Classification with Multiple Prototypes

James C. Bezdek
Thomas R. Reichherzer
Department of Computer Science
University of West Florida
Pensacola, FL 32514, USA

Gek Lim
Yianni Attikiouzel
Center for Intell. Inf. Proc. Systems
Dept. of Electrical and Electronic Eng.
University of Western Australia
Nedlands, Perth, 6009, Western Australia

Abstract

We compare learning vector quantization, fuzzy learning vector quantization, and a deterministic scheme called the dog-rabbit (DR) model for generation of multiple prototypes from labeled data for classifier design. We also compare these three models to three other methods: a clumping method due to C. L. Chang; our modification of C.L. Chang's method; and a derivative of the batch fuzzy c-means algorithm due to Yen and C.W. Chang. All six methods are superior to the labeled subsample means, which yield 11 errors with 3 prototypes. Our modified Chang's method is, for the Iris data used in this study, the best of the six schemes in one sense; it finds 11 prototypes that yield a resubstitution error rate of 0. In a different sense, the DR method is best, yielding a classifier that commits only 3 errors with 5 prototypes.

1. Introduction

There are four types of class labels - crisp, fuzzy, probabilistic and possibilistic. Let integer c denote the number of classes, $1 < c < n$, and define three sets of label vectors in $\Re^c$ as follows :

$$N_{pc} = [0,1]^c - \{(0,0,\ldots 0)^T\} \quad ; \qquad (1a)$$

$$N_{fc} = \left\{ \mathbf{y} \in N_{pc} : \sum_{i=1}^{c} y_i = 1 \right\} \quad ; \qquad (1b)$$

$$\begin{aligned} N_{hc} &= \left\{ \mathbf{y} \in N_{fc} : y_i \in \{0,1\} \forall i \right\} \\ &= \left\{ \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_c \right\} \end{aligned} \quad . \qquad (1c)$$

N_{hc} is the canonical (unit vector) basis of Euclidean c-space. The i-th vertex of N_{hc}, $\mathbf{e}_i = (0, 0, \ldots, \underbrace{1}_{i}, \ldots, 0)^T$, is the crisp label for class i, $1 \le i \le c$. N_{fc}, a piece of a hyperplane, is the convex hull of N_{hc}. The vector $\mathbf{y} = (0.1, 0.6, 0.3)^T$ is a fuzzy or probabilistic label vector; its entries lie between 0 and 1, and sum to 1. The interpretation of $\mathbf{y}$ depends on its origin. If $\mathbf{y}$ is a label vector for some $\mathbf{x} \in \Re^p$ generated by, say, the fuzzy c-means clustering method, we call $\mathbf{y}$ a fuzzy label for $\mathbf{x}$. If $\mathbf{y}$ came from a method such as maximum likelihood estimation in mixture decomposition, $\mathbf{y}$ would be a probabilistic label. N_{pc}, the unit hypercube in $\Re^c$, *excluding the origin*, contains possibilistic label vectors such as $\mathbf{z} = (0.7, 0.2, 0.7)^T$. Note that $N_{hc} \subset N_{fc} \subset N_{pc}$.

Examples of *alternating optimization* (AO) algorithms that generate each of the four kinds of labels, as well as a set

$\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c\} \subset \Re^p$ of prototypes (or centers) for clusters in X from unlabeled object data are :

Label	Model/ AO Algorithm	Ref.
Crisp	Crisp c-means/ HCM	[1]
Fuzzy	Fuzzy c-means/ FCM	[2]
Prob.	Statistical mixture/EM	[3]
Poss.	Poss. c-means/ PCM	[4]

