Some Novel Classifiers Designed Using Prototypes Extracted by a New Scheme Based on Self-Organizing Feature Map

Arijit Laha and Nikhil R. Pal, *Senior Member, IEEE*

***Abstract*—We propose two new comprehensive schemes for designing prototype-based classifiers. The scheme addresses all major issues (number of prototypes, generation of prototypes, and utilization of the prototypes) involved in the design of a prototype-based classifier. First we use Kohonen's self-organizing feature map (SOFM) algorithm to produce a minimum number (equal to the number of classes) of initial prototypes. Then we use a dynamic prototype generation and tuning algorithm (DYNAGEN) involving merging, splitting, deleting, and retraining of the prototypes to generate an adequate number of useful prototypes. These prototypes are used to design a "1 nearest multiple prototype (1-NMP)" classifier. Though the classifier performs quite well, it cannot reasonably deal with large variation of variance among the data from different classes. To overcome this deficiency we design a "1 most similar prototype (1-MSP)" classifier. We use the prototypes generated by the SOFM-based DYNAGEN algorithm and associate with each of them a zone of influence. A norm (Euclidean)-induced similarity measure is used for this. The prototypes and their zones of influence are fine-tuned by minimizing an error function. Both classifiers are trained and tested using several data sets, and a consistent improvement in performance of the latter over the former has been observed. We also compared our classifiers with some benchmark results available in the literature.**

***Index Terms*—Dynamic prototype generation, nearest neighbor (NN) classifier, prototype-based classifier, self-organizing feature map (SOFM).**

I. Introduction

A CLASSIFIER designed from a data set $X = \{\mathbf{x}_i | i = 1, \ldots, n;\ \mathbf{x}_i \subset \Re^p\}$ can be defined as a function $\mathcal{D}: \Re^p \rightarrow N_c$, where $N_c = \{\mathbf{e}_i | i = 1, \ldots, c;\ \mathbf{e}_i \in \Re^c\}$ is the set of label vectors, p is the number of features, and c is the number of classes. If $\mathcal{D}$ is a fuzzy classifier, then $c_{ij} \geq 0$ and $\sum_{j=1}^{c} c_{ij} = 1$. If $\mathcal{D}$ is a crisp classifier, $\mathbf{e}_i$ is a basis vector with components $c_{ij} = 0\ \forall i \neq j$ and $c_{ii} = 1$; consequently, here also $\sum_{j=1}^{c} c_{ij} = 1$. However, for a possibilistic classifier $\sum_{j=1}^{c} c_{ij} \leq c$ [3]. Designing a classifier involves finding a good $\mathcal{D}$. $\mathcal{D}$ may be specified parametrically, e.g., Bayes classifier [1], or nonparametrically, e.g., nearest neighbor (NN) classifiers (crisp and fuzzy) [1], [3], nearest prototype (NP) classifiers (crisp or fuzzy) [1], [3], neural networks [4], etc.

Manuscript received October 29, 1999; revised May 31, 2001. This paper was recommended by Associate Editor B. J. Oommen.

A. Laha is with the National Institute of Management Calcutta, Alipore, Calcutta 700 027, India (e-mail: arijitl@yahoo.com).

N. R. Pal is with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700 035, India (e-mail: nikhil@isical.ac.in).

Publisher Item Identifier S 1083-4419(01)08548-X.

Although Bayes classifier is statistically optimal, it requires complete knowledge of prior probabilities $p_j;\ j = 1, 2, \ldots, c$ and class densities $g(\mathbf{x}|j);\ j = 1, 2, \ldots, c$, which is almost never possible in practical cases. Usually no knowledge of the underlying distribution is available except that it can be estimated from the samples.

Among the nonparametric classification schemes the (NN) algorithms are the most straightforward. This family of algorithms is known as k-NN algorithms. Given a set of n labeled training samples $X = \{(\mathbf{x}_i, \mathrm{l}_i | i = 1, \ldots, n;\ \mathbf{x}_i \in \Re^p, \mathrm{l}_i \in N_c)\}$ the crisp k-NN $(1 \leq k < n)$ classifier assigns a sample $\mathbf{x} \notin X$ as follows.

- ◇ Find the set of k samples $\{(\mathbf{x}_i, \mathrm{l}_i)\} \in X$ closest to $\mathbf{x}$.
- ◇ Assign $\mathbf{x}$ to the class from which majority of k closest neighbors has come.

All crisp classifiers assign an absolute label to a sample tested. But in real world situation this is often disadvantageous. It would be better to have some measure of confidence available for different alternative decisions. If more than one decision have close confidence values then they may be examined more closely using additional information (if available) instead of immediately committing to a decision that might incur heavy penalty. This has motivated development of several variants of fuzzified k-NN algorithms [2], [3].

Despite their simplicity, a straightforward implementation of an NN classifier may require large memory (the complete set of training samples X has to be kept in memory) and may involve large computational overhead (the distance between $\mathbf{x}$ and each of the training data points has to be computed). There have been several attempts to minimize the computational overhead of k-NN algorithm [5], [6], some of which approximate the k-NN scheme. However, even with most of them, the entire data set needs to be maintained. The prototype-based classifiers overcome these drawbacks. The training data set is represented by a set of prototypes $V = \{(\mathbf{v}_i, \mathrm{l}_i) | i = 1, \ldots, \hat{c}\}$, where $\hat{c} \geq c$. Each prototype can be thought of as a representative of a subset of X.

While designing a prototype-based classifier, one faces three fundamental issues.

- How to generate the prototypes.
- How many prototypes are to be generated.
- How to use the prototypes for designing the classifier.

Depending on the schemes adopted to address these questions, there are a large number of classifiers. To illustrate the point we discuss the simplest of them, i.e., the NP classifiers

with a single prototype per class. The number of prototypes is equal to the number of classes in the training data, i.e., $\hat{c} = c$. In this kind of classifier the prototype for a class is usually generated by taking the mean of the training data vectors from that class. The third issue is commonly addressed using a distance function δ. A vector $\mathbf{x} \in \Re^p$ is classified using the prototype set $V = \{\mathbf{v}_i, \mathbf{l}_i | i = 1, \ldots, c, \mathbf{v}_i \in \Re^p, \mathbf{l}_i \in N_c\}$, where c is the number of classes and $\mathbf{l}_i$ is the label vector associated with $\mathbf{v}_i$ as follows.

$$\begin{aligned} \text{Decide} \quad & \mathbf{x} \in \text{ class } i \\ \Leftrightarrow \quad & \mathcal{D}_{V,\delta}(\mathbf{x}) = \mathbf{l}_i \\ \Leftrightarrow \quad & \delta(\mathbf{x}, \mathbf{v}_i) \leq \delta(\mathbf{x}, \mathbf{v}_j) \qquad \forall j \neq i. \end{aligned}$$

This simple scheme and its variants work quite well in many problem domains. However, such a classifier has been proved inadequate if the data from one class are distributed into more than one cluster or if the data from two different classes cannot be separated by a single hyperplane, as demonstrated in the famous "XOR" data [7]. Therefore, for a generalized classifier design one must keep provision for multiple prototypes for a class.

If multiple prototypes are used for a class, the three issues concerning the prototypes become more difficult to address, but in this case a lot of sophisticated techniques become available for dealing with them. Usually the three issues can be addressed independently, but in some cases the strategy used for answering one issue may depend on the strategy used for dealing with others or a single strategy may take care of more than one issue.

