On the Efficient Design of a Prototype-Based Classifier Using Differential Evolution

Luiz A. Soares Filho and Guilherme A. Barreto
Department of Teleinformatics Engineering, Federal University of Ceará
Av. Mister Hull, S/N, Campus of Pici, Center of Technology
CEP 60455-970, Building 725, Fortaleza, Ceará, Brazil
Email: luizflh@gmail.com, gbarreto@ufc.br

***Abstract*—In this paper we introduce an evolutionary approach for the efficient design of prototype-based classifiers using differential evolution (DE). For this purpose we amalgamate ideas from the Learning Vector Quantization (LVQ) framework for supervised classification by Kohonen [1], [2], with the DE-based automatic clustering approach by Das et al. [3] in order to evolve supervised classifiers. The proposed approach is able to determine both the optimal number of prototypes per class and the corresponding positions of these prototypes in the data space. By means of comprehensive computer simulations on benchmarking datasets, we show that the resulting classifier, named LVQ-DE, consistently outperforms state-of-the-art prototype-based classifiers.**

I. Introduction

Prototype-based classification (PBC) encompasses a large family of supervised pattern classification methods and algorithms. Like the K-nearest neighbor (KNN) method [4], PBC is a local classification method in the sense that classification boundaries are approximated locally. Instead of making use of all the training data points, however, PBC relies on a set of appropriately chosen prototype vectors (a.k.a. reference, centroid or codebook vectors). Thus, PBC requires a much smaller number of items which must be stored and to which a new data point must be compared for classification than KNN.

Furthermore, PBC possesses two desirable properties which are hard to find in standard multilayer perceptron (MLP) and support vector machine (SVM) classifiers. Firstly, due to the inherent local way of building classification boundaries, interpretation of the decisions in terms of local explanatory rules associated to each prototype is facilitated. Secondly, prototype-based classifiers are easily endowed with adaptive strategies for adding and deleting prototypes to fit the current data distribution, a valuable property specially in evolving, nonstationary environments.

Prototype-based classifiers have been designed in the literature basically in three different ways. One approach comes from Bayesian decision theory. In this case, the design of the classifier involves two steps: 1) the construction of models for the probability densities of the different classes and 2) the construction of the classification boundaries using the criterion of maximum a posteriori (MAP) probability. If, for example, the designer assumes that (i) class-specific probability densities are well approximated by Gaussian densities, (ii) the classes are equiprobable, and (iii) the input features are uncorrelated and have equal variances, the MAP classifier reduces to a popular class of PBC algorithms known as *minimum distance* (MD) classifiers [4].

The second approach comes from the idea of directly estimating the discriminant functions for multiclass classification problems. In PBC, this is done using a set of prototype vectors for each class, and classification is based on the distance between a data point and the class to which its closest prototype belongs to, much the same way as done by MD classifiers. Often an Euclidean distance measure is used, but in principle any distance measure can be used. Two of the most common methods for the construction of prototype-based discriminant functions are, respectively, the Learning Vector Quantization (LVQ) [1], [2] and the ARTMAP [5] networks.

Finally, a third approach for building PBC can be used to build prototype-based (supervised) classifiers from prototype-based clustering (i.e. unsupervised) methods, such as the Self-Organizing Map (SOM) [6]. Firstly, the training data samples are submitted to a SOM network with a certain number of prototypes. Then, once training is finished, SOM prototypes are labeled according to a majority voting scheme, i.e. a given prototype is assigned the label of the most frequent class among the training samples which are closest to it. As for MD and LVQ classifiers, classification is based on the distance between a data point and the class to which its closest prototype belongs to.

Usually, the possibility of using a set of prototype vectors for each class gives more flexibility to the second and third PBC methods just described. However, the number of prototypes per class has to be defined *a priori* for the second method or can only be determined *a posteriori* (i.e. after the labelling phase) for the third method. Often, these numbers are far from optima, requiring from the user a great deal of experimentation with the data.

Bearing this in mind, in this paper we introduce an evolutionary approach for the efficient design of prototype-based classifiers using differential evolution (DE) [7]. The idea is to amalgamate ideas from the LVQ approach to supervised classification with the DE-based automatic clustering approach by Das et al. [3] in order to design prototype-based classifiers. The proposed approach is able to determine both the optimal number of prototypes per class and the corresponding positions of these prototypes in the data space. We show by means of comprehensive computer simulations on benchmarking datasets that the resulting classifier consistently outperforms state-of-the-art prototype-based classifiers.

978-1-4799-4461-3/14/$31.00 ©2014 IEEE

The remainder of the paper is organized as follows. In Section II we briefly describe the prototype-based classifiers to be evaluated in this paper. The proposed LVQ-DE method is described in detail in Section III. In Section IV the simulation scenarios are described and the obtained results are discussed. The paper is concluded in Section V.

II. Prototype-Based Classifiers

Let us consider a set of training input-output patterns $\{(\mathbf{x}_l, y_l)\}_{l=1}^{N}$, where $\mathbf{x}_l \in \mathbb{R}^p$ denotes the l-th input pattern and $y_l \in \mathcal{C}$ denotes its corresponding class label. Note that y_l is a discrete variable (of either numerical or nominal nature) which may assume only one out of K values in the finite set $\mathcal{C} = \{\omega_1, \omega_2, \ldots, \omega_K\}$.

Given a set of labeled prototype vectors $\mathbf{m}_i \in \mathbb{R}^p$, $i = 1, \ldots, M$, for all the prototype-based classifiers to be described in this section, class assignment for a new input pattern $\mathbf{x}(t)$ is based on the following decision criterion:

$$\text{Class of } \mathbf{x}(t) = \text{Class of } \mathbf{m}_c(t), \tag{1}$$

where

$$c = \arg \min_{i=1,\ldots,M} \{\|\mathbf{x}(t) - \mathbf{m}_i\|\}, \tag{2}$$

in which $\|\cdot\|$ denotes the euclidean distance measure and c is the index of the nearest prototype among the M ones available. In the following paragraphs we briefly described the learning rules for finding the positions of the prototypes $\mathbf{m}_i$, $i = 1, \ldots, M$ in the data space.

