Pergamon

Neural Networks, Vol. 8, No. 5, pp. 729–743, 1995
Copyright © 1995 Elsevier Science Ltd
Printed in USA. All rights reserved
0893–6080/95 $9.50 + .00

0893-6080(95)00024-0

CONTRIBUTED ARTICLE

Two Soft Relatives of Learning Vector Quantization

JAMES C. BEZDEK AND NIKHIL R. PAL

The University of West Florida

(*Received* 6 *October* 1992; *revised and accepted* 1 *February* 1995)

Abstract—*Learning vector quantization often requires extensive experimentation with the learning rate distribution and update neighborhood used during iteration towards good prototypes. A single winner prototype controls the updates. This paper discusses two soft relatives of LVQ: the soft competition scheme (SCS) of Yair et al. and fuzzy LVQ = FLVQ. These algorithms both extend the update neighborhood to all nodes in the network. SCS is a sequential, deterministic method with learning rates that are partially based on posterior probabilities. FLVQ is a batch algorithm whose learning rates are derived from fuzzy memberships. We show that SCS learning rates can be interpreted in terms of statistical decision theory, and derive several relationships between SCS and FLVQ. Limit analysis shows that the learning rates of these two algorithms have opposite tendencies. Numerical examples illustrate the difficulty of choosing good algorithmic parameters for SCS. Finally, we elaborate the relationship between FLVQ, Fuzzy c-Means, Hard c-Means, a batch version of LVQ and SCS.*

Keywords—c-Means Clustering, Fuzzy Clustering, Fuzzy LVQ, Learning Vector Quantization, Soft Competition Scheme.

1. INTRODUCTION

Numerical clustering algorithms organize a set of unlabeled feature vectors $X = \{x_1, x_2, \ldots, x_n\} \subset \mathcal{R}^p$ into clusters or natural groups (Hartigan, 1975; Duda & Hart, 1973; Jain & Dubes, 1988). To characterize solution spaces for clustering and classifier design, let c denote the number of clusters, $1 < c < n$, and set:

$$N_{pc} = \{y \in \mathcal{R}^c | y_i \in [0, 1] \forall i,\ y_i > 0 \exists i\} \quad \text{(1a)}$$

possibilistic labels;

$$N_{fc} = \left\{ y \in N_{pc} | \sum_{i=1}^{c} y_i = 1 \right\} \quad \text{(1b)}$$

fuzzy/probabilistic labels;

$$N_{hc} = \{y \in N_{fc} | y_i \in \{0, 1\} \forall i\} \quad \text{(1c)}$$

crisp labels.

Figure 1 depicts these sets for $c = 3$ classes. N_{hc} is the canonical (unit vector) basis of Euclidean c-space; N_{fc}, a subset of a hyperplane, is its convex hull; and N_{pc} is the unit hypercube in $\mathcal{R}^c$ minus the origin, $N_{pc} = [0, 1]^c - \{\theta\}$. These three sets are *label vectors*; each point in them provides a set of class labels for either a real object, or a numerical characterization ($x_k \in \mathcal{R}^p$) of it. The ith vertex of N_{hc},

$$e_i = (0, 0, \ldots, \underbrace{1}_{i}, \ldots, 0)^T,$$

is the crisp label for class i, $1 \le i \le c$.

It is important to see that fuzzy and probabilistic labels lie in the same set. The vector $y = (0.1, 0.6, 0.3)^T$ in N_{fc} is a constrained label

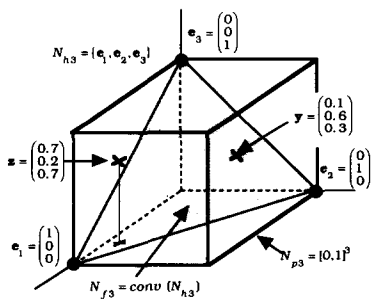

FIGURE 1. Hard, fuzzy, probabilistic and possibilistic label vectors (for c = 3 classes).

Acknowledgments: Anonymous referees of an earlier version of this paper provided many valuable suggestions for its improvement. Reviewing papers is a thankless job, and reviewing them well is hard. The comments sent to us were really invaluable in helping us clarify our thinking on soft relatives of LVQ. The connection between LVQ, FLVQ, SCS and the c-Means families given in this paper was fully understood by us only during the rewrite. For this, as well as numerous other improvements, we owe them our thanks and we give it.

Requests for reprints should be sent to J. C. Bezdek, Department of Computer Science, The University of West Florida, Pensacola, FL 32514, USA.

vector; its entries lie between 0 and 1, and are constrained to sum to 1. If y is generated by, say, the fuzzy c-means clustering method (Bezdek, 1981), we call it a fuzzy label for some x_k, and interpret its values as the membership of x_k in each of the classes represented by the rows of y. Thus, 0.6 is the membership of x_k in class 2. If y came from a method such as maximum likelihood estimation in mixture decomposition (Titterington, Smith & Makov, 1985), it would be a probabilistic label, and 0.6 would be the (posterior) probability $p(2|x_k)$ that, given x_k, it came from class 2.

N_{pc} is called *possibilistic* label vector space. Vectors in it such as $z=(0.7, 0.2, 0.7)^T$ have each entry between 0 and 1, but the components of z do not necessarily sum to 1. z might be generated as the label for some x_k by, for example, the possibilistic c-means clustering model (Krishnapuram & Keller, 1993), or by a feed-forward neural (classifier) network that has unipolar sigmoidal transfer functions at each of c output nodes (Haykin, 1994). In this case z_i can be regarded as the possibility that x_k belongs to class i.

Clustering in unlabeled data X is the assignment of (hard, fuzzy, probabilistic, or possibilistic) label vectors to the points in X, and hence, to the objects generating X. If the labels are hard, we hope they identify c natural subgroups in X. Clustering is also called *unsupervised learning*, the word *learning* referring here to learning the correct labels (and possibly vector prototypes or quantizers) for good subgroups in the data. c-partitions of X are sets of (cn) values $\{u_{ik}\}$ satisfying some or all of the following conditions. Let $U_{(k)}$ be the k-th column of U:

$$U_k \in N_{pc} \qquad \forall k; \tag{2a}$$

$$0 < \sum_{k=1}^{n} u_{ik} < n \qquad \forall i; \tag{2b}$$

$$\sum_{i=1}^{c} u_{ik} = 1 \qquad \forall k. \tag{2c}$$

Using eqns (2) with the values $\{u_{ik}\}$ arrayed as a $(c \times n)$ matrix $U = [u_{ik}]$, we define:

$$M_{pcn} = \{U \in \mathscr{R}^{cn} | U \text{ satisfies (2a) and (2b)}\}; \tag{3a}$$

$$M_{fcn} = \{U \in M_{pcn} | U \text{ satisfies (2c)}; \tag{3b}$$

$$M_{hcn} = \{U \in M_{fcn} | u_{ik} = 0 \text{ or } 1 \forall i \text{ and } k\}. \tag{3c}$$

Eqns (3a)–(3c) define, respectively, the sets of possibilistic, fuzzy/probabilistic, and crisp c-partitions of X. Each column of U in $M_{pcn}(M_{fcn}, M_{hcn})$ is a label vector from $N_{pc}(N_{fc}, N_{hc})$. The reason these matrices are called *partitions* for all cases except the probabilistic context follows from the interpretation of column k as the membership of x_k in the i subsets of X defined by the rows of U. If U is probabilistic, its rows define the posterior probabilities of each x_k in sample X of being from one of c probability distributions. We indicate the statistical context by replacing $U = [u_{ik}]$ with $P = [p_{ik}] = [p(i|x_k)]$. Observe that $M_{hcn} \subset M_{fcn} \subset M_{pcn}$.

A *classifier* is any function D: $\mathscr{R}^p \mapsto N_{pc}$. The value $y = D(z)$ is the label vector for z in $\mathscr{R}^p$. D is a *crisp classifier* if $D[\mathscr{R}^p] = N_{hc}$. Since definite class assignments are usually the ultimate goal of classification and clustering, outputs of algorithms that produce label vectors in N_{pc} or N_{fc} are commonly transformed into crisp labels. Most often, non-crisp labels are converted to crisp ones using the function H: $N_{pc} \mapsto N_{hc}$,

$$H(y) = e_i \Leftrightarrow \|y - e_i\| \leq \|y - e_j\| \Leftrightarrow y_i \geq y_j; j \neq i. \tag{4}$$

In eqn (4) $\| * \|$ is the Euclidean norm on $\mathscr{R}^c$. If $y = D(z)$, H simply finds the crisp label vector e_i in N_{hc} closest to y. Alternatively, H finds the *maximum coordinate* of y, and assigns this crisp label to z. The rationale for using H depends on the algorithm that produces label vector y. For example, the justification for using eqn (4) for outputs from the k-nearest neighbor rule is simple majority voting. If y is gotten from mixture decomposition, using H is Bayes rule-label z by its class of maximum posterior probability. And if the labels are fuzzy, this step is called defuzzification of U by the maximum membership rule. We give these procedures a common name; we will call the use of H *hardening*.