Object data are represented as X = {$\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$} in *feature space* $\Re^p$. The k-th object (a ship, patient, stock market report, pixel, etc.) has $\mathbf{x}_k$ as it's numerical representation; x_{jk} is the j-th characteristic (or *feature*) associated with object k. A *classifier*, any function $\mathbf{D} : \Re^p \mapsto N_{pc}$, specifies c decision regions in $\Re^p$. Training a classifier means identification of the parameters of **D** if it is explicit; or representing the boundaries of **D** algorithmically if it is implicit. The value **y** = **D**(**z**) is the label vector for **z** in $\Re^p$. **D** is a *crisp classifier* if $\mathbf{D}[\Re^p] = N_{hc}$. 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 (**y**) assigned to object vectors **z** during the operational (i.e., classification) phase are almost always converted to crisp ones through hardening of **y** with the function

$$\mathbf{H}(\mathbf{D}(\mathbf{z})) = \mathbf{H}(\mathbf{y}) = \mathbf{e}_i \Leftrightarrow$$
$$\|\mathbf{y} - \mathbf{e}_i\| \le \|\mathbf{y} - \mathbf{e}_j\| \Leftrightarrow$$
$$y_i \ge y_j \quad ; \ j \ne i \quad . \tag{2}$$

In (2) the distance is Euclidean, $\delta_E(\mathbf{y}, \mathbf{e}) = \|\mathbf{y} - \mathbf{e}\|_I = \sqrt{(\mathbf{y} - \mathbf{e})^T(\mathbf{y} - \mathbf{e})}$. If the design data are labeled, finding **D** is called *supervised learning*. Then 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_{pc}. *Testing* a classifier **D** designed with X_{tr} means: 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* $E_{\mathbf{D}}(X_{te} | X_{tr})$; our notation indicates that **D** was trained with X_{tr}, and tested with X_{te}. $E_{\mathbf{D}}$ is usually the performance index by which **D** is judged, and for convenience, we refer to it simply as the error rate. $E_{\mathbf{D}}(X|X)$ is called the *resubstitution* error rate. Resubstitution uses the same data for training and testing, so it usually produces an optimistic error rate, but this is not an impediment to using it to compare different designs.

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-nearest neighbor* ($\mathbf{D}_{k\text{-}nn}$) rule [5,6] can require too much storage and CPU time for efficient deployment. Here we discuss 6 ways to replace X_{tr} with a set of prototypes **V** that can be used as a substitute for X_{tr} (e.g., in the nearest neighbor rule) without appreciable degradation in $E_{\mathbf{D}_{k\text{-}nn}}((X_{te} | X_{tr})$. In this case $\mathbf{D}_{k\text{-}nn}$ becomes a nearest prototype design with error rate $E_{\mathbf{D}}(X_{te} | \mathbf{V})$

2. Nearest prototype classifiers

Once the prototypes **V** are found (and possibly relabeled if the data have physical labels), they can be used to define a crisp *nearest prototype* (1-np) classifier, say $\mathbf{D}_{\mathbf{V},\delta}$:

The *nearest prototype* (1-np) Classifier. Given *any* c prototypes $\mathbf{V} = \{\mathbf{v}_j \in \Re^p : 1 \le j \le c\}$, one $\mathbf{v}_j$ /class, and *any* dis-similarity measure δ on $\Re^p$: for any $\mathbf{z} \in \Re^p$:

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

Ties in (3) are arbitrarily resolved. The crisp 1-np design can be implemented using prototypes from *any* algorithm that produces them. Equation (3) defines a crisp classifier, even when **V** comes from a fuzzy, probabilistic or possibilistic algorithm. When one or more classes are represented by multiple prototypes, there are two ways to extend the 1-np design. We can simply use equation (3), recognizing that **V** contains more than one prototype for at least one of the c classes. Or we can extend the 1-np design to a k-np rule, wherein the k nearest prototypes are used to conduct a vote about the label that should be assigned to input **z**. This amounts to operating the k-nn rule using prototypes (points built from the data) instead of neighbors (points in the data). We opt here for the simpler choice, which is formalized as the

The *nearest multiple prototype* (1-nmp) classifier. Given *any* N_p prototypes $\mathbf{V}=\{\mathbf{v}_{ij}\in\Re^p : 1\le i\le c; 1\le j\le n_{pj}\}$, where n_{pj} is the number of prototypes for class j, $N_p=\sum_{j=1}^{c} n_{pj}$; and *any* dis-similarity measure δ on $\Re^p$: for any $\mathbf{z}\in\Re^p$:

$$\begin{aligned}&\text{Decide } \mathbf{z}\in \text{class } i \Leftrightarrow \mathbf{D}_{\mathbf{V},\delta}(\mathbf{z})=\mathbf{e}_i \Leftrightarrow\\ &\exists s\in\{1,\ldots,n_{pi}\}\ni \delta(\mathbf{z},\mathbf{v}_{is})\le\delta(\mathbf{z},\mathbf{v}_{jt})\\ &\forall\ j\ne i \text{ and } t\in\{1,\ldots,n_{pj}\}\end{aligned} \quad . \ (3')$$

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 are ready to turn to methods for finding multiple prototypes.