Minimum Distance-to-Centroid (MDC) classifier [4]: For this classifier, we have $M = K$, i.e. the number of prototypes (M) is equal to the number of classes (K). In this case, the prototype of the i-th class is computed as the centroid of class i as

$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in \omega_i} \mathbf{x}, \quad i = 1, \ldots, K, \tag{3}$$

where n_i is the number of training examples of class i.

A. LVQ classifiers

The LVQ is very popular family of competitive learning algorithms for supervised pattern classification. As a consequence, several applications and variants of basic LVQ algorithms are available elsewhere [8]–[14]. In this paper, we briefly describe five LVQ network variants. The performances of these variants will be compared with that of the DE-based PBC design method to be proposed in Section III.

The Optimized-Learning-Rate (OLVQ) [1]: For the whole family of LVQ classifiers we have $M > K$, i.e. the number of prototypes (M) is higher than the number of classes (K). As a consequence, different prototypes may share the same label. Let c be defined as in Eq. (2) for a new input pattern $\mathbf{x}(t)$. Then, the prototype $\mathbf{m}_c$ is updated as follows

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) + s(t)\alpha_c(t)[\mathbf{x}(t) - \mathbf{m}_c(t)], \tag{4}$$

where $s(t) = +1$ if the classification is correct, and $s(t) = -1$ if the classification is wrong. The optimal learning rate scheduling is given by

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 + s(t)\alpha_c(t-1)}. \tag{5}$$

It should be noted that since $\alpha_c(t)$ can also increase, it is specially important that it shall not rise above 1. This constraint can be forced in the implementation of the algorithm itself. For the initial values it is recommended to start with $\alpha_i = 0.3$.

LVQ-2.1 [1]: In this algorithm, two prototypes $\mathbf{m}_i$ and $\mathbf{m}_j$ that are the nearest neighbors to $\mathbf{x}(t)$ are now updated simultaneously. One of them ($\mathbf{m}_j$, for example) must belong to the correct class and the other to a wrong class, respectively. Thus, the learning rules of the LVQ2.1 algorithm are given by

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \tag{6}$$

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_j(t)], \tag{7}$$

where $\mathbf{x}(t)$ must satisfy the following condition:

$$\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s, \quad \text{where } s = \frac{1-w}{1+w}, \tag{8}$$

where d_i and d_j are the Euclidean distances of $\mathbf{x}(t)$ from $\mathbf{m}_i$ and $\mathbf{m}_j$, respectively. A relatively 'window' width from 0.2 to 0.3 is recommended.

LVQ-3 [1]: For scenarios in which $\mathbf{x}(t)$, the nearest prototypes $\mathbf{m}_i$ and $\mathbf{m}_j$ belong to the same class, the following updating rule is applicable:

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) + \epsilon\alpha(t)[\mathbf{x}(t) - \mathbf{m}_k(t)], \tag{9}$$

for $k \in \{i, j\}$, with $\mathbf{x}$ falling into the 'window'. In a series of experiments carried out in [1], feasible values of ϵ ranging from 0.1 to 0.5 were found, relating to w = 0.2 or 0.3. The optimal values for ϵ seems to depend on the size of the window, being smaller for narrower windows. An important feature of the LVQ3 algorithm is that it is self-stabilizing, in the sense that the optimal placements of the prototypes do not change in continued learning.

Soft LVQ [15], [16]: The Soft LVQ algorithm is based on a statistical modeling of the given data distribution, i.e. the probability density is described by a mixture model. It is assumed that every component j of the mixture generates data which belongs to only one of the K classes. Following [17], the probability density of the data is approximated by

$$p(\mathbf{x}|W) = \sum_{i=1}^{K} \sum_{j:class(\mathbf{m}_j)=\omega_i} P(j)p(\mathbf{x}|j), \tag{10}$$

where $\sum_j P(j) = 1$ and the conditional density $p(\mathbf{x}|j)$ is a function of the prototype $\mathbf{m}_j$. A possible choice is the normalized exponential form $p(\mathbf{x}|j) = K(j) \cdot \exp\{f(\mathbf{x}, \mathbf{m}_j, \sigma_j^2)\}$. A Gaussian mixture is assumed in [15] with $K(j) = (2\pi\sigma_j^2)^{(} - p/2)$ and $f(\mathbf{x}, \mathbf{m}_j, \sigma_j^2) = -d(\mathbf{x}, \mathbf{m}_j)$, where $d(\cdot, \cdot)$ is the squared Euclidean distance, and every component is assumed to have equal variance $\sigma_j^2 = \sigma^2$ and equal prior probability $P(j) = 1/M$, $\forall j$.

Thus, Soft LVQ maximizes the following likelihood ratio

$$L = \prod_{l=1}^{N} \frac{p(\mathbf{x}_l, \omega_l|W)}{p(\mathbf{x}_l|W)}, \tag{11}$$

with respect to the prototype locations by means of gradient ascent. $p(\mathbf{x}_l, \omega_l|W)$ is the probability density that sample $\mathbf{x}_l$

is generated jointly with a component of the correct class ω_l. This local density corresponds to the inner sum in Eq. (10). The Soft LVQ learning rule is obtained by taking the derivatives of the cost function $E = \log(L)$ with respect to $\mathbf{m}_j$ as follows.