Clustering algorithms produce *partitions*, which are *sets* of label vectors. For fuzzy partitions, the usual method of defuzzification is the application of eqn (4) to each column $U_{(k)}$ of U. The crisp maximum membership partition U_{MM} in M_{hcn} corresponding to any $U \in M_{pcn}$ has as its kth column, $1 \leq k \leq n$:

$$U_{MM,(k)} = H(U_{(k)}) = e_i \Leftrightarrow u_{ik} \geq u_{jk} \quad j = 1, 2, \ldots, c, \quad j \neq i. \tag{5}$$

The conversion of a probabilistic partition $P = [p_{ik}] \in M_{fcn}$ by Bayes rule (decide $x_k \in$ class i if and only if $p(i|x_k) \geq p(j|x_k)$ for $j \neq i$) also results in a crisp partition, P_{MP}, which is entirely analogous to the maximum membership partition produced by eqn (5). If there are ties in eqn (5), they are resolved arbitrarily as long as the appropriate constraint is preserved.

We discuss three algorithms that generate prototypes from unlabeled data. Prototype representation (or vector quantization) is based on the idea illustrated in Figure 2. The vector v_i is taken as a prototypical representation for all the vectors in the hard cluster $X_i \subset X$. There are many synonyms for

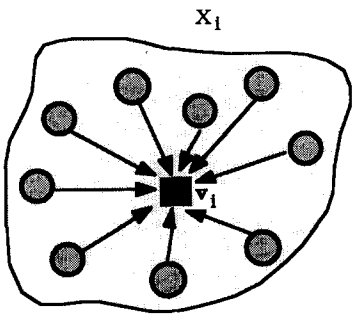

FIGURE 2. Representation of many vectors by one prototype.

the word prototype: for example, vector quantizer, signature, template, codevector, paradigm, centroid, exemplar. In the context of clustering v_i is often called the *cluster center* of hard cluster $X_i \subset X$.

Many families of algorithms are prototype generators. There are, roughly speaking, four approaches: (i) pattern recognition models such as the leader algorithm (Hartigan, 1975), sequential hard c-means (Duda & Hart, 1973), and batch hard, fuzzy (Bezdek, 1981) and possibilistic (Krishnapuram & Keller, 1993) c-means: (ii) statistical models such as mixture decomposition (Titterington, Smith & Makov, 1985); (iii) network models such as Kohonen's self-organizing feature maps and its many generalizations (Kohonen, 1989; Pal, Bezdek & Tsao, 1993); and (iv) vector quantizer approaches such as the generalized Lloyd algorithm (Gersho & Gray, 1992).

Once prototypes are found (and possibly relabeled if the data have physical labels), they can be used to define a hard nearest prototype (1-*NP*) classifier, say $D_{NP,V}$:

1.1. Crisp Nearest Prototype (1-NP) Inner Product Classifier

Given prototypes $V = \{v_k | 1 \le k \le c\}$ and $z \in \mathcal{R}^p$:

$$\text{Decide } z \in i \Leftrightarrow D_{NP,V}(z) = e_i \Leftrightarrow \|z - v_i\|_A \le \|z - v_j\|_A : \quad 1 \le j \le c, j \ne i. \quad (6)$$

In eqn (6) A is any *positive definite* $p \times p$ weight matrix, it renders the norm in eqn (6) an inner product norm,

$$\|z - v_i\|_A = \sqrt{(z - v_i)^T A (z - v_i)}.$$

Eqn (6) defines a hard classifier, even though its parameters may come from a fuzzy or probabilistic algorithm. It would be careless to call $D_{NP,V}$ a fuzzy classifier, for example, just because fuzzy c-means produced the prototypes. The crisp 1-*NP* design can be implemented using prototypes from *any* algorithm that produces them. For fuzzy c-means and decomposition of normal mixtures, U_{MM} and P_{MP} are, respectively, generated implicitly by the 1-*NP* rule at eqn (6).

Our brief discussion of the 1-*NP* classifier here is aimed primarily at clarifying the geometry of and role for the prototypes generated by LVQ type algorithms, and their relationship to the 1-*NP* rule at eqn (6). However, it is worth noting that prototypes generated by LVQ and its relatives (and any other prototype generation method, for that matter) *are* often used to define the classifier $D_{NP,V}$.

The common denominator in all VQ schemes is a mathematical definition of how well prototype v_i represents X_i. Any measure of similarity on $\mathcal{R}^p$ can be used to define this match. The usual choice is distance (dissimilarity), the most convenient is squared distance, the most common is squared Euclidean distance. Local (sequential) methods attempt to optimize some function of the c squared distances $\{\|x_k - v_i\|^2; 1 \le i \le c\}$ at each x_k in X_i. Global (batch) methods usually seek extrema of some function of all (cn) distances $\{\|x_k - v_i\|^2; 1 \le i \le c \text{ and } 1 \le k \le n\}$.[1]

LVQ attempts to minimize an objective function that places all of its emphasis on the winning prototype for each data point. However, information due to data point x is carried by *all* of the c distances $\{\|x - v_r\|\}$. Many authors have suggested modifications to LVQ that update all c quantizers during each updating epoch, thereby eliminating the need to define an update neighborhood. We study two models of this type: a fuzzy model called *fuzzy learning vector quantization* (FLVQ) (Tsao, Bezdek & Pal, 1992); and a deterministic learning algorithm that uses learning rates that are partially based on probabilities called the *soft competition scheme* (SCS) (Yair, Zeger & Gersho, 1992).

Sections 2 and 3 describe the LVQ and SCS models. Each section contains a specification of an algorithm for optimizing its model. Section 4 exhibits a relationship between SCS and statistical mixtures. We show that SCS is related to a mixture of c multivariate normal distributions with circular covariance matrix structure. Section 5 discusses the Hard and Fuzzy c-Means models, with particular emphasis on their limiting behavior. Section 6 describes FLVQ, and analyzes it as a function of one of its parameters. Section 7 contains a numerical example that uses the IRIS data to compare certain aspects of LVQ, SCS and FLVQ. We compare limiting properties of SCS to LVQ in Section 8. We also discuss the relationship of these generalizations to the batch hard and fuzzy c-means models/

[1] Do not confuse our use of the terms local and global *methods* with the local and global *extrema* found by a particular method.

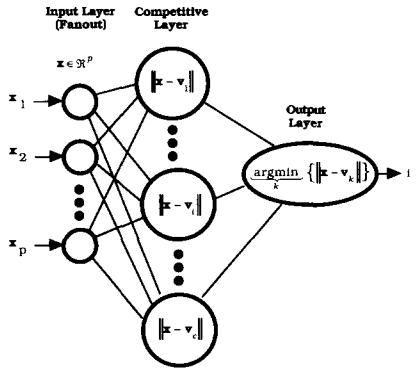

FIGURE 3. The LVQ competitive learning network.

algorithms. Section 9 contains our conclusions and some ideas for further research.

2. LEARNING VECTOR QUANTIZATION

The primary goal of LVQ is representation of many points by fewer prototypes. Identification of clusters is implicit, but not active, in LVQ's pursuit of this goal. The salient features of the LVQ model are contained in Figure 3. The input layer of an LVQ network is connected directly to the output layer. The circles in Figure 3 are *nodes*, and the prototypes are node weights.[2] The prototypes $V = (v_1, v_2, \ldots, v_c)$, $v_i \in \mathcal{R}^p$ for $1 \le i \le c$, are the (unknown) vector quantizers we seek.

When an input vector x is submitted to this network, distances are computed between x and each v_k. The output nodes "compete," a (minimum distance) winner node, say v_i, is found; and it is then updated using one of several update rules. We give a brief specification of LVQ. (There are many other versions of LVQ; this one is usually regarded as the standard form).