We now briefly review some commonly used schemes for addressing these issues. The optimal number of prototypes required depends on both the interclass as well as the intraclass distribution of the data. One can use suggestions of a domain expert, or some clustering algorithm to find cluster centroids in the training data set with enough prototypes to represent each class. However, most of the clustering algorithms (e.g., c-means algorithm) require the number of clusters to be supplied externally or to be determined using some cluster validity index. The k-nn algorithms [1] use each data point in the training data set as a prototype.

Given the number of prototypes, there are many procedures for generating them, e.g., clustering algorithms like c-means [1], fuzzy c-means [2], mountain clustering method [8], etc. Other approaches include learning vector quantization (LVQ) methods, NN-based methods such as Kohonen's SOFM, random search methods, gradient search methods, etc. Each method has its own advantages and limitations. Often more than one method are used together for producing a good set of prototypes.

Once the prototypes are generated, there are several ways of using the prototypes in the classifier. To mention a few, in an NP classifier a data point is assigned the class label of the prototype closest to it. In a fuzzy rule-based classifier, each prototype can be used to generate a fuzzy rule to define a fuzzy rulebase [14]. This rulebase is then used for classifying the data.

In the present paper, two new approaches to design prototype-based classifiers are proposed. The problem of finding the required number of prototypes as well as the prototypes themselves are addressed together. A set of c (the number of classes) initial prototypes is generated from a set of labeled training data (without using the label information) using Kohonen's SOFM algorithm. Then the prototypes are labeled using the class information following a "most likely class" heuristic. The proposed algorithm, during the tuning stage evaluates the performance of the prototypes and depending on the evaluation result, prototypes are deleted, merged, and split. This is continued until the performance of the prototypes stabilize. The procedure takes into account the intraclass and interclass distribution of the training data and tries to produce an "optimal" set of prototypes. This set of prototypes is then used to design an NP classifier. This is our first classifier, i.e., "1-nearest multiple prototype (1-NMP) classifier." This 1-NMP classifier has some limitations as will be demonstrated later. To address these limitations the set of prototypes is further processed through a fine-tuning stage. During this stage a zone of influence for each prototype is defined. A "similarity measure" and an error function is defined. Then the prototypes' position and their zones of influence are modified iteratively in order to minimize the error function. Any unknown data point is then classified according to its maximum similarity with the prototypes. We call this classifier "1-most similar prototype (1-MSP) classifier." Both classifiers are tested with several data sets and compared with some benchmark results.

II. Generation of the Initial Set of Prototypes

We use Kohonen's self-organizing feature map (SOFM) algorithm [9] for generating the initial set of prototypes. For the sake of completeness, first we give a brief description of the SOFM and then proceed to describe the generation and labeling scheme for the initial set of prototypes.

A. Kohonen's SOFM Algorithm

The SOFM, denoted here by $A^D_{SOFM}: \Re^p \to V(\Re^q)$, is often advocated for visualization of metric–topological relationships and distributional density properties of feature vectors (signals) $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ in $\Re^p$. Usually X is transformed into a display lattice of $q \leq 3$ dimensions. In this article we concentrate on $(m \times n)$ displays in $\Re^2$.

Input vectors $\mathbf{x} \in \Re^p$ are distributed by a fanout layer to each of the $(m \times n)$ output nodes in the competitive layer. Each node in this layer has a weight vector $\mathbf{w}_{ij}$ attached to it.

SOFM begins with a (usually) random initialization of the weight vectors $\mathbf{w}_{ij}$. For notational clarity we suppress the double subscripts. Now let $\mathbf{x} \in \Re^p$ enter the network and let t denote the current iteration number. A^D_{SOFM} finds $\mathbf{w}_{r,t-1}$ that best matches $\mathbf{x}$ in the sense of minimum Euclidean distance in $\Re^p$. Then $\mathbf{w}_{r,t-1}$ and the other weights in its spatial neighborhood are updated using the rule

$$\mathbf{w}_{i,t} = \mathbf{w}_{i,t-1} + \alpha_t g_t(\mathbf{x} - \mathbf{w}_{i,t-1}). \tag{1}$$

Here α_t is the learning parameter and g_t is a Gaussian-type function with spread σ_t, $g_t = \exp^{-dist^2(r,i)/\sigma_t^2}$. α_t and σ_t both decrease with time t. The topological neighborhood also decreases with time. This scheme, when repeated long enough, usually preserves spatial order in the sense that weight vectors

which are metrically close in $\Re^p$ generally have, at termination of the learning procedure, visually close images in the viewing plane $V(\Re^p)$, and the distribution of the weight vectors in $\Re^p$ will reflect the distribution of the training data X.

B. Generation and Labeling of Prototypes

In this investigation we use a one-dimensional (1-D) SOFM, but the algorithm can be extended to a two-dimensional (2-D) SOFM also. First, we train a 1-D SOFM using the training data. Although the class information is available, SOFM training does not use it. The number of nodes in the SOFM is the same as the number of classes c. This choice is inspired by the fact that the minimum number of prototypes that are required is equal to the number of classes. At the end of the training, the weight vector distribution of the SOFM will reflect the distribution of the input data. These unlabeled prototypes are then labeled using class information. For each of N input feature vectors we identify the prototype closest to it, i.e., the winner node. Since no class information is used during the training, some prototypes may become the winner for data from more than one class, particularly when the classes overlap or touch each other. For each prototype $\mathbf{v}_i$ we compute a score D_{ij}, which is the number of data points from class j to which $\mathbf{v}_i$ is the closest prototype. Due to strong interaction among the neighboring nodes of the SOFM during the training, some prototypes may be so placed that for no input data they are the closest prototypes; i.e., D_{ij} is 0 for all j. We reject such prototypes. For the remaining prototypes the class label C_i of the prototype $\mathbf{v}_i$ is determined as

$$C_i = \underbrace{\arg\max}_{j} D_{ij}. \tag{2}$$

This scheme will assign a label to each of the c prototypes, but such a set of prototypes may not classify the data satisfactorily. For example, from (2) it is clear that $\sum_{j \neq i} D_{ij}$ data points will be wrongly classified by the prototype $\mathbf{v}_i$. Hence, we need further refinement of the initial set of prototypes $V_0 = \{\mathbf{v}_{10}, \mathbf{v}_{20}, \ldots, \mathbf{v}_{c0}\} \subset \Re^p$.

III. Generating a Better Set of Prototypes

The prototypes generated by the SOFM algorithm represent the overall distribution of the input data. A set of prototypes useful for a classification job must be capable of dealing with class specific characteristics (such as class boundaries) of the data. In this section we present an algorithm (DYNAGEN) for generating a set of prototypes that is a better representative of the distribution of training data and takes into account the class specific characteristics. The strategy involves a modification procedure starting with the initial set of prototypes V_0. In each iteration the prototype set is used in an NP classifier and the classification performance is observed. Depending on the performance of the individual prototypes in the classifier, prototypes are modified, merged, split, and deleted, leading to a new set of prototypes. This process of modification is repeated until the number of prototypes and their performance stabilize within an acceptable level. The final set of prototypes, when used to design prototype-based classifiers is expected to enhance the performance of the classifier. We use the term "performance of a prototype" to indicate the "performance of the prototype when used in an NP classifier."

