Fuzzy Sets and Systems 18 (1986) 237–256
North-Holland

GENERALIZED *k*-NEAREST NEIGHBOR RULES*

James C. BEZDEK, Siew K. CHUAH
Computer Science Department, University of South Carolina, Columbia, SC 29208, USA

David LEEP
Boeing Aerospace Company, Seattle, WA 98124, USA

Received February 1985

This paper discusses a suitable framework for generalizing the k-nearest neighbor (k-NNR) algorithms to cases where the design labels are not necessarily crisp, i.e., not binary-valued. The proposed framework imbeds all crisp k-NNR's into a larger structure of fuzzy k-NNR's. The resultant model enables neighborhood voting to be a *continuous* function of local labels at a point to be classified. We emphasize that the decision itself may be crisp even when a fuzzy k-NNR is utilized. The usefulness of this extension of the conventional technique is illustrated by comparing the observed error rates of four classifiers (the hard k-NNR, two fuzzy k-NNR's, and a fuzzy 1-nearest prototype rule (1-NPR)) on three data sets: Anderson's Iris data, and samples from (synthetic) univariate and bivariate normal mixtures. Our conclusions: all four designs yield comparable (usually within 4%) error rates; the Fuzzy c-Means (FCM) based k-NNR is usually the best design; the FCM/1-NPR is the most efficient and perhaps most useful of the four designs; and finally, that generalized NNR's are an important and useful extension of the conventional ones.

Keywords: Classifier design, c-Means algorithm, Fuzzy sets, Nearest neighbors, Pattern recognition.

1. Introduction and conclusions

A conventional pattern recognition system is a decision rule that assigns one of c labels to (some representation of) each test sample submitted to the rule. In this paper objects will be characterized by feature vectors $\boldsymbol{x} \in R^s$ and the set of class labels by $I_c = \{1, 2, \ldots, c\}$. Any function $D: R^s \to I_c$ is a classifier function or decision function on R^s: its action partitions R^s into c disjoint regions, the inverse images of the elements of I_c. If one desires to include a rejectance (no decision) class, it can be realized by taking $c = c + 1$.

Statistical decision theory presumes that the c object classes captured by D are jointly distributed as a mixture of component probability density functions (pdf):

$$f(\boldsymbol{x}) = \sum_{j=1}^{c} p_j g(\boldsymbol{x} \mid j) \text{ is the mixture,} \tag{1a}$$

with

$$0 \leq p_j \leq 1, \quad p_j \text{ the prior probability of class } j \ \forall j, \tag{1b}$$

$$g(\boldsymbol{x} \mid j) = \text{the pdf of class } j \ \forall j. \tag{1c}$$

Bayes' rule relates the three factors in (1a) to each other as follows:

$$p(j \mid \boldsymbol{x}) = p_j g(\boldsymbol{x} \mid j)/f(\boldsymbol{x}), \quad 1 \leq j \leq c. \tag{2}$$

In the sequel we shall denote the posterior vector whose elements are the left side of (2) as $\boldsymbol{p}(\boldsymbol{x}) = (p(1 \mid \boldsymbol{x}), p(2 \mid \boldsymbol{x}), \ldots, p(c \mid \boldsymbol{x}))^{\mathrm{T}}$. Any decision rule that is a monotone increasing function of either side of (2) is called a Bayes decision rule. In particular, if the factors at (1) are known, the rule

$$D_{\mathrm{B}}(\boldsymbol{x}) = i \Leftrightarrow p(i \mid \boldsymbol{x}) > p(j \mid \boldsymbol{x}) \; \forall j \neq i, \tag{3}$$

with ties resolved arbitrarly, is called the *optimal Bayes classifier* on R^s with respect to $f(\boldsymbol{x})$ because D_{B} minimizes the probability of misclassifying $\boldsymbol{x}$. This error rate is defined, for any classifier function D as

$$E(D) = 1 - \int\int \cdots \int \max_i \{p_i g(\boldsymbol{x} \mid i)\} \, \mathrm{d}\boldsymbol{x}, \tag{4}$$

the integral being taken over the appropriate inverse images of D in R^s. The optimality of D_{B} with respect to E, i.e., $E_{\mathrm{B}} = E(D_{\mathrm{B}}) \leq E(D) \; \forall D : R^s \rightarrow I_c$ is well known [1], but is actually true only when the loss matrix associated with the problem is the 0–1 loss matrix. In this special instance D_{B} also minimizes the overall risk for all rules D. Henceforth, we consider only the 0–1 loss matrix case.