2.1. The (Unlabeled Data) LVQ Algorithm (Kohonen, 1989).

Store. Unlabeled Object Data $X = \{x_1, x_2, \ldots, x_n\} \subset \mathcal{R}^p$

Pick. ☛ $1 < c < n$ ☛ $\alpha_1 \in (0, 1)$ ☛ $N =$ max. iterations ☛ $\varepsilon > 0$

Guess. $V_0 = (v_{1,0}, v_{2,0}, \ldots, v_{c,0}) \in \mathcal{R}^{cp}$

Iterate. For $t = 1, 2, \ldots, N$;
For $k = 1, 2, \ldots, n$:

Find $\|x_k - v_{i,t-1}\| = \min_{1 \le j \le c}\{\|x_k - v_{j,t-1}\|\}$. (7a)

Update the winner:

$$v_{i,t} = v_{i,t-1} + \alpha_t(x_k - v_{i,t-1}) \quad (7b)$$

Next k
Adjust learning rate $\alpha_t \leftarrow \alpha_0(1 - t/N)$

Next t
$V \leftarrow V_N$

Use. Prototypes V. For example, a crisp label matrix for X can be generated by applying the 1-NP (nearest prototype) rule to the data:

$$u_{LVQ_k} = \left\{ \begin{array}{ll} 1; & \|x_k = v_i\| \le \|x_k = v_j\|,\ 1 \le j \le c, j \ne i \\ 0; & \textit{otherwise} \end{array} \right\}, \quad 1 \le k \le n. \quad (8)$$

The update scheme or learning rule in eqn (7b) has a simple geometric interpretation. The winning prototype $v_{i,t-1}$ is simply rotated towards the current data point, and is constrained to move towards it along the vector $(x_k - v_{i,t-1})$ which connects it to x_k. The amount of shift depends on the value of the *learning rate* parameter $\alpha_t \in [0, 1]$. There is no update if $\alpha_t = 0$, and when $\alpha_t = 1$, $v_{i,t}$ becomes x_k. The optional labeling phase generates $U_{LVQ} = [u_{LVQ_{ik,N}}]$ at eqn (8), which is a $c \times n$ matrix whose columns are in N_{hc}. U_{LVQ} is almost always a hard c-partition of X (constraint (2b) may not be satisfied).

Learning rule (7b) is obtained by assuming that each $x \in \mathcal{R}^p$ is distributed according to an unknown time invariant probability density function $F(x)$. LVQ's objective is to find a set of v_i's such that the expected value of the square of the discretization error

$$E(\|x - v_i\|^2) = \int \int_{\mathcal{R}^r} \cdots \int \|x - v_i\|^2 F(x)\, dx. \quad (9)$$

is minimized. In this expression v_i is the winning prototype for each x, and will of course vary as x ranges over $\mathcal{R}^p$. A sample function of the optimization problem is $e = \|x - v_i\|^2$. An optimal set of v_i's can be approximated by applying local gradient descent to the sample function for every data point x from F (Kohonen, 1991). LVQ uses Euclidean distance, which corresponds to the update rule shown in eqn (7b), since $\nabla_v(\|x - v\|^2) = -2(x - v)$. The origin of this rule, its geometry, and a discussion of LVQ's relationship to sequential hard c-means appears in Pal, Bezdek and Tsao (1993).

LVQ attempts to minimize an objective function that places all of its emphasis on the winning

[2] The p components $\{v_{ij}\}$ of v_i are often regarded as weights or connection strengths of the edges that connect the p inputs to node i.

prototype for each data point. This is reflected in eqn (7b), which alters only the winner for each x submitted. This is reasonable, but it ignores global information about the geometric structure of the data that is represented in the remaining $(c-1)$ losing distances from x to the other prototypes. In this sense LVQ updating is somewhat like using the sup norm,

$$\|x - v\|_\infty = \max_{j=1}^{p}\{|x_j - v_j|\}.$$

to measure distance in $\mathscr{R}^p$. The extreme value in the absolute difference between pairs of coordinates dominates all others, and ignores them. In the same way, LVQ updating is a very harsh local strategy that ignores global relationships between the winner and the rest of the prototypes.

Using an inner product norm on $\mathscr{R}^p$ ameliorates the harshness of the sup norm by counting contributions from each pair of coordinate differences in the overall distance calculation. In the same way, we think that other nodes in the LVQ network should be allowed to influence the update of the winner, and perhaps, be updated themselves. This presents two questions. First, which other nodes should be accounted for (*what is the update neighborhood*)? Second, how much influence should each non-winner node that is recognized exert (*what is the learning rate distribution*)? Modifications of LVQ are usually motivated by a desire to solve one or both of these problems.

If the prototypes are to be useful quantizers, we think that information about structure in the data that is captured by the prototypes should be used as part of the criterion that determines the best prototypes. For LVQ and its relatives, information due to data point x is carried by *all* of the c distances $\{\|x - v_r\|\}$. Hence, it seems reasonable to define good quantizers in terms of a criterion that recognizes not only the local importance of the winner prototype, but also the importance of the other $(c-1)$ distances of non-winner prototypes relative to the winner distance. We circumvent the need to define an update neighborhood by expanding the neighborhood to include all c nodes. And we do it in the belief that vector quantizers based on both *local* (winner) and *global* (non-winner) information about the relationship of x to the prototypes will be better representatives of the overall structure in X than those based on local information alone. FLVQ and SCS recognize the winner as the most important prototype during the update cycle, but also give recognition to structural relationships between it and the other $c-1$ nodes. This answers the first question posed in the previous paragraph. The second problem—how much should each non-winner count?—is handled quite differently by these two algorithms. First, we consider the probabilistic scheme.

3. THE SOFT COMPETITION SCHEME

Yair, Zeger and Gersho (1992) recognized the limitations of LVQ just itemized. They proposed two vector quantization models, namely, a *Stochastic Relaxation Scheme* (SRS) and a *Soft Competition Scheme* (SCS) to address these issues. Both algorithms eliminate the need to define an update neighborhood by extending the update to all c nodes; and they use learning rates that are functions of the c distances $\{\|x - v_r\|\}$. In SRS each codevector is updated probabilistically, where the probability of updating a codevector is a function of its distance from the training vector presented at that instant. We will not consider SRS further in this note.

The other model given by Yair, Zeger and Gersho (1992) is the deterministic SCS algorithm (the algorithm is deterministic because its steps are not stochastically controlled, but it does use probabilities as part of the learning rates). In SCS all c prototypes are simultaneously updated by a scheme which directs them, like LVQ, towards the current training vector. The step size of each update is scaled by the probability of that prototype being the winner. At time (iterate) t, the probability of the ith prototype winning is defined as

$$p_{ik,t} = \frac{e^{-\beta_t \|x_k - v_{i,t-1}\|^2}}{\sum_{j=1}^{c} e^{-\beta_t \|x_k - v_{j,t-1}\|^2}}, \tag{10}$$

where

$$\lim_{t\to\infty}\{\beta_t\} = \infty.$$

The probability $p_{ik,t}$ is one factor in the SCS update equation. The choice for β_t is further refined by defining $\beta_t = \hat{\gamma}^{t/c}/T_0$. Here T_0 is regarded as an initial "temperature," and $\hat{\gamma}$ is a constant which Yair, Zeger, & Gersho (1992) stipulate should be greater than 1. The quantity $(1/\beta_t)$ is regarded as the temperature, so as $t \to \infty$, $\beta_t \to \infty$, and $T \to 0$, somewhat analogous to simulated annealing.

Next, let $n_{i,t} = n_{i,t-1} + p_{ik,t}$ (*approximately* the total number of times that v_i has been updated, this parameter is reset to 1 whenever iteration counter t is a perfect square). Yair, Zeger, & Gersho (1992) use this to define the other factor of their learning rates:

$$\eta_{ik,t} = \left(\frac{1}{n_{i,t}}\right) = \left(\frac{1}{n_{i,t-1} + p_{ik,t}}\right). \tag{11}$$

The overall learning rate for SCS is then taken as the product $\eta_{ik,t} \cdot p_{ik,t}$. This number replaces the LVQ multiplier α_t in eqn (7b), resulting in the SCS codevector update equation

$$v_{i,t} = v_{i,t-1} + (\eta_{ik,t} \cdot p_{ik,t})(x_k - v_{i,t-1}). \tag{12}$$

SCS starts with a low value of β_t (i.e., with approximately uniform $\{p_{ik,t}\}$), and then β_t is slowly increased with time. As a result, at the beginning of the process no codevector is strongly attracted to a particular class. With time codevectors become more strongly separated from each other as $p_{ik,t}$ begins to peak around the Euclidean winner, but at the same time $\eta_{ik,t} \to 0$. Thus in the limit (as iterate t goes to infinity) SCS behaves like the winner-take-all (LVQ) competition.

We point out that the innermost loop (on c) in the SCS algorithm below generates the c numbers $\{p_{ik,t}\}$, which satisfy $0 \le p_{ik,t} \le 1$ and

$$\sum_{i=1}^{c} p_{ik,t} = 1.$$

Consequently, $p_t(x_k) = (p_{1k,t}, p_{2k,t}, \ldots, p_{ck,t})^T$ is a probabilistic label vector for x_k, $p_t(x_k) \in N_{fc}$. Since each

$$\eta_{ik,t} = \left(\frac{1}{n_{i,t}}\right) \le 1,$$

the sum of the learning rates for fixed input vector x_k at any iterate t satisfies the following constraint:

$$0 < \sum_{i=1}^{c} \eta_{ik,t} \cdot p_{ik,t} \le 1.$$

We specify our implementation of SCS.