In the t-th iteration, the prototype set V_{t-1} from previous iteration is used to generate the new set of prototypes V_t. The labeled prototypes V_{t-1} are used to classify a set of training data. Let W_i be the number of training data to which prototype $\mathbf{v}_i$ is the closest one. Let $S_i = \max_j\{D_{ij}\} - D_{iC_i}$. Thus, when $\mathbf{v}_i$ is labeled as a prototype for class C_i, S_i training data points will be correctly classified by $\mathbf{v}_i$ and $F_i = \sum_{j \neq C_i} D_{ij}$ data points will be incorrectly classified. Consequently, $W_i = S_i + F_i$ and $W_i = \sum_j D_{ij}$.

In the training data $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ let there be N_j points from class j. The refinement stage uses just *two* parameters K_1 and K_2 to dynamically generate $c+1$ retention thresholds known as global retention threshold α and a set of classwise retention threshold β_k (one for each class), to evaluate the performance of each prototype. α and β_k are computed dynamically (not fixed) for the tth iteration using the following formulae:

$$\alpha_t = \frac{1}{K_1 \mid V_{t-1} \mid}$$

$$\beta_{kt} = \frac{1}{K_2 \mid V_{kt}^* \mid}$$

where $V_{kt}^* = \{\mathbf{v}_i \mid \mathbf{v}_i \in V_{t-1}, C_i = k\}$. We emphasize that the algorithm uses just two (not $c+1$) user supplied parameters.

Based on the classification performance of the prototypes, different operations are performed to generate a new set of prototypes. First, we define the various operations. Later, while describing the algorithm, we state when to use these operations.

1) Merging of a Prototype With Respect to a Class: Let a prototype $\mathbf{v}_i$ represent D_{ik} training data from class k. To merge $\mathbf{v}_i$ w.r.t. class k we identify the prototype $\mathbf{v}_l$ closest to $\mathbf{v}_i$ where $C_l = k$ (i.e., $\mathbf{v}_l$ also is a prototype for class k). Let us denote X_{ij} as the set of training data vectors from class j whose nearest prototype is $\mathbf{v}_i$. When we merge $\mathbf{v}_i$ with $\mathbf{v}_l$ w.r.t. class k, $\mathbf{v}_l$ is updated according to

$$\mathbf{v}_l = \frac{W_l \mathbf{v}_l + \sum_{\mathbf{x} \subset X_{ik}} \mathbf{x}}{W_l + D_{ik}}. \tag{3}$$

Note that we do not say here when to merge. This will be discussed later.

2) Modifying a Labeled Prototype: A prototype $\mathbf{v}_i$ is modified according to the following equation:

$$\mathbf{v}_i = \frac{\sum_{\mathbf{x} \in X_{iC_i}} \mathbf{x}}{D_{iC_i}}. \tag{4}$$

3) Splitting a Prototype: A prototype $\mathbf{v}_i$ is split into r new prototypes for r different classes according to (5). For each of r new prototypes $\mathbf{v}_l$ for class C_l we compute

$$\mathbf{v}_l = \frac{\sum_{\mathbf{x} \subset X_{iC_l}} \mathbf{x}}{D_{iC_l}}. \tag{5}$$

The prototype $\mathbf{v}_i$ is deleted. So after the splitting the number of prototypes is increased by $r-1$.

4) Deleting a Prototype: A prototype, say $\mathbf{v}_i$, is just deleted so that number of prototypes is reduced by one. Now we are in a position to describe the DYNAGEN algorithm for dynamic generation and enhancement of the prototypes.

Algorithm DYNAGEN:

Repeat for all $\mathbf{v}_i \in V_{t-1}$ (starting with $\mathbf{v}_i$ having smallest W_i) until the termination condition is satisfied.

If $W_i \neq D_{iC_i}$ and $W_i < \alpha_i N$ and there is at least another prototype for class C_i
then delete $\mathbf{v}_i$. (Global deletion)

/* If a prototype is not pure one (i.e., it represents data from more than one class) and does not represent a reasonable number of points, it fails to qualify to become a prototype. However, if there is no other prototype for class C_i the prototype is retained */

Else if $W_i > \alpha_i N$ but $D_{ij} < \beta_{ji} N_j$ for all classes
then merge $\mathbf{v}_i$ for the classes for which $D_{ij} > 0$ and delete $\mathbf{v}_i$. (Merge and delete)

/* The prototype represents a reasonable number of points, but not reasonable number of points from any particular class, so it can not qualify as a prototype for any particular class. But we cannot ignore the prototype completely. We logically first split $\mathbf{v}_i$ into s prototypes $\mathbf{v}_{i1}, \mathbf{v}_{i2}, \ldots \mathbf{v}_{is}$, $s \leq c$, s is the total number of classes for which $D_{ij} > 0$, and then merge $\mathbf{v}_{ij}$ to its closest prototype from class j. $\mathbf{v}_i$ is then deleted. */

Else if $W_i > \alpha_i N$ and $D_{iC_i} > \beta_{C_i i} N_{C_i}$ but $D_{ij} < \beta_{ji} N_j$ for all $j \neq C_i$
then merge $\mathbf{v}_i$ with respect to all the classes other than C_i for which $D_{ij} > 0$ using (3) and modify $\mathbf{v}_i$ using (4). (Merge and modify)

/* The prototype represents points from more than one class; however, the points from only one class are well represented by the prototype. According to our labeling scheme the prototype is labeled with the most represented class. Thus, we merge $\mathbf{v}_i$ with respect to the classes other than C_i using (3) and then modify $\mathbf{v}_i$ by (4). */

Else if $W_i > \alpha_i N$ and $D_{ij} > \beta_{ji} N_j$ for more than one class
then merge $\mathbf{v}_i$ w.r.t. classes for which $D_{ij} < \beta_{ji} N_j$ by (3) and split $\mathbf{v}_i$ into new prototypes for the classes for which $D_{ij} > \beta_{ji} N_j$ by (5).
Add these new prototypes to the new set of prototypes V_t. (Merge and split)

/* The prototype represents reasonably well the number of points from more than one class. So we merge the prototype with respect to the classes whose data are not well represented and split the prototype into one for each class whose data are reasonably well represented by $\mathbf{v}_i$. */

Let V_t be the union of all unaltered prototypes of V_{t-1} and the modified as well as new prototypes. Run the SOFM algorithm on V_t with *winner-only update* (i.e., no neighbor is updated) strategy using the same training data as input.

/* At this stage we want only to fine tune the prototypes. If the neighbors are also updated the prototypes again might migrate to represent points from more than one class. */

5) Termination Conditions: The algorithm terminates under any one of the following two conditions.

a) Achievement of a satisfactory recognition score defined in terms of percentage of correct classifications (ϵ).
b) A specified number of iterations ($I_{\max}$) is reached.

Proper use of condition a) requires some knowledge of the data. However, even if we do not have the same, we can always set a high (conservative) percentage for (ϵ), say 95%.

Condition b) is used to protect against infinite looping of the algorithm for some data with highly complex structures for which the chosen values of ϵ may not be reachable.

We now design a nearest multiple prototype (1-NMP) classifier using the set of prototypes generated by DYNAGEN.

A. Results of 1-NMP Classifier

We report the performance of the classifier for ten data sets. The data sets are divided into two groups A and B. Group A consists of six data sets including **Iris, Glass, Breast Cancer, Vowel, Norm4**, and **Two-Dishes** while group B contains **Cone-torus, Normal Mixture, Sat-image** and **Phoneme**. For group A data sets, some results are available in the literature but the details of the experimental paradigm (such as training test division, computational protocols, etc.) used are not available. Hence, we have randomly divided each data set in group A into two approximately equal partitions. We then use these partitions alternately as training and test sets and report the performance. For the group B data sets, many benchmark results for different classifiers are documented in [17] along with computational protocols. We have used the same partitions as well as compared our results with those reported in [17].