Being optimal relative to (4), E_{B} is often called upon to serve as a bench-mark when comparing sub-optimal designs. Unfortunately, E_{B} is almost always impossible to compute even when $f(\boldsymbol{x})$ is known; and is obviously not calculable when $f(\boldsymbol{x})$ is unknown (which is of course the usual case in practice). Thus, we count, say $\bar{E}(D)$, the average number of mistakes made by D on a labelled test set, and interpret $\bar{E}(D)$ either as (i) an estimate of the 'true but unknown' E_{B}, or (ii) an estimate of the future performance of D on fresh samples drawn in a manner similar to the test data. Loosely speaking, using $\bar{E}$ for (i) asks for an estimate of how well *any* D might do, whereas in (ii) $\bar{E}$ predicts how well *this* D may do. Moreover, $\bar{E}$ is clearly dependent on both the test data used to compute it and the design data used to find D. Because of these difficulties, the second use of $\bar{E}(D)$ – viz., to predict the future performance of D – is much less controversial than its quality as an estimator of E_{B}: this is the view of $\bar{E}$ adhered to below.

There are two broad approaches to the design of a classifier based on the mixture assumptions. In the *parametric* approach, labelled or unlabelled design data is used to estimate (parameters of) the functions on the right-hand side of (2); and the resultant densities are used to compute an approximate $\boldsymbol{p}(\boldsymbol{x})$ for use in (3). Alternately, labelled design samples can be used to, given $\boldsymbol{x}$, estimate $\boldsymbol{p}(\boldsymbol{x})$ directly by examining the local distribution of class labels in the neighborhood of $\boldsymbol{x}$. The family of algorithms based on this latter idea are the k-nearest neighbor rules (k-NNR): one calls them *non-parametric* because the left-hand side of (2) is estimated directly from the design labels without resort to an intermediate computation of the parameters of the right-hand side of (2). Although k-NNR's can be used to estimate E_{B} via lower and upper (asymptotic) bounds, we shall not consider this aspect of k-NNR's below (see [2–4] for further discussion along this line).

In essence, k-NNR's simply find the k NN's to $\boldsymbol{x}$ and then count the votes for each class among these neighbors. There is a well established hierachy (cf. [4] for an excellent survey) of k-NNR's in the literature, viz., the 1-NNR, k-NNR, (k, l)-NNR, and (k, l_i)-NNR, which are progressively more sophisticated in their requirements for giving $\boldsymbol{x}$ a label in I_c. Common to all of these algorithms, however, is the stipulation that the design labels are hard (non-fuzzy, crisp), i.e., that each neighbor to $\boldsymbol{x}$ is attached (wholly) to one and only one of the c classes. Recently, a great deal of interest has arisen concerning the relaxation of this requirement through the use of *fuzzy* label vectors, and consequently to fuzzy k-NNR's [5, 6]. One of the main objectives of the present study is to define a NNR formalism that unifies this work and embodies all of the previous k-NNR's in a single framework. This will provide a basis for further extensions, interpretations, and analysis of new generalizations of the extant theory and its applications. A second objective of this paper is to compare the performance of two fuzzy k-NNR's to the hard k-NNR; and to compare all three NNR's to a fuzzy one-Nearest Prototype Rule (1-NPR). Section 2 discusses the generalized k-NNR formalism. Section 3 briefly reviews 1-NPR's in general, and the FCM/1-NPR in particular. Section 4 describes Jóźwik's algorithm. Section 5 describes the data and computing protocols for the numerical examples of Section 6. Section 7 reports our conclusions, which are summarized briefly as follows:

- The FCM/k-NNR yields the best overall performance, especially for mixtures of normal densities, of the four classifiers tested.
- The fuzzy 1-NPR is computationally superior (less storage and CPU time) than all three k-NNR's, and yields error rates that are quite comparable to NNR designs.
- All four designs produce comparable results on normal mixtures; and rules that use (given) labelling information are somewhat better than those that ignore it for *very small* design sample sets.
- Fuzzy generalizations of conventional k-NNR's *do* improve predicted performance rates in most instances.

