a matrix. Table 1 describes the structure of an individual. Furthermore, each prototype p_j, $1 \leq j \leq \mathbf{r}$, of an individual $X_{i,G}$ has a class $x_{p\omega,j}$.

The current PGS of the IPADE algorithm must be inserted once as one of the individuals of the population, initializing the rest of the individuals with random prototypes extracted from TR. It is important to point out that every solution must have the same structure that PGS, thus they must have the same number of prototypes per class, and the classes must have the same arrangement in the matrix $X_{i,G}$.

Table 1. Encoding of a set of prototypes in a individual $X_{i,G}$

	Attribute 1	**Attribute 2**	...	**Attribute** D	**Class**
Prototype 1	$\mathbf{x}_{p1,1}$	$\mathbf{x}_{p2,1}$	...	$\mathbf{x}_{pD,1}$	$\mathbf{x}_{p\omega,1}$
Prototype 2	$\mathbf{x}_{p1,2}$	$\mathbf{x}_{p2,2}$	...	$\mathbf{x}_{pD,2}$	$\mathbf{x}_{p\omega,2}$
...					
Prototype r	$\mathbf{x}_{p1,r}$	$\mathbf{x}_{p2,r}$	...	$\mathbf{x}_{pD,r}$	$\mathbf{x}_{p\omega,r}$

The *RandToBest* mutation strategy generates the mutant matrix $V_{i,G}$ in respect to each individual $X_{i,G}$, in the current population. The operations of addition, subtraction and scalar product are carried out as typical matrices. This is the justification for the individuals having the same structure. In order for the mutation operator to make sense, the operators must act over the same attributes and over prototypes of the same class in all cases. After applying this operator, we check that the mutant matrix $V_{i,G}$ has generated correct values for all features of the prototypes. This procedure checks 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.

The trial matrix $U_{i,G}$ is generated with a binomial crossover. In PG, the mutant matrix $V_{i,G}$ exchanges its prototypes with $X_{i,G}$ to generate $U_{i,G}$.

The selection operator decides which individual between $X_{i,G}$ and $U_{i,G}$ should survive in the next generation $G + 1$. The NN rule, with k=1 (1NN), guides this operator. The instances in TR are classified with the prototypes encoded in $X_{i,G}$ or $U_{i,G}$ by the 1NN rule with a *leave-one-out* validation scheme, and their corresponding fitness values are measured as the number of successful hits relative to the total number of classifications. We try to maximize this value.

5 Experimental Framework and Analysis of Results

This section presents the experimental framework (Subsection 5.1) and the comparative study between IPADECS and IPADE (Subsection 5.2).

5.1 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 the *accuracy* rate [14], and the *reduction rate* measured as: $1 - size(PGS)/size(TR)$.

We use 20 data sets from the KEEL-dataset repository[1] [13]. Table 2 shows, for each data set, the number of examples (#Ex.), the number of attributes (#Atts.), and the number of classes (#Cl.). The data sets considered are partitioned using the ten fold cross-validation (10-fcv) procedure.

The configuration parameters of all the methods used in the comparison are shown in Table 3. In this table, the values of the parameters F_l, F_u, $iterSFGSS$

[1] `http://sci2s.ugr.es/keel/datasets`

Table 2. Summary description for used classification data sets

Data Set	#Ex.	#Atts.	#Cl.	Data Set	#Ex.	#Atts.	#Cl.
bupa	345	6	2	nursery	12690	8	5
car	1.728	6	4	pageblocks	5.472	10	5
ecoli	336	7	8	pima	768	8	2
german	1.000	20	2	ring	7400	20	2
glass	214	9	7	segment	2.310	19	7
haberman	306	3	2	thyroid	7.200	21	3
heart	270	13	2	twonorm	7.400	20	2
iris	150	4	3	wine	178	13	3
monks	432	6	2	wisconsin	683	9	2
newthyroid	215	5	3	zoo	101	17	7

Table 3. Parameter specification for the methods employed in the experimentation

Algorithm	*Parameters*
IPADE	iterations of Basic DE = 500, iterSFGSS = 8, iterSFHC =20, Fl=0.1, Fu=0.9
IPADECS	PopulationSize = 10, iterations of Basic DE = 500 iterSFGSS = 8, iterSFHC = 20, Fl=0.1, Fu=0.9

and $iterSFHC$ are the recommended values established in [12]. 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 KEEL software tool [13].

5.2 Analysis of Results

Table 4 shows the results obtained in test data. This table collects the average results in terms of accuracy and reduction rate, obtained over the 20 data sets considered. The best result for each data set is remarked in bold.

Considering only average results could lead us to erroneous conclusions. Due to this fact, we will accomplish statistical comparisons over multiple data sets based on non-parametric tests [15], [16]. Specifically, we use the Wilcoxon Signed-Ranks test and we obtain a p-value of **0.0019** in the comparison between IPADECS and IPADE.

Looking at Table 4 we want to outline some interesting comments:

- IPADECS obtains the best average results in terms of accuracy in comparison with the IPADE algorithm. Furthermore, the Wilcoxon test supports this statement with a p-value lower than 0.05.
- As we can observe in Table 4, the original IPADE achieves a higher reduction rate than IPADECS. This behavior arises because IPADECS uses a stronger optimization process that allows to introduce more prototypes in PGS, specifically in the addition stage to increase the accuracy capabilities.

