

# Multiple-Prototype Classifier Design

James C. Bezdek, *Fellow, IEEE*, Thomas R. Reichherzer, Gek Sok Lim, and Yianni Attikiouzel, *Fellow, IEEE*

**Abstract**—Five methods that generate multiple prototypes from labeled data are reviewed. Then we introduce a new sixth approach, which is a modification of Chang’s method. We compare the six methods with two standard classifier designs: the 1-nearest prototype (1-np) and 1-nearest neighbor (1-nn) rules. The standard of comparison is the resubstitution error rate; the data used are the Iris data. Our modified Chang’s method produces the best consistent (zero errors) design. One of the competitive learning models produces the best minimal prototypes design (five prototypes that yield three resubstitution errors).

**Index Terms**—Competitive learning, Iris data, modified Chang’s method (MCA), multiple prototypes, nearest neighbor (1-nn) rule.

## I. INTRODUCTION

PERHAPS the most basic idea in pattern recognition is the class label. There are four types of labels—crisp, fuzzy, probabilistic, and possibilistic. Let integer  $\hat{c}$  denote the number of classes  $1 < \hat{c} < n$  and define three sets of label vectors in  $\mathcal{R}^{\hat{c}}$ , as follows:

$$N_{p\hat{c}} = \{\mathbf{y} \in \mathcal{R}^{\hat{c}}: y_i \in [0, 1] \forall i, y_i > 0 \exists i\} \\ = [0, 1]^{\hat{c}} - \{(0, 0, \dots, 0)^T\} \quad (1a)$$

$$N_{f\hat{c}} = \left\{ \mathbf{y} \in N_{p\hat{c}}: \sum_{i=1}^{\hat{c}} y_i = 1 \right\} \quad (1b)$$

$$N_{h\hat{c}} = \{\mathbf{y} \in N_{f\hat{c}}: y_i \in [0, 1] \forall i\} \\ = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{\hat{c}}\}. \quad (1c)$$

$N_{h\hat{c}}$  is the canonical (unit vector) basis of Euclidean  $\hat{c}$ -space. The  $i$ th vertex of  $N_{h\hat{c}}$ ,  $\mathbf{e}_i = (0, 0, \dots, \underbrace{1}_i, \dots, 0)^T$ , is the

crisp label for class  $i$ ,  $1 \leq i \leq \hat{c}$ .  $N_{f\hat{c}}$ , a piece of a hyperplane, is the convex hull of  $N_{h\hat{c}}$ . The vector  $\mathbf{y} = (0.1, 0.6, 0.3)^T$  is a fuzzy or probabilistic label vector; its entries lie between zero and one and sum to one.  $N_{p\hat{c}}$ , the unit hypercube in  $\mathcal{R}^{\hat{c}}$ , excluding the origin, contains possibilistic label vectors, such as  $\mathbf{z} = (0.7, 0.2, 0.7)^T$ . Note that  $N_{h\hat{c}} \subset N_{f\hat{c}} \subset N_{p\hat{c}}$ .

Object data are represented as  $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$  in feature space  $\mathcal{R}^p$ . The  $k$ th object (a ship, patient, stock market report, pixel, etc.) has  $\mathbf{x}_k$  as its numerical representation;  $\mathbf{x}_{jk}$  is the  $j$ th characteristic (or feature) associated with object  $k$ . Examples of alternating optimization (AO) algorithms that

generate each of the four kinds of labels as well as a set  $V = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{\hat{c}}\} \subset \mathcal{R}^p$  of point prototypes (or cluster centers) for clusters in  $X$  from unlabeled object data are *hard  $\hat{c}$ -means* (HCM), Duda and Hart [1]; *fuzzy  $\hat{c}$ -means* (FCM), Bezdek [2]; probabilistic mixtures (EM), Titterington *et al.* [3]; and *possibilistic  $\hat{c}$ -means* (PCM), Krishnapuram and Keller [4].

A classifier, any function  $D: \mathcal{R}^p \mapsto N_{p\hat{c}}$ , specifies  $\hat{c}$  decision regions in  $\mathcal{R}^p$ . Training a classifier means identification of the parameters of  $D$  if it is explicit or representing the boundaries defined by  $D$  algorithmically if it is implicit. The value  $\mathbf{y} = D(\mathbf{z})$  is the label vector for  $\mathbf{z}$  in  $\mathcal{R}^p$ .  $D$  is a *crisp classifier* if  $D[\mathcal{R}^p] = N_{h\hat{c}}$ . New, unlabeled object data that enter feature space after *crisp* decision regions are defined simply acquire the label of the region they land in. If the classifier is fuzzy, probabilistic, or possibilistic, labels  $\mathbf{y}$  assigned to object vectors  $\mathbf{z}$  during the operational (i.e., classification) phase are almost always converted to crisp ones through the hardening of  $\mathbf{y}$  with the function

$$\mathbf{H}[D(\mathbf{z})] = \mathbf{H}(\mathbf{y}) = \mathbf{e}_i \Leftrightarrow \|\mathbf{y} - \mathbf{e}_i\| \\ < \|\mathbf{y} - \mathbf{e}_j\| \Leftrightarrow y_i > y_j, \quad j \neq i. \quad (2)$$

In (2), the distance is Euclidean,  $\delta_E(\mathbf{y}, \mathbf{e}) = \|\mathbf{y} - \mathbf{e}\| = \sqrt{(\mathbf{y} - \mathbf{e})^T(\mathbf{y} - \mathbf{e})}$ , and ties are resolved arbitrarily. If the design data are labeled (that is, if we have training data that possess class label vectors in  $N_{p\hat{c}}$ ), finding  $D$  is called *supervised learning*. In supervised classifier design,  $X$  is usually crisply partitioned into a *design* (or training) set  $X_{tr}$  with *label matrix*  $L_{tr}$  and a *test set*  $X_{te} = (X - X_{tr})$  with *label matrix*  $L_{te}$ . Columns of  $L_{tr}$  and  $L_{te}$  are label vectors in  $N_{p\hat{c}}$ .

Testing a classifier  $D$  designed with  $X_{tr}$  means finding its *error rate* (or estimated probability of misclassification). The standard method for doing this is to submit  $X_{te}$  to  $D$  and count mistakes ( $L_{te}$  must have crisp labels for data in  $X_{te}$  in order to do this). This yields the *apparent* error rate  $E_D(X_{te}|X_{tr})$ ; our notation indicates that  $D$  was trained with  $X_{tr}$  and tested with  $X_{te}$ .  $E_D$  is often the performance index by which  $D$  is judged because it measures the extent to which  $D$  generalizes to the test data.

When  $X = X_{tr} = X_{te}$ , the error rate  $E_D(X|X)$  is called the *resubstitution* error rate. Some authors call  $D$  a *consistent* classifier if and only if  $E_D(X|X) = 0$ . Resubstitution uses the same data for training and testing, so it usually produces an optimistic error rate. That is,  $E_D(X|X)$  is not as reliable as  $E_D(X_{te}|X_{tr})$  for assessing the *generalization* ability of  $D$ , but this is not an impediment to using  $E_D(X|X)$  as a basis for *comparison* of different designs. Moreover, unless  $n$  is very large compared to  $p$  and  $\hat{c}$  (an often used rule of thumb is  $n \geq 100p\hat{c}$ ), the credibility of either error rate is questionable.

Manuscript received May 17, 1996; revised December 3, 1996 and July 5, 1997. This research was supported by ONR Grant N00014-96-1-0642.

J. C. Bezdek and T. R. Reichherzer are with the Department of Computer Science, University of West Florida, Pensacola, FL 32514 USA (e-mail: jbezdek@ai.uwf.edu).

G. S. Lim and Y. Attikiouzel are with the Center for Intelligent Information Processing Systems, Department of Electrical and Electronic Engineering, University of Western Australia, Nedlands, Perth, 6009 Western Australia.

Publisher Item Identifier S 1094-6977(98)01528-4.

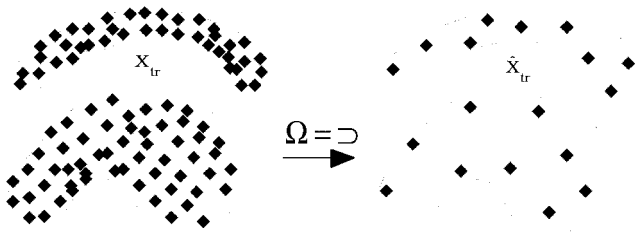


Fig. 1. Editing by selection of labeled data in  $X_{tr}$ .

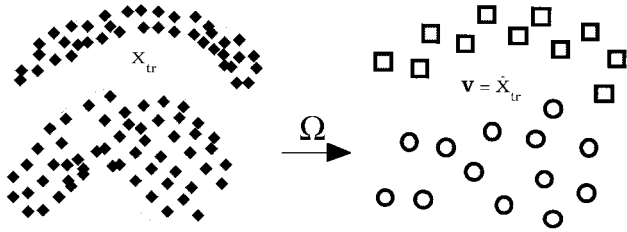


Fig. 2. Editing by replacement of  $X_{tr}$  with labeled prototypes  $\mathbf{V}$ .

Although the implicit use of the methods we discuss is for classification (and hence, good generalization potential), the data used in our examples does not justify worrying about the difference between  $E_D(X|X)$  and  $E_D(X_{te}|X_{tr})$ .

Classifier performance is largely dependent on the quality of  $X_{tr}$ . If  $X_{tr}$  is large enough and its substructure is well delineated, we expect classifiers trained with it to yield small error rates. On the other hand, when the training data are large in dimension  $p$  and/or number  $n$ , classifiers such as the  $k$ -nn ( $\mathbf{D}_{k-nn}$ ) rule [5] can require too much storage and CPU time for efficient deployment. To circumvent time and storage problems caused by very large data sets, many authors have studied ways to transform the original data  $X_{tr}$  into a smaller, but equally useful data set, say,  $\hat{X}_{tr}$ , so that  $E_D(X_{te}|X_{tr}) \approx E_D(X_{te}|\hat{X}_{tr})$ . The  $k$ -nn classifier  $\mathbf{D} = \mathbf{D}_{k-nn}$  is often used to decide whether this objective has been met.