1) If class of $\mathbf{x}(t) = \omega_j$, then

$$\Delta\mathbf{m}_j(t) = \frac{\alpha(t)}{\sigma^2}(P_y(j|\mathbf{x}(t)) - P(j|\mathbf{x}(t)))(\mathbf{x}(t) - \mathbf{m}_j(t)). \quad (12)$$

2) If class of $\mathbf{x}(t) \neq \omega_j$, then

$$\Delta\mathbf{m}_j(t) = -\frac{\alpha(t)}{\sigma^2}P(j|\mathbf{x}(t))(\mathbf{x}(t) - \mathbf{m}_j(t)). \quad (13)$$

where $\alpha_1 > 0$ is the learning rate, and $P_y(j|\mathbf{x}(t))$ and $P(j|\mathbf{x}(t))$ are assignment probabilities given, respectively, by

$$P_y(j|\mathbf{x}(t)) = \frac{\exp f(\mathbf{x}(t), \mathbf{m}_j, \sigma^2)}{\sum_{i:class(\mathbf{m}_i)=y} \exp f(\mathbf{x}(t), \mathbf{m}_i, \sigma^2)} \quad (14)$$

and

$$P(j|\mathbf{x}(t)) = \frac{\exp f(\mathbf{x}(t), \mathbf{m}_j, \sigma^2)}{\sum_i \exp f(\mathbf{x}(t), \mathbf{m}_i, \sigma^2)}, \quad (15)$$

with respect to one example $(\mathbf{x}, y)$. As expected, the update rules reflect the fact that prototypes with $class(\mathbf{m}_j) = y$ must be attracted by the training sample, while prototypes carrying any other class label must be repelled.

It is worth noting that the performance of the Soft LVQ algorithm highly depends on the hyperparameter σ^2, since it determines the value of the assignment probabilities as can be seen in Eqs. (14) and (15). Also, it directly controls the strength of the attractive and repulsive forces in the learning equations (12) and (13), respectively.

In the limit $\sigma^2 \to 0$, the soft LVQ algorithm reduces to a learning-from-mistakes scheme, i.e. only in case of erroneous classification, the closest correct and incorrect prototype are updated, behaving much similar to the original LVQ2 algorithm. In the Soft LVQ, however, a larger number of prototypes is adapted at each learning step. In fact, all training samples lying in an active region around the decision boundary cause an update of the prototype constellation.

LVQ with Training Count (LVQTC) [18]: LVQTC represents a modification of the original LVQ scheme by attributing training counters to each neuron, which record its training statistics. The additional neuron attributes are exploited during training and classification. During training, they help to replace (i.e pruning) neurons with poor training performance and to create new neurons when they re needed. During classification, they provide an estimate of the reliability of the classification given by each neuron.

An important consequence of keeping track of the training count for each neuron is that he number of neurons assigned to each class is no longer required to be proportional to the global probability for the class, as in standard LVQ. In the latter the density of neurons of each class in data space represents the only ingredient to (statistically) control classification in overlapping regions. In LVQTC, this role is largely taken over by the training counters. That can be exploited by assigning relatively few neurons to classes which are concentrated in small regions of the data space, and more neurons to classes which are spread out over large regions. In this way, with a given total number of neurons, one can better represent the shapes of the class distributions. Due to the lack of space, we omit the technicalities of the LVQTC algorithm implementation. The interested reader are referred to the original paper [18].

III. The Proposed Approach

In this section we firstly describe the DE algorithm as stated in Das & Suganthan [19]. Then, we introduce our scheme for designing prototype-based classifiers based on the automatic clustering procedure proposed by Das *et al.* [3].

A. Basics of Differential Evolution

Differential Evolution (DE) [7] is a population-based global optimization algorithm that uses a floating-point (real-coded) representation. The i-th individual vector (chromosome) of the population at generation t with d components, defining its dimension, can be simply represented as a vector of real-numbers as

$$\mathbf{z}_i(t) = [z_{i,1}(t) \quad z_{i,2}(t) \quad \cdots \quad z_{i,d}(t)]. \quad (16)$$

For each individual vector $\mathbf{z}_k(t)$ that belongs to the current population, DE randomly samples three other individuals, i.e., $\mathbf{z}_i(t)$, $\mathbf{z}_l(t)$, and $\mathbf{z}_m(t)$, from the same generation, with distinct k, i, l, and m. It then calculates the (component-wise) difference of $\mathbf{z}_i(t)$ and $\mathbf{z}_l(t)$, scales it by a scalar F (usually $F \in [0, 1]$), and creates a trial offspring $\mathbf{u}_i(t)$ by adding the result to $\mathbf{z}_m(t)$. In vector notation, we have

$$\mathbf{u}_i(t) = \mathbf{z}_m(t) + F(\mathbf{z}_i(t) - \mathbf{z}_l(t)). \quad (17)$$

A crossover operation is carried out component-wise with $\mathbf{z}_k(t)$ (parent vector) and $\mathbf{u}_i(t)$ (trial vector) in order to produce offspring $\mathbf{z}'_k(t)$. It is implemented component-wise as follows:

$$z'_{kj}(t) = \begin{cases} u_{ij}(t), & \text{If } j \in \mathcal{J} \\ z_{kj}(t), & \text{otherwise} \end{cases} \quad (18)$$

where $z_{kj}(t)$ and $u_{ij}(t)$ refer to the j-th element of the vectors $\mathbf{z}_k(t)$ and $\mathbf{u}_i(t)$, respectively. $\mathcal{J}$ is the set of element indices that will undergo perturbation (or in other words, the set of crossover points). Different methods can be used to determine the set $\mathcal{J}$. The algorithm of the *binomial crossover*, one of the most frequently used in practice, is presented below.