Table 4. Results obtained

Datasets	IPADE		IPADECS	
	AccTest	Red. Rate	AccTest	Red. Rate
bupa	0.5775±0.0545	0.9894±0.0021	**0.6567±0.0848**	0.9842±0.0055
car	0.8096±0.0298	0.9931±0.0012	**0.8692±0.0205**	0.9914±0.0021
ecoli	0.7714±0.0615	0.9683±0.0026	**0.7892±0.0654**	0.9524±0.0042
german	0.7160±0.0233	0.9966±0.0008	**0.7180±0.0325**	0.9920±0.0032
glass	0.6306±0.1092	0.9533±0.0053	**0.6909±0.1113**	0.9393±0.0131
haberman	0.7287±0.0715	0.9898±0.0027	**0.7445±0.0640**	0.9891±0.0049
heart	0.8185±0.0560	0.9877±0.0052	**0.8370±0.0983**	0.9831±0.0059
iris	**0.9667±0.0447**	0.9763±0.0030	0.9467±0.0400	0.9748±0.0036
monks	0.8576±0.0974	0.9920±0.0024	**0.9120±0.0476**	0.9910±0.0021
newthyroid	0.9630±0.0348	0.9783±0.0045	**0.9818±0.0302**	0.9835±0.0021
nursery	0.5299±0.0282	0.9993±0.0002	**0.6479±0.0458**	0.9992±0.0003
page-blocks	0.9291±0.0084	0.9983±0.0002	**0.9335±0.0155**	0.9974±0.0005
pima	0.7528±0.0586	0.9955±0.0010	**0.7684±0.0467**	0.9916±0.0031
ring	0.8165±0.0342	0.9989±0.0004	**0.8970±0.0103**	0.9956±0.0012
segment	0.8753±0.0142	0.9950±0.0009	**0.9208±0.0097**	0.9919±0.0017
thyroid	**0.9404±0.0036**	0.9990±0.0002	0.9399±0.0036	0.9989±0.0003
twonorm	0.9764±0.0070	0.9995±0.0002	**0.9766±0.0069**	0.9993±0.0002
wine	**0.9497±0.0678**	0.9800±0.0025	0.9441±0.0352	0.9794±0.0040
wisconsin	**0.9699±0.0197**	0.9960±0.0011	0.9642±0.0194	0.9951±0.0019
zoo	0.9464±0.0702	0.9087±0.0071	**0.9633±0.0823**	0.9086±0.0054
AVERAGE	0.8263±0.1370	0.9848±0.0216	**0.8551±0.1138**	0.9819±0.0231

6 Concluding Remarks

In this contribution, we have presented a data reduction technique called IPADECS which iteratively learns the most adequate number of prototypes per class and their positioning for the NN algorithm. This technique is an extension our previous proposal IPADE with a different codification scheme. The results obtained show that IPADECS is statistically better than IPADE, which showed to outperform previously proposed methods.

Acknowledgments. This work was supported by Project TIN2008-06681-C06-01. I. Triguero holds an scholarship from the University of Granada.

References

1. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13, 21–27 (1967)
2. Garcia, E.K., Feldman, S., Gupta, M.R., Srivastava, S.: Completely Lazy Learning. IEEE Transactions on Knowledge and Data Engineering 22(9), 1274–1285 (2010)
3. Liu, H., Motoda, H. (eds.): Instance Selection and Construction for Data Mining. The International Series in Engineering and Computer Science. Springer, Heidelberg (2001)
4. García, S., Cano, J.R., Herrera, F.: A memetic algorithm for evolutionary prototype selection: A scaling up approach. Pattern Recognition 41(8), 2693–2709 (2008)
5. Lam, W., Keung, C., Liu, D.: Discovering Useful Concept Prototypes for Classification Based on Filtering and Abstraction. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(8), 1075–1090 (2002)

6. Chang, C.: Finding Prototypes For Nearest Neighbor Classifiers. IEEE Transactions on Computers 23(11), 1179–1184 (1974)
7. Chen, C.H., Jóźwik, A.: A sample set condensation algorithm for the class sensitive artificial neural network. Pattern Recognition Letters 17(8), 819–823 (1996)
8. Sánchez, J.S.: High training set size reduction by space partitioning and prototype abstraction. Pattern Recognition 37(7), 1561–1564 (2004)
9. Kohonen, T.: The self organizing map. Proceedings of the IEEE 78(9), 1464–1480 (1990)
10. Cervantes, A., Galván, G., Isasi, I.M.: AMPSO: A New Particle Swarm Method for Nearest Neighborhood Classification. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics 39(5), 1082–1091 (2009)
11. Triguero, I., García, S., Herrera, F.: IPADE: Iterative Prototype Adjustment for Nearest Neighbor Classification. IEEE Transactions on Neural Networks 21(12), 1984–1990 (2010)
12. Ferrante, N., Tirronen, V.: Scale factor local search in differential evolution. Memetic Computing 1(2), 153–171 (2009)
13. Alcalá-Fdez, J., Fernandez, A., Luengo, J., Derrac, J., García, S., Sánchez, L., Herrera, F.: 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)
14. Alpaydin, E.: Introduction to Machine Learning, 2nd edn. The MIT Press, Cambridge (2010)
15. García, S., Herrera, F.: An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. Journal of Machine Learning Research 9, 2677–2694 (2008)
16. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental Analysis of Power. Information Sciences 180, 2044–2064 (2010)