Iris data [10] have 150 points in four dimensions that are from three classes, each with 50 points. Glass data [11] consist of 214 samples with nine attributes from six classes. Breast Cancer [12] have 569 points in 30 dimensions from two classes. A normalized version of the Breast Cancer data is generated by dividing each component by the largest value of that component. The Vowel [15] data set consists of 871 samples of discrete phonetically balanced speech samples for the Telugu vowels in consonant–vowel nucleus–consonant (CNC) form. It is a three-dimensional (3-D) data containing the first three formant frequencies. The 871 data points are unevenly distributed over six classes. The data set Norm4 [13] is a sample of 800 points consisting of 200 points each from the four components

TABLE I
PERFORMANCE OF THE 1-NMP CLASSIFIER FOR GROUP A DATA SETS

Data Set	Size		No. of prototypes		% of Error		Average Test Error
	Trng.	Test	Initial	Final	Trng.	Test	
Iris	75	75	3	5	2.66%	8.0%	
	75	75	3	5	4.0%	5.33%	6.66%
Glass	105	109	6	28	16.19%	34.86%	
	109	105	6	26	19.26%	33.33%	34.09%
Breast Cancer	284	285	2	4	10.91%	11.58%	
	285	284	2	5	9.12%	12.67%	12.13%
Normalised Breast Cancer	284	285	2	8	8.8%	5.61%	
	285	284	2	4	6.66%	10.56%	8.08%
Vowel	434	437	6	21	18.4%	19.22%	
	437	434	6	15	20.82%	22.35%	20.78%
Norm	400	400	4	4	4.25%	3.5%	
	400	400	4	4	4.0%	4.0%	3.75%
Two-Dishes	750	750	2	2	6.13%	5.33%	
	750	750	2	2	5.6%	6.53%	5.93%

of a mixture of four class 4-variate normals. The Two-Dishes is a synthetically generated data set consisting of 1500 2-D data points distributed uniformly over two well-separated dishes of different radii. Each dish has 750 points.

In group B the Cone-Torus data set has 400 2-D points in both the training and test sets [17], [18]. There are three classes each representing a different shape, a cone, a half torus, and a Gaussian shape. In the Normal Mixture data each of the two classes come from a normal distribution of 2-D points [17], [19]. The training set contains 250 points and the test set contains 1000 points. The Sat-image data set is generated from Landsat Multi-Spectral Scanner image data [17], [20]. The present data set covers an area of 82 $\times$ 100 pixels portion of the whole image. Each feature vector has 36 components containing the gray values of nine (a 3 $\times$ 3 neighborhood) pixels captured by four sensors operating in different spectral regions. The data set has seven classes representing different kinds of ground covers. The training set has 500 points and the test set has 5935 points. The Phoneme data set contains vowels coming from 1809 isolated syllables (e.g., pa, ta, pan, ...) in French and Spanish [17], [20]. Five different attributes are chosen to characterize each vowel; they are the amplitudes of the five first harmonics *AHi*, normalized by the total energy *Ene* (integrated on all frequencies) *AHi/Ene*. Each harmonic is signed positive when it corresponds to a local maximum of the spectrum and negative otherwise. The Phoneme data set has two classes, nasal and oral. The training set contains 500 points and the test set contains 4904 points.

Tables I and II summarize the classification performances of 1-NMP classifier for group A and group B data sets, respectively. We used the values $K_1 = 3$, $K_2 = 6$, $\epsilon = 95\%$, and $I_{max} = 10$ for all data sets. The % of error column shows the percentage of misclassification for training and test data.

It is well known that classes 2 and 3 of Iris have some overlap and the typical resubstitution error with an NP classifier defined by three prototypes obtained by some clustering algorithm is 15–16 (i.e., about 10% error with three prototypes). The average test performance of the proposed system with five prototypes is quite good resulting in only 6.66% error.

Glass data show a high percentage of error; this is possibly unavoidable because a scatterplot (Fig. 1) of the first two principal components shows that the data for class 3 are almost randomly distributed among the data points from other classes. In fact, the points from class 3 (represented by $+$) are not visible in the scatterplot. In [14], the recognition score for the glass data is 64.4%, i.e., about 35% error. Our classifier could realize more than 66% accuracy with 26 prototypes in the best test result and the average test error is 34.09%.

TABLE II
PERFORMANCE OF THE 1-NMP CLASSIFIER FOR GROUP B DATA SETS

Data Set	Size		No. of prototypes		% of Error	
	Trng.	Test	Initial	Final	Trng.	Test
Cone-torus	400	400	3	12	18.25%	22.5%
Normal Mixture	250	1000	2	4	14.0%	9.5%
Sat-image	500	5935	7	27	13.6%	15.65%
Phoneme	500	4904	2	5	21.8%	21.31%

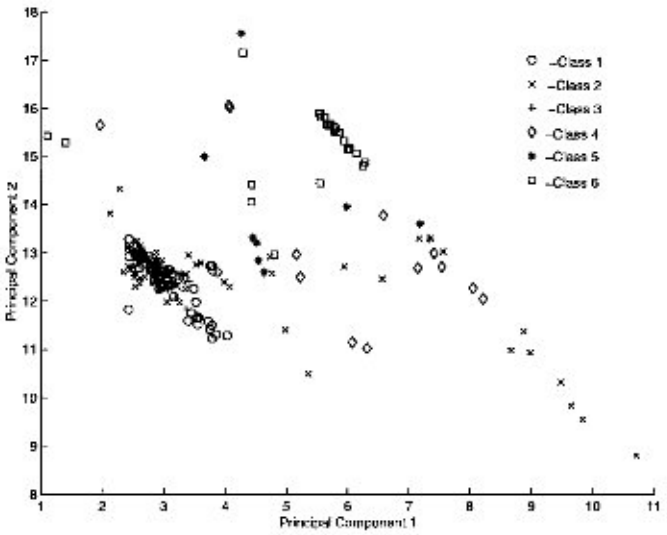

Fig. 1. Scatterplot of the glass data along the two most significant principal components.

Breast Cancer data have been used in [12] to train a linear programming-based diagnostic system by a variant of multisurface method (MSM) called MSM-tree and about 97.5% accuracy was obtained. Breast Cancer data of a similar kind have also been used in a recent study [14] with 74.0% accuracy with 100 rules. Our classifier could achieve as low as 12.13% average test error with only four to five prototypes and it is quite good. With the normalized breast cancer data the 1-NMP classifier exhibits a much better performance.

Although the Vowel data set has three features, like other authors [15], we used only the first two features. Bayes classifier for this data set [16] gives an overall recognition score of 79.2%. In Fig. 2, the scatterplot (different classes are represented by different characters) of Vowel data depicts that there is substantial overlap among different classes and hence some misclassification is unavoidable. The proposed classifier could achieve an average of 79.22% correct classification.

The performance on Norm4 [13] with only four prototypes, i.e., one prototype per class, is quite good. In this case, the DYNAGEN-based classifier could achieve up to 96% accuracy with only four prototypes.