3. Three sequential CL models

Sequential learning models update estimates at iterate (t-1) of the $\{\mathbf{v}_i\}$ at iterate t (one iteration is one pass through X) upon presentation of an $\mathbf{x}_k$ from X using the general form, i = 1, 2, ..., c:

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

In (4) $\{\alpha_{ik,t}\}$ is the *learning rate distribution* over the c nodes for input $\mathbf{x}_k$ at iterate t. The principle difference between various competitive learning models lies in (i) the subset of nodes that get updated at each iterate, and (ii) values of the $\{\alpha_{ik,t}\}$. 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 nodes for each presentation of an input. The learning rate distribution for LVQ is well known:

$$\alpha_{ik,t}^{LVQ}=\begin{Bmatrix}\alpha_t & ; & i=\underbrace{\arg\min}_{r}\{\|\mathbf{x}_k-\mathbf{v}_{r,t-1}\|\}\\ 0 & , & r=1,2,\ldots c\ ;\ r\ne i\end{Bmatrix} . (5)$$

In (5) α_t is usually initialized at some value in (0, 1), and decreased linearly with t. The model underlying GLVQ-F contains LVQ as a subcase and is discussed extensively elsewhere [8]. The GLVQ-F update rule for the prototypes **V** at iterate t in the special (and simple) case m=2 uses the following learning rate distribution in equation (4), i = 1, 2, ..., c:

$$\alpha_{ik,t}^{GLVQ-F} = 2c\alpha_t\left(\sum_{r=1}^{c}\left(\|\mathbf{x}_k-\mathbf{v}_{i,t-1}\|^2\Big/\|\mathbf{x}_k-\mathbf{x}_{r,t-1}\|^2\right)\right)^{-2} . (6)$$

As in (5), α_t in (6) - now one <u>component</u> of the *learning rates* $\{\alpha_{ik,t}\}$ - is usually proportional to 1/t, and the constant (2c) is absorbed in it without loss.

Like GLVQ-F, the DR algorithm [10] may update all c nodes 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 update equation (4) that is used by the DR model:

$$\alpha_{ik,t}^{DR}=\begin{Bmatrix}(A) & ; i=\underbrace{\arg\min}_{r}\{\|\mathbf{x}_k-\mathbf{v}_{r,t-1}\|\}\\ (B) & , r=1,2,\ldots c\ ;\ r\ne i\end{Bmatrix} . (7)$$

where

$$(A)=\left(\frac{2\|\mathbf{x}_k-\mathbf{v}_{i,t-1}\|}{\left(1+\|\mathbf{x}_k-\mathbf{v}_{i,t-1}\|\right)^{f_{ik,t-1}}}\right), \text{ and}$$

$$(B)=\left(\frac{\|\mathbf{x}_k-\mathbf{v}_{i,t-1}\|}{\left(\Lambda+\|\mathbf{x}_k-\mathbf{v}_{i,t-1}\|\right)}\right)\left(\frac{2\|\mathbf{x}_k-\mathbf{v}_{r,t-1}\|}{\left(1+\|\mathbf{x}_k-\mathbf{v}_{r,t-1}\|\right)^{f_{jk,t-1}}}\right)$$

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* $\Lambda > 0$. Suppose $\mathbf{v}_{i,T-1}$ to be the winning prototype with $\|\mathbf{x}_k - \mathbf{v}_{i,t-1}\| > R_f$. All c nodes are updated using (7) 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 nodes), and it also encourages non-winners to look for other data to chase. Movement of (i.e., updating) the i-th prototype ceases when $f_{ik,t} > f_M$. Thus, termination of updating is done node by node.