Two common editing schemes are *selection and replacement*. Selection finds a *proper subset*  $\hat{X}_{tr} \subset X_{tr}$ . Replacement uses a transformation  $\Omega: \mathbb{R}^p \mapsto \mathbb{R}^p$  to find  $\hat{X}_{tr} = \Omega[X_{tr}]$ . The process of subset selection is a special case of replacement. Replacements are almost always labeled prototypes, such as  $\mathbf{V}$  produced by  $\Omega$ . In this paper,  $\Omega$  is any one of seven prototype generation algorithms, all of which are compared to one selection algorithm.

Fig. 1 depicts selection. The density of labeled data over each cluster in the left side of Fig. 1 is high. A selected subset (or skeleton) of the original data are shown on the right. This approach has many variants and is well summarized in Devijver and Kittler [6]. The aim is to condense  $X_{tr}$ , while approximately preserving the shape of the decision boundaries set up by training  $\mathbf{D}$  with it.

Fig. 2 illustrates replacement by multiple prototypes. In this scheme,  $X_{tr}$  is *replaced* by  $\mathbf{V}$ , a set of labeled prototypes for classes 1 ( $\square$ ) and 2 ( $\circ$ ). Note that there is more than one prototype per class.

We discuss six multiple-prototype generation schemes: 1) *learning vector quantization* (LVQ) [7], 2) a family of fuzzy

LVQ models called GLVQ-F [8], [9], 3) the deterministic *dog-rabbit* (DR) model of Lim *et al.* [10], [11],<sup>1</sup> 4) a deterministic hierarchical clumping model due to Chang [12], 5) our modification of Chang's algorithm, and 6) a modification of batch fuzzy  $\hat{c}$ -means [13]. These six schemes will be compared to the standard nearest prototype (1-np) and nearest neighbor (1-nn) rule classifiers.

## II. 1-NP CLASSIFIERS

Synonyms for the word prototype include *vector quantizer* (VQ), signature, template, codevector, paradigm, centroid, and exemplar. There are many approaches to prototype generation. A nonexhaustive list includes *sequential* competitive learning models, such as crisp (adaptive)  $\hat{c}$ -means [1], LVQ [7], GLVQ-F [8], GLVQ [9], the DR model [10], [11], and probabilistic schemes such as SCS [14]. *Batch* prototype generator models include crisp and fuzzy  $\hat{c}$ -means [2], possibilistic  $\hat{c}$ -means [4], statistical models such as mixture decomposition [3], and VQ approaches such as the generalized Lloyd algorithm [15].

The common denominator in most point prototype generation schemes is a mathematical definition of how well  $\mathbf{v}_i$  represents a crisp subset  $X_i$  of  $X$ . Any measure of similarity on  $\mathbb{R}^p$  can be used. The usual choice is distance (dissimilarity), while the most convenient is squared Euclidean distance. Local methods attempt to optimize some function of the  $\hat{c}$ -squared distances  $\{\|\mathbf{x}_k - \mathbf{v}_i\|^2: 1 \leq i \leq \hat{c}\}$  at each  $\mathbf{x}_k$  in  $X_i$ . Global methods seek extrema of some function of all  $\hat{c}n$  distances  $\{\|\mathbf{x}_k - \mathbf{v}_i\|^2: 1 \leq i \leq \hat{c} \text{ and } 1 \leq k \leq n\}$ . Once the prototypes  $\mathbf{V}$  are found (and possibly relabeled if the data have physical labels), they can be used to define the crisp 1-np classifier  $\mathbf{D}_{\mathbf{V}, \delta}$ .

*The 1-np Classifier:* Given any  $\hat{c}$  prototypes  $\mathbf{V} = \{\mathbf{v}_j: 1 \leq j \leq \hat{c}\}$ , so there is one  $\mathbf{v}_j$ /class, and any dissimilarity measure  $\delta$  on  $\mathbb{R}^p$ : for any  $\mathbf{z} \in \mathbb{R}^p$

$$\text{Decide } \mathbf{z} \in \text{class } i \Leftrightarrow \mathbf{D}_{\mathbf{V}, \delta}(\mathbf{z}) = \mathbf{e}_i \Leftrightarrow \delta(\mathbf{z}, \mathbf{v}_i) < \delta(\mathbf{z}, \mathbf{v}_j), \quad \forall j \neq i. \quad (3)$$

Ties in (3) are arbitrarily resolved. The crisp 1-np design can be implemented by using prototypes from *any* algorithm that produces them. Equation (3) defines a crisp classifier even when  $\mathbf{V}$  comes from a fuzzy, probabilistic, or possibilistic algorithm. It would be careless to call  $\mathbf{D}_{\mathbf{V}, \delta}$  a fuzzy classifier, for example, just because fuzzy  $\hat{c}$ -means produced  $\mathbf{V}$ .

The geometry of  $\mathbf{D}_{\mathbf{V}, \delta}$  is shown in Fig. 3 using  $\delta_E$  for  $\delta$  in (3). This 1-np design erects a linear boundary between the  $i$ th and  $j$ th classes, viz., the hyperplane HP through the midpoint of and perpendicular to  $(\mathbf{v}_i - \mathbf{v}_j)$ . Fig. 3 illustrates the labeling decision in (3);  $\mathbf{z}$  is assigned to class  $i$  because it is closest to the  $i$ th prototype. Be careful not to confuse 1-np's, which are new vectors made *from* the data, with 1-nn's, which are labeled points *in* the data. In Fig. 3, the prototype nearest to  $\mathbf{z}$  is  $\mathbf{v}_i$  and the neighbor nearest to  $\mathbf{z}$  is  $\mathbf{x}_{nn} \in X_i$ .

All 1-np designs that use inner-product norms erect (piecewise) linear decision boundaries. Thus, the *geometry* of 1-np classifier *boundaries* is fixed by the way distances are

<sup>1</sup>In [10] and [11], the input vectors are called "rabbits" and the prototypes trying to catch them are the "dogs"—hence, DR for dog-rabbit.

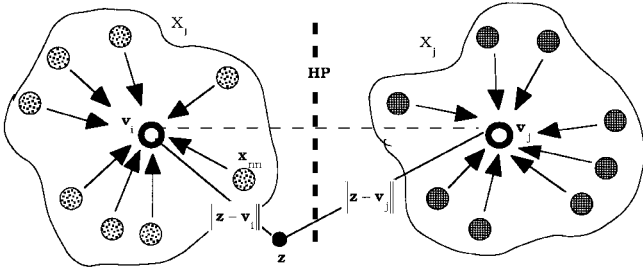


Fig. 3. Geometry of the 1-np classifier for the Euclidean norm.

measured in the feature space and *not* by geometric properties of the model that produces the cluster prototypes. The location in  $\mathbb{R}^p$  of the prototypes determines the location and orientation of the  $\hat{c}(\hat{c} - 1)/2$  hyperplanes that separate each pair of prototypes. The *geometry of the prototypes* does depend on both the clustering model and data used to produce them. Hence, 1-np designs based on different prototype generating schemes can certainly be expected to yield different performance as 1-np classifiers, even though they all share the same type of decision surface structure.

When one or more classes have multiple prototypes, as shown in Fig. 2, there are two ways to extend the 1-np design. We can simply use (3), recognizing that  $\mathbf{V}$  contains more than one prototype for at least one of the  $\hat{c}$  classes. Or we can extend the 1-np design to a  $k$ -np rule, wherein the  $k$  np's are used to conduct a vote about the label that should be assigned to input  $\mathbf{z}$ . This amounts to operating the  $k$ -nn rule by using prototypes (points built from the data) instead of neighbors (points in the data). We opt here for the simpler choice, which is formalized in the following.

*The Nearest Multiple-Prototype (1-nmp) Classifier:* Given any  $c$  prototypes  $\mathbf{V} = \{\mathbf{v}_{ij}: 1 \leq i \leq \hat{c}; 1 \leq j \leq n_{pi}\}$ , where  $n_{pi}$  is the number of prototypes for class  $i$ ;  $c = \sum_{j=1}^{\hat{c}} n_{pj}$ , and any dissimilarity measure  $\delta$  on  $\mathbb{R}^p$ , for any  $\mathbf{z} \in \mathbb{R}^p$

Decide  $\mathbf{z} \in \text{class } i \Leftrightarrow \mathbf{D}_{\mathbf{v}, \delta}(\mathbf{z}) = \mathbf{e}_i \Leftrightarrow \exists s \in \{1, \dots, n_{pi}\}$  such that

$$\delta(\mathbf{z}, \mathbf{v}_{is}) < \delta(\mathbf{z}, \mathbf{v}_{jt}), \quad \forall j \neq i \text{ and } t \in \{1, \dots, n_{pj}\}. \quad (3')$$

As in (3), ties in (3') are resolved arbitrarily. We use the same notation for the 1-np and 1-nmp classifiers, relying on context to identify which one is being discussed. Now we turn to methods for finding multiple prototypes.

### III. THREE COMPETITIVE LEARNING MODELS FOR MULTIPLE PROTOTYPES

When labeled data are crisply partitioned into  $\hat{c}$  subsets,  $X = X_1 \cup \dots \cup X_{\hat{c}}$  with  $X_i \cap X_j = \emptyset$  for all  $i \neq j$ , the natural choice for  $\mathbf{V}$  in the 1-np design is to compute the cluster or subsample mean vectors  $\bar{\mathbf{V}} = (\bar{\mathbf{v}}_1, \dots, \bar{\mathbf{v}}_{\hat{c}})$ , where  $\bar{\mathbf{v}}_i = \sum_{\mathbf{x} \in X_i} \mathbf{x} / |X_i|$ ,  $1 \leq i \leq \hat{c}$ .  $\mathbf{D}_{\bar{\mathbf{V}}, \delta_E}$  is the only single prototype classifier discussed here.