3.1. Soft Competition Scheme (Yair, Zeger & Gersho (1992))

Store. Unlabeled Object Data $X = \{x_1, x_2, \ldots, x_n\} \subset \mathcal{R}^p$

Pick. ☛ $1 < c < n$ ☛ $\hat{\gamma} > 1$ ☛ N = max. iterations ☛ T_0 = initial temperature

Guess. $V_0 = (v_{1,0}, v_{2,0}, \ldots, v_{c,0}) \in \mathcal{R}^{cp}$

Iterate. For $t = 1$ to N: $\beta_t = \hat{\gamma}^{t/c}/T_0$
For $k = 1$ to n
For $i = 1$ to c

$$p_{ik,t} = e^{-\beta_t \|x_k - v_{i,t-1}\|^2} \Big/ \sum_{j=1}^{c} e^{-\beta_t \|x_k - v_{j,t-1}\|^2} \quad (10)$$

If (t = a perfect square) $n_{i,t} = 1$
else $n_{i,t} = n_{i,t-1} + p_{ik,t}$

$$\eta_{ik,t} = (1/n_{i,t}) \quad (11)$$

$$v_{i,t} = v_{i,t-1} + (\eta_{ik,t} \cdot p_{ik,t})(x_k - v_{i,t-1}) \quad (12)$$

Next i
Next k
Next t
$V \leftarrow V_N$

Use. Prototypes V. For example, a crisp label matrix for X can be generated by applying the $1 - NP$ (nearest prototype) rule to the data:

$$u_{SCS_{ik}} = \left\{ \begin{array}{ll} 1; & \|x_k - v_i\| \le \|x_k - v_j\|,\ 1 \le j \le c, j \ne i \\ 0; & \text{otherwise} \end{array} \right\},$$
$$1 \le k \le n.$$

4. THE RELATIONSHIP OF SCS TO NORMAL MIXTURES

There is a strong relationship between SCS and statistical decision theory that is not discussed by Yair, Zeger, & Gersho (1992). This section is devoted to exposing the relationship between SCS learning rates and mixtures of normal distributions. To begin, assume that X is drawn from a mixed population of c p-variate statistical distributions, say with random vector variables $\{X_i\}$, that have $\{\pi_i\}$ as their *prior probabilities* and $\{g(x|i)\}$ as their class-conditional *probability density functions* (PDFs). That is

$$\pi_i \quad (13a)$$

prior probability of class i;

$$g(x|i) \quad (13b)$$

class conditional PDF of class i.

The convex combination of the class conditional PDFs, viz.,

$$f(x) = \sum_{i=1}^{c} \pi_i g(x|i) \quad (13c)$$

is itself a PDF whose distribution is called a *mixture* of the components $\{\pi_i g(x|i)\}$. Let the *posteriori probability* that, given x, x came from class i, be denoted by $\pi(i|x)$. Bayes rule relates the elements of eqn (13) to the probabilities $\{\pi(i|x)\}$ as follows:

$$\pi(i|x) = \frac{\pi_i g(x|i)}{f(x)}. \quad (14)$$

For a particular x_k, eqn (14) becomes, if all populations are known,

$$\pi(i|x_k) = \frac{\pi_i g(x_k|i)}{f(x_k)}.$$

For a sample of n points $X = \{x_1, x_2, \ldots, x_n\}$ assumed to be drawn from eqn (13c), *independently* and *identically distributed* (iid) with PDFs as in eqn (13), the $c \times n$ *posterior matrix* $\Pi = [\pi(i|x_k)] = [\pi_{ik}]$ of posterior probabilities satisfies constraints (2). Consequently, $\Pi \in M_{fcn}$, the space of constrained c-partitions of X. π_{ik} is a probability that plays much the same role in statistical pattern recognition that the fuzzy membership value u_{ik} plays in fuzzy pattern recognition.

In the general theory of mixtures, the component *probability density functions* (PDFs) $g(x|i)$ can be quite arbitrary (that is, continuous, discrete or both). Predictably, the case that dominates applications is when every component of $f(x)$ is *multivariate normal*. In this case the PDF $g(x|i)$ of component i has the familiar form

$$n(\mu_i, \Sigma_i) = g(x|i) = \frac{e^{-\frac{1}{2}\|x-\mu_i\|^2_{\Sigma_i^{-1}}}}{(2\pi)^{\frac{p}{2}}\sqrt{\det \Sigma_i}}, \tag{15a}$$

where

$$\mu_i = (\mu_{i1}, \mu_{i2}, \ldots, \mu_{ip})^T$$
is the *population mean vector* of class i; (15b)

and

$$\Sigma_i^{-1} = [\mathrm{cov}(X_i)]^{-1} = \begin{bmatrix} \sigma_{i,11} & \sigma_{i,12} & \cdots & \sigma_{i,1p} \\ \sigma_{i,21} & \sigma_{i,22} & \cdots & \sigma_{i,2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{i,p1} & \sigma_{i,p2} & \cdots & \sigma_{i,pp} \end{bmatrix}^{-1}, \tag{15c}$$

is the (positive definite) *population covariance* matrix of class i. $\sigma_{i,jk} = \mathrm{cov}(X_{ij}, X_{ik})$ is the population covariance between variables j and k for class i. The norm in (15a) is an inner product norm called the *Mahalanobis norm* computed in the usual way with the population parameters (μ_i, Σ_i), i.e., $\|x - \mu_i\|^2_{\Sigma_i^{-1}} = (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)$.

We make two simplifying assumptions about the mixture of normals obtained by substituting eqn (15a) into eqn (13c). For each class i, $1 \le i \le c$, we assume that

(i) $\pi_i = 1/c$ (16a)
(all classes are equally likely)

and

(ii) $\Sigma_i = \sigma^2 I$ (16b)
(all classes have covariance which is a scalar multiple of the identity).

From eqn (16b) $\Sigma_i^{-1} = 1/\sigma^2 I$ and $\sqrt{\det(\Sigma_i)} = \sigma$ for every class, so the Mahalonobis norm becomes a multiple of the Euclidean norm, $\|x - \mu_i\|^2_{\Sigma_i^{-1}} = 1/\sigma^2 \|x - \mu_i\|^2$. For this special case Bayes rule at eqn (14) takes the form

$$\pi(i|x) = \frac{(1/c)e^{-(1/(2\sigma^2))\|x-\mu_i\|^2)}/((2\pi)^{p/2}\sigma)}{\left[\sum_{j=1}^{c}\left(\frac{1}{c}\right)e^{-(1/(2\sigma^2))\|x-\mu_j\|^2)}\right] \Big/ ((2\pi)^{p/2}\sigma)}$$
$$= \frac{e^{-(1/(2\sigma^2))\|x-\mu_i\|^2)}}{\left[\sum_{j=1}^{c} e^{-(1/(2\sigma^2))\|x-\mu_j\|^2)}\right]}. \tag{17}$$

For a given x_k this becomes

$$\pi(i|x_k) = e^{-(1/(2\sigma^2))\|x_k-\mu_i\|^2)} \Big/ \sum_{j=1}^{c} e^{-(1/(2\sigma^2))\|x_k-\mu_j\|^2)}. \tag{18}$$

Comparing eqn (10) with eqn (18), if we define $\beta_t = 1/(2\sigma^2)$ and $v_{i,t-1} = \mu_i$ for $i = 1$ to c, then $p_{ik,t}$ and $\pi(i|x_k)$ are identical. Thus the component $p_{ik,t}$ of the SCS learning rate used by Yair, Zeger, & Gersho, (1992), can be interpreted as an estimate of the posterior probability of x_k being from class i under the assumptions in eqn (16). However, this choice for β_t does not ensure

$$\lim_{t\to\infty}\{\beta_t\} = \infty.$$

To achieve this t is used in the definition for β_t, i.e., $\beta_t = (\hat{\gamma}^{t/c}/T_0)$. Thus, at time t, if we take $\sigma^2 = (T_0\hat{\gamma}^{-t/c}/2)$, then $p_{ik,t}$ and $\pi(i|x_k)$ are identical, and $\beta_t = (1/2\sigma^2)$ ensures

$$\lim_{t\to\infty}\{\beta_t\} = \infty.$$

In summary, $p_{ik,t}$ can be interpreted as the posterior probability that x_k is from class i when all classes are equally likely, and class i is modeled as a p-variate normal distribution with parameters $(\mu_i = v_i, \Sigma_i = (T_0\hat{\gamma}^{-t/c}/2)I)$.