We conclude Section 7 with some conjectures and ideas for further research. The Appendix gives a pseudo-code procedure for the generalized k-NNR.

2. The generalized NNR

A compact description of k-NN decision rules and algorithms for implementing them is afforded by introducing some terminology associated with V_{cn}, the vector space of real $c \times n$ matrices. Specifically, let $W = [w_{ik}] \in V_{cn}$, and suppose its entries satisfy three constraints:

$$w_{ik} \in [0, 1], \quad 1 \leq i \leq c;\ 1 \leq k \leq n, \tag{5a}$$

$$\sum_{i=1}^{c} w_{ik} = 1, \quad 1 \leq k \leq n, \tag{5b}$$

$$0 < \sum_{k=1}^{n} w_{ik} < n, \quad 1 \leq i \leq c. \tag{5c}$$

We denote by M_{fcn} the set of all such matrices:

$$M_{fcn} = \{W \in V_{cn} \mid w_{ik} \text{ satisfies (5)}\}, \tag{6a}$$

and by M_{cn} the subset of M_{fcn} for which every entry of W is either 0 or 1:

$$M_{cn} = \{W \in M_{fcn} \mid w_{ik} \in \{0, 1\}\ \forall i, k\}. \tag{6b}$$

It is also convenient to name the subsets of R^c that contain single column vectors of such matrices. Let

$$N_{fc} = \left\{\boldsymbol{w} \in R^c \mid 0 \leq w_i \leq 1\ \forall i, \sum_{i=1}^{c} w_i = 1\right\}, \tag{6c}$$

$$N_c = \{\boldsymbol{w} \in N_{fc} \mid w_i \in \{0, 1\}\ \forall i\}. \tag{6d}$$

In the present context elements of these four sets have the following names and interpretations:

$\boldsymbol{w} \in N_c \sim$ hard label vectors,

$\boldsymbol{w} \in (N_{fc} - N_c) \sim$ fuzzy label vectors,

$W \in M_{cn} \sim$ hard c-partitions of X,

$W \in (M_{fcn} - M_{cn}) \sim$ fuzzy c-partitions of X,

where $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ is a set of n points in R^s. The terms hard and fuzzy *c-partitions of X* accrue to M_{cn} and M_{fcn} upon regarding w_{ik} as $w_i(\boldsymbol{x}_k)$, where $w_i: X \to [0, 1]$ is the membership function for the i-th partitioning subset of X, exhibited through its values along row i of W on all n points in X. The number w_{ik} is the *grade of membership* of $\boldsymbol{x}$ in w_i, and when $W \in M_{cn}$, its columns are just the hard label vectors in N_c of any n points. Reference [7] contains a lengthy discussion of the geometric and philosophical ramifications of these definitions, as well as several numerical examples exemplifying their usefulness. It is also convenient to have a notation for each column vector of W: let $\boldsymbol{W}^{(k)} = (w_{1k}, w_{2k}, \ldots, w_{ck})^T$, so that for $W \in M_{fcn}$, we have the association

$$W = [\boldsymbol{W}^{(1)}\ \boldsymbol{W}^{(2)}\ \cdots\ \boldsymbol{W}^{(k)}\ \cdots\ \boldsymbol{W}^{(n)}] \in M_{fcn}, \tag{7a}$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$X = \{\boldsymbol{x}_1\ \ \boldsymbol{x}_2\ \ \cdots\ \boldsymbol{x}_k\ \ \cdots\ \boldsymbol{x}_n\} \subset R^s, \tag{7b}$$

where $\downarrow$ "$\boldsymbol{W}^{(k)}$ labels $\boldsymbol{x}_k$". Note that each $\boldsymbol{W}^{(k)}$ is a label vector in N_{fc}. For example, if $\boldsymbol{x}_8$ belongs entirely to class 4 among $c = 5$ classes, then $\boldsymbol{W}^{(8)} = (0, 0, 0, 1, 0)^T$; on the other hand, if the membership of $\boldsymbol{x}_5$ is divided among classes 1, 2, and 4, say, in the percentages 40%, 22%, and 38%, respectively, then $\boldsymbol{W}^{(5)} = (0.40, 0.22, 0, 0.38, 0)^T$. $\boldsymbol{W}^{(8)}$ is a hard label, $\boldsymbol{W}^{(5)}$ is a fuzzy one. A pair (X, W) satisfying (7) is called a hard (or fuzzy) *labelled data set* according as $W \in M_{cn}$ (or $(M_{fcn} - M_{cn})$).