Unlike Chang's method, none of the CL methods just described uses the labels of points in X_{tr} during training to guide iterates towards 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_{tr}. The relabeling algorithm uses L_{tr} to attach the most likely physical label to each $\mathbf{v}_i$. Let $\hat{c}$ be the number of classes in X_{tr}, labeled by the crisp vectors $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{\hat{c}}\} = N_{h\hat{c}}$ Now define p_{ij}, i=1,2, ..., $\hat{c}$, j=1,2, ..., 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 $N_{f\hat{c}}$, and c columns $\mathbf{p}_i$ in $N_{p\hat{c}}$. We assign label $\mathbf{e}_j$ to $\mathbf{v}_i$ when $\mathbf{H}(\mathbf{p}_i) = \mathbf{e}_j$. The assignment is formalized as

$$\text{label } j \leftarrow \mathbf{v}_i \Leftrightarrow \mathbf{H}(\mathbf{p}_i) = \mathbf{e}_j \; ; \quad i = 1,2,\ldots,c \; ; \; j = 1,2,\ldots,\hat{c} \tag{8}$$

Finally, X_{te} is submitted to the classifier and its error rate is computed and tabulated using the $c \times c$ confusion matrix $C = [c_{ij}]$ = [# labeled class j I but were really class i].

4. Numerical results

Following Chang [11], we use Anderson's Iris data [12] as an experimental data set . Iris contains 50 (physically labeled) vectors in $\Re^4$ for each of c=3 classes of Iris subspecies. For reference, the resubstitution error rate for the supervised 1-np design that uses the class means of each subset of Iris listed in Table 1 as single prototypes in (3) is 11 errors in 150 submissions using the Euclidean norm, $E_{\mathbf{D}_{\overline{\mathbf{V}},\delta_E}}$ (Iris|Iris) = 7.33%.

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

Table 1. Subsample (mean) prototypes $\overline{\mathbf{V}}$ in $\Re^4$ for Iris

Computing protocols, control parameters, initialization, termination, iteration and robustness are discussed in [14].

LVQ	# $\mathbf{v}_i$'s : LVQ
$\begin{pmatrix}50 & 0 & 0\\ 0 & 47 & 3\\ 0 & 1 & 49\end{pmatrix}$	Class 1 : 2 Class 2 : 3 Class 3. 3
GLVQ-F	# $\mathbf{v}_i$'s : GLVQ-F
$\begin{pmatrix}50 & 0 & 0\\ 0 & 47 & 3\\ 0 & 1 & 49\end{pmatrix}$	Class 1 : 2 Class 2 : 3 Class 3. 3
DR	# $\mathbf{v}_i$'s : DR
$\begin{pmatrix}50 & 0 & 0\\ 0 & 47 & 3\\ 0 & 1 & 49\end{pmatrix}$	Class 1 : 1 Class 2 : 4 Class 3. 3

Table 2. Typical confusion matrices and class representatives for 8 terminal prototypes

When the three CL algorithms are instructed to seek c = 8 prototypes, the error rate for all three 1-nmp designs typically remains at 2.33% as shown in Table 2. Table 2 suggests that the replacement of IRIS with 8 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.

Table 3 reports best case results (as number of resubstitution errors) using each algorithm for various values of c. Shaded cells show the best case in each row. 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 resulted in 5 prototypes that produced only 3 resustitution errors when used in (3'). These prototypes are shown in Table 4.

c →	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

Table 3. Number of resubstitution errors of the 1-nmp designs : best case results

This shows that the Iris data can be well represented by five labeled prototypes. At the other extreme, increasing c past c = 8 has little effect on the best case results. Taken together, these observations suggest that Iris (and more generally, any labeled data set X) has some upper and lower bounds in terms of high quality representation by multiple prototypes for classifier design.

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

Table 4. Five DR prototypes that yield 3 resubstitution errors with the 1-nmp rule on Iris