The Two-Dishes data is a synthetic one generated for investigating the limitations of the 1-NMP classifier. Although a straight line can separate the two classes, the 1-NMP classifier achieves an accuracy of 96% with two prototypes only. Our tuning algorithm could have produced more prototypes and

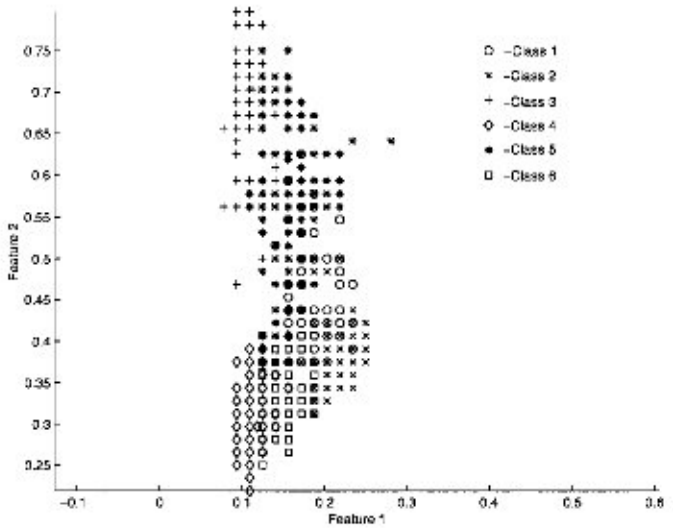

Fig. 2. Scatterplot of first two features of the vowel data.

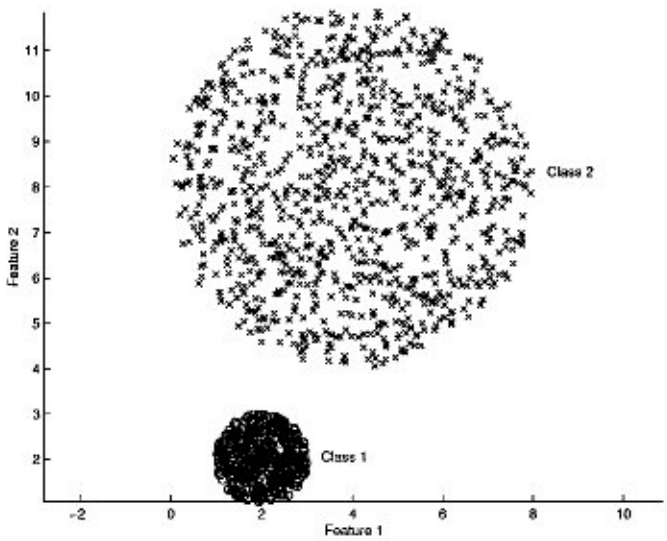

Fig. 3. Scatterplot of the Two Dishes data.

thereby could reduce the error, but we used only two prototypes to demonstrate the usefulness of the classifier described in the following section.

Table II shows that for all of the group B data sets, the 1-NMP classifier generalizes quite well. For two data sets the performance on the test sets is little better than that on the training sets while for the other two cases the performance on the training sets is little better.

As evident from the results obtained, a 1-NMP classifier designed with the prototypes generated by the DYNAGEN algorithm results in a very efficient classifier. As with any prototype-based classifier, the performance of this classifier depends on the quality of the prototype set used, i.e., how faithfully the distribution of the data is represented by the set of prototypes. In our case, the DYNAGEN algorithm is ultimately responsible for the quality of the prototype set. On closer examination of the DYNAGEN algorithm, one can find the following general tendencies acting on the prototypes.

- The DYNAGEN algorithm tries to generate a very small number of prototypes while at the same time tries to keep the error rate as low as possible. This is achieved by means of two opposing actions: 1) splitting of prototypes and 2) deletion of prototypes. The former leads to an increase in the number of prototypes with a view to ensuring that class boundaries are taken care of while the latter decreases the number of prototypes by deleting prototypes, which do not represent an adequate number of data points. These two actions together try to ensure that there are enough prototypes to represent the distribution of the data and each prototype represents a substantial number of data points. In other words, the number of prototypes does not increase in an uncontrolled manner to become comparable with the number of training data points.
- The DYNAGEN algorithm tries to place each prototype at the center of a dense set of points. This is achieved by the retraining of the prototype set using SOFM algorithm with the winner-only update scheme at the end of each modification cycle. This step reduces the effect of outlier data points on the positioning of the prototypes in the feature space.

The above capabilities make the tuning algorithm suitable for generating quality prototypes even for complex training data sets with linearly nonseparable classes or for data sets for which points from the same class are divided into more than one cluster. These desirable qualities of the prototype set is reflected in the performance of the classifier.

However, the result obtained for the Two-Dishes data set (Fig. 3) indicates that just placement of prototypes at the center of each class in some situations can turn into a weakness of the algorithm. Though the 1-NMP classifier with two prototypes generates only 5.93% error rate for Two-Dishes, it is not good enough. Fig. 3 shows that the data from two classes are linearly separable and each class forms a distinct cluster. But the two classes have considerably different variances. One can reasonably expect that this data set should be classified with very high accuracy with only two prototypes, but it is beyond the capability of the 1-NMP classifier. This is due to the fact that the tuning process places a prototype at the center of a cluster and in case of clusters having considerably different variances, such a set of prototypes, when used in an (NP) classifier, may not classify the data accurately.

To address this problem, we propose a new strategy of fine-tuning the prototypes and design a new classifier. In this new strategy each prototype is associated with a zone of influence. This modification is expected to improve the classification performance over the NP classifier and it might reduce the number of prototypes as well. The scheme is described in the next section.

IV. New Classifier and Fine-Tuning of Prototypes

We describe the design of a new prototype-based classifier, the "1 most similar prototype (1-MSP)" classifier, and the fine-tuning of the prototype set in order to achieve better classification performance.

A. The 1-MSP Classifier

Given a set of labeled prototypes V, we define a zone of influence for each prototype. The influence of a prototype $\mathbf{v}_i$ at a point $\mathbf{x} \in \Re^p$ is defined by the equation

$$\alpha(\mathbf{v}_i, \mathbf{x}) = \exp^{-\|\mathbf{v}_i - \mathbf{x}\|^2/\sigma_i} \tag{6}$$

where $\sigma_i > 0$ is a parameter associated with prototype $\mathbf{v}_i$. $\alpha(\mathbf{v}_i, \mathbf{x})$ is a decreasing function of the Euclidean distance $\|\mathbf{v}_i - \mathbf{x}\|$. The surface of constant influence for a prototype is a hypersphere in $\Re^p$ centering at the prototype. Note that other choices of α are also possible.

The function $\alpha(\mathbf{v}_i, \mathbf{x})$ also serves as a measure of similarity between the prototype $\mathbf{v}_i$ and the point $\mathbf{x}$. The 1-MSP classifier is designed to classify a data point $\mathbf{x}$ as follows:

$$\begin{aligned} \text{Decide} \quad & \mathbf{x} \subset \text{class } i \\ \Leftrightarrow \quad & \mathcal{D}_{V,\alpha}(\mathbf{x}) = l_i \\ \Leftrightarrow \quad & \alpha(\mathbf{x}, \mathbf{v}_i) \geq \alpha(\mathbf{x}, \mathbf{v}_j) \qquad \forall j \neq i. \end{aligned}$$

B. Fine Tuning of the Prototypes

Let V be a set of prototypes ($|V| = c \geq$ the number of classes) generated from the training data X by the DYNAGEN algorithm discussed in Section III or by some other means. Let $\mathbf{x}_i \in X$ be from class c and $\mathbf{v}_{ci}$ be the prototype for class c having the greatest similarity α_{ci} with $\mathbf{x}_i$. Also let $\mathbf{v}_{\neg ci}$ be the prototype from incorrect classes having the greatest similarity $\alpha_{\neg ci}$ with $\mathbf{x}_i$. Thus

$$\alpha_{ci} = \exp^{-\|\mathbf{x}_i - \mathbf{v}_{ci}\|/\sigma_{ci}}$$

and

$$\alpha_{\neg ci} = \exp^{-\|\mathbf{x}_i - \mathbf{v}_{\neg ci}\|/\sigma_{\neg ci}}$$

where σ_{ci} and $\sigma_{\neg ci}$ are the values of the σ parameters associated with the prototypes $\mathbf{v}_{ci}$ and $\mathbf{v}_{\neg ci}$, respectively.