5. HARD AND FUZZY c-MEANS

The most widely used objective function model for fuzzy clustering in X is the weighted within groups sum of squared errors objective function J_m, which is used to define the constrained optimization problem

$$\min_{(U,V)}\left\{J_m(U, V; X) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m\|x_k - v_i\|^2_A\right\} \tag{19}$$

where $U \in M_{fcn}$, $V = (v_1, v_2, \ldots, v_c)$ is a vector of

(unknown) cluster centers (prototypes), $v_i \in \mathcal{R}^p$ for $1 \le i \le c$ and $\|x\|_A = \sqrt{x^T A x}$ is any inner product norm. Optimal partitions U^* of X are taken from pairs (U^*, V^*) that are local minimizers of J_m. Approximate optimization of J_m by the *Fuzzy c-Means algorithm* is based on iteration through the following necessary conditions for its local extrema:

5.1. Fuzzy c-Means Theorem (Bezdek, 1981)

If $\|x_k - v_i\|_A > 0$ for all i and k, then $(U, V) \in M_{fcn} \times \mathcal{R}^{cp}$ may minimize J_m only if, for $m > 1$,

$$u_{ik} = \left(\sum_{j=1}^{c} (\|x_k - v_i\|_A / \|x_k - v_j\|_A)^{\frac{2}{m-1}} \right)^{-1} \quad \forall i, k; \tag{20a}$$

and

$$v_i = \sum_{k=1}^{n} (u_{ik})^m x_k \Big/ \sum_{k=1}^{n} (u_{ik})^m \; \forall i. \tag{20b}$$

Perhaps the most popular algorithm for approximating solutions of eqn (19) is *alternating optimization* (AO) iteration through eqn (20a) and eqn (20b), commonly known as the FCM-AO algorithm (Bezdek, 1981). The most problematical choice for FCM-AO is the weighting exponent m, which can take any value in $(1, \infty)$. Most users of FCM experiment with this parameter, and find a value in the range [1.1, 5] that yields a suitable interpretation of substructure in the data. FLVQ as defined later provides two things: a strong link from FCM to LVQ, and a way for FCM users to find (roughly) a suitable value for m without extensive trials and errors.

Some limiting properties of equations (20) that are important for this study are (Bezdek, 1981):

$$\lim_{m \to 1} \left\{ \left(\sum_{j=1}^{c} \left(\|x_k - v_i\|_A \Big/ \|x_k - v_j\|_A \right)^{\frac{2}{m-1}} \right)^{-1} \right\} = \left\{ \begin{array}{ll} 1; & \|x_k - v_i\|_A < \|x_k - v_j\|_A \forall j \neq i \\ 0; & \text{otherwise} \end{array} \right\} \forall i, k. \tag{21a}$$

Using this result, we take the same limit in eqn (20b), obtaining:

$$\lim_{m \to 1} \left\{ \left(v_i = \sum_{k=1}^{n} (u_{ik})^m x_k \Big/ \sum_{k=1}^{n} (u_{ik})^m \right) \right\} = \frac{\sum_{x_k \in X_i} x_k}{n_i} \quad \forall i, \tag{21b}$$

where $X = X_1 \cup \ldots X_i \cup \ldots X_c$ is the hard c-partition of X defined by the right side of eqn (21a) with

$$\sum_{k=1}^{n} u_{ik} = n_i = |X_i|.$$

If we use these results in eqn (19), we have:

$$\lim_{m \to 1} \left(\min_{(U, V)} \left\{ J_m(U, V; X) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m \|x_k - v_i\|_A^2 \right\} \right) = \min_{(U, V)} \left\{ J_1(U, V; X) = \sum_{k=1}^{n} \sum_{i=1}^{c} u_{ik} \|x_k - v_i\|_A^2 \right\}. \tag{22}$$

$J_1(U, V; X)$ is the classical within-groups sum of squared errors objective function. Eqn (22) is the *hard c-means* (HCM) *model*. Moreover, the right sides of eqn (21a) and eqn (21b) are the necessary conditions for local extrema of J_1. Observe that crisp membership assignments in the right side of eqn (21a) use the 1-*NP* rule at eqn (6). HCM-AO clustering is iteration through the right hand sides of eqn (21a) and eqn (21b).

In a weak sense then, the HCM model (objective function + algorithm) is a bridge between LVQ and FCM. However, HCM and FCM are batch algorithms, whereas the sequential implementation of LVQ given earlier is more like the sequential hard-c means model (Pal, Bezdek, & Tsao, 1993). For the present article, the most important points are that each of these algorithms generates c prototypes, and they are all driven by objective functions that involve sums of squared errors. The main differences between LVQ, SCS, HCM, FCM and FLVQ lies in the way squared errors are weighted and the way prototypes are updated.

To properly analyze the behavior of FLVQ we will also need the limits of eqn (20) as m goes to infinity:

$$\lim_{m \to \infty} \left\{ \left(\sum_{j=1}^{c} \left(\|x_k - v_i\|_A \Big/ \|x_k - v_j\|_A \right)^{\frac{2}{m-1}} \right)^{-1} \right\} = \frac{1}{c} \quad \forall i, k; \tag{23a}$$

$$\lim_{m \to \infty} \left\{ \left(v_i = \sum_{k=1}^{n} (u_{ik})^m x_k \Big/ \sum_{k=1}^{n} (u_{ik})^m \right) \right\} = \frac{\sum_{k=1}^{n} x_k}{n} = \bar{v} \quad \forall i. \tag{23b}$$

where $\bar{v} = \sum_{x \in X} x/n$ is the grand mean of X.

6. FUZZY LEARNING VECTOR QUANTIZATION

In this section we review FLVQ and provide a new analysis of its behavior as a function of m using the limits shown in Section 5. The connection between

FCM and LVQ was first discussed by Huntsberger & Ajjimarangsee (1990) who suggested fuzzification of LVQ by replacing the learning rates $\{\alpha_{ik,t}\}$ usually found in rules such as eqn (7b) with the fuzzy membership values $\{u_{ik,t}\}$ computed with FCM formula eqn (20a). While this approach was innovative, it was to some extent unmotivated. Moreover, their method still required choosing m, and it seemed to improperly mix the objectives of LVQ (vector quantization) and FCM (clustering). This led to a revision of LVQ that was first called FKCN (Tsao, Bezdek, & Pal, 1992) that has subsequently become known as FLVQ.

As noted earlier, the choice of m for the FCM model is very important. Eqn (21a) shows that when m is small (close to 1), FCM-AO tends to produce almost crisp label vectors. Each column of U from FCM-AO must sum to 1. Because of this, if updates are based on some function of eqn (20a) and one u_{ik} is close to 1, the update for node i may be very large compared to the other updates. If, additionally, the current prototypes from FCM-AO have an unfavorable geometry compared to the central tendencies of clusters in the data, some prototypes may move rapidly towards a cluster, while others may move but little. This effect is illustrated in Figure 4 for the data set $X = X1 \cup X2$.

In Figure 4, prototype v_1 is closer to every point in X than v_2 is. The result of this is that for any m at $c = 2$, the class 1 memberships $\{u_{1k}\}$ of every point in X computed with eqn (20a) will be higher than the class 2 memberships $\{u_{2k}\}$. Since $u_{1k} + u_{2k} = 1$ for all k, the two rows of membership matrices produced with eqn (20a) for any m will look like this:

$$U(m) = \begin{bmatrix} \leftarrow \cdots (> 0.5) \cdots \rightarrow \\ \leftarrow \cdots (< 0.5) \cdots \rightarrow \end{bmatrix} \xrightarrow{m \to 1}$$

$$U(1) = \begin{bmatrix} \leftarrow \cdots (\rightarrow 1) \cdots \rightarrow \\ \leftarrow \cdots (\rightarrow 0) \cdots \rightarrow \end{bmatrix}.$$

So, when m is close to 1, memberships of points in both X_1 and X_2 in class 1 will be close to 1. The effect of this is that prototype v_1 in Figure 4 will migrate towards the grand mean $\bar{v}$ of X, and v_2 will not change much.

On the other hand, if m is large (say > 7) all of the

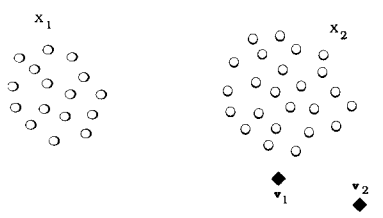

FIGURE 4. A geometric situation where a low value of m may produce bad prototypes.

u_{ik}'s will be nearly $1/c$ as implied by eqn (23a). In this case both prototypes in Figure 4 will be pulled towards the data very slowly by FCM-AO. This will happen because every $(u_{ik,t})^m \approx (1/c^m)$. So when m is large, every prototype will be updated to almost the same very small extent (e.g., with $c = 3$ and $m = 7$, every $u^m_{ik,t} \cong 0.0004$). This will also be the case for any competitive learning scheme whose update rate is a monotonic function of the $\{u_{ik,t}\}$.

Thus, neither low nor high values of m seem desirable. However, if we start with a high value of m, and then slowly reduce it *during iteration*, this undesirable situation is avoided. Motivated by this behavior, we define:

$$\omega_{ik,t} = (u_{ik,t})^{m_t} = \left(\sum_{j=1}^{c} (\|x_k - v_{i,t}\|_A / \|x_k - v_{j,t}\|_A)^{\frac{2}{m_t - 1}} \right)^{-m_t} \quad \forall i, k, \quad (24a)$$

and

$$v_{i,t} = v_{i,t-1} + \sum_{k=1}^{n} \omega_{ik,t}(x_k - v_{i,t-1}) \Big/ \sum_{s=1}^{n} \omega_{is,t} \quad \forall i, \quad (24b)$$

where

$$m_t = m_0 + t[(m_f - m_0)/T] = m_0 + t\Delta m;\ m_f,\ m_0 > 1;\ t = 1, 2, \ldots N. \quad (24c)$$

Eqn (24b) can be rewritten as

$$v_{i,t} = \sum_{k=1}^{n} \omega_{ik,t} x_k \Big/ \sum_{s=1}^{n} \omega_{is,t}.$$

Comparing this to eqn (20b), eqn (24c) asserts that when $m_0 = m_f = m$ is fixed, FLVQ is FCM-AO. The learning rates in eqn (24a) were chosen so that this would be true. Our competitive learning scheme based on eqn (24) has three objectives: (i) to overcome the two problems we identified for LVQ (which nodes to update and how to use the non-winner prototypes in the determination of learning rates); (ii) to circumvent (to some extent) the problem of how to choose m for FCM; and (iii) to provide a substantial link between the c-means and LVQ families.