5. Other multi-prototype designs

C.L. Chang's 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, $E_{D_{\mathbf{V}_n},\delta_E}(X|\mathbf{V}_n) = 0$. Now find $(i,j) = \underbrace{\arg\min}_{s \neq t}\{[\|\mathbf{x}_s - \mathbf{x}_t\|]\}$. Merge $\mathbf{x}_i$ and $\mathbf{x}_j$ using $\mathbf{v}_{ij} = (M\mathbf{x}_i + N\mathbf{x}_j)/(M+N)$, where M and N are the number of merger parents of $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively. Now set $\mathbf{V}_{n-1} = X - \{\mathbf{x}_i, \mathbf{x}_j\} + \mathbf{v}_{ij}$ and then calculate $E_{D_{\mathbf{V}_{n-1}},\delta_E}(X|\mathbf{V}_{n-1})$ 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 (i) the error rate increases; or (ii) $\mathbf{x}_i$ 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 non-mergeable prototypes and continues. Note that when there is a merger, $\mathbf{v}_{ij}$ automatically inherits the class label of its parents, and that the test data are fixed (all of X). Continue 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., $E_{D_{\mathbf{V}_c},\delta_E}(X|\mathbf{V}_c) = 0$. Chang reports in [12] that his method finds c = 14 prototypes that replace Iris and preserve a zero resubstitution error rate. The prototypes were not listed in [11].

We modified Chang's approach in two ways (cf. Appendix). 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. Second, we altered the search for candidates to merge two ways. First, we partition the distance matrix into c submatrices blocked by common labels, and look for the minimum in each sublock. This eliminates candidate pairs with different labels. Then, we attempt to merge the minimum of label-matched pairs. If this fails (because the prototype produced by merger yields an error), we look at the next candidates. And we continue looking in ascending order of distance until either (i) a merger can be done; or (ii) no merger is possible. The algorithm terminates when (ii) occurs. This is an effective strategy because merging the closest points of the same label is sufficient, but not necessary in order to preserve the error rate at zero. These two simple modifications led to c = 11 prototypes that yield zero resubstitution error.

Yen and Chang [13] modifed 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, 2,3. 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).

5. Conclusions and discussion

Table 5 summarizes the best results achieved by the seven algorithms used in our study. What does Table 5 entitle us to conclude? First, our results are of course specialized to just one data set, and generalizations to other data warrant caution.

Algorithm	**Min. errors in 150 tries**	**# v's**
Labeled 1-np ($\bar{\mathbf{V}}$)	11	3
1-nmp designs		
LVQ	3	7
GLVQ-F (m=2)	3	8
DR	3	5
Chang	0	14
Mod. Chang	0	11
MFCM-3	8	7

Table 5. Summary of the best error rates achieved by the 6 methods

All six 1-nmp designs use the labeled data more effectively than the 1-np design based on the labeled sample means. The minimum error rate (zero) is not realized by the minimum number of prototypes (five). If the determining criterion for choosing multiple prototypes is *minimum error rate*, then our modification of Chang's method 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 *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 ideally suited to finding multiple prototypes that yield a few errors with fewer prototypes than the modified Chang's method.

Finally, we comment on the results reported by Yen and Chang [13]. Their best 1-nmp (batch-design) classifier was inferior to the best results achieved by all of the 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.

6. References

[1] Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis*, Wiley Interscience, NY.

[2] Bezdek, J. C. and Pal, S.K. (1992). *Fuzzy Models for Pattern Recognition, IEEE* Press, Piscataway, NJ.

[3] Titterington, D., Smith, A. and Makov, U. (1985). *Statistical Analysis of Finite Mixture Distributions*, Wiley, NY.

[4] Krishnapuram, R. and Keller, J. (1993). A Possibilistic Approach to Clustering, *IEEE Trans. Fuzzy Systems*, 1(2), 98-110.

[5] Dasarathy, B.V. (1990). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, CA.

[6] Devijver, P. and Kittler, J. (1982).*Pattern Recognition: A Statistical Approach*, Prentice-Hall, Englewood Cliffs, NJ.

[7] Kohonen, T. (1989). *Self-organization and Associative Memory*. Springer-Verlag, Berlin, Germany, 3rd ed.

[8] Karayiannis, N., Bezdek, J.C., Pal, N.R., Hathaway, R.J. and Pai, P. (1995). Repairs to GLVQ : A new family of competitive learning schemes, in press, *IEEE Trans. Neural Networks*,.

[9] Pal, N.R., Bezdek, J.C., Tsao, E.C. (1993). *Generalized Clustering Networks and Kohonen's Self-Organizing Scheme. IEEE Trans. Neural Networks*,

[10] Lim. G.S., Alder, M. and Hadingham, P. (1992). Adaptive quadratic neural nets, *Patt. Recog. Letters*, 13, 325-329.