Now we seek multiple prototypes  $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_c\} \subset \mathbb{R}^p$ ,  $\hat{c} < c \ll n$  that are good representatives—for classifier

design—of the input data.<sup>2</sup> Sequential *competitive learning* (CL) models are a natural choice for finding multiple prototypes. Usually, upon presentation of an  $\mathbf{x}_k$  from  $X$ , updated estimates of the  $\{\mathbf{v}_i\}$  at iterate  $t$  (one iteration is one pass through  $X$ ) are computed as

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}), \quad i = 1, 2, \dots, c. \quad (4)$$

In (4),  $\{\alpha_{ik,t}\}$  is the *learning rate distribution* over the  $c$  prototypes for input  $\mathbf{x}_k$  at iterate  $t$ . The principle difference between various competitive learning models lies in 1) the subset of prototypes that get updated at each iterate and 2) the values of the  $\{\alpha_{ik,t}\}$ . It is always possible to define the  $\{\alpha_{ik,t}\}$  to include the update neighborhood, so this is usually what is specified. Unsupervised LVQ updates only the winner (i.e., the  $\mathbf{v}_i$  closest to  $\mathbf{x}_k$ ) at each input, whereas GLVQ-F and the DR algorithm may update all  $c$  prototypes for each presentation of an input. The learning rate distribution for LVQ is

$$\alpha_{ik,t}^{\text{LVQ}} = \begin{cases} \alpha_t; & i = \arg \min \{ \|\mathbf{x}_k - \mathbf{v}_{r,t-1}\| \} \\ 0; & r = 1, 2, \dots, c; r \neq i \end{cases}, \quad 1 \leq i \leq c. \quad (5)$$

In (5),  $\alpha_t$  is usually initialized at some value in (0, 1) and decreases with  $t$ . Kohonen [8] gives conditions under which LVQ terminates at a fixed point of the iterate sequence, defined via (4) and (5), that requires a nonlinear decrease in  $\alpha_t$  [e.g.,  $\alpha_t \propto (1/t)$ ]. Termination occurs without comparison of successive estimates of  $\mathbf{V}$ . Since we use  $\|\mathbf{V}_t - \mathbf{V}_{t-1}\|$  for termination control, we chose to decrease  $\alpha_t$  linearly with  $t$ .

The model underlying GLVQ-F contains LVQ as a subcase and is discussed extensively elsewhere [8]. GLVQ-F is based on minimizing a sum of squared errors associated with replacing unlabeled data set  $X_{\text{tr}}$  by the  $c$  prototypes  $\mathbf{V}$ . The function to be minimized is

$$\begin{aligned} L(\mathbf{x}_k; \mathbf{V}) &= \sum_{r=1}^c u_r \|\mathbf{x}_k - \mathbf{v}_r\|^2 \\ &= \sum_{r=1}^c \left\{ \sum_{j=1}^c \left[ \frac{\|\mathbf{x}_k - \mathbf{v}_r\|^{2/(m-1)}}{\|\mathbf{x}_k - \mathbf{v}_j\|^{2/(m-1)}} \right] \right\}^{-1} \\ &\quad \cdot \|\mathbf{x}_k - \mathbf{v}_r\|^2. \end{aligned} \quad (6)$$

In (6), the vector  $\mathbf{u} = (u_1, u_2, \dots, u_c)^T \in N_{fc}$  is a fuzzy label vector; its entries are the *memberships* of  $\mathbf{x}_k$  in each of the  $\hat{c}$  classes represented by the prototypes  $\mathbf{V}$ . The real number  $m > 1$  in (6) is a parameter that affects the quality of representation and speed of termination of the GLVQ-F algorithm, which is just steepest descent applied to the function in (6). The GLVQ-F update rule for the prototypes  $\mathbf{V}$  at iterate  $t$ , in the special (and simple) case  $m = 2$ , uses the following

<sup>2</sup>Good prototypes for classifier design are not necessarily the same (even in form) as those used for other purposes. For example, good prototypes for compression, transmission, and reconstitution of images may be quite poor as representatives of classes for the purpose of pixel labeling in the same image.

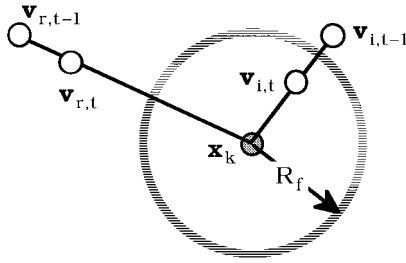


Fig. 4. Control of learning rates in the DR algorithm.

learning rate distribution in (4):

$$\alpha_{ik,t}^{\text{GLVQ-F}, m=2} = 2c\alpha_t \underbrace{\left[ \sum_{r=1}^c \left( \frac{\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|^2}{\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\|^2} \right) \right]^{-2}}_{u_{ik,t-1}^2}, \quad 1 \leq i \leq c. \quad (7)$$

$\alpha_t$  in (7)—now one component of the *learning rates*  $\{\alpha_{ik,t}\}$ —is treated the same way as in (5), and the constant  $(2c)$  is absorbed in it without loss. Limiting properties of GLVQ-F are [8] 1) as  $m$  approaches infinity, all  $c$  prototypes receive equal updates and the  $\mathbf{v}_i$ 's all converge to the grand mean of the data, whereas 2) as  $m$  approaches one from above, only the winner is updated and GLVQ-F reverts to LVQ. Finally, we mention that the winning prototype in GLVQ-F for  $m = 2$  receives the largest (fraction) of  $\alpha_{ik,t}$  at iterate  $t$ , that other prototypes receive a share that is *inversely* proportional to their distance from the input and that the GLVQ-F learning rates satisfy the additional constraint that  $\sum_{i=1}^c \alpha_{ik,t} \leq 1$  when  $m = 2$ .

The third sequential CL model used here is the deterministic DR algorithm [10], [11]. The basic idea for our implementation can be found in [10]; an alternate implementation is discussed in [11]. Like GLVQ-F, the DR algorithm may update all  $c$  prototypes for each input. Unlike GLVQ-F, the DR algorithm is not based on an optimization problem. Rather, its authors use intuitive arguments to establish the learning rate distribution for (4) that is used by the DR model, as shown in (8) at the bottom of the page.

In (8),  $\Lambda > 0$  is a user-specified constant that inhibits movement of the nonwinning prototypes toward  $\mathbf{x}_k$ ; and  $\{f_{ik,t} \geq 1\}$  is a user-specified distribution of *fatigue factors* for the DR algorithm. In our implementation, the fatigue factors are not necessarily updated at the same time across  $i$ , and may not be functions of iterate number  $t$ . Rather, control of these exponents depends on circumstances at individual prototypes. Fig. 4 illustrates how the learning rates are controlled.

The DR user must specify an initial distribution for the  $\{f_{ik,t} \geq 1\}$  and four constants: a *rate of change of fatigue factor*  $\Delta f > 0$ , a *maximum fatigue*  $f_M$ , a *fence radius*  $R_f > 0$ ,

and an *inhibition constant*  $\Lambda > 0$ . Now suppose  $\mathbf{v}_{i,t-1}$  to be the winning prototype with  $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\| > R_f$ , as shown in Fig. 4. All  $c$  prototypes are updated by using (8) in (4). Following this, the distance  $\|\mathbf{x}_k - \mathbf{v}_{i,t}\|$  is compared to  $R_f$ . If  $\|\mathbf{x}_k - \mathbf{v}_{i,t}\| < R_f$ , the closest dog is now inside the fence around  $\mathbf{x}_k$  and is slowed down by increasing its fatigue  $f_{ik,t} \leftarrow f_{ik,t-1} + \Delta f$ . This inhibits future motion of this prototype a little (relative to the other prototypes), and it also encourages nonwinners, such as  $\mathbf{v}_{r,t}$ , to look for other data to chase.

When the winning prototype gets very close to (a group of) inputs, we want it to stop moving altogether, so we also check the current value of  $f_{ik,t}$  against  $f_M$ . Movement of (i.e., updating) the  $i$ th prototype ceases when  $f_{ik,t} > f_M$ . Thus, termination of updating is done prototype by prototype, and DR stops when all of the prototypes are “close enough”—as measured by their rates of change of fatigue exceeding the maximum—to the subset of data for which they are the winner.

The dependency of  $\{\alpha_{ik,t}^{\text{DR}}\}$  on the parameters  $\{f_{ik,t}, \Delta f, f_M, R_f, \Lambda\}$  is complicated by the functional form of (8). However, we can say that these rates ensure that the winning prototype receives the largest (fraction) of  $\alpha_{ik,t}$  at iterate  $t$  until the winning prototype closes in on its rabbits. At this point, other prototypes may start receiving a larger fraction of the update even though they are nonwinners. The DR learning rates do not satisfy any additional constraints. A brief specification of LVQ, GLVQ-F for  $m = 2$ , and the DR algorithms, as used in our examples, are given in Table XI.

None of the CL methods just described uses the labels of points in  $X_{\text{tr}}$  during training to guide iterates toward a good  $\mathbf{V}$ . Consequently, at the end of the learning phase, the  $c$  prototypes have *algorithmic* labels that may or may not correspond to the *physical* labels of  $X_{\text{tr}}$ . The relabeling algorithm discussed next uses the labels in  $L_{\text{tr}}$  to attach the most likely (as measured by a percentage of labeled neighbors) physical label to each  $\mathbf{v}_i$ .

Recall that  $\hat{c}$  is the number of classes in  $X_{\text{tr}}$  labeled by the crisp vectors  $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{\hat{c}}\} = N_{h\hat{c}}$ . Now define  $p_{ij}$ ,  $i = 1, 2, \dots, \hat{c}$ ,  $j = 1, 2, \dots, c$  to be the percentage (as a decimal) of training data from class  $i$  closest to  $\mathbf{v}_j$  via the 1-np rule  $\mathbf{D}_{\mathbf{V}, \delta_E}$ . Define the matrix  $P = [p_{ij}]$ .  $P$  has  $\hat{c}$  rows in  $N_{fc}$  and  $c$  columns  $\mathbf{p}_j$  in  $N_{pc}$ . We assign label  $\mathbf{e}_i$  to  $\mathbf{v}_j$  when  $\mathbf{H}(\mathbf{p}_j) = \mathbf{e}_i$

$$\text{label } i \leftarrow \mathbf{v}_j \Leftrightarrow \mathbf{H}(\mathbf{p}_j) = \mathbf{e}_i, \quad i = 1, 2, \dots, \hat{c}, \quad j = 1, 2, \dots, c. \quad (9)$$