Since m_t in eqn (24c) is variable, we can have three families of Fuzzy LVQ or FLVQ algorithms, depending on the choice of the initial (m_0) and final (m_f) values of m. For $t \in \{1, 2, \ldots, N\}$,

$$m_0 > m_f \Rightarrow \{m_t\} \downarrow m_f: \text{Descending FLVQ} = \downarrow \text{FLVQ} \quad (25a)$$

$$m_0 < m_f \Rightarrow \{m_t\} \uparrow m_f: \text{Ascending FLVQ} = \uparrow \text{FLVQ} \quad (25b)$$

$$m_0 = m_f \Rightarrow m_t \equiv m_0 \equiv m: \text{FLVQ} \equiv \text{FCM}. \quad (25c)$$

We have included ↑FLVQ here for completeness. However, its properties as functions of m_t seem counter to the intuitively desirable properties shared by SCS and ↓FLVQ. We do not recommend the use of ↑FLVQ for this reason. Here we concentrate on and describe the implementation of ↓FLVQ based on eqn (24) which is used in the numerical examples of Section 7.

6.1. The Descending Fuzzy LVQ (↓FLVQ) Algorithm (Tsao, Bezdek, & Pal, 1992)

Store. Unlabeled Object Data $X = \{x_1, x_2, \ldots, x_n\} \subset \mathscr{R}^p$

Pick. ☛ $1 < c < n$ ☛ $\| \ \|_A$ ☛ N = max. iterations ☛ $\varepsilon > 0$ ☛ $7 > m_0 > m_f > 1.1$

Guess. $V_0 = (v_{1,0}, v_{2,0}, \ldots, V_{c,0}) \in \mathscr{R}^{cp}$

Iterate. For $t = 1, 2, \ldots, N$:

$m_t = m_0 + t[(m_f - m_0)/T] = m_0 + t\Delta m$

For $k = 1$ to n:

a. $\omega_{ik,t} = (u_{ik,t})^{m_t}$

$$= \left(\sum_{j=1}^{c} (\|x_k - v_{i,t-1}\|_A / \|x_k - v_{j,t-1}\|_A)^{\frac{2}{m_t - 1}} \right)^{-m_t} \quad \forall i, k$$

b. $$v_{i,t} = v_{i,t-1} + \sum_{k=1}^{n} \omega_{ik,t}(x_k - v_{i,t-1}) \Big/ \sum_{s=1}^{n} \omega_{is,t}$$

c. If $E_t = \|V_t - V_{t-1}\|_{err} = \sum_{i=1}^{c} \|v_{i,t} - v_{i,t-1}\|$

$$= \sum_{i=1}^{c} \sum_{j=1}^{p} |v_{ij,t} - v_{ij,t-1}| < \varepsilon \text{ stop; else}$$

Next k

Next t

$V \leftarrow V_t$; $U \leftarrow U_t$

Use. Prototypes V and/or U

As with LVQ and SCS, the prototypes produced by ↓FLVQ can be used with eqn (6) to produce a crisp partition of X, and also to define a 1-NP classifier. Our implementation of ↓FLVQ is necessarily batch, and this preserves its relationship to FCM-AO. Unlike LVQ and SCS, which are both terminated by iterate limit N, FCM-AO and ↓FLVQ are terminated when successive estimates for V become close, as measured by $\|V_t - V_{t-1}\|_{err}$.

Another difference worth noting is that unlike LVQ and FCM-AO, ↓FLVQ does not optimize a fixed objective function. All we can say about this is that since ↓FLVQ uses eqn (20) at each iteration with $m = m_t$, every full step of ↓FLVQ uses a pair (U_t, V_t) that are necessary for a local extrema of J_{m_t}. We point out the constraints $7 > m_0 > m_f > 1.1$ in our specification of ↓FLVQ. In our experience these are useful limits for m that should prevent numerical instability. In other words, stay away from 1 and infinity.

The c numbers $\{u_{ik,t}\}$ satisfy $0 \le u_{ik,t} \le 1$ and

$$\sum_{i=1}^{c} u_{ik,t} = 1.$$

Consequently, the vector $u_t(x_k) = (u_{1k,t}, u_{2k,t}, \ldots, u_{ck,t})^T$ is a fuzzy label vector for x_k, $u_t(x_k) \in N_{fc}$. This means that the sum of the ↓FLVQ learning rates for input vector x_k at any iterate t satisfies the same constraint as the SCS learning rates:

$$0 < \sum_{i=1}^{c} \omega_{ik,t} \le 1.$$

To understand how m_t acts to control the distribution and values of the learning rates $\{\omega_{ik,t}\}$ in FLVQ, we discuss ↓FLVQ in more detail. The general situation can be understood by examining the learning rates at eqn (24a) for fixed c, $\{v_{i,t}\}$ and m_t. In this case,

$$\omega_{ik,t} = \left(\kappa \|x_k - v_{i,t}\|_A^{2/(m_t - 1)}\right)^{-m_t} = \kappa^{-m_t}\left(\|x_k - v_{i,t}\|_A^{-2m_t/(m_t - 1)}\right), \quad (26)$$

where

$$\kappa = \sum_{j=1}^{c} (1/\|x_k - v_{j,t}\|_A)^{2/(m_t - 1)}$$

is a positive constant. From eqn (26) we see that the contribution of x_k to the next update of the node weights is inversely proportional to their distances from it, so the winner for this k is the $v_{i,t-1}$ closest to x_k. Larger values of m_t lead to fuzzier values of $u_{ik,t}$ (values closer to $1/c$), and $\Sigma u_{ik,t} = 1 \Rightarrow \Sigma \omega_{ik,t} \le 1$. So, in the initial stages of ↓FLVQ large values of m_t (near m_0) yield updates with lower individual learning rates.

In the initial stages of SCS (for low values of t) $p_{ik,t} \approx 1/c$, and since the counters $\{n_{i,t}\}$ all start at 1, at the beginning of the SCS learning process each codevector is (more or less) updated to the same

extent. In other words $(\eta_{i,t} \cdot p_{ik,t}) \approx (\eta_{j,t} \cdot p_{jk,t})$ for all i and j at low values of t. What happens for ↓FLVQ? In this case we start with a high value of $m = m_0$. For high values of m, $u_{ik,t} \approx 1/c \forall i$, and as a result $\omega_{ik,t} = (u_{ik,t})^{m_t} \approx \omega_{jk,t} = (u_{jk,t})^{m_t}$ for all i and j at low values of t. Thus, in ↓FLVQ all c prototypes will have about the same importance at the beginning of iteration, with learning rates at each x_k that are roughly uniformly distributed across the c nodes during updates. Thus, ↓FLVQ and SCS start with very similar configurations.

As iteration continues $p_{ik,t}$ for SCS and $u_{ik,t}$ for ↓FLVQ both tend to peak at the Euclidean winner. For SCS, $p_{ik,t} \to 1$ when node i is the winner, but $\eta_{ik,t} \to 0$ so the overall SCS learning rate $\eta_{ik,t} \cdot p_{ik,t} \to 0$. On the other hand, for ↓FLVQ $u_{ik,t} \to 1$ when node i is the winner but since $m_t \to 1$, the overall learning rate for this method also goes to 1, $\omega_{ik,t} = u_{ik,t}^{m_t} \to 1$. As $m_t \searrow m_f$ (m_t gets closer to 1), more and more of the update is given to the winner node. In other words, the lateral distribution of learning rates is a function of t, which in ↓FLVQ sharpens at the winner node (for each x_k) as $m_t \searrow m_f$. Indeed, the learning rate characteristics of ↓FLVQ are roughly opposite to the usual behavior imposed on them by other competitive learning schemes. In LVQ and SCS all c learning rates at x_k decrease towards 0 as t increases, but in ↓FLVQ, the winner learning rate tends to increase towards 1 during learning, while the other c-1 rates tend towards zero at each x_k. Thus, SCS behaves much more like LVQ as iteration proceeds than ↓FLVQ does.