Algorithm 1 Selecting crossover points for set $\mathcal{J}$

```
1: procedure BINOMIAL CROSSOVER(d, p_r)
2:     j* ~ U(1, d)
3:     J ← J ∪ {j*}
4:     for each j ∈ {1, 2, ..., d} do
5:         if U(0, 1) < p_r and j ≠ j* then
6:             J ← J ∪ {j}
7:         end if
8:     end for
9: end procedure
```

In binomial crossover, the crossover points are randomly selected from the set of possible crossover points, $\{1, 2, \ldots, d\}$, where d is the chromosome dimension. In this

algorithm, p_r is the crossover rate, i.e. the probability that the considered crossover point will be included. The larger p_r, the more crossover points will be selected compared to a smaller value. This means that more elements of the trial vector $\mathbf{u}_i(t)$ will be used to produce the offspring, and less of the parent vector $\mathbf{z}_k(t)$.

Because a probabilistic decision is made as to the inclusion of a crossover point, it may happen that no points may be selected, in which case the offspring will simply be the original parent $\mathbf{z}_k(t)$. To enforce that at least one element of the offspring differs from the parent, the set $\mathcal{J}$ is initialized to include a randomly selected point j^*.

Finally, selection operator is applied to determine which of the parent or the offspring will survive to the next generation. This is implemented as follows:

$$\mathbf{z}_k(t+1) = \begin{cases} \mathbf{z}'_k(t), & \text{If } f(\mathbf{z}'_k(t)) > f(\mathbf{z}_k(t)) \\ \mathbf{z}_k(t), & \text{If } f(\mathbf{z}'_k(t)) \leq f(\mathbf{z}_k(t)) \end{cases} \tag{19}$$

where $f(\cdot)$ is the objective function to be maximized.

B. A Prototype-Based Classifier Using DE

The prototype-based classifier to be proposed in this section is optimally designed by means of DE. The proposed approach is highly motivated by recent successful applications of the DE algorithm to partitional clustering as surveyed in [3], [20]. More specifically, the design procedure can be viewed as an extension to supervised pattern classification of the automatic clustering approach introduced by Das *et al.* [3], henceforth denoted by DAK (from Das-Abraham-Konar) method. The details of the DAK method and its extension proposed in the current paper are given in the following paragraphs.

In the DAK method, each chromosome i in the population defines a clustering solution with a certain number of cluster centers optimally positioned in the pattern space. One of the features of the DAK method is to automatic determine the *optimal number* of cluster centroids (or prototypes) in an unsupervised scenario.

For this purpose, they specified a chromosome whose first K_{max} components define *activation threshold* values associated with exactly K_{max} cluster centers. For pattern vectors of dimension p, the remaining $p * K_{max}$ components of the i-th chromosome contain the corresponding cluster centers for that individual. Thus, the i-th chromosome of the DAK method is coded as follows:

$$\mathbf{z}_i(t) = [T_{i,1}\ T_{i,2}\ \cdots\ T_{i,K_{max}} \mid \mathbf{m}_1^{(i)}\ \mathbf{m}_2^{(i)}\ \cdots\ \mathbf{m}_{K_{max}}^{(i)}] \tag{20}$$

For example, let $T_{i,j}$ be the activation threshold of the j-th cluster centroid of the i-th chromosome. Thus, the j-th cluster center in the i-th chromosome is active or selected for partitioning the associated data set if $T_{i,j} > 0.5$. Otherwise, if $T_{i,j} < 0.5$, the particular j-th cluster is inactive.

During the course of execution of the DE algorithm, when a new offspring chromosome is created according to (18) and (19), at first, the activation threshold values are used to select the active cluster centroids.

If due to mutation some threshold $T_{i,j}$ in an offspring exceeds 1 or becomes negative, it is forced back either to 1 or 0, respectively. Also, if it is found that all activation thresholds are smaller than 0.5, one must randomly select two thresholds and reinitialize them to a random value between 0.5 and 1.0 in order to have always a minimum number of two clusters.

In LVQ-like supervised classifiers, the prototypes are labeled and multiple prototypes per class are allowed. Thus, in order to extend the DAK method to allow the design of prototype-based classifiers we need to define the maximum allowed number of prototypes per class, L_{max}. In general, $1 \leq L_{max} \ll n_i$, where n_i is the number of training examples of the i-th class. Then we must define activation threshold values for the prototype within a class.

In the extended DAK (EDAK) method, considering the i-th chromosome in the population, the first L_{max} activation threshold values correspond to L_{max} prototype vectors of the first class ω_1:

$$\mathbf{T}_i^{(\omega_1)} = [T_{i,1}^{(\omega_1)}\ T_{i,2}^{(\omega_1)}\ \cdots\ T_{i,L_{max}}^{(\omega_1)}] \in \mathbb{R}^{L_{max}}, \tag{21}$$

with the corresponding set of L_{max} prototypes given by

$$\mathbf{W}_i^{(\omega_1)} = [\mathbf{m}_{i,1}^{(\omega_1)}\ \mathbf{m}_{i,2}^{(\omega_1)}\ \cdots\ \mathbf{m}_{i,L_{max}}^{(\omega_1)}] \in \mathbb{R}^{p \cdot L_{max}}, \tag{22}$$

where p is the dimension of the pattern vectors.