We illustrate the labeling algorithm in (9). Suppose  $X_{\text{tr}}$  has  $\hat{c} = 3$  classes labeled with the crisp vectors  $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\} = N_{h3}$ . Let  $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4)$  be four prototypes found by some algorithm. Let  $P$  be the  $3 \times 4$  percentage matrix shown in Table I. Labeling algorithm (9) assigns  $\mathbf{v}_1$  to class 1,  $\mathbf{v}_2$

$$\alpha_{ik,t}^{\text{DR}} = \left\{ \begin{array}{l} \left[ \frac{2\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|}{(1 + \|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|)^{f_{ik,t-1}}} \right], \quad i = \underbrace{\arg \min}_r \{\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\|\} \\ \left[ \frac{\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|}{(\Lambda + \|\mathbf{x}_k - \mathbf{v}_{i,t-1}\|)} \right] \left[ \frac{2\|\mathbf{x}_k - \mathbf{v}_{r,t-1}\|}{(1 + \|\mathbf{x}_k - \mathbf{v}_{r,t-1}\|)^{f_{rk,t-1}}} \right], \quad r = 1, 2, \dots, c; r \neq i \end{array} \right\} \quad (8)$$

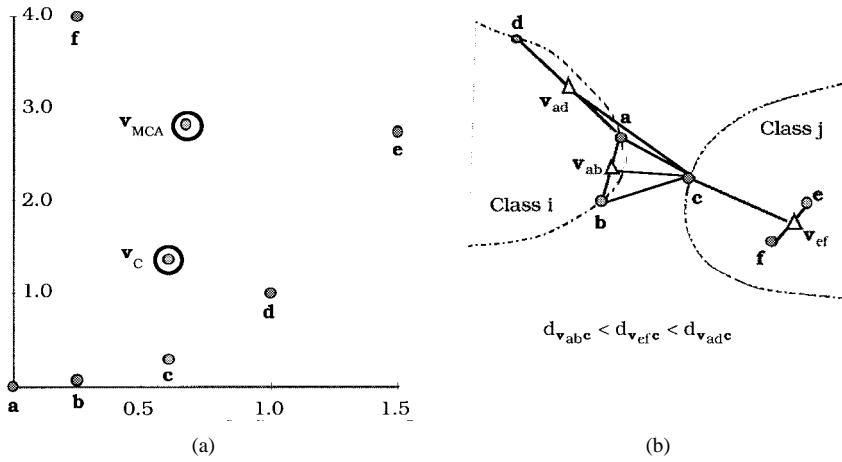


Fig. 5. (a) Effect of the merger formula and (b) illustration of both models.

TABLE I  
EXAMPLE OF THE MULTIPLE-PROTOTYPE LABELING ALGORITHM

	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$
$\mathbf{e}_1$	0.57	0.10	0.13	0.20
$\mathbf{e}_2$	0.15	0.10	0.15	0.60
$\mathbf{e}_3$	0.05	0.40	0.40	0.15

$\mathbf{H}(\mathbf{p}_1)=\mathbf{e}_1$     $\mathbf{H}(\mathbf{p}_2)=\mathbf{e}_3$     $\mathbf{H}(\mathbf{p}_3)=\mathbf{e}_3$     $\mathbf{H}(\mathbf{p}_4)=\mathbf{e}_2$

and  $\mathbf{v}_3$  to class 3, and  $\mathbf{v}_4$  to class 2. Whenever  $c > \hat{c}$ , we will have more than one prototype for at least one of the labeled classes and will use the 1-np rule at (3') instead of the 1-np rule at (3).

#### IV. THE CHANG AND MODIFIED CHANG ALGORITHMS (MCA'S)

Now we discuss three methods that are not based on sequential competitive learning: Chang's algorithm [12], an improved version of it, and a batch method due to Yen and Chang [13].

Chang [13] discussed one of the earliest multiple-prototype classifier schemes. The method begins by assuming every point in a labeled data set  $X$  is its own prototype; so let  $\mathbf{V}_n = X$ . Consequently, the 1-np rule at (3) or the 1-nmp rule at (3') error rate is zero,  $E_{\mathbf{D}_{\mathbf{V}_n}, \delta_E}(X|\mathbf{V}_n) = 0$ . Now find  $(i, j) = \arg \min_{s \neq t; \mathbf{x}_s, \mathbf{x}_t \in \mathbf{V}_n} \{\|\mathbf{x}_s - \mathbf{x}_t\|\}$ . Tentatively merge these two points

by using the weighted mean  $\mathbf{v}_{ij} = (M\mathbf{x}_i + N\mathbf{x}_j)/(M + N)$ , where  $M, N$  are the number of merger parents of  $\mathbf{x}_i$  and  $\mathbf{x}_j$ , respectively. Initially,  $M$  and  $N$  have the value one. When two data points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  are merged,  $\mathbf{v}_{ij} = (\mathbf{x}_i + \mathbf{x}_j)/2$  and  $\mathbf{v}_{ij}$  has two merger parents. Subsequently, if  $\mathbf{v}_{ij}$  and  $\mathbf{x}_k$  are merged, then  $M = 2, N = 1, \mathbf{v}_{ijk} = (2\mathbf{v}_{ij} + \mathbf{x}_k)/3$ , etc.

Next, update the prototypes by setting  $\mathbf{V}_{n-1} \leftarrow X - \{\mathbf{x}_i, \mathbf{x}_j\} + \mathbf{v}_{ij}$ , and calculate  $E_{\mathbf{D}_{\mathbf{V}_{n-1}}, \delta_E}(X|\mathbf{V}_{n-1})$  by using the 1-nmp rule at (3'). If the error rate is still zero and if  $\mathbf{x}_i$  and  $\mathbf{x}_j$  have the same label, accept the merger and continue. If either 1) the error rate increases or 2)  $\mathbf{x}_i$  and  $\mathbf{x}_j$  have different labels, do not merge  $\mathbf{x}_i$  and  $\mathbf{x}_j$ . In this case, Chang regards  $\mathbf{x}_i$  and  $\mathbf{x}_j$  as currently nonmergeable prototypes and continues.

When there is a merger, the child inherits the class label of its parents and it replaces them in the current prototype set. Especially important is that the *test* data are *fixed* (all of  $X$ ). Continue this procedure until further merging produces an error, and at this point stop, having found  $c$  prototypes  $\mathbf{V}_c$  that replace the  $n$  labeled data  $X$  and that preserve a resubstitution error rate of zero, i.e.,  $E_{\mathbf{D}_{\mathbf{V}_c}, \delta_E}(X|\mathbf{V}_c) = 0$ . An implementation of this scheme, based on minimal spanning trees is given by Chang.

We modified Chang's approach, here called the MCA, in two ways. First, instead of using the weighted mean  $\mathbf{v}_{ij} = (M\mathbf{x}_i + N\mathbf{x}_j)/(M + N)$  to merge prototypes we used the simple arithmetic mean [see Fig. 5(a)]. Second, we altered the search for candidates to merge by partitioning the distance matrix into  $c$  submatrices blocked by common labels, and we always looked for the minimum only in each block [see Fig. 5(b)]. This eliminates consideration of candidate pairs with different labels. Like Chang, we attempt to merge the minimum of label-matched pairs. If this fails (because the prototype produced by the merger yields an error), we look at the next best candidates, and so on. Search continues in ascending order of distance until either 1) a merger can be done or 2) no merger is possible in any class. The MCA algorithm terminates when 2) occurs. This is more effective than Chang's approach because merging the closest points of the same label may be sufficient, but is not necessary, to preserve the zero error rate.

Fig. 5 illustrates the operation of and highlights differences between the Chang and MCA models. Using the number of merger parents as weights guarantees that Chang's prototypes are either isolated singletons or true centroids of the points they represent. Using simple averaging instead results in the points that are not close to each other drawing MCA prototypes *away* from points that are close to each other. Fig. 5(a) shows the effect of changing the merger formula from Chang's weighted mean to the simple arithmetic mean.

In Fig. 5(a), the six points labeled  $\mathbf{a}$  to  $\mathbf{f}$  are increasingly distant from each other as  $\mathbf{a} \rightarrow \mathbf{f}$ . The centroid of  $\mathbf{a}$  to  $\mathbf{f}$  in Fig. 5(a) is Chang's prototype  $\mathbf{v}_c = (\mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d} + \mathbf{e} + \mathbf{f})/6$ . Each point has equal weight, regardless of the relative distances between pairs of points. For the MCA prototype,

$\mathbf{v}_{MCA} = \mathbf{a}/32 + \mathbf{b}/32 + \mathbf{c}/16 + \mathbf{d}/8 + \mathbf{e}/4 + \mathbf{f}/2$ . The points closest together are merged first (in this case,  $\mathbf{a}$  and  $\mathbf{b}$  by both methods; but in MCA, these points have the *least* overall effect on the final position of the prototype). This enables MCA prototypes to (have a better chance) to correctly label points in their own classes that are *NOT* located near the central tendency of the clusters.

Fig. 5(b) illustrates the Chang and MCA approaches. Let  $d_{ab}$  denote the (Euclidean) distance between  $\mathbf{a}$  and  $\mathbf{b}$  and likewise for other distances in the diagram. Suppose the closest existing class  $j$  prototype in the neighborhood is shown to be  $\mathbf{v}_{ef}$  and that  $d_{ab}$  is the minimum distance in the current node set. Both Chang and MCA will consider replacing  $\mathbf{a}$  and  $\mathbf{b}$  with the class  $i$  prototype  $\mathbf{v}_{ab}$ , and neither will do so. Chang's method applies two tests: are the labels of  $\mathbf{a}$  and  $\mathbf{b}$  the same? (yes) and will this prototype cause an error for the new 1-np classifier? (yes). Since the distance from  $\mathbf{v}_{ab}$  to  $\mathbf{c}$  is smaller than the distance from  $\mathbf{v}_{ef}$  to  $\mathbf{c}$ , the point  $\mathbf{v}_{ab}$  will misclassify  $\mathbf{c}$ , so the proposed merger is rejected by both algorithms.

At this point, Chang's method will next inspect possible mergers between  $\mathbf{a}$  and  $\mathbf{c}$  and then  $\mathbf{b}$  and  $\mathbf{c}$ ; both will be rejected, due to a failure to have the same class labels. MCA skips these calculations, since the minimum distance search is constrained to submatrices for the same class within the current distance matrix. Then, both algorithms will consider merging  $\mathbf{a}$  and  $\mathbf{d}$ , and both will do this.

In the terminology of Section I, the Chang and MCA models are both consistent designs—that is, they are defined to maintain zero resubstitution errors. Both of these designs are replacement classifiers in the sense of Fig. 2. In the next section, we compare these two designs to the standard 1-nn rule.

## V. NUMERICAL COMPARISON OF THREE CONSISTENT DESIGNS

The standard 1-nn rule classifier is specified as follows.