We use an error function E

$$E = \sum_{\mathbf{x}_i \in X} (1 - \alpha_{ci} + \alpha_{\neg ci})^2. \tag{7}$$

Such an error function has been used by Chiu [21] in the context of designing a fuzzy rule-based classifier. The fine-tuning algorithm presented here involves minimization of E following the steepest descent search to modify the prototypes and their zones of influence.

```
Prototype Modification Algorithm
Begin
Set the learning parameters η_v and η_σ.
Set a parameter reduction factor 0 < ε < 1.
Set the maximum number of iteration maxiter.
Compute E_0 using (7) for V_0 = (v_1^0, v_2^0, ..., v_c^0).
Compute the misclassification M_0 of 1-MSP classifier using V_0.
While (t < maxiter) do
  For each x_i ∈ X
    Find the prototypes v_ci^{t-1} and v_¬ci^{t-1}.
    Compute α_ci and α_¬ci.
    Modify the prototypes v_ci^{t-1} and v_¬ci^{t-1} and their
    zones of influence using the following equations.
```

$$\begin{aligned} \mathbf{v}_{ci}^t &= \mathbf{v}_{ci}^{t-1} - \eta_m \frac{\partial E}{\partial \mathbf{v}_{ci}^{t-1}} \\ &= \mathbf{v}_{ci}^{t-1} + \eta_m (1 - \alpha_{ci} + \alpha_{\neg ci}) \frac{\alpha_{ci}}{\sigma_{ci}^{t-1}} (\mathbf{x}_i - \mathbf{v}_{ci}) \end{aligned}$$

$$\begin{aligned} \mathbf{v}_{\neg ci}^t &= \mathbf{v}_{\neg ci}^{t-1} + \eta_m \frac{\partial E}{\partial \mathbf{v}_{\neg ci}^{t-1}} \\ &= \mathbf{v}_{\neg ci}^{t-1} - \eta_m (1 - \alpha_{ci} + \alpha_{\neg ci}) \frac{\alpha_{\neg ci}}{\sigma_{\neg ci}^{t-1}} (\mathbf{x}_i - \mathbf{v}_{\neg ci}) \end{aligned}$$

$$\begin{aligned} \sigma_{ci}^t &= \sigma_{ci}^{t-1} - \eta_s \frac{\partial E}{\partial \sigma_{ci}^{t-1}} \\ &= \sigma_{ci}^{t-1} + \eta_s (1 - \alpha_{ci} + \alpha_{\neg ci}) \frac{\alpha_{ci}}{\sigma_{ci}^{t-1^2}} \|\mathbf{x}_i - \mathbf{v}_{ci}\|^2 \end{aligned}$$

$$\begin{aligned} \sigma_{\neg ci}^t &= \sigma_{\neg ci}^{t-1} + \eta_s \frac{\partial E}{\partial \sigma_{\neg ci}^{t-1}} \\ &= \sigma_{\neg ci}^{t-1} - \eta_s (1 - \alpha_{ci} + \alpha_{\neg ci}) \frac{\alpha_{\neg ci}}{\sigma_{\neg ci}^{t-1^2}} \|\mathbf{x}_i - \mathbf{v}_{\neg ci}\|^2. \end{aligned}$$

```
    Compute E_t using (7) for the new set of
    prototypes V_t.
    Compute the misclassification M_t of 1-MSP
    classifier using V_t.
    If M_t > M_{t-1} or E_t > E_{t-1}
      then
      η_m ← (1 − ε)η_m
      η_s ← (1 − ε)η_s
      V_t ← V_{t-1}
        /*If the error is increased, then possibly
        the learning coefficients are too large.
        So, decrease the learning coefficients, and
        retain V_{t-1} and continue.*/
    If M_t = 0 or E_t = 0
      then Stop.
  End While
  End
```

In each iteration for each data point the algorithm finds *the prototype which has the maximum possibility of being responsible for correctly classifying the point* and *the prototype which has the maximum possibility of being responsible for wrongly classifying the point*. Then the parameters associated with these two prototypes are modified so that the possibility of correct classification of the data point increases while that of wrong classification decreases.

When the algorithm terminates we have the final set of prototype V_{Final}, which is expected to give a very low error rate when used with the 1-MSP classifier.

C. Implementation and Results of 1-MSP Classifier

To implement the above prototype refinement and classification scheme, one question needs to be answered. How do we get an initial estimate of the zones of influence, i.e., σ_is for the initial prototypes?

In our implementation we used the prototype set generated by our previous SOFM-based DYNAGEN algorithm. If we set the initial values of σ_is as

$$\sigma_i = \text{constant} \qquad \forall i$$

theoretically the tuning starts with a set of prototypes whose performance in a 1-MSP classifier is the same as its performance in a 1-NMP classifier since

$$\alpha(\mathbf{v}_i, \mathbf{x}_k) \geq \alpha(\mathbf{v}_j, \mathbf{x}_k) \Leftrightarrow \|\mathbf{v}_i - \mathbf{x}_k\| \leq \|\mathbf{v}_j - \mathbf{x}_k\| \quad \forall \mathbf{v}_i, \mathbf{v}_j, \mathbf{x}_k \in \Re^p.$$

Then with the progress of tuning, the prototypes and σ_is will be modified to reduce E. However, there are a few practical considerations. If the constant is not chosen judiciously, the tuning may not yield the desired result. To elaborate this point, if the σ_is are too small, then there will be a sharp fall of α with distance and this may cause loss of important digits due to finite precision on a digital computer. Consequently, unless high precision is used, the performance of the 1-NMP and 1-MSP classifier may be slightly different. On the other hand, if the σ_is are too large for a data point, all the prototypes may produce high α values and, during training, the reduction of the error may be too slow making the tuning less effective. We get an initial estimate of the zones of influence (σ_is) for the prototypes as follows.

For each prototype $\mathbf{v}_i^0$ in the set $V_0 = \{\mathbf{v}_i^0 | i = 1, \ldots, \hat{c}, \mathbf{v}_i^0 \in \Re^p\}$ let X_i be the set of training data closest to $\mathbf{v}_i^0$. For each $\mathbf{v}_i^0$ a set

$$S_i = \left\{ \sigma_{ij} \,\middle|\, j = 1, \ldots, p, \sigma_{ij} = \left(\sqrt{\left(\sum_{\mathbf{x}_k \in X_i} (x_{kj} - v_{ij})^2 \right)} \right) \Big/ |X_i| \right\}$$

is computed. Then the set of initial σ_is is computed as

$$S = \left\{ \sigma_i \,\middle|\, i = 1, \ldots, \hat{c}, \sigma_i = \left(\sum_{\sigma_{ij} \in S_i} \sigma_{ij} \right) \Big/ p \right\}.$$

As evident from the above definitions, σ_i is the average of the standard deviations of the individual components of data vectors $\mathbf{x} \in X_i$ about the prototype $\mathbf{v}_i^0$. This is just a choice for the initial σ_is; other choices are possible too.