Similarly, the subsequent L_{max} activation threshold values correspond to L_{max} prototype vectors of the second class ω_2, i.e.

$$\mathbf{T}_i^{(\omega_2)} = [T_{i,1}^{(\omega_2)}\ T_{i,2}^{(\omega_2)}\ \cdots\ T_{i,L_{max}}^{(\omega_2)}] \in \mathbb{R}^{L_{max}}, \tag{23}$$

with the corresponding set of L_{max} prototypes given by

$$\mathbf{W}_i^{(\omega_2)} = [\mathbf{m}_{i,1}^{(\omega_2)}\ \mathbf{m}_{i,2}^{(\omega_2)}\ \cdots\ \mathbf{m}_{i,L_{max}}^{(\omega_2)}] \in \mathbb{R}^{p \cdot L_{max}}. \tag{24}$$

This coding process is repeated until the last class ω_K components:

$$\mathbf{T}_i^{(\omega_K)} = [T_{i,1}^{(\omega_K)}\ T_{i,2}^{(\omega_K)}\ \cdots\ T_{i,L_{max}}^{(\omega_K)}] \in \mathbb{R}^{L_{max}}, \tag{25}$$

with the corresponding set of L_{max} prototypes given by

$$\mathbf{W}_i^{(\omega_K)} = [\mathbf{m}_{i,1}^{(\omega_K)}\ \mathbf{m}_{i,2}^{(\omega_K)}\ \cdots\ \mathbf{m}_{i,L_{max}}^{(\omega_K)}] \in \mathbb{R}^{p \cdot L_{max}}. \tag{26}$$

Grouping together all the definitions from Eqs. (21) to (26), the i-th chromosome at generation t coding a prototype-based classifier is represented as follows:

$$\mathbf{z}_i(t) = [\mathbf{T}_i^{(\omega_1)} \mid \cdots \mid \mathbf{T}_i^{(\omega_K)} \mid \mathbf{W}_i^{(\omega_1)} \mid \cdots \mid \mathbf{W}_i^{(\omega_K)}]. \tag{27}$$

From the exposed, the total dimensionality of the chromosome $\mathbf{z}_i(t)$ is

$$\begin{aligned} d &= \dim(\mathbf{z}_i(t)) = K \cdot L_{max} + K \cdot p \cdot L_{max}, \\ &= K \cdot L_{max} \cdot (1+p). \end{aligned} \tag{28}$$

As a typical example, the hypothetical chromosome shown in Figure 1 corresponds to a prototype-based classifier for a two-dimensional (i.e. $p = 2$), 2-class problem (i.e. $K = 2$), $L_{max} = 3$. with 3 prototypes assigned to the first class and 2 prototypes assigned to the second class. More examples on real-world datasets will be shown in Section IV.

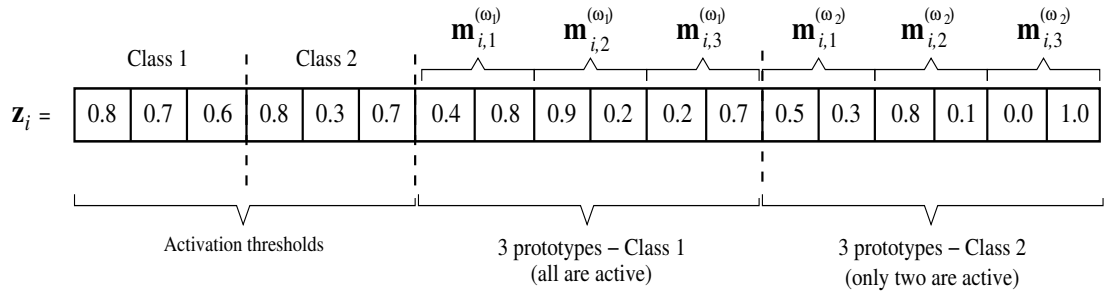

Fig. 1. Hypothetical chromosome structure for the EDAK method, corresponding to a prototype-based classifier for a two-dimensional problem. The resulting classifier has 3 prototypes assigned to the first class and two prototypes to the second class. Note that the second prototype of the second class (i.e. $\mathbf{m}_{i,2}^{(\omega_2)}$) is not active because its threshold activation is below 0.5.

C. Our Fitness Function

Since we are interested in building efficient pattern classifiers, it is natural to use a fitness function that takes into consideration a measure of the classification performance, such as the recognition rate. However, it is interesting to have the highest classification rate which is possible to achieve using the smallest number of prototypes. Bearing this in mind, the following fitness function is used by the EDAK method for the evaluation of the chromosomes at generation t:

$$f(\mathbf{z}_i(t)) = CC(\mathbf{z}_i(t)) - \lambda \cdot Q(\mathbf{z}_i(t)), \tag{29}$$

where $CC(\mathbf{z}_i(t))$ is the correct classification rate of the i-th chromosome at generation t for the validation set and $Q(\mathbf{z}_i(t))$ is the number of *active* prototypes of the classifier associated with the i-th chromosome.

The penalty factor $\lambda > 0$ is used to control the influence of the number of active prototypes on the fitness function. Since the goal of the EDAK method is to maximize $f(\mathbf{z}_i(t))$, high (small) values of λ tend to produce classifiers with small (high) numbers of prototypes. The final solution to be found by the EDAK method is a compromise between a good classification rate and a sufficient number of prototypes. The final solution found by the EDAK method will be referred to from now on as the LVQ-DE classifier.

It is worth mentioning that λ is a hyperparameter of the EDAK method and its value affects considerably the performance of the proposed approach. Obviously, more complex classification problems (i.e. with nonlinear, disjoint or highly overlapping decision boundaries) probably demand more prototypes and, hence, smaller values for the hyperparameter λ. By means of comprehensive experimentation, we have found that values in the range from 0.0025 to 0.0075 led to good performances of the EDAK method on the evaluated datasets.

IV. Simulations and Discussions

In this section we report the results of the evaluation of the performance of the EDAK method in building efficient LVQ-DE classifiers. For the sake of completeness, we compare the performance of the proposed LVQ-DE classifier with those achieved by the prototype-based classifiers described in Section II. It worth mentioning that the performance comparison is not totally fair because the only two approaches capable of determining automatically the number of prototypes is the proposed EDAK method and the LVQTC algorithm [18].

All the classifiers were implemented in Matlab R2012a (64 bits) running under Ubuntu 14.04 LTS on a notebook equipped with microprocessor Intel Core i5-2500K, 3.3GHz, with two parallel RAM units model Kingston HyperX Blu 4GB 1600MHz DDR3.