Given a labelled data set (X, W), we ordinarily divide it into two parts: a design set X_d, $|X_d| = n_d$, and a test set X_t, $|X_t| = n_t$, satisfying

$$X_d \cup X_t = X, \tag{8a}$$

$$X_d \cap X_t = \phi \text{ (the empty set)}, \tag{8b}$$

$$n_d + n_t = n, \tag{8c}$$

From the association in (7) there follows the partitioned association (perhaps after suitable relabelling):

$$W = [\boldsymbol{W}_d^{(1)} \cdots \boldsymbol{W}_d^{(n_d)} \mid \boldsymbol{W}_t^{(n_d+1)} \cdots \boldsymbol{W}_t^{(n_t)}] = [W_d \mid W_t], \tag{9a}$$

$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$

$$X = \{\boldsymbol{x}_1 \cdots \boldsymbol{x}_{n_d} \mid \boldsymbol{x}_{n_d+1} \cdots \boldsymbol{x}_n\} = \{X_d \mid X_t\}. \tag{9b}$$

So (X_d, W_d) is a labelled design set, and (X_t, W_t) is a labelled test set; note that W_d and W_t are in M_{fcn_d} and M_{fcn_t} respectively. We call $(n_d : n_t)$ the *partition ratio* corresponding to the division of X into design and test sets.

Now we define the quantities needed to make (X_d, W_d) a k-NNR. Specifically, let $\boldsymbol{x}$ be any vector in R^s (*it might be* in X_d), let Δ be a metric on R^s, and set

$$\Delta(\boldsymbol{x}, \boldsymbol{y}) = \text{the } \Delta\text{-distance between } \boldsymbol{x}, \boldsymbol{y} \in R^s, \tag{10a}$$

$$I_d = \{1, 2, \ldots, n_d - 1\}: \text{ integers in } I_d \text{ are potential numbers of nearest neighbors to any point } \boldsymbol{x}, \tag{10b}$$

$$N_k(\boldsymbol{x}) = \text{the } k \text{ vectors in } X_d \text{ } \Delta\text{-closest to } \boldsymbol{x}, \text{ where } k \in I_d. \quad N_k(\boldsymbol{x}) \text{ are the } k\text{-NN of } \boldsymbol{x}, \boldsymbol{x} \notin N_k(\boldsymbol{x}), \tag{10c}$$

$$J_k(\boldsymbol{x}) = \text{the indices of the points in } N_k(\boldsymbol{x}), \text{ ordered so that } \Delta(\boldsymbol{x}, \boldsymbol{x}_i) \leq \Delta(\boldsymbol{x}, \boldsymbol{x}_{i+1}),\ i = 1, 2, \ldots, k-1, \tag{10d}$$

$$\boldsymbol{L}_k(\boldsymbol{x}) = \sum_{j \in J_k(\boldsymbol{x})} \boldsymbol{W}_d^{(j)}/k; \quad k \in I_d. \tag{10e}$$

Note that $\boldsymbol{L}_k(\boldsymbol{x}) \in N_{fc}$, i.e., it is a label vector for $\boldsymbol{x}$. The i-th component of $\boldsymbol{L}_k(\boldsymbol{x})$, say $L_{ik}(\boldsymbol{x})$, is easily interpreted: $L_{ik}(\boldsymbol{x})$ is the membership in class i assigned to $\boldsymbol{x}$ by the labels of its k NN's. Thus, $\boldsymbol{L}_k(\boldsymbol{x})$ is, in general, a *fuzzy* label vector for $\boldsymbol{x}$. Indeed, $k\boldsymbol{L}_k(\boldsymbol{x})$ is a *hard* label vector for $\boldsymbol{x}$ if and only if *all* of the columns $\{\boldsymbol{W}_d^{(j)}\}$ used to compute it via (10e) are hard labels. If a hard *decision* is required (ultimately it usually is, although the membership of $\boldsymbol{x}$ in each class may be more useful in intermediate processing goals), one could elect to assign $\boldsymbol{x}$ to the class of maximum membership. A more stringent decision philosophy would be to require (even the maximum!) $L_{ik}(\boldsymbol{x})$ to exceed a prespecified threshold before converting $\boldsymbol{L}_k(\boldsymbol{x})$ to a hard label (i.e., hard decision). Thus, we are led to the fuzzy (k, l_i) rule:

Definition 1. *The generalized* (k, l_i)*-NNR classifier.* Let $k \in I_d$, (X_d, W_d), be a labelled (hard *or* fuzzy) data set, and let $\boldsymbol{L}_k(\boldsymbol{x}) = (L_{1k}(\boldsymbol{x}), L_{2k}(\boldsymbol{x}), \ldots, L_{ck}(\boldsymbol{x}))^T$ be the label vector for any $\boldsymbol{x} \in R^s$ computed via (10e). Finally, let $\{l_1, l_2, \ldots, l_c\}$ be class voting thresholds. The function $D_k : R^s \to I_c$ defined as

$$D_k(\boldsymbol{x}) = i \Leftrightarrow L_{ik}(\boldsymbol{x}) > l_i \tag{11}$$

is the generalized (k, l_i)-NNR.

Rule (11) is not well defined unless the numbers $\{l_i\}$ are properly constrained.

For example, $\sum_{i=1}^{c} L_{ik}(\boldsymbol{x}) = 1$ because $\boldsymbol{L}_k(\boldsymbol{x}) \in N_{fc}$, so every l_i should be between 0 and 1. If the l_i's are large enough, D_k can fail to make any decision. Conversely, if the l's are small enough, ties are inevitable, so a suitable set of tie-breaking procedures is needed to make D_k a function. For example, suppose $c = 4$ and $\boldsymbol{L}_k(\boldsymbol{x}) = (0.65, 0.10, 0.25, 0)^{\mathrm{T}}$. If $l_i = 0.7$, D_k is undefined; if $l_i = 0.6$, $D_k(\boldsymbol{x}) = 1$, and if $l_i = 0.20\ \forall i$, then $D_k(\boldsymbol{x}) = 1$ or 3 is double-valued, so is not a (classifier) function. For our purpose it suffices to note that due caution must be exercised in choosing the $\{l_i\}$ so that D_k is always singe-valued on R^s and when it is, (11) is the proper generalization of all hard k-NNR families having equivalent conventions. The virtues of (11) are twofold: First, it enables one to discriminate against selected classes by raising or lowering l_i for a specific class as the application demands. And second, the k-NNR's are extended to fuzzy labels of X_d in a natural way that completely imbeds the original methodology. The primary advantage in allowing W_d to be a fuzzy labelling of X_d is that the label vector $\boldsymbol{L}_k(\boldsymbol{x})$ of $\boldsymbol{x}$ defined at (10e) can be a *continuous* function of the label information about mixed classes in its neighborhood. If, e.g., $k = c = 3$ and $W_d \in M_{sn}$ is hard, the *only possible values* for $L_{ik}(\boldsymbol{x})$ – the membership assigned to $\boldsymbol{x}$ for class i by the labels in $J_k(\boldsymbol{x})$ – are the four numbers $0, \frac{1}{3}, \frac{2}{3}$, and 1; whereas all values in $[0, 1]$ are possible (subject to constraint (6c)) for the elements of $\boldsymbol{L}_k(\boldsymbol{x})$ if $W_d \in M_{f3n}$ is a set of fuzzy design labels. Fuzzy k-NNR's ostensibly provide a continuous 'tuning mechanism' for thresholding votes as opposed to the set of preselected bands (the possible discrete values of $L_{ik}(\boldsymbol{x})$) imposed by hard k-NNR's. Consequently, one may expect more precise performance when the data represent truly mixed classes. Returning to our example with $k = c = 3$, taking equal values of l_i at any value less than 0.67 (ignoring roundoff) in (11) yields the conventional hard 3-NNR with simple majority (at least 2 out of 3) voting. Moreover, this happens if and only if (say) $L_{ik}(\boldsymbol{x})$ is the unique maximum (because $\sum L_{ik}(\boldsymbol{x}) = 1$) of the $L_{jk}(\boldsymbol{x})$, so (11) has, *for this special case,* the more familiar form of simple majority rule:

$$D_k(\boldsymbol{x}) = i \ \Leftrightarrow\ L_{ik}(\boldsymbol{x}) > L_{jk}(\boldsymbol{x})\ \forall j \neq i. \tag{12}$$