*The 1-nn Classifier:* Given a crisply labeled data set in  $\mathcal{R}^p$ ,  $X = X_1 \cup \dots \cup X_c$ ;  $X_i \cap X_j = \emptyset$  for  $i \neq j$ , and any dissimilarity measure  $\delta$  on  $\mathcal{R}^p$ , for any  $\mathbf{z} \in \mathcal{R}^p$

$$\text{Decide } \mathbf{z} \in \text{class } i \Leftrightarrow \mathbf{D}_{1\text{-nn}, \delta}(\mathbf{z}) = \mathbf{e}_i \Leftrightarrow \mathbf{x}_s \in X_i$$

and

$$\delta(\mathbf{z}, \mathbf{x}_s) < \delta(\mathbf{z}, \mathbf{x}_s) < \delta(\mathbf{z}, \mathbf{x}_j), \quad \forall j \neq s. \quad (10)$$

Ties in (10) are arbitrarily resolved. Dasarathy [17] recently discussed a method for selecting a consistent subset from  $X$  for use with the 1-nn rule. His method is based on finding nearest *unlike* neighbor subsets in  $X$ . Dasarathy calls his scheme the MCS method, in which MCS stands for *minimal consistent set*. Although the term minimal implies that MCS finds the smallest consistent subset of  $X$  for the 1-nn rule, Dasarathy admits the possibility that it is not, since no proof of minimality is given.

The important point here is that MCS is to Fig. 1 as the Chang and MCA methods are to Fig. 2: the former is a selection method, while the latter are replacement methods. All three methods use the labels during training, and they all guarantee consistency (zero resubstitution error rate). Thus, the results of Dasarathy are very useful for comparing selection to

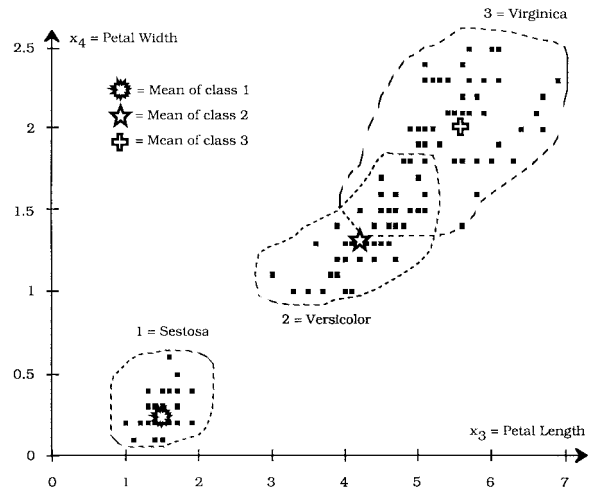


Fig. 6. Iris data: feature three versus feature four.

TABLE II  
SUBSAMPLE (MEAN) PROTOTYPES  $\bar{\mathbf{v}}$  IN  $\mathcal{R}^4$  FOR IRIS

Symbol	Name	$x_1$	$x_2$	$x_3$	$x_4$
	$\bar{\mathbf{v}}_1$	5.01	3.43	1.46	0.25
	$\bar{\mathbf{v}}_2$	5.94	2.77	4.26	1.33
	$\bar{\mathbf{v}}_3$	6.59	2.97	5.55	2.03

replacement or, equivalently, comparing an edited 1-nn design to the 1-nnp designs based on the Chang and MCA models.

Following Chang [12] and Dasarathy [17], we use Anderson's Iris data [16] as the experimental data set. In the examples, we take  $\text{Iris} = X = X_{\text{tr}} = X_{\text{te}}$ . Iris contains 50 (physically labeled) vectors in  $\mathcal{R}^4$  for each of  $\hat{c} = 3$  classes of Iris subspecies. Fig. 6 is a scatterplot of the third and fourth features of Iris, which also shows the subsample mean for each of the three classes in these two dimensions. Table II lists the coordinates of the means. Class 1 is well separated from classes 2 and 3 in these two dimensions; classes 2 and 3 show some overlap in the central area of the figure, and this region contains the vectors that are usually mislabeled by 1-np designs. The dashed boundaries indicate the physically labeled clusters.

Chang reports that his method finds  $c = 14$  prototypes that replace Iris and preserve a zero resubstitution error rate; the prototypes are not listed in [12]. Dasarathy reports in [17] that MCS finds 15 points in the Iris data that result in zero resubstitution errors. If MCS is truly minimal, this suggests that Chang's replacement method is, for Iris at least, slightly superior to MCS subset selection. The 15 points in Iris found by MCS were not listed in [17]. Our modification of Chang's method finds 11 prototypes built from Iris that yield consistency for the 1-nnp design with which they were used. The 11 prototypes are listed in Table III. This confirms that our modifications of Chang's method serve as advertised; that is, MCA does reduce the number of nearest multiple prototypes while preserving consistency.

Our implementation of MCA can probably be made more efficient, and a different merger scheme might reduce the number of prototypes needed for consistency even more.

TABLE III  
11 MCA PROTOTYPES THAT YIELD ZERO RESUBSTITUTION ERRORS WITH THE 1-NMP RULE ON IRIS

Class 1				Class 2				Class 3			
4.94	3.36	1.38	0.28	5.69	2.33	3.88	1.16	4.90	2.50	4.50	1.70
				6.70	2.98	4.82	1.56	6.29	2.71	5.01	1.68
				5.86	3.08	4.55	1.54	5.83	2.86	4.98	1.88
				6.15	2.60	5.00	1.55	6.05	2.40	5.30	1.45
								6.19	3.00	5.33	2.31
								7.02	3.09	6.14	2.18

Subsequent work [18] has shown that the MCS method is not minimal, and this example shows that replacement can, for the methods discussed, be more efficient than selection.

## VI. NUMERICAL COMPARISON OF FIVE INCONSISTENT DESIGNS

How do we use 1-np and 1-nmp classifiers to compare unsupervised learning algorithms? The method employed here is to first derive the prototypes  $\mathbf{V}$  from labeled data  $X$  without using the labels (that is, we pretend there are no labels) during the training phase. Then, (9) is used to get class labels for the prototypes. Finally,  $X$  is submitted to the classifier and its resubstitution error rate is computed. Error counts are conveniently tabulated by using the  $c \times c$  confusion matrix  $C = [c_{ij}] = [\# \text{ labeled class } j \text{ but were really class } i]$  that can be constructed during this process. The error rate (in percent) is

$$\begin{aligned} \mathbf{E}_{\mathbf{D}_{\mathbf{V}, \delta}}(X_{\text{te}}|X_{\text{tr}}) &= 100 * \left[ 1 - \left( \frac{\# \text{ right}}{|X_{\text{te}}|} \right) \right] \\ &= 100 * \left[ 1 - \left( \frac{\text{tr}(C)}{|X_{\text{te}}|} \right) \right]. \end{aligned} \quad (11)$$

For reference, the resubstitution error rate for the supervised 1-np design that uses the class means of each subset of Iris in (3), listed in Table II and plotted in Fig. 6 as single prototypes, is 11 errors in 150 submissions by using the Euclidean norm, i.e.,  $\mathbf{E}_{\mathbf{D}_{\mathbf{V}, \delta E}}(\text{Iris}|\text{Iris}) = 7.33\%$ . Next, we discuss the computational protocols used by the three CL methods outlined in Section III.

### A. Initialization

The following method was used to generate an initial set of prototypes  $\mathbf{V}_0$ :

$$\text{Minimum of feature } j: m_j = \underbrace{\min\{x_{jk}\}}_k: j = 1, 2, \dots, p \quad (12)$$

and

$$\text{Maximum of feature } j: M_j = \underbrace{\max\{x_{jk}\}}_k: j = 1, 2, \dots, p. \quad (13)$$

The Cartesian product  $hb(\mathbf{m}, \mathbf{M}) = [m_1, M_1] \times \dots \times [m_p, M_p]$  is a *hyperbox* in  $\mathbb{R}^p$ . The main diagonal of  $hb(\mathbf{m}, \mathbf{M})$  connects  $\mathbf{m}$  and  $\mathbf{M}$  with the line segment  $\{\mathbf{m} + \alpha(\mathbf{M} - \mathbf{m}); 0 \leq \alpha \leq 1\}$ . Initial prototypes for all

TABLE IV  
INITIAL PROTOTYPES FOR IRIS AT  $c = 6$  COMPUTED WITH (14)

$\mathbf{v}_{1,0} = (4.30 \ 2.00 \ 1.00 \ 0.10) = \mathbf{m}$
$\mathbf{v}_{2,0} = (5.02 \ 2.48 \ 2.18 \ 0.58)$
$\mathbf{v}_{3,0} = (5.74 \ 2.96 \ 3.36 \ 1.06)$
$\mathbf{v}_{4,0} = (6.46 \ 3.44 \ 4.54 \ 1.54)$
$\mathbf{v}_{5,0} = (7.18 \ 3.92 \ 5.72 \ 2.02)$
$\mathbf{v}_{6,0} = (7.90 \ 4.40 \ 6.90 \ 2.50) = \mathbf{M}$

three algorithms were

$$\mathbf{v}_{i,0} = \mathbf{m} + \left( \frac{i-1}{c-1} \right) (\mathbf{M} - \mathbf{m}), \quad i = 1, 2, \dots, c. \quad (14)$$

Thus,  $\mathbf{v}_{1,0} = \mathbf{m} = (m_1, m_2, \dots, m_p)^T$ ;  $\mathbf{v}_{c,0} = \mathbf{M} = (M_1, M_2, \dots, M_p)^T$ ; and the remaining  $(c - 2)$  initial prototypes are uniformly distributed along the diagonal of  $hb(\mathbf{m}, \mathbf{M})$ . To illustrate, Table IV shows the initial prototypes produced by (14) with the Iris data at  $c = 6$ . Algorithmic outputs reported for other values of  $c$  were obtained from similar initializations by using (14). The control parameters of each CL algorithm that we used are listed in Table XI; some experimentation with them is discussed at the end of this section.

### B. Termination

The primary termination criterion is to compare successive estimates of the prototypes with the 1-norm.  $\|\mathbf{V}_t - \mathbf{V}_{t-1}\|_1 = \sum_{r=1}^c \|\mathbf{v}_{r,t} - \mathbf{v}_{r,t-1}\|_1 = \sum_{j=1}^p \sum_{r=1}^c |v_{jr,t} - v_{jr,t-1}|$  is compared to cutoff threshold  $\varepsilon$ . If this fails, secondary termination occurs at the iterate limit  $T$  specified in Table XI. We tested three thresholds:  $\varepsilon = 0.1, 0.01$ , and  $0.001$ . The DR algorithm has a third termination criterion (the prototype-by-prototype cutoff) that can (and often does) occur inside the main iteration, as shown in Table XI.