Table III summarizes the result of the 1-MSP classifier for six data sets in group A. We have used the same training and test division for all data sets as used for the 1-NMP classifier. For most of the data sets, the results show marked improvement over those of the 1-NMP classifier, while for the rest the performance remains almost the same. We emphasize the Two-Dishes data set to explain our motivation for the design of the 1-MSP classifier. The result shows an average of 0.07% error rate for Two-Dishes with only two prototypes, and the best result obtained shows zero error.

However, the 1-MSP classifier fails in the case of the original Breast Cancer data set. Our algorithm assumes hyperspherical clusters in the feature space. This is expected when, in a cluster, the standard deviations of all features are nearly equal. But the Breast Cancer data have 30 features, of which the majority have values of the order of 10^{-2} and some are of the order of 10^{3}, and the standard deviations of different features are considerably different. Our initialization scheme computes the average of these standard deviations to associate a hyperspherical zone of influence with each prototype. Since most of the components have small values, the σ_is are small and αs for most of the data points also become very small, leading to loss of precision and huge misclassification rates. However, if the σ_is are increased artificially (by a factor of say 400), the performance matches that of the 1-NMP classifier, but the reduction of the error function E becomes very slow during training. Such data could be better handled if the zones of influence of the prototypes are not limited to hypersphere. But such schemes (like using Mahalanobis distance instead of Euclidean distance) would lead to more computational overhead. So, we get around this by normalizing the data over each component. The Normalized Breast Cancer data shows an excellent performance of the 1-MSP classifier.

Apparently the 1-MSP classifier has some similarity with the radial basis function (RBF) network, since each prototype is associated with a zone of influence, and the strength of the influence is determined by a Gaussian function of Euclidean distance from the prototype. The hidden nodes in an RBF network also use the Gaussian functions as activation functions (there are other possibilities too). However, in the conventional RBF network, a linear aggregation function is used between the hidden layer and the output layer. There is complete connection between the hidden layer and the output layer. If we think of an RBF-like architecture for the 1-MSP classifier, we cannot have the complete connection between the hidden layer and the output layer. The aggregation function in each output node would be a *max* operator, not a linear function. Moreover, the α-function used here need not necessarily be a basis function. As we shall see, in general, the RBF network needs more nodes for similar performances. However, our algorithm can be used as a preprocessing stage for the RBF classifier, where the weight vectors of RBF nodes can be initialized with the positions of the prototypes and the spreads of the activation functions can be initialized with the value of the corresponding σ. To make a comparison, we have classified the group A data sets (same as in Table III) using RBF networks with the same number of RBF nodes as the number of prototypes used by the 1-MSP classifier. We have used the routines available in the Neural Network Toolbox of Matlab 5. For all experiments we use 2.0 for the spread of the RBFs. The results for the RBF classifiers are summarized in Table IV.

Table IV reveals a poor performance of the RBF classifiers compared to our 1-MSP classifiers for most of the data sets. For Iris and Normalized Breast Cancer it shows a slightly better performance. These results clearly suggest the superiority of the

TABLE III
PERFORMANCE OF THE 1-MSP CLASSIFIER FOR GROUP A DATA SETS

Data Set	Size		No. of prototypes	% of Error		Average Test Error
	Trng.	Test		Trng.	Test	
Iris	75	75	5	2.66%	2.66%	
	75	75	5	2.66%	4.0%	3.33%
Glass	105	109	28	14.28%	34.86%	
	109	105	26	13.76%	27.62%	31.24%
Normalized Breast Cancer	284	285	8	5.63%	4.21%	
	285	284	4	4.91%	7.04%	5.62%
Vowel	434	437	21	16.82%	18.53%	
	437	434	15	17.39%	17.51%	18.02%
Norm	400	400	4	4.25%	4.25%	
	400	400	4	4.0%	3.5%	3.87%
Two-Dishes	750	750	2	0%	0.13%	
	750	750	2	0%	0%	0.07%

TABLE IV
PERFORMANCE OF THE RBF NETWORKS FOR GROUP A DATA SETS

Data Set	Size		Network configuration	% of Error		Average Test Error
	Trng.	Test		Trng.	Test	
Iris	75	75	4-5-3	4.0%	2.66%	
	75	75	4-5-3	2.66%	2.66%	2.66%
Glass	105	109	9-28-6	22.8%	49.54%	
	109	105	9-26-6	30.27%	43.8%	46.67%
Normalized B. Cancer	284	285	30-8-2	3.17%	2.45%	
	285	284	30-4-2	3.51%	5.98%	4.21%
Vowel	434	437	2-21-2	55.06%	56.97%	
	437	434	2-15-2	48.28%	50.46%	53.71%
Norm	400	400	3-4-4	36.5%	36.25%	
	400	400	3-4-4	37.25%	38.5%	37.37%
Two-Dishes	750	750	2-2-2	9.6%	10.13%	
	750	750	2-2-2	6.66%	6.0%	8.06%

TABLE V
PERFORMANCE OF THE 1-MSP CLASSIFIER FOR GROUP B DATA SETS

Data Set	Size		No. of prototypes	% of Error	
	Trng.	Test		Trng.	Test
Cone-torus	400	400	12	16.5%	14.75%
Normal Mixture	250	1000	4	13.4%	9.7%
Sat-image	500	5935	27	13.4%	15.6%
Phoneme	500	4904	5	19.8%	20.53%

1-MSP classifier over RBF networks with comparable configurations especially for data sets with complex class structures.

Table V summarizes the performance of the 1-MSP classifiers for the group B data sets. For each of the data sets, the training error is lower for the 1-MSP classifier than the 1-NMP classifier. The test error for Cone-torus data decreases substantially. Test error for Normal Mixture and Sat-image remains almost the same while, for the Phoneme data, there is some improvement for the test set.

These four data sets, as mentioned earlier, have been extensively investigated in [17]. In Tables VI and VII, we reproduce some results using a multilayer perceptron (MLP) and an RBF, respectively, on these data sets. To make a fair comparison, for both networks, we have chosen two architectures having the number of nodes closest to the number of prototypes used by our 1-MSP classifier.

Comparing Table V with Table VI shows that for Cone-torus data, the performance of the 1-MSP classifier with 12 prototypes is comparable to the average performance of the MLP with ten and 15 hidden nodes. For Normal Mixture, the performance of 1-MSP with four prototypes is also comparable with the performance of MLP networks with five and ten hidden nodes. While for Sat-image the 1-MSP classifier produces

TABLE VI
RESULTS WITH MLP NETWORKS FOR GROUP B DATA SETS

Data Set	Network Size	Trng Error	Test Error
Cone-torus	2-10-3	15.25%	14.25%
	2-15-3	13.50%	12.00%
	Average	14.37%	13.12%
Normal Mixture	2-5-2	12.00%	10.00%
	2-10-2	12.80%	10.30%
	Average	12.40%	10.15%
Sat-image	4-20-6	79.20%	75.92%
	4-65-6	24.40%	23.08%
	Average	51.80%	49.50%
Phoneme	5-5-2	14.00%	18.23%
	5-10-2	16.80%	21.04%
	Average	15.40%	19.63%