Further, the substance of Cover and Hart's classic result [8] in this special case is that $k\boldsymbol{L}_k(\boldsymbol{x}) \in N_c$ and

$$\boldsymbol{L}_k(\boldsymbol{x}) \to \boldsymbol{p}(\boldsymbol{x}) \quad \text{as } (k, n) \to \infty \text{ with } k/n \text{ property constrained.} \tag{13}$$

In turn, D_k at (12) 'approaches' the optimal performance of D_{B}. It is beyond the scope of this study to do more than observe that, for a fuzzy label sequence $\{\boldsymbol{L}_k(\boldsymbol{x})\}$ indexed on n, that if $\{k\boldsymbol{L}_k(\boldsymbol{x})\} \to k\hat{\boldsymbol{L}}(\boldsymbol{x}) \in N_c$, there is surely a suitable extension of the Cover and Hart result for the fuzzy k-NNR case. In what follows, we are content to compare several instances of (12) numerically, with a view towards deciding whether or not fuzzy k-NNR's do in fact improve the expected classifier performance of D_k over their hard counterparts. Does this make any sense? It does, because $D_k(\boldsymbol{x}) = i$ in (11) is a hard *decision* about $\boldsymbol{x}$ even when the labels W_d used to get it are fuzzy. What cannot be fuzzy for our current objective is the set of *testing* labels W_t for X_t. With a view towards comparing the performance of various classifiers on R^s we make

Definition 2. *Empirical error rate of D.* Let D be any decision function, $D: R^s \to I_c$, and let (X_t, W_t) be a hard test set, $W_t \in M_{cn_t}$. The empirical error rate of D is

$$\bar{E}(D) = \sum_{j=1}^{n_t} e(\boldsymbol{x}_j)/n_t \tag{14a}$$

where $\boldsymbol{x}_j \in X_t$ and

$$e(\boldsymbol{x}_j) = \begin{cases} 1, & D(\boldsymbol{x}_j) = i \text{ and } W_{it}^{(j)} = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{14b}$$

$\bar{E}(D)$ is just the average number of labelling errors made by D on X_t. $W_{it}^{(j)}$ is the i-th component of $\boldsymbol{W}_t^{(j)}$, the hard labels for $\boldsymbol{x}_j$.

We have not, in Definition 2, specified any relationship between the samples X_t and, if used, any design samples X_d used to train (or find) the classifier function D being tested. This topic is well discussed elsewhere (cf. [1, 4]). Suffice it to say that X_t and X_d should be independent, as they will be, e.g., when the partitioning scheme for X exhibited at (9) is used.

The next detail requiring attention is the value of k: how many neighbors should be used? Different k's obviously result in different classifiers and hence various $\bar{E}(D_k)$'s. If we let k vary from 1 to some maximum value, say $k_{\max} \in I_d$ we end up with a sequence of empirical error rates, $\{\bar{E}(D_1), \bar{E}(D_2), \ldots, \bar{E}(D_{k_{\max}})\}$. An inelegant but pragmatic way to choose k is to select the k^* that produces the smallest error rate. We shall call $(k^*, \bar{E}(D_{k^*}))$ the 'optimal' number of nearest neighbors and predicted error rate for (X_d, W_d) when k^* is chosen so that

$$\bar{E}(D_{k^*}) = \min\{\bar{E}(D_j): 1 \leq j \leq k_{\max}\}. \tag{15}$$

k^* is clearly a function of many variables, some of which (e.g., the actual samples X_d and X_t used) are beyond theoretical analysis. In view of this, (15) is about the best we can do.