7. NUMERICAL EXAMPLES

In this section we illustrate and compare LVQ, SCS and FLVQ by calculating centroids obtained by applying these three algorithms to Anderson's IRIS data (Anderson, 1935). IRIS contains 50 (physically labeled) vectors in $\mathcal{R}^4$ for each of $c = 3$ classes of IRIS subspecies. IRIS has been used in many papers to illustrate various clustering (unsupervised) and classifier (supervised) designs. One way we can assess relative performance is to compare the numerical values of terminal centroids to the physically labeled subsample means.

A second way to validate prototype generating algorithms with this data is to find three terminal prototpyes for IRIS, relabel them if necessary so that the algorithmic labels correspond to the physical class labels, and then use them as a basis for the 1-*NP* classifier at eqn (6). Submitting all 150 points in IRIS to $D_{NP,V}$ and counting the mistakes results in an estimate of $D_{NP,V}$'s error rate for that V. This is called the resubstitution error rate. We know this error rate is a little optimistic, but it is fine for comparisons of competing designs. Typical resubstitution error rates for IRIS with supervised designs are 0–5 mistakes; and for unsupervised designs such as the three discussed here, around 16 mistakes.

We used the two initializations shown in Table 1. The vectors shown as initialization I_1 are the sample means of the physically labeled points in the three 50 point subsets of IRIS. The second initialization, I_2 in Table 1, is computed using the following method. For data set $X = \{x_1, \ldots, x_n\} \subset \mathcal{R}^p$, let data point k and initial prototype i be $x_k = (x_{k1}, x_{k2}, \ldots, x_{kp})^T$ and $v_i = (v_{i1}, v_{i2}, \ldots, v_{ip})^T$ respectively. Compute the feature ranges

$$\text{Minimum of feature } j: \quad m_j = \underbrace{\min}_{k}\{x_{kj}\}: \quad j = 1, 2, \ldots, p; \tag{27a}$$

$$\text{Maximum of feature } j: \quad M_j = \underbrace{\max}_{k}\{x_{kj}\}: \quad j = 1, 2. \tag{27b}$$

With these, compute the jth component of the ith initial prototype as:

$$v_{ij} = m_j + (i-1)\left(\frac{M_j - m_j}{c-1}\right); \quad i = 1, 2, \ldots, c; j = 1, 2, \ldots, c. \tag{28}$$

Formula (28) disperses initial prototype values uniformly along each feature range $[m_j, M_j]$. For example, $v_1 = m = (m_1, m_2, \ldots, m_p)^T$, $v_c = M = (M_1, M_2, \ldots, M_p)^T$, and so on.

None of the algorithms studied here use class information (that is, are supervised) during learning (i.e., while finding the prototypes). The confusion

TABLE 1
Two Initializations for the Numerical Experiments

Initial Centroids I_1 = (Means)					Initial Centroids I_2			
5.006	3.428	1.462	0.246	$\leftarrow v_{1,0} \rightarrow$	4.300	2.000	1.000	0.100
5.936	2.770	4.260	1.326	$\leftarrow v_{2,0} \rightarrow$	6.100	3.200	3.950	1.300
6.588	2.974	5.552	2.026	$\leftarrow v_{3,0} \rightarrow$	7.900	4.400	6.900	2.500

TABLE 2
Centroids and Outputs of Sample Mean, LVQ, SCS and ↓FLVQ 1-*NP* Classifiers on the IRIS data when initialized with l_1

Initial Centroids l_1				Final Centroids l_1				Confusion Matrix		
5.006	3.428	1.462	0.246	5.006	3.428	1.462	0.246	50	0	0
5.936	2.770	4.260	1.326	5.936	2.770	4.260	1.326	0	46	4
6.588	2.974	5.552	2.026	6.588	2.974	5.552	2.026	0	7	43
				Final Centroids: LVQ $N = 50$, $\alpha_0 = 0.6$				Confusion Matrix		
	Init. Same as Above			4.999	3.420	1.463	0.248	50	0	0
				5.873	2.746	4.366	1.414	0	47	3
				6.813	3.079	5.682	2.083	0	13	37
				Final Centroids: SCS $N = 50$, $\hat{\gamma} = 1.3$, $T_0 = 40$				Confusion Martrix		
	Init. Same as Above			5.006	3.425	1.465	0.247	50	0	0
				5.884	2.743	4.370	1.414	0	47	3
				6.776	3.047	5.634	2.031	0	13	37
				Final Centroids: ↓FLVQ $N = 50$, $m_0 = 5$, $m_f = 1.5$				Confusion Matrix		
	Init. Same as Above			5.006	3.420	1.474	0.252	50	0	0
				5.884	2.748	4.371	1.411	0	47	3
				6.821	3.064	5.697	2.063	0	14	36

TABLE 3
Centroids and Outputs of the SCS 1-*NP* Classifier on the IRIS Data

Set	Init.	$\hat{\gamma} = 1.30$, $T_0 = 40$				Confusion Matrix		
A	l_1	5.006	3.425	1.465	0.247	50	0	0
		5.884	2.743	4.370	1.414	0	47	3
		6.776	3.047	5.634	2.031	0	13	37
		$\hat{\gamma} = 1.15$, $T_0 = 40$				Confusion Matrix		
B	l_1	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
		$\hat{\gamma} = 1.30$, $T_0 = 40$				Confusion Matrix		
C	l_2	5.006	3.425	1.465	0.247	50	0	0
		5.884	2.743	4.370	1.414	0	47	3
		6.776	3.047	5.634	2.031	0	13	37
		$\hat{\gamma} = 1.15$, $T_0 = 40$				Confusion Matrix		
D	l_2	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
		$\hat{\gamma} = 1.30$, $T_0 = 60$				Confusion Matrix		
E	l_2	5.008	3.378	1.548	0.284	50	0	0
		6.272	2.884	4.945	1.690	3	0	47
		6.292	2.884	4.945	1.690	0	0	50
		$\hat{\gamma} = 1.30$, $T_0 = 70$				Confusion Matrix		
F	l_2	5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0
		5.843	3.057	3.758	1.199	50	0	0

matrices listed in Tables 2 and 3 are found by applying the nearest prototype classifier based on the final prototoypes from a particular algorithm to each of the 150 points in IRIS. The ij-th entry of the confusion matrix records the number of times real physical label i was instead given label j by the algorithm.

Table 2 reports the results of nearest prototype classification of IRIS by using the centroids recommended by LVQ, SCS and FLVQ in 1-*NP* rule 6. The first set of rows shows the confusion matrix associated with $D_{NP,V}$ when $V_{\text{final}} = V + 0 = I_1$, the physical subsample means. If we know the labels, the sample means yield a classifier that commits 11 errors; 4 class 2 points are labeled class 3; and 7 class 3 points are labeled class 2. All three algorithms produce very similar centroids. The confusion matrices for the LVQ and SCS based 1-*NP* designs are identical, showing 16 resubstitution errors. FLVQ is very nearly the same, committing one more error than LVQ and SCS on a class 3 data point.

It is shown elsewhere that LVQ can terminate at very bad centroids when initialized with vectors outside the convex hull of the IRIS data (Pal, Bezdek, & Tsao, 1993). To test stability of the results in Table 2 to V_0, the initialization of the prototypes, we made another set of runs with the same algorithmic parameters as shown in Table 2, but with the initialization I_2 shown in Table 1. The centroids produced by all three algorithms were identical (to three decimal places) to those shown in Table 2. This does not establsh that these algorithms are insensitive to initialization, but it gives us some confidence that the IRIS data are rather well structured. Thus, there are combinations of initializations and algorithmic parameters for all three algorithms that produce very similar and predictable results.

Our implementation of SCS found it very sensitive to the choice of and interaction between $\hat{\gamma}$ and T_0. Yair, Zeger, & Gersho (1992, p. 303) state that "It is important that the initial temperature not be chosen too large, for in such a case the codevectors may tend to merge together, yielding a poor codebook." Elsewhere, however Yair, Zeger, & Gersho (1992, p. 302), state that "The algorithm starts with a low value of β [our β_t] , for which $P_n(i)$ [our $p_{ik,t}$] is approximately uniform. That is, for low values of β (high temperatures) the codevectors are not yet attracted to a certain partition, and they all migrate towards the data presented." These two statements suggest that there is a range over which T_0 yields good results.

Table 3 studies the effect on SCS outputs to the parameters $\hat{\gamma}$ and T_0. All runs reported in this table used $N = 50$; rows A are repeated from Table 2. First compare A, B, C and D, all of which have $T_0 = 40$. Changing $\hat{\gamma}$ from 1.30 to 1.15 using either I_1 or I_2 has the dramatic result of forcing all three SCS centroids to terminate at $\bar{v} = (5.843, 3.057, 3.758, 1.199)^T$, the grand mean of IRIS. This has the predictably bad effect on the 1-*NP* design of it committing 100 mistakes in both cases.