### C. Iteration

We drew samples randomly from  $X$  without replacement. One iteration corresponds to one pass through  $X$ . Each algorithm was run five times for each case discussed to see how different input sequences affected the terminal prototypes. For the less stringent termination criteria ( $\varepsilon = 0.1$  and  $0.01$ ), we sometimes obtained different terminal prototypes for different runs. For  $\varepsilon = 0.001$ , this effect was nearly (but not always) eliminated. Most of the runs using  $\varepsilon = 0.001$  were completed in less than 300 passes through  $X$ . DR, via its prototype-by-prototype criterion, often terminated in less than 50 iterations.

TABLE V  
TYPICAL PROTOTYPES, CONFUSION MATRICES, AND ERROR RATES FOR SIX PROTOTYPES.  $\alpha_0 = 0.4$  AND  
 $T = 500$  FOR LVQ AND GLVQ-F; DR PARAMETERS  $\{\{f_{ik,t}\}, \Delta f, f_M, R_f, \Lambda\}$  IN TABLE XI

LVQ Labels	LVQ prototypes	GLVQ-F Labels	GLVQ-F (m=2) prototypes	DR Labels	DR prototypes
1	4.69 3.12 1.39 0.20	1	4.75 3.15 1.43 0.20	1	5.08 3.45 1.44 0.22
1	5.23 3.65 1.50 0.28	1	5.24 3.69 1.50 0.27	2	5.78 2.63 3.99 1.20
2	5.52 2.61 3.90 1.20	2	5.60 2.65 4.04 1.24	2	6.61 3.00 4.45 1.39
2	6.21 2.84 4.75 1.57	2	6.18 2.87 4.73 1.56	2	6.09 2.98 4.60 1.40
3	6.53 3.06 5.49 2.18	3	6.54 3.05 5.47 2.11	3	6.06 2.83 4.95 1.78
3	7.47 3.12 6.31 2.02	3	7.44 3.07 6.27 2.05	3	6.74 3.12 5.60 2.24
	$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 14 & 36 \end{pmatrix}$		$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 50 & 0 \\ 0 & 14 & 36 \end{pmatrix}$		$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$
	Error rate = 9.33 %		Error rate = 9.33 %		Error rate = 2.66 %

TABLE VI  
TYPICAL PROTOTYPES, CONFUSION MATRICES, AND ERROR RATES FOR SEVEN PROTOTYPES.  $\alpha_0 = 0.4$   
AND  $T = 500$  FOR LVQ AND GLVQ-F; DR PARAMETERS  $\{\{f_{ik,t}\}, \Delta f, f_M, R_f, \Lambda\}$  IN TABLE XI

LVQ Labels	LVQ prototypes	GLVQ-F Labels	GLVQ-F (m=2) prototypes	DR Labels	DR prototypes
1	4.68 3.11 1.39 0.20	1	4.74 3.15 1.43 0.20	1	5.06 3.42 1.45 0.21
1	5.23 3.65 1.50 0.28	1	5.24 3.69 1.50 0.27	2	5.58 2.49 3.89 1.10
2	5.53 2.62 3.93 1.21	2	5.57 2.61 3.96 1.21	2	5.69 2.88 4.18 1.29
2	6.42 2.89 4.59 1.43	2	6.26 2.92 4.54 1.43	2	6.09 2.95 4.63 1.40
3	6.57 3.09 5.52 2.18	3	6.62 3.09 5.56 2.16	2	6.64 3.00 4.56 1.41
3	7.47 3.12 6.31 2.02	3	7.50 3.05 6.35 2.06	3	6.11 2.82 4.91 1.78
3	5.99 2.75 5.02 1.79	3	6.04 2.79 4.95 1.76	3	6.72 3.10 5.57 2.22
	$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$		$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 46 & 4 \\ 0 & 1 & 49 \end{pmatrix}$		$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$
	Error rate = 2.66 %		Error rate = 3.33 %		Error rate = 2.66 %

TABLE VII  
TYPICAL CONFUSION MATRICES AND CLASS REPRESENTATIVES FOR EIGHT TERMINAL PROTOTYPES

c = 8 prototypes		
LVQ	GLVQ-F	DR
$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$	$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$	$C = \begin{pmatrix} 50 & 0 & 0 \\ 0 & 47 & 3 \\ 0 & 1 & 49 \end{pmatrix}$
E = 2.66 %	E = 2.66 %	E = 2.66 %
Class 1 : 2	Class 1 : 2	Class 1 : 1
Class 2 : 3	Class 2 : 3	Class 2 : 4
Class 3 : 3	Class 3 : 3	Class 3 : 3

#### D. Results

The results shown in Tables V–VII are typical cases; those in Table VIII are the best case we saw in each instance. Our main objective is to compare the methods rather than obtain an optimal design for Iris. Indeed, it may be that with enough experimentation, any of the CL models will yield best-case (or better!) results.

Table V exhibits terminal prototypes found by each algorithm at  $c = 6$  as well as the resultant 1-nmp error rates they

produce when used in (3') on all of IRIS. Each of the three physical clusters is represented by two prototypes by both LVQ and GLVQ-F, and the overall error rate produced by these two classifiers is 9.33%. The DR model performs much better, finding six prototypes that produce only four errors when used with (3'). Note especially that DR uses only one prototype for class 1, three for class 2, and two for class 3.

The prototypes in Table V are plotted in Fig. 7 against a background created by roughly estimating the convex hull of each physical class in these two dimensions by eye. Some of the prototypes are hard to see because their coordinates are very close in these two dimensions. We draw attention to the LVQ and GLVQ-F prototypes that seem to lie on the boundary between classes 2 and 3 by enclosing these points with a jagged star. These prototypes are the ones that incur most of the misclassifications that are committed by the LVQ and GLVQ-F 1-nmp classifiers. Notice that there is no DR prototype in this region at  $c = 6$ . Instead, DR opts for only one class 1 prototype, thereby enabling it to better represent the boundary region between classes 2 and 3 with the prototypes



TABLE VIII  
NUMBER OF RESUBSTITUTION ERRORS OF THE  
1-nmp CL DESIGNS: BEST-CASE RESULTS

$c \rightarrow$	3	4	5	6	7	8	9	15	30
LVQ	17	24	14	14	3	4	4	4	4
GLVQ-F	16	20	19	14	5	3	4	4	4
DR	10	13	3	3	3	4	3	6	3

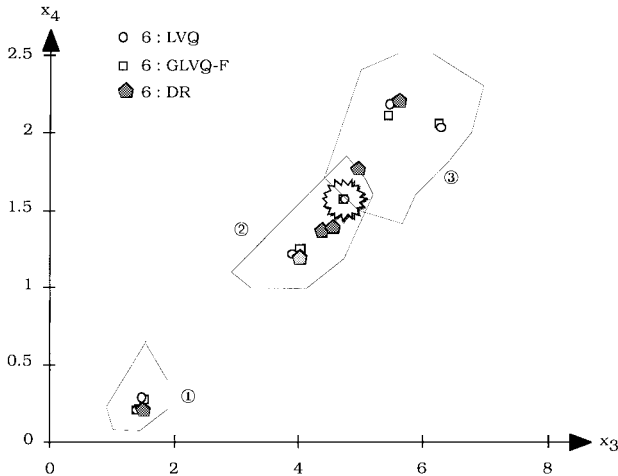


Fig. 7. Terminal prototypes at  $c = 6$ .

shown for it. This is a real difference between and decided advantage for the DR model compared to the two LVQ designs.

Table VI lists the same information as Table V for  $c = 7$ . There is a sharp drop in the error rate for the LVQ and GLVQ-F 1-nmp designs. Be careful to note that the seventh prototype is not “added” to the previous six; rather, new prototypes are found by each algorithm. The error rates in Table VI are very low for designs that are not based on using the labels during training. Note that LVQ and GLVQ-F continue to use two prototypes for each of classes 1 and 2, and add a third representative for class 3 at  $c = 7$ . Contrast this to DR, which still has one for class 1, four for class 2, and two for class 3 prototypes. Adding a seventh prototype does not improve the DR 1-nmp design because two of the seven prototypes are almost identical to one used at  $c = 6$ .

Fig. 8 shows that the crucial “boundary” prototypes from LVQ and GLVQ-F in the  $c = 6$  case have roughly “divided” into two sets of new prototypes, shown again by the jagged star. LVQ and GLVQ-F essentially “catch up” with DR in the region of overlap by now representing class 3 with three prototypes instead of two.

When the three CL algorithms are instructed to seek  $c = 8$  prototypes, the error rate for all three 1-nmp designs typically remains at 2.66%, as shown in Table VII. At  $c = 9$ , the results are quite similar to those shown for  $c = 8$ .

We can conclude from Tables V–VII that the replacement of IRIS with eight or nine prototypes found by any of the three CL algorithms results in a 1-nmp design that is quite superior to the labeled 1-np design based on the  $c = 3$  subsample means. Moreover, the DR model yielded consistently better results than either LVQ or GLVQ-F in almost every case we tested.

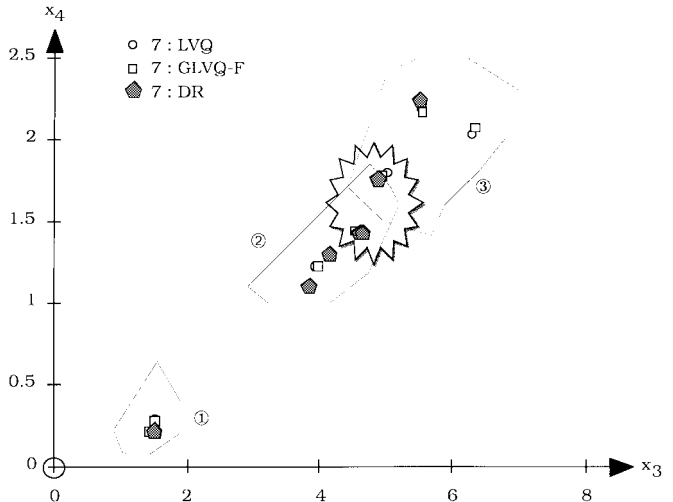


Fig. 8. Terminal prototypes for  $c = 7$ .