At this point we have a completely specified procedure for the generalized k-NNR. The procedure is given in the Appendix in 'pseudo-code' format; in the sequel we shall refer to it as PROC.KNNR. Each of the variables passed to PROC.KNNR alters its output $(k^*, \bar{E}(D_{k^*}))$. In order to study the effect of various labelling schema for the same design set, we fix *all* of the inputs to PROC.KNNR except W_d, the labels attached to X_d. In this case $(k^*, \bar{E}(D_{k^*}))$ become functions of W_d alone; one can envision a truly gruesome notation for the dependency of $\bar{E}(D_{k^*})$ on the labels; e.g., $(\bar{E}(D_{k^*})[(W_d)])$. We ease this problem by identifying $(k^*, \bar{E}(D_{k^*}))$ with the algorithm used to generate the label matrix W_d. Specifically then, how *can* different W_d's arise? For a fixed design set X_d several possibilities exist:

Type (i). $W_d \in M_{cn_d}$ is hard (given).

Type (ii). $W_d \in M_{fcn_d}$ is fuzzy (given).

Type (iii). A hard W_d as in (i) is converted into fuzzy W_d as in (ii).

Type (iv). W_d (and perhaps even c!) is unknown. Use X_d to produce: (a) W_d as

in (i); or (b), W_d as in (ii) by applying, respectively, a hard or fuzzy clustering algorithm to X_d.

In this list 'given' can mean assigned or observed by man or machine; 'converted' means either by a human or with an algorithmic process; and 'produce' means by an algorithmic process (e.g., by clustering X_d). Type (i) is presumably unique, and corresponds to the usual hard k-NNR situation. Here is an example of (ii), i.e., a natural physical process in which it is *possible, reasonable,* and *desirable* to assign fuzzy labels to the vectors in X_d: a sensor searching for some object (against an unfamiliar background) may be covering the assigned search area by scanning. At any instant, the object-recognition system attached to the sensor is paying attention to a limited field of view; if the boundaries of this field are not sharp, i.e., if the sensor's response to a pointlike object fades gradually with displacement from the current center of attention, then in collecting a set of pattern vectors to characterize the sought object under various circumstances, it is natural to assign to each vector a fuzzy label representing the degree to which the object was near the center of attention. Possibility (iii) begs a question: if hard labels are *given,* what motivation can there be for converting them into fuzzy ones? It is easy to envision cases where the measurements (features) of $\boldsymbol{x}_k$'s $\in X_d$ from different classes exhibit considerable overlap in R^s even though their labels, if hard, imply that they are in some sense well separated. Conversion as in (iii) simply attempts to make the labels of the $\boldsymbol{x}_k$'s more accurately reflect the relationship between their features. Thus, Type (iii) labels are sought simply to improve the (expected) error rate of the k-NNR upon which they are based. As an aside, we point out that the Type (iii) situation is entirely analogous to the relationship between Hard and Fuzzy c-Means: there are an infinite number of fuzzy generalizations of the (unique) hard algorithm. Both situations are ultimately traceable to the extension of membership values from $\{0, 1\}$ to $[0, 1]$; and the trick in either case is to find a 'good', i.e., *better* fuzzy algorithm than the hard one based on a Type (i) W_d.

The last class, Type (iv), is the 'unsupervised learning' problem. Unlabelled data is first labelled by a hard or fuzzy clustering algorithm; and then the labels are themselves used with X_d in the k-NNR classifier mode. This is clearly the most difficult (c itself may be unknown!) and least tractable of the four possibilities; we list it here only for completeness. In the examples to follow we send one Type (i) and two Type (iii) sets of labels to PROC.KNNR with all other variables fixed for each data set. The purpose of these experiments is threefold: to demonstrate that Type (iii) processing does reduce the Type (i) error rate; to compare two Type (iii) processes; and to compare the performance of two generalized k-NNR's to a fuzzy 1-NPR. With this latter objective in mind, we turn to a brief description of NPR's.

3. The nearest prototype rule: FCM

Given a set X_d constituted as above, the label matrix W_d (hard or fuzzy) is one way to represent information about the class identity of each individual in X_d.

Another (not necessarily equivalent) representation of this information is by *class prototypes,* i.e., vectors in R^s that are presumed to be paradigmatic of the features one might expect for 'ideal' class representatives. Each class may have multiple prototypes (in fact, if n_{di} is the number of vectors labelled as class i by some $W_d \in M_{cn_d}$, $\sum_i n_{di} = n_d$, then the hard k-NNR is an $\{n_{di}\}$ multiple-prototype rule).