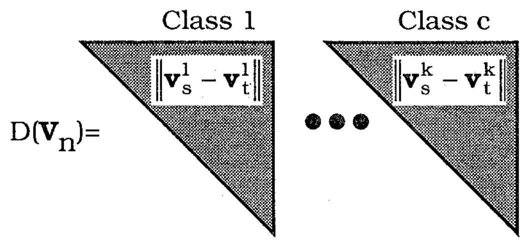
[11] Chang, C.L. (1974). Finding prototypes for nearest neighbor classification, *IEEE Trans. Comp.*, 23(11).

[12] Anderson, E. (1935). The IRISes of the Gaspe peninsula, *Bull. Amer. IRIS Soc.* , 59, 2-5.

[13] Yen, J. and Chang, C.W. (1994). A multi-prototype fuzzy c-means algorithm, *Proc. 2nd EUFIT*, Aachen, 539-543.

[14] Bezdek, J. C., Reicherzer, T. , Lim, G. and Attikiouzel, Y. (1995). Multiple prototype classifier design, in review, *IEEE Trans. SMC*.

Appendix. The modified Chang algorithm

Store	☞ Labeled object data $X = \underbrace{\mathbf{x}_1^1,\cdots,\ \mathbf{x}_{n_1}^1}_{\text{class 1}},\cdots,\underbrace{\mathbf{x}_1^c,\cdots,\ \mathbf{x}_{n_1}^c}_{\text{class c}}$
Pick	☛ $\delta_E(\mathbf{x},\mathbf{v}) = \|\mathbf{x}-\mathbf{v}\|_I = \sqrt{(\mathbf{x}-\mathbf{v})^T(\mathbf{x}-\mathbf{v})}$ for similarity of data to prototypes
Initialize	☛ $\mathbf{V}_n \leftarrow X$; $E_{\mathbf{V}_n,\delta_E}(X \mid \mathbf{V}_n) = 0$
	While $E_{\mathbf{V}_\omega,\delta_E}(X \mid \mathbf{V}_\omega) = 0$: ❶ Compute the partitioned upper triangular distance matrix  Class 1 Class c $D(\mathbf{V}_n)=$ $\|\mathbf{v}_s^1-\mathbf{v}_t^1\|$ $\|\mathbf{v}_s^k-\mathbf{v}_t^k\|$ ● ● ● ❷ Find $\left(\substack{k^*\\i},\substack{k^*\\j}\right) = \underbrace{\arg\min}_{1\le k\le c\ ;\ s\ne t}\left\{\|\mathbf{v}_s^k-\mathbf{v}_t^k\|\right\}$. Compute $\mathbf{v}^{k^*} = (\mathbf{v}_i^{k^*}+\mathbf{v}_j^{k^*})/2$, and update $\mathbf{V}_{n-1}^* \leftarrow \mathbf{V}_n - \{\mathbf{v}_i^{k^*},\ \mathbf{v}_j^{k^*}\} + \mathbf{v}^*$. If $E_{\mathbf{V}_{n-1},\delta_E}(X \mid \mathbf{V}_{n-1}^*) = 0$; $\mathbf{V}_{n-1} \leftarrow \mathbf{V}_{n-1}^*$, compute $D(\mathbf{V}_{n-1})$; continue. Else return to $D(V_n)$ and find the next pair $\left(\substack{h^*\\i},\substack{h^*\\j}\right)$ that solves $\left(\substack{h^*\\i},\substack{h^*\\j}\right) = \underbrace{\arg\min}_{1\le k\le c\ ;\ s\ne t}\left\{\|\mathbf{v}_s^k-\mathbf{v}_t^k\|\right\}$ and $\left(\substack{h^*\\i},\substack{h^*\\j}\right) \ne \left(\substack{k^*\\i},\substack{k^*\\j}\right)$. Attempt to merge $\mathbf{v}^{h^*} = (\mathbf{v}_i^{h^*}+\mathbf{v}_j^{h^*})/2$. Repeat step ❷ until no merger is possible. Terminate with $\mathbf{V}_c \ni E_{\mathbf{V}_c,\delta_E}(X \mid \mathbf{V}_c) = 0$.