The experiments discussed so far led us to wonder how few prototypes were needed by the 1-nmp rule to achieve good results. And conversely, going in the other direction, at what point does prototype representation become counterproductive? Table VIII reports the best-case results (as number of resubstitution errors) we saw by using each algorithm for various values of  $c$ .

First, we can observe that on passing from  $c = 3$  to  $c = 4$ , even the best-case error rate for all three models increased, followed by a decrease on passing from  $c = 4$  to  $c = 5$ . One run of DR (shaded in Table VIII) resulted in five prototypes that produced only three resubstitution errors when used in (3'). These prototypes are shown in Table IX. This shows that the Iris data can be well represented by five labeled prototypes.

At the other extreme, increasing  $c$  above  $c = 7$  or 8 has little effect on the best-case results. Taken together, these observations suggest that Iris (and more generally, any labeled data set) has some upper and lower bounds in terms of high-quality representation by multiple prototypes for classifier design. There seems to be little hope, however, of discovering this on a better than case-by-case basis.

It is also clear from Table VIII that the DR model provides the best results for every value of  $c$ . We conjecture that the reason for this is that the control structure for this model is fundamentally very different from both LVQ and GLVQ-F. It seems that DR rapidly closes in on a single prototype for class 1 (which, for Iris, is really all that is needed), terminates updates for this prototype, and by its increased fatigue factors encourages the remaining prototypes to seek other data to represent, which they do. It would be a mistake to generalize this belief to other data without much more computational evidence; however, we believe that when a small number of errors can be tolerated in exchange for a small number of multiple prototypes, the DR algorithm will prove to be superior to both LVQ and GLVQ-F.

### E. Robustness

Finally, we comment on the sensitivity of each CL model to changes in its control parameters. We did not experiment with

TABLE IX  
FIVE DR PROTOTYPES THAT YIELD THREE RESUBSTITUTION ERRORS WITH THE 1-nmp RULE ON IRIS

Class 1				Class 2				Class 3			
5.03	3.38	1.50	0.31	5.59	2.70	4.02	1.28	6.12	2.88	5.06	1.82
				6.45	2.88	4.60	1.46	6.95	3.05	5.83	2.12

TABLE X  
SUMMARY OF THE BEST ERROR RATES ACHIEVED BY THE EIGHT METHODS

Algorithm	Classifier Type	Min. errors in 150 tries	# of points used by the classifier
<i>Consistent designs</i>			
MCS	1-nn	0	15
Chang	1-nmp	0	14
MCA	1-nmp	0	11
<i>Inconsistent designs</i>			
Labeled means ( $\bar{V}$ )	1-np	11	3
MFCM-3	1-nmp	8	7
LVQ	1-nmp	3	7
GLVQ-F ( $m=2$ )	1-nmp	3	8
DR	1-nmp	3	5

changes in  $m$  for GLVQ-F. Certainly this parameter affects terminal prototypes; however, we doubt that this will cause radical changes in the results given above. We varied  $\alpha_0$  from 0.4 to 0.6 in both LVQ and GLVQ-F without noticeable changes in typical results. The DR algorithm has more parameters to vary, and we spent a little time experimenting with them before settling on the values listed in Table XI. For example, we made runs of DR with  $\Lambda$  ranging from 1 to 9 and found little difference in the average case outputs at every value of  $c$  shown in Table VIII. In particular, the average minimum number of DR errors occurred at  $c = 6$  prototypes for  $\Lambda = 1, 5,$  and  $9$ . All three CL algorithms are sensitive—but not alarmingly so—to changes in their control parameters. Usually—but not always—when the number of errors was the same for competing designs, the vectors that were misclassified were also identical.

The last method we discuss is due to Yen and Chang [13], who modified the (batch) fuzzy  $c$ -means algorithm so that it can be used to produce multiple prototypes for each class by an algorithm they called MFCM- $n$ ,  $n = 1, 2,$  and  $3$ . The theory of their method is well discussed elsewhere, so we are content here to show their results on Iris. Specifically, Yen and Chang compare four outputs: FCM,  $c = 3$ , 16 errors; MFCM-1,  $c = 3$ , 16 errors; MFCM-2,  $c = 5$  with (1, 2, 2) labeled prototypes for classes (1, 2, 3), 14 errors; and their best result, MFCM-3,  $c = 7$ , with (1, 3, 3) labeled prototypes for classes (1, 2, 3), 8 errors. For convenience, we refer to MFCM as a CL model in Table X (in fact, it is a CL model, but it is batch, so the competition is global, not local).

## VII. CONCLUSIONS AND DISCUSSION

Table X summarizes the best results achieved by the eight algorithms discussed in our study. What does Table X entitle us to conclude? First, our results are of course specialized to just one data set, and generalizations to other data warrant caution.

### A. Conclusions

The most effective classifier for a consistent design is MCA, our modification of Chang's algorithm. If zero resubstitution

error is a desirable criterion, it appears that MCA is a better choice than Chang's method, at least for the Iris data. Whether MCA is reliably better than either MCS or Chang's method requires an in-depth study with many data sets. We suspect that data exist for which each of these methods produces a consistent classifier with the minimum number of selected or replacement points. All six 1-nmp designs use the labeled data more effectively in the sense of a smaller number of resubstitution errors than the 1-np design, based on the labeled sample means. This indicates that better classifier performance is certainly possible and very likely by using multiple prototypes. Our tests at low and high values for  $c$  suggest that there is probably an optimal range for the number of prototypes that should be used to replace labeled training data. Since a theoretical derivation of the best number of prototypes seems optimistic, it is probably the case that this must be discovered by the trial-and-error process illustrated by Table VIII. None of the CL models produced a consistent design; the best 1-nmp CL model (DR) did realize the minimum number of errors with the minimum number of prototypes.

### B. Discussion

If the determining criterion for choosing multiple prototypes is *consistency*, then MCA might be the method of choice. On the other hand, we can imagine applications (image compression comes to mind) where it is very important to find a (possibly inconsistent) design that offers the *minimum number of prototypes*. If this is important enough, developers may be willing to sacrifice a little accuracy to achieve this objective. In this case, the DR algorithm seems well suited to finding multiple prototypes that yield a few errors with fewer prototypes than the other multiple-prototype methods. The tradeoff is clear: no errors with more prototypes versus some errors with fewer prototypes.

We did not study the question of how to find an optimal path for continuing our modification of Chang's method to carry it beyond zero errors, but the DR results (three errors with five prototypes) suggest that paths exist that might bring us close to this. Specifically, some sort of partial minimal-spanning-tree technique might be used to extend our modification so that the best path beyond no errors could be found for one error, two errors, etc. A solution to this problem would provide users a way to find some (not necessarily) minimal number of prototypes that guaranteed any prespecified resubstitution error rate that the data would support. Perhaps a hybrid approach can be developed to do this. For example, the DR algorithm could be used to find multiple prototypes that yield some resubstitution error rate greater than zero. And then, MCA could be applied to the DR prototypes in the hope of reducing their number, while not increasing the current error rate. We think this would make an excellent follow-up study to this report.

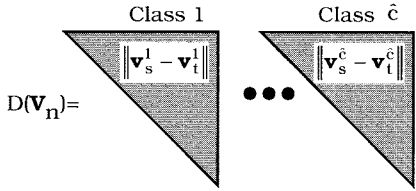
TABLE XI  
LVQ, GLVQ-F ( $m = 2$ ) AND DR ALGORITHMS

Store	<ul style="list-style-type: none"> <li>☛ Object Data <math>\mathbf{X} = \mathbf{X}_{tr} \in \mathfrak{R}^P</math> and crisp label matrix <math>L_{tr}</math> of <math>\mathbf{X}_{tr}</math></li> </ul>
Pick	<ul style="list-style-type: none"> <li>☛ Euclidean norm for similarity of data to prototypes :  <math>\delta_E(\mathbf{x} - \mathbf{v}) = \ \mathbf{x} - \mathbf{v}\ _1 = \sqrt{(\mathbf{x} - \mathbf{v})^T (\mathbf{x} - \mathbf{v})}</math></li> <li>☛ number of prototypes : <math>3 \leq c \leq 30</math></li> <li>☛ maximum number of iterations : <math>T = 1000</math></li> <li>☛ termination criterion : <math>\varepsilon = 0.1, 0.01</math> and <math>0.001</math></li> <li>☛ norm for termination : <math>\ \mathbf{V}_t - \mathbf{V}_{t-1}\ _{err} = \sum_{r=1}^c \ \mathbf{v}_{r,t} - \mathbf{v}_{r,t-1}\ _1 = \sum_{j=1}^p \sum_{r=1}^c  v_{jr,t} - v_{jr,t-1} </math></li> <li>☛ initial learning rate : <math>\alpha_0 = 0.6</math> (LVQ and GLVQ-F only)</li> <li>☛ weighting exponent : <math>m = 2</math> (GLVQ-F only)</li> <li>☛ fatigue distribution: <math>\{f_{ik,0} = 1 ; 1 \leq i \leq c\}</math> (DR only)</li> <li>☛ ROC of <math>\{f_{ik,t}\}</math> : <math>\Delta f = 0.1</math> (DR only)</li> <li>☛ maximum fatigue: <math>f_M = 5</math> (DR only)</li> <li>☛ fence radius : <math>R_f = 0.2</math> (DR only)</li> <li>☛ inhibition factor: <math>\Lambda = 5</math> (DR only)</li> </ul>
Compute	<ul style="list-style-type: none"> <li>☛ initial prototypes : <math>\mathbf{V}_0 = (\mathbf{v}_{10}, \mathbf{v}_{20}, \dots, \mathbf{v}_{c0}) \in \mathfrak{R}^{cP}</math> with (14)</li> </ul>
Iterate	<p>For <math>t = 1</math> to <math>T</math>:</p> <p>For <math>k = 1</math> to <math>n</math>: Randomly select <math>\mathbf{x} \in X</math> ; <math>X \leftarrow X - \{\mathbf{x}\}</math> ; <math>\mathbf{x}_k \leftarrow \mathbf{x}</math></p> <p>Find <math>\alpha_{ik,t}^{LVQ}</math> with (5) for <math>i=1, \dots, c</math></p> <p>(or) Find <math>\alpha_{ik,t}^{GLVQ-F}</math> with (7) for <math>i=1, \dots, c</math></p> <p>(or) Find <math>\alpha_{ik,t}^{DR}</math> with (8) for <math>i=1, \dots, c</math></p> <p><math>\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + \alpha_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1})</math> for <math>i=1, \dots, c</math></p> <p><math>\ \mathbf{x}_k - \mathbf{v}_{i,t}\  &lt; R_f \Rightarrow f_{ik,t} \leftarrow f_{ik,t} + \Delta f</math> (DR only)</p> <p><math>f_{ik,t} &gt; f_M \Rightarrow \mathbf{v}_{i,t}^* \leftarrow \mathbf{v}_{i,t}</math> (this prototype is locked) (DR only)</p> <p>Next <math>k</math></p> <p>If <math>E_t = \ \mathbf{V}_t - \mathbf{V}_{t-1}\ _{err} \leq \varepsilon</math>, Stop and put <math>\mathbf{V} \leftarrow \mathbf{V}_t</math>; Else</p> <p>Adjust learning rate <math>\alpha_t \leftarrow \alpha_0(1 - t / T)</math></p> <p>Next <math>t</math></p> <p>If <math>t=T</math> : put <math>\mathbf{V} \leftarrow \mathbf{V}_T</math></p>