The classifiers were evaluated on five benchmarking datasets available for download from the public internet repository of the University of California at Irvine (`http://archive.ics.uci.edu/ml/`). The chosen datasets were the Vertebral Column, Glass Identification, Heart Disease Cleveland, Breast Cancer Wisconsin and Vehicle Silhouettes. These datasets are briefly describe in the following paragraphs.

Vertebral Column ($N = 310$**,** $p = 6$**,** $K = 3$): The Vertebral column dataset contains six relevant features (biomechanical attributes), correspond to the following parameters of the spino-pelvic system: 1) angle of pelvic incidence, 2) angle of pelvic tilt, 3) lordosis angle, 4) sacral slope, 5) pelvic radius and 6) grade of slipping. The data set has three classes and the objective is to classify each pattern vector into normal (100 objects), disk hernia (60) or spondilolisthesis (150 objects).

Glass Identification ($N = 214$**,** $p = 9$**,** $K = 6$): The data were sampled from six different types of glass: 1) building windows float processed (70 objects); 2) building windows nonfloat processed (76 objects); 3) vehicle windows float processed (17 objects); 4) containers (13 objects); 5) tableware (9 objects); and 6) headlamps (29 objects). Each type has nine features: 1) refractive index; 2) sodium; 3) magnesium; 4) aluminum; 5) silicon; 6) potassium; 7) calcium; 8) barium; and 9) iron.

Heart Disease Cleveland ($N = 297$**,** $p = 13$**,** $K = 2$): This database contains 13 attributes: 1) age, 2) sex, 3) chest pain type, 4) resting blood pressure, 5) serum cholesterol, 6) fasting blood sugar, 7) resting electrocardiographic results, 8) maximum heart rate achieved, 9) exercise induced angina, 10) ST depression induced by exercise relative to rest, 11) the slope of the peak exercise ST segment, 12) number of major vessels colored by fluoroscopy, and 13) type of defect (normal, fixed, or reversable). The data set has two classes and the goal is to distinguish presence (137 objects) from absence (160) of heart disease in the patient. The original dataset has $N = 303$ objects, but six objects with missing attributes were eliminated.

Breast Cancer Wisconsin ($N = 683$**,** $p = 9$**,** $K = 2$): The Wisconsin breast cancer database contains nine relevant features: 1) clump thickness; 2) cell size uniformity; 3) cell

TABLE I. RESULTS ON THE DATASET *Vertebral Column*.

Classifier	Median	St.Dev.	# Prototypes		
			ω_1	ω_2	ω_3
MDC	74.19	4.80	1	1	1
OLVQ	75.80	5.58	5	5	5
LVQ-2.1	**75.80**	**4.75**	**15**	**15**	**15**
LVQ-3	74.19	5.64	15	15	15
Soft LVQ	69.35	4.51	30	30	30
LVQTC	70.96	5.02	15	15	15
LVQDE ($\lambda = 0.005$)	**77.42**	**4.03**	**2**	**3**	**2**

TABLE II. RESULTS ON THE DATASET *Glass Identification*.

Classifier	Median	St.Dev.	# Prototypes					
			ω_1	ω_2	ω_3	ω_4	ω_5	ω_6
MDC	75.00	5.19	1	1	1	1	1	1
OLVQ	87.5	5.19	12	12	12	12	12	12
LVQ-2.1	87.5	5.40	8	8	8	8	8	8
LVQ-3	**90.0**	**5.03**	**12**	**12**	**12**	**12**	**12**	**12**
Soft LVQ	76.25	5.14	12	12	12	12	12	12
LVQTC	87.5	5.53	12	12	12	12	12	12
LVQDE ($\lambda = 0.0075$)	**92.50**	**4.60**	**1**	**3**	**1**	**1**	**1**	**2**

shape uniformity; 4) marginal adhesion; 5) single epithelial cell size; 6) bare nuclei; 7) bland chromatin; 8) normal nucleoli; and 9) mitoses. The data set has two classes. The objective is to classify each data vector into benign (239 objects) or malignant tumors (444 objects).

Vehicle Silhouettes ($N = 946$, $p = 18$, $K = 4$): This dataset contains 946 instances divided into 4 classes (OPEL, SAAB, BUS and VAN). The purpose is to classify a given silhouette as one of those four types of vehicle, using a set of 18 features intended to characterize shape, such as circularity, radius ratio, compactness, scaled variance along major and minor axes, etc. The vehicle may be viewed from one of many different angles.

For tests with using any of the aforementioned datasets, the available samples were randomly divided into training (60%), validation (20%) and testing (20%) sets. The training and validation sets are used to determine the best hyperparameters for a given classifier (e.g. training rate and window length w). For this purpose, the holdout scheme is repeated for 100 runs, with the training and validation instances chosen randomly at each run.

A. Parameters of the Evaluated Classifiers

Common parameters: All the evaluated classifiers (except for the MDC classifier), including the ones trained along the course of the generations of the EDAK method, are trained for 200 epochs. Furthermore, the maximum number of prototypes per class (L_{max}) varies with the dataset. Our approach is to set it initially to half the number of instances of the largest class (i.e. the one with the largest number of instances). Finally, all the numerical results shown in tables correspond to average values over 100 executions of a training/testing cycle for each classifier of interest.

OLVQ Classifier: For this classifier, the initial value of the optimized learning rate α_i is set to 0.3 (see Eq. (5)). Along the course of learning, the value of $\alpha_c(t)$ is reset to 0.3 if it goes above 1.

LVQ-2.1 and LVQ-3 classifier: For these classifiers, the learning rate $\alpha(t)$ is decreased linearly along the iterations from 0.1 to 0.001. The window length w is set to 0.25.