TABLE VII
RESULTS WITH RBF NETWORKS FOR GROUP B DATA SETS

Data Set	Network Size	Trng Error	Test Error
Cone-torus ($\sigma = 3$)	2-10-3	16.50%	13.75%
	2-15-3	17.00%	14.00%
	Average	16.75%	13.87%
Normal Mixture ($\sigma = 1.5$)	2-5-2	14.00%	9.50%
	2-10-2	12.40%	10.00%
	Average	13.20%	9.75%
Sat-image ($\sigma = 10$)	4-15-6	14.80%	17.02%
	4-20-6	12.80%	15.52%
	Average	13.80%	16.27%
Phoneme ($\sigma = 2$)	5-5-2	21.00%	23.65%
	5-10-2	18.80%	21.31%
	Average	19.90%	22.48%

13.4% training error and 15.6% test error, the MLP even with 64 hidden nodes produces 24.4% training error and 23.08% test error. The performance of MLP with 20 hidden nodes is very poor (79.2% training error and 75.92% test error). For Phoneme data, the MLP with five hidden nodes produces a little better result than 1-MSP with five prototypes. Comparing Table VII with Table V we find that the performance of 1-MSP classifier and RBF network is comparable for Normal Mixture; for Sat-image and Phoneme the 1-MSP classifier performs a little better than the RBF network, while for Cone-torus the performance of RBF is a little better than the 1-MSP classifier.

V. CONCLUSION

We have presented two comprehensive schemes for designing a prototype-based classifier. The schemes address all major issues involved with the design. Based on the training data set, our SOFM-based DYNAGEN algorithm takes care of prototype generation as well as determination of an adequate number of prototypes needed for the classification by an NP classifier. The performance of the 1-NMP classifier is quite good. However, as pointed out earlier, the NP classifier cannot take care of large variations in the variances of the data from different classes. To equip the classifier to deal with such a situation it must use the variance of the data represented by each prototype.

In our second classifier (1-MSP), we associate with each prototype a zone of influence that tries to cover the spread of the data represented and thereby it accounts for the class variance. To measure the proximity we use a (Euclidean) norm-induced

similarity measure that implicitly defines the limit of the zone of influence of each prototype. Due to the choice of Euclidean norm, the surfaces of constant influence are hyperspherical in shape. So 1-MSP performs better for the data having hyperspherical or near-hyperspherical class/subclass structure.

Apparently one can think of two ways of using Mahalanobis distance to deal with data having nonhyperspherical class/subclass structure. The first is to run some clustering algorithm on the whole training data and compute the covariance matrix for each cluster. However, in this method, the class information is ignored and a cluster may contain data from different classes. This may particularly defeat the purpose of the exercise. The second is to cluster the data from each class separately to generate the prototypes and compute the covariance matrix of the data points closest to each prototype. This "closeness" criterion is again based on Euclidean distance. So the computation of Mahalanobis distance based on the covariance matrix thus obtained may not solve the problem unless it is updated during training, which may not be simple. Moreover, although this option uses the class label information, it cannot capture interaction between classes and this has to be accounted for during refinement of the prototypes. We are currently investigating such possibilities for dealing with most general types of class/subclass structures by the 1-MSP classifier.

ACKNOWLEDGMENT

The authors would like to thank the referees for their valuable suggestions that have helped to improve the manuscript considerably.

REFERENCES

[1] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1974.
[2] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.
[3] J. C. Bezdek *et al.*, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Norwell, MA: Kluwer, 1999.
[4] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1999.
[5] F. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 750–753, July 1975.
[6] B. Dasarathy, *Nearest Neighbor Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1991.
[7] J. Zurada, *Introduction to Artificial Neural Systems*. St. Paul, MN: West, 1992.
[8] R. R. Yager and D. P. Filev, "Approximate clustering by the mountain method," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 1279–1283, Aug. 1994.
[9] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, pp. 1464–1480, Sept. 1990.
[10] E. Anderson, "The IRISes of the Gaspe peninsula," *Bull. Amer. IRIS Soc.*, vol. 59, pp. 2–5, 1935.
[11] R.C. Holte, "Very simple classification rules perform well on most commonly used data sets," *Mach. Learn.*, vol. 11, pp. 63–91, 1993.
[12] O. L. Mangasarian, W. N. Street, and W. H. Wolberg, "Breast cancer diagnosis and prognosis via linear programming," *Oper. Res.*, vol. 43, no. 4, pp. 570–577, 1995.
[13] N. R. Pal and J. C. Bezdek, "On cluster validity for the fuzzy c-means model," *IEEE Trans. Fuzzy Syst.*, vol. 3, pp. 370–379, June 1995.
[14] H. Ishibuchi, T. Nakashima, and T. Murata, "Performance evaluation of fuzzy classifier systems for multi-dimensional pattern classification problems," *IEEE Trans. Syst., Man, Cybern. B*, vol. 29, pp. 601–618, Oct. 1999.
[15] S. K. Pal and D. Dutta Majumder, "Fuzzy sets and decision making approaches in vowel and speaker recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-7, pp. 625–629, Aug. 1977.
[16] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 683–697, Sept. 1992.
[17] L. Kuncheva, *Fuzzy Classifiers*. New York: Physica-Verlag, 2000.
[18] University of Wales, Bangor. [Online]. Available: http://www.bangor.ac.uk/mas00a/Z.txt and http://www.bangor.ac.uk/mas00a/Zte.txt.
[19] University of Oxford, Department of Statistics. [Online]. Available: http://www.stats.ox.ac.uk/ riplay/PRNN.
[20] University College London. [Online]. Available: ftp://ftp.dice.ucl.ac.be/pub/neural-nets/ELENA.
[21] S. L. Chiu, "Fuzzy model identification based on cluster estimation," *J. Intell. Fuzzy Syst.*, vol. 2, pp. 267–278, 1994.

Arijit Laha received the B.Sc. (with honors) and M.Sc. degrees in physics from the University of Burdwan, India, in 1991 and 1993, respectively. He received the M.Tech. degree in computer science from the Indian Statistical Institute, Calcutta, in 1997.

From August 1997 to May 1998 he worked as a Senior Software Engineer, Wipro Infotech Global R&D, Bangalore, India. From May 1998 to December 1999 he worked on a real-time expert system development project at the Indian Statistical Institute. Currently, he is a Lecturer, Department of Computer Science, National Institute of Management, Calcutta. His research interests include pattern recognition, data compression, neural networks, fuzzy systems, and expert systems.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree (with honors) in physics and the M. S. degree in business management from the University of Calcutta, India, in 1979 and 1982, respectively. He received the M.Tech. and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, in 1984 and 1991, respectively.

Currently, he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute. From September 1991 to February 1993, July 1994 to December 1994, October 1996 to December 1996, and January 2000 to July 2000, he visited the Computer Science Department, University of West Florida, Pensacola. He was also a Guest Faculty of the University of Calcutta. His research interests include image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers. He has coauthored a book entitled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing* (Norwell, MA: Kluwer, 1999), coedited a volume of *Advances in Pattern Recognition and Digital Techniques, ICAPRDT99*, and edited the book *Pattern Recognition in Soft Computing Paradigm* (Singapore: World Scientific, 2001). He is an Associate Editor of the *International Journal of Fuzzy Systems* and *International Journal of Approximate Reasoning*. He is also an Area Editor of *Fuzzy Sets and Systems*.

Dr. Pal is an Associate Editor of IEEE TRANSACTIONS ON FUZZY SYSTEMS and IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS–PART B (electronic version).