Another point worth mentioning concerns the possibility that one or more labeled classes may end up with *no* representative prototypes. This cannot happen with the MCS, Chang, and MCA methods, but it can happen with the CL models (see [9] for an example). The most likely cause of this is very poor (unlucky) initialization. Since the data are labeled, the occurrence of some class being ignored will always be known to the user, and in this case, other initializations should be tried. Indeed, any method that requires initialization is subject to problems of this kind, so more than one should be tried as a matter of good development practice. A more problematical reason for underrepresentation is that the labeled classes are so mixed that neither selection nor replacement can effectively reduce the original data. This can be the case, for example, when the data comprise concentric clusters. Again, the user will know that this has happened, and the failure of a method such as MCS or MCA to provide much reduction in  $X_{tr}$  will signal the user that the entire data set should be used.

Finally, we comment on the results reported by Yen and Chang [13]. Their best 1-nmp (batch-designed) classifier was inferior to the best results achieved by all of the sequential CL models. We suspect that sequential updating encourages “localized” prototypes, which are able, when there is more than one per class, to position themselves better with respect to subclusters that may be present within the same class. This leads us to conjecture that batch algorithms are at their best when used to erect 1-np designs and that sequential models are more effective for 1-nmp classifiers. There is little doubt that the DR model is the best of the three CL methods tried for this data. We speculate that this is due to its prototype-by-prototype control structure, which provides *very* “localized” behavior, a property that seems ideally suited to finding multiple prototypes. The fact that DR never placed two prototypes in class 1 of Iris supports this, but we reserve judgment until more experimental evidence exists. This will be the subject of our next investigation.

TABLE XII  
MODIFIED CHANG ALGORITHM

Store	$\mathfrak{R}^p$ Labeled object data $X = X^1 \cup \dots \cup X^{\hat{c}} = X_{tr} \in \mathfrak{R}^p$ ordered with $\hat{c}$ classes as $X = \underbrace{\mathbf{x}_1^1, \dots, \mathbf{x}_{n_1}^1}_{\text{class 1}}, \dots, \underbrace{\mathbf{x}_1^{\hat{c}}, \dots, \mathbf{x}_{n_{\hat{c}}}^{\hat{c}}}_{\text{class } \hat{c}}$ , where $n_i =  X^i , i = 1, 2, \dots, \hat{c}$ .
Pick	Euclidean norm for similarity of data to prototypes : $\delta_E(\mathbf{x}, \mathbf{v}) = \ \mathbf{x} - \mathbf{v}\ _1 = \sqrt{(\mathbf{x} - \mathbf{v})^T (\mathbf{x} - \mathbf{v})}$
Set	$\mathbf{V}_n \leftarrow X$ ; $E_{\mathbf{V}_n, \delta_E}(X \mathbf{V}_n) = 0$ While $E_{\mathbf{V}_n, \delta_E}(X \mathbf{V}_n) = 0$ : ① Compute the partitioned upper triangular distance matrix <div style="text-align: center;">  </div> ② Find $(k^*, j^*) = \arg \min_{1 \leq k \leq \hat{c}; s \neq t} \{\ \mathbf{v}_s^k - \mathbf{v}_t^k\ \}$ . Compute $\mathbf{v}^{k^*} = (\mathbf{v}_i^{k^*} + \mathbf{v}_j^{k^*})/2$ , and update $\mathbf{V}_{n-1} \leftarrow \mathbf{V}_n - \{\mathbf{v}_i^{k^*}, \mathbf{v}_j^{k^*}\} + \mathbf{v}^{k^*}$ . If $E_{\mathbf{V}_{n-1}, \delta_E}(X \mathbf{V}_{n-1}) = 0$ ; $\mathbf{V}_{n-1} \leftarrow \mathbf{V}_{n-1}$ , compute $D(\mathbf{V}_{n-1})$ ; continue. Else return to $D(\mathbf{V}_n)$ and find the next pair $(h^*, h^*)$ that solves $(h^*, h^*) = \arg \min_{1 \leq k \leq \hat{c}; s \neq t} \{\ \mathbf{v}_s^k - \mathbf{v}_t^k\ \}$ and $(h^*, h^*) \neq (k^*, k^*)$ . Attempt to merge $\mathbf{v}^{h^*} = (\mathbf{v}_i^{h^*} + \mathbf{v}_j^{h^*})/2$ . Repeat step ② until no merger is possible. Terminate with $\mathbf{V}_c \ni E_{\mathbf{V}_c, \delta_E}(X \mathbf{V}_c) = 0$ .

## APPENDIX

See Tables XI and XII.

## REFERENCES

- [1] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Interscience, 1973.
- [2] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.
- [3] D. Titterton, A. Smith, and U. Makov, *Statistical Analysis of Finite Mixture Distributions*. New York: Wiley, 1985.
- [4] R. Krishnapuram and J. Keller, "A possibilistic approach to clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 98–110, May 1993.
- [5] B. V. Dasarthy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1990.
- [6] P. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
- [7] T. Kohonen, *Self-Organization and Associative Memory*, 3rd ed. Berlin, Germany: Springer-Verlag, 1989.
- [8] N. Karayiannis, J. C. Bezdek, N. R. Pal, R. J. Hathaway, and P. Pai, "Repairs to GLVQ: A new family of competitive learning schemes," *IEEE Trans. Neural Networks*, vol. 7, pp. 1062–1071, Sept. 1996.
- [9] N. R. Pal, J. C. Bezdek, and E. C. Tsao, "Generalized clustering networks and Kohonen's self-organizing scheme," *IEEE Trans. Neural Networks*, vol. 4, July 1993.
- [10] G. S. Lim, M. Alder, and P. Hadingham, "Adaptive quadratic neural nets," *Pattern Recognit. Lett.*, vol. 13, pp. 325–329, 1992.
- [11] P. McKenzie and M. Alder, "Initializing the EM algorithm for use in Gaussian mixture modeling," in *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, ed. E. S. Gelsema and L. N. Kanal, Eds. New York: Elsevier, pp. 91–105, 1994.
- [12] C. L. Chang, "Finding prototypes for nearest neighbor classification," *IEEE Trans. Comput.*, vol. C-23, Nov. 1974.
- [13] J. Yen and C. W. Chang, "A multi-prototype fuzzy c-means algorithm," in *Proc. 2nd EUFIT*, Aachen, Germany, 1994, pp. 539–543.
- [14] E. Yair, K. Zeger, and A. Gersho, "Competitive learning and soft competition for vector quantizer design," *IEEE Trans. Signal Processing*, vol. 40, pp. 294–309, Feb. 1992.
- [15] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
- [16] E. Anderson, "The IRISes of the Gaspe peninsula," in *Proc. Bull. Amer. IRIS Soc.*, vol. 59, 1935, pp. 2–5.
- [17] B. V. Dasarthy, "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, Mar. 1994.
- [18] L. I. Kuncheva and J. C. Bezdek, "Nearest prototype classification: Clustering, genetic algorithms or random search," this issue, pp. 160–164.



**James C. Bezdek** (M'80–SM'90–F'92) received the Ph.D. degree from Cornell University, Ithaca, NY, in 1973.

His interests include pattern recognition, fishing, computational neural networks, skiing, image processing, blues music, medical computing, and motorcycles.

Dr. Bezdek is the founding Editor of the IEEE TRANSACTIONS ON FUZZY SYSTEMS.



**Thomas R. Reichherzer** received the Vordiplom degree in computer science at the University of Ulm, Ulm, Germany, and the M.S. degree in computer science from both the University of West Florida, Pensacola, and the University of Ulm, in 1996.

He is presently with the Institute of Human & Machine Cognition (IHMC), University of West Florida. His research interests include pattern recognition, artificial intelligent agents, and natural language processing.



**Gek Sok Lim** received the B.Sc. degree in computer science from the University of Western Australia, Perth, in 1991. She submitted her Ph.D. dissertation to the Department of Electrical and Electronic Engineering, University of Western Australia, in early 1997.

She is currently with Lucent Technology. She was a Research Officer with the Centre for Intelligent Information Processing Systems (CIIPS), Department of Electrical and Electronic Engineering, University of Western Australia, from 1992 to 1996. She worked with NEC, Singapore, in cooperation with the Institute of Systems Science (ISS) and the Supercomputing Centre, National University of Singapore, Singapore, from 1991 to 1992.



**Yianni Attikiouzel** (M'74–SM'90–F'97) received the B.Sc. degree in electrical engineering with first-class honors and the Ph.D. degree from the University of Newcastle-upon-Tyne, U.K., in 1969 and 1973, respectively.

He is a Professor of Electrical and Electronic Engineering and Director of the Centre for Intelligent Information Processing Systems, Department of Electrical and Electronic Engineering, University of Western Australia, Perth. He has been active in the areas of adaptive signal processing, information technology, medical electronics, and artificial neural networks. His work has been published in more than 200 refereed papers in international journals and conferences. He is the author of two books, and he recently co-edited an IEEE Press book on computational intelligence.

Dr. Attikiouzel is a Fellow of the Institute of Electrical Engineers, U.K., and IEAust.