Next, compare sets C and F in Table 3 to see that it is not just a change of $\hat{\gamma}$ that has this effect on SCS, for in this case you will see that the same result occurs with $\hat{\gamma}$ fixed at 1.30 but T_0 increased from 40 to 70. Finally, look at sets C, E and F for I_2 and $\hat{\gamma} = 1.30$ fixed. Intermediate between the good result at $T_0 = 40$ and the worst result at $T_0 = 70$ is the case $T_0 = 60$, for which SCS terminates with a good estimate of the first centroid, but identical vectors for the second and third prototypes, resulting in a 1-*NP* error rate of 50 mistakes. Table 3, and many other experiments with other values for $\hat{\gamma}$ and T_0 not reported here, suggest that SCS is very sensitive to good choices for these two parameters.

8. ON THE RELATIONSHIP BETWEEN c-MEANS AND COMPETITIVE LEARNING SCHEMES

In eqn (19) and eqn (20) the weighting exponent m for J_m is fixed, but in eqn (24) it is a variable. Since m is replaced by a parameter whose value depends on the number of iterations that have elapsed, m_t plays a role that is somewhat analogous to $\alpha_{ik,t}$ in LVQ. To see this, remember that

$$\sum_{i=1}^{c} u_{ik,t} = 1$$

for each x_k in X. In consequence, the learning rates in eqn (24a) that are applied to all c nodes via eqn (24b) for each x_k are dependent on each other, and themselves must satisfy the condition

$$\sum_{i=1}^{c} \omega_{ik,t} \leq 1.$$

The effect of controlling the learning rates this way is best understood by considering a simple example. Suppose $c = 5$ and $m_t = 4$ at some iterate. Two label vectors for x_k for the five nodes, and the resultant learning rate distributions they induce via eqn (24a) are shown below:

$$u(x_k) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \omega(x_k) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \text{ for any } m_t; \qquad (29a)$$

and

$$\hat{u}(x_k) = \begin{pmatrix} 0.1 \\ 0.6 \\ 0.0 \\ 0.2 \\ 0.1 \end{pmatrix} \Rightarrow \hat{\omega}(x_k)$$

$$= \begin{pmatrix} 0.0001 \\ 0.1296 \\ 0.0000 \\ 0.0016 \\ 0.0001 \end{pmatrix} (m_t = 4 \text{ is illustrated}). \quad (29b)$$

In eqn (29a) node 2 is the crisp winner since it receives all of the membership of this data point in any of the five clusters. From eqn (24a) it follows that for any value of m_t the learning rates applied to this data point will also be crisp, and will be the same as the labels used to compute them, as shown in eqn (29a). Thus, when a single node can win all of the membership, none of the non-winner nodes are allowed to influence the update in eqn (24b) for that data point. In this special case, FLVQ reverts to an LVQ-like strategy, but only for data points that have crisp memberships.

On the other hand, if the distribution of memberships for x_k is truly fuzzy, as in eqn (29b), exponentiation of the membership values by m_t has a noticeable effect on the role played by each node in the update scheme. The winner node in eqn (29b) in the sense of maximum membership (which is, as previously noted, also the minimum distance prototype) is still node 2. But in this second case, non-winner nodes with non-zero memberships will also participate in the determination of how much to change their corresponding weight vectors for that data point. Finally, if $m_0 = m_f$ then clearly FCM = FLVQ.

If all n membership columns in U from the FCM formula (20a) were crisp, eqn (24b) would become a batch version, LVQ-style update, with

$$v_{i,t} = v_{i,t-1} + \sum_{x_k \in X_i} (x_k - v_{i,t-1})/n_{i,t},$$

where $n_{i,t}$ is the number of points in the ith crisp cluster of X at iterate t. The previous estimate for $v_{i,t-1}$ can be eliminated from this last equation by distributing the sum over the minus sign, leaving the HCM update formula on the right side of eqn (21b). Suppose eqn (7b) in LVQ is replaced with this batch update formula, and calculation of U_{LVQ} as in eqn (8) is required at each pass (remember that LVQ does not do so) through the data. Call this *extended batch* LVQ (EBLVQ). Then FLVQ reduces to EBLVQ whenever U is crisp, and further, EBLVQ is precisely HCM. In this sense FLVQ is a true generalization of both LVQ and HCM that integrates their models in perhaps the strongest possible way.

9. CONCLUSIONS

We think that structural information due to data point x is carried by all of the c distances $\{\|x - v_r\|\}$. We have discussed two soft relatives of LVQ that define good prototypes in terms of criteria that recognize not only the local importance of the winner (minimum distance) prototype, but also the global importance of the other $(c-1)$ distances of non-winner prototypes relative to the winner distance. We believe that vector quantizers based on both *local* (winner) and *global* (non-winner) information about the relationship of x to the prototypes will be better representatives of the overall structure in X than those based on local information alone.

SCS and ↓FLVQ both recognize the winner as the most important prototype during the update cycle, but also give recognition to structural relationships between it and the other c-1 nodes. Both of these algorithms expand the update neighborhood to include all c nodes; and both allow all c prototypes to participate in setting the amount by which each node gets updated at every pass through the data.

In ↓FLVQ and SCS, adjustments to each prototype are made inversely proportional to its distance from x. In both schemes the largest share of each update is accorded to the winner, and proportionately smaller shares are given to each of the other c-1 non-winner nodes. One of the most intriguing properties of ↓FLVQ is that it provides a means for circumventing the question of how to choose the weighting exponent m in FCM-AO.

We have shown that the learning rates for SCS are related to estimates for the posterior probabilities of a certain mixture of normal distributions. Further, we showed that FLVQ and Fuzzy c-Means are equivalent in one special case, and as a subcase of this, that FLVQ reduces to Hard c-Means when the partition generated is crisp.

In practice, our experience is that ↓FLVQ is much more stable to changes in its parameters than SCS. The complexity of these two algorithms makes it hard to offer more than a conjecture about this; here is ours. First, batch algorithms seem inherently more stable to small changes in algorithmic parameters than sequential ones, simply because the effect of changes is spread across all n data points before it is felt. And secondly, the control strategy of SCS may cause some instability. We draw attention to the line in the SCS algorithm that resets the counter $n_{i,t}$ to 1 at every t a perfect

square. The effect of this is to modulate the learning rates so that the distribution of $\eta_{ik,t} \cdot p_{ik,t}$ looks like a sawtooth wave bounded by 1 from above and 0 from below. The reset occurs at the iterate numbers 1, 4, 9, 16, 25, . . . , t^2 . . . , so the width between pulses is successively longer. For large enough t, the factor $\eta_{ik,t} = 1/n_{i,t}$ tends towards zero, and in the limit will go to zero. Nonetheless, the behavior of $\eta_{ik,t} \cdot p_{ik}$, for finite iterate limit N is quite different than the usual Kohonen learning rate, i.e., $\alpha_t = \alpha_0(1 - t/N)$, which goes to zero smoothly with t. Our examples only ran SCS for $N = 50$ sweeps through IRIS. Consequently, the learning rates were reset 7 times. It might be that longer learning times would eradicate this, but we doubt it. Once SCS settled at the grand mean of IRIS in our examples, it stayed there. This behavior is an interesting facet of SCS that might be profitably pursued in a future investigation.

REFERENCES

Anderson, E. (1935). The IRISes of the Gaspe peninsula. *Bulletin American IRIS Society,* **39**, pp. 2–15.

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms.* New York: Plenum.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis.* New York: Wiley.

Gersho, A., & Gray, R. (1992). *Vector quantization and signal compression.* Boston: Kluwer.

Hartigan, J. (1975). *Clustering algorithms.* New York: Wiley.

Haykin, S. (1994). *Neural networks: A comprehensive foundation.* New York: Macmillan.

Huntsberger, T., & Ajjimarangsee, P. (1990). Parallel Self-Organizing Feature Maps for Unsupervised Pattern Recognition, *International Journal of General Systems,* **16**, p. 357.

Jain, A., & Dubes, R. (1988). *Algorithms that cluster data,* Englewood Cliffs: Prentice Hall.

Kohonen, T. (1989). *Self-organization and associative memory* (3rd ed.), Berlin: Springer Verlag.

Kohonen, T. (1991). Self-organizing maps: optimization approach. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas (Eds.), *Artificial neural networks,* New York: Elsevier (pp. 981–990).

Krishnapuram, R., & Keller, J. (1993). A Possibilistic Approach to Clustering. *IEEE Trans. Fuzzy Systems,* **1**(2), 98–110.

Pal, N. R., Bezdek, J. C., & Tsao, E. (1993). Generalized Clustering Networks and Kohonen's Self-Organizing Scheme. *IEEE Trans. Neural Networks,* **4**(4), 549–558.

Titterington, D., Smith, A., & Makov, U. (1985). *Statistical analysis of finite mixture distributions.* New York: Wiley.

Tsao, E. C. K., Bezdek, J. C., & Pal, N. R. (1992). Fuzzy Kohonen Clustering Networks. *Pattern Recognition,* **27**(5), 1994, 757–764.

Yair, E., Zeger, K., & Gersho, A. (1992). Competitive learning and soft competition for vector quantizer design, *IEEE Trans. SP,* **40**(2), 294–309.