Soft LVQ classifier: Following the algorithm presented in [15], the parameter σ^2 is reduced exponentially at each epoch t' as $\sigma^2('+1) = \beta(t')\sigma^2(t')$, where the decay parameter is itself annealed as $\beta('+1) = \beta(t')^\gamma$. We set $\sigma^2(0) = 1.0$, $\beta(0) = 0.9$ and $\gamma = 1.1$. The learning rate $\alpha(t)$ used in Eqs. (12) and (13) is decreased linearly along the iterations from 0.1 to 0.001.

LVQTC classifier: This classifier in particular has a great number of training parameter to specify. Following the algorithm presented in [18], the learning rate parameters α_r and α_w are initially set to 0.1, but they are monotonically decreased for each successive epoch by a factor $F = 0.01$. The cutoff parameter P_{prn} for neuron pruning/creation is set to 5. At the end of the last epoch, no neuron pruning and creation is made. The parameters $f_{cmax} = 0.976$, $P_{min} = 7.8636$ and $D_{max} = 0.8643$ are needed in order to determine if the classification by the nearest neuron can be considered unreliable.

LVQDE classifier: The population size is set to 50, the crossover rate is set to $p_r = 0.6$ and the scaling parameter F is set to 0.1. The following values of the penalty factor λ of the fitness function were tested: $\{0.25, 0.5, 0.75 and 1.0\}$. Best value for each dataset is determined with the help of the corresponding validation set.

B. Results

The first numerical results comparing the performances of the proposed LVQDE classifier with 6 other state-of-the-art PBCs described in Section II are shown in Table I for the Vertebral Column dataset. As mentioned previously, the results shown in tables corresponding to average values over 100 executions of a training/validation/testing cycle for each classifier of interest. In these tables we report the values of median of the classification rate over the 100 executions, the corresponding standard deviation and the final optimal number of prototypes[1]. As can be seen, the performance of the LVQDE classifier was clearly the best among the evaluated classifiers. In addition, this performance was achieved with the smallest number of prototypes among all the LVQ-based classifiers. The results achieved by the two best classifiers are highlighted in boldface for the sake of comparison.

In Table II we report the results of the second performance comparison experiment, this time using the Glass Identification dataset. Again, the proposed LVQDE classifier clearly performed better than the other six ones. Again, the final number of prototypes of the LVQDE classifier is very small, when compared to the other LVQ-based classifiers.

The results of the third and fourth sets of computer experiments are shown in Tables III and IV, respectively. Results in Table III are for the Heart Disease Cleveland dataset,

[1]Of course, for the MDC classifier, by definition, there is always one prototype per class which is the centroid of the class.

TABLE III. RESULTS ON THE DATASET *Heart Disease Cleveland*.

Classifier	Median	St.Dev.	# Prototypes	
			ω_1	ω_2
MDC	81.48	5.15	1	1
OLVQ	81.48	4.86	5	5
LVQ-2.1	77.77	5.09	15	15
LVQ-3	79.62	4.68	10	10
Soft LVQ	**83.33**	**5.09**	**5**	**5**
LVQTC	79.62	5.29	5	5
LVQDE ($\lambda = 0.01$)	**83.33**	**4.48**	**1**	**1**

TABLE IV. RESULTS ON THE DATASET *Breast Cancer Wisconsin*.

Classifier	Median	St.Dev.	# Prototypes	
			ω_1	ω_2
MDC	96.29	1.79	1	1
OLVQ	96.29	1.50	10	10
LVQ-2.1	96.66	1.48	10	10
LVQ-3	**97.03**	**1.34**	**10**	**10**
Soft LVQ	96.66	1.28	10	10
LVQTC	96.29	2.41	5	5
LVQDE ($\lambda = 0.005$)	**97.03**	**1.14**	**1**	**1**

TABLE V. RESULTS ON THE DATASET *Vehicle Silhouettes*.

Classifier	Median	St.Dev.	# Prototypes			
			ω_1	ω_2	ω_3	ω_4
MDC	44.31	3.19	1	1	1	1
OLVQ	**67.06**	**2.95**	**20**	**20**	**20**	**20**
LVQ-2.1	59.88	3.85	20	20	20	20
LVQ-3	64.67	2.68	10	10	10	10
Soft LVQ	52.69	4.27	10	10	10	10
LVQTC	56.58	3.04	10	10	10	10
LVQDE ($\lambda = 0.0075$)	**65.86**	**2.78**	**2**	**2**	**2**	**2**

while the results reported in Table IV are for the Breast Cancer Wisconsin dataset. An analysis of these tables reveals that the performances of the LVQDE classifier in terms of median values of the classification rate are equivalent to the Soft LVQ classifier (Table IV) and to LVQ-3 classifier (Table IV). However, it is worth pointing out that the classification performances of the LVQDE classifier were achieved using fewer numbers of prototypes. Furthermore, it is also worth emphasizing that the optimum setting of training parameters of the Soft LVQ and the LVQ-3 classifiers, including the number of prototypes per class, were achieved after exhaustive rounds of experimentation with the data. For the LVQDE design, however, the parameter setting is relatively loose, in the sense that the user should not dedicate too much time to it, letting for example the evolution process find optimum values for the number of prototype per class. This is clearly another advantage of the proposed EDAK method over the other prototype-based classifiers evaluated in this paper.

For the sake of completeness, in Figure 2 we show the convergence of the proposed EDAK method along the generations of the DE algorithm for the Wisconsin breast cancer dataset. For each generation, we show the fitness values of the individuals that produced the best LVQDE classifier and the worst LVQDE. We also show the average fitness value of that particular generation in order to quantify how far is in average a typical individual is from the best and worst ones. As can been seen, the observed convergence process is extremely well-behaved, stabilizing around 100 generations with practically the best, worst and average fitness values converging to the same value except for minor random fluctuations).

Finally, the results of the last set of computer experiments are shown in Table V. This experiment is very interesting in the sense that it shows that a dramatically decrease in the number

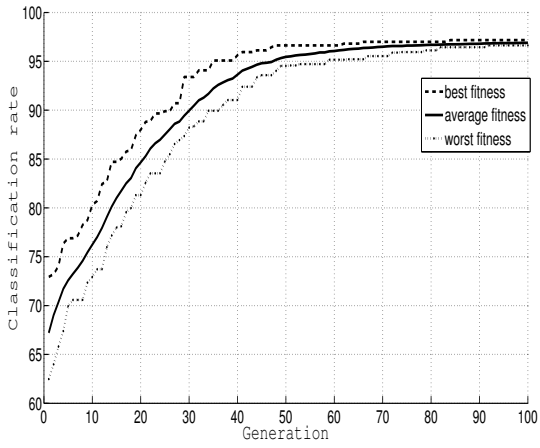

Fig. 2. Evolution of the fitness values of the best and worst individuals of the population along the generations of the DE algorithm for the Breast Cancer Wisconsin dataset. Evolution of the average fitness value of the population is also shown.

of prototypes found automatically by the LVQDE classifier results in a performance comparable to that of the best LVQ-based classifier tuned manually (the OLVQ classifier, in this case).

V. CONCLUSIONS

In this paper we proposed the EDAK method for the efficient design of prototype-based classifiers using the differential evolution (DE) global optimization method. The EDAK method can be viewed as an extension to supervised pattern classification of the automatic clustering approach introduced by Das *et al.* [3], denoted by DAK (from Das-Abraham-Konar)

method, for convenience. By means of exhaustive computer simulations on four benchmarking datasets, we have shown that the proposed EDAK method in providing an efficient prototype-based classifier (named LVQDE) consistently outperforms the state-of-the-art prototype-based classifiers, among them several ones belonging to the LVQ family of neural network classifiers.

Currently, we are evaluating the EDAK method and the resulting LVQDE classifier on additional benchmarking datasets in order to have a more comprehensive notion of its performance for different classification scenarios. Since we evaluated only one typical configuration of the DE algorithm (i.e. DE/rand/1/bin), we are also testing the performance of the EDAK method for different configurations, such as DE/rand/1/exp, DE/best/1/(bin or exp), and DE/rand-to-best/n/(bin or exp).

VI. Acknowledgment

The authors thank CNPq (grant 309841/2012-7) for the financial support and NUTEC (Fundação Núcleo de Tecnologia Industrial do Ceará) for providing the laboratory infrastructure for the execution of the research activities reported in this paper.

References

[1] T. Kohonen, "Improved versions of learning vector quantization," in *Proceedings of the 1990 International Joint Conference on Neural Networks (IJCNN'90)*, vol. 1, 1990, pp. 545–550.

[2] ——, "Learning vector quantization," in *The Handbook of Brain Theory, Neural Networks*, 2nd ed., M. A. Arbib, Ed. MIT Press, 2003, pp. 631–635.

[3] S. Das, A. Abraham, and A. Konar, "Automatic clustering using an improved differential evolution algorithm," *IEEE Transactions on Systems, Man, and cybernetics, Part A*, vol. 38, no. 1, pp. 218–237, 2008.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. John Wiley & Sons, 2006.

[5] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 698–713, 1992.

[6] T. Kohonen, "Essentials of the self-organizing map," *Neural Networks*, vol. 37, pp. 52–65, 2013.

[7] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[8] M. Biehl, K. Bunte, and P. Schneider, "Analysis of flow cytometry data by matrix relevance learning vector quantization," *PLoS One*, vol. 8, no. 3, p. e59401, 2013.

[9] S. R. Arya and B. Singh, "Implementation of distribution static compensator for power quality enhancement using learning vector quantisation," *IET Generation, Transmission & Distribution*, vol. 7, no. 11, pp. 1244–1252, 2013.

[10] A. W. Witoelar, A. Ghosh, J. J. G. de Vries, B. Hammer, and M. Biehl, "Window-based example selection in learning vector quantization," *Neural Computation*, vol. 22, no. 11, pp. 2924–2961, 2010.

[11] M. Blachnik and W. Duch, "LVQ algorithm with instance weighting for generation of prototype-based rules," *Neural Networks*, vol. 24, no. 8, pp. 824–830, 2011.

[12] C.-Y. Chang, Y.-C. Hong, P.-C. Chung, and C.-H. Tseng, "A neural network for thyroid segmentation and volume estimation in CT images," *IEEE Computational Intelligence Magazine*, vol. 6, no. 4, pp. 43–55, 2011.

[13] M. Biehl, A. Ghosh, and B. Hammer, "Dynamics and generalization ability of LVQ algorithms," *Journal of Machine Learning Research*, vol. 8, no. Feb, pp. 323–360, 2007.

[14] B. Hammer and T. Villmann, "Generalized relevance learning vector quantization," *Neural Networks*, vol. 15, no. 8-9, pp. 1059–1068, 2002.

[15] S. Seo and K. Obermayer, "Soft learning vector quantization," *Neural Computation*, vol. 15, no. 7, pp. 1589–1604, 2003.

[16] S. Seo, M. Bode, and K. Obermayer, "Soft nearest prototype classification," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 390–398, 2003.

[17] P. Schneider, M. Biehl, and B. Hammer, "Hyperparameter learning in robust soft LVQ," in *Proceedings of the 17th European Symposium on Artificial Neural Networks (ESANN'09*, 2009, pp. 517–522.

[18] R. Odorico, "Learning vector quantization with training count (LVQTC)," *Neural Networks*, vol. 10, no. 6, pp. 1083–1088, 1997.

[19] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[20] S. Paterlini and T. Krink, "Differential evolution and particle swarm optimisation in partitional clustering," *Computational Statistics & Data Analysis*, vol. 50, pp. 1220–1247, 2006.