The virtue of NPR's is in their compression of the n_d points into some smaller number. The limit of this process is to represent all n_{di} points in class i by exactly one class prototype, say $\boldsymbol{v}_i \in R^s$. Once the c prototypes are found, the 1-NPR is easy to implement:

Definition 3. *The 1-NPR classifier.* Let $\boldsymbol{v} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_c)^T$, $\boldsymbol{v}_i \in R^s \forall i$, found by any means whatsoever, be class prototypes for the labels in I_c. Let $\boldsymbol{x} \in R^s$. The function $D_{\boldsymbol{v}} : R^s \to I_c$ defined as

$$D_{\boldsymbol{v}}(\boldsymbol{x}) = i \;\Leftrightarrow\; \Delta(\boldsymbol{x}, \boldsymbol{v}_i) < \Delta(\boldsymbol{x}, \boldsymbol{v}_j)\ \forall j \neq i, \tag{16}$$

where Δ is any metric on R^s and ties are broken arbitrarily, is called the 1-*nearest prototype rule* associated with the pair $(\Delta, \boldsymbol{v})$.

Note first that $\tilde{E}(D_{\boldsymbol{v}})$ at (14) is well defined for $D_{\boldsymbol{v}}$'s as in (16): $\tilde{E}$ will serve as a common denominator for comparing NN and NP rules. 1-NPR's are discussed at length in many works [1, 4, 7]. If Δ is induced by an inner product, e.g., the action of $D_{\boldsymbol{v}}$ can be written in terms of affine functions ($\Delta(\boldsymbol{x}, \boldsymbol{v}_i)$ can be replaced by $\boldsymbol{x}^T\boldsymbol{v}_i + k_i$), with the subsequent geometric characterization of $D_{\boldsymbol{v}}$ as a 'linear' classifier.

Given (X_d, W_d) with W_d hard, an obvious set of prototypes to use are the sample means of each labelled subset in X_d: letting $[w_{ik}] = W_d$,

$$\boldsymbol{m}_i = \sum_{k=1}^{n} w_{ik}\boldsymbol{x}_k \Big/ \sum_{k=1}^{n} w_{ik}. \tag{17}$$

The vector $\boldsymbol{m} = (\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_c)$ defines $D_{\boldsymbol{m}}$ at (16). $D_{\boldsymbol{m}}$, is, for a given Δ, the 1-NPR classifier called the *Hard c-Means* (*HCM*) 1-*NPR,* regardless of whether W_d is given (Type (i)), or computed (Type (iv)(a)). There are many alternatives to (17). Of these, we are particularly interested in the following: given X_d, let us apply the *Fuzzy c-Means* (*FCM*) algorithm (cf. [7] or [13]) to it. At termination, there are two outputs, (W_d) and $\boldsymbol{v}$:

$$X_d \to \boxed{\text{PROC.FCM}} \to \begin{cases} W_d = W_{fcm} \in M_{fcn_d} \text{ a fuzzy label matrix for } X_d, \\ \text{either Type (iii) or Type (iv)(b), to use in the} \\ \text{FCM/}k\text{-NNR at (11)} \end{cases} \tag{18a}$$

$$\to \begin{cases} \boldsymbol{v} = \boldsymbol{v}_{fcm} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_c) \text{ a set of prototypes} \\ \text{for } X_d, \text{ to use in (16) with } \Delta \text{ as a fuzzy 1-NPR} \end{cases} \tag{18b}$$

We shall henceforth refer to $D_{\boldsymbol{v}}$ with $(\Delta, \boldsymbol{v})$ from (18b) as a FCM/1-NPR and to

D_k when labels (18a) are used in (11) as a FCM/k-NNR. It is clear that the implemenability and computational efficiency of D_v from (18b) is superior to any D_k at (11) using labels from (18a): it is *not* clear that the two representations of the 'classifier information' resident in the fuzzy labels (W_{fcm}) and prototypes (v_{fcm}) are equivalent. Previous studies [10, 11, 12] have compared the FCM/1-NPR with the HCM/1-NPR, MLE/1-NPR, and Hard/k-NNR's (MLE refers to maximum likelihood estimation: both labelled and unlabelled forms have been investigated). In all of these studies the FCM/1-NPR has proven superior in terms of minimizing $\bar{E}(D)$: an objective of the examples below is to determine whether this continues to hold for comparisons with fuzzy k-NNR's.

A complete description of FCM including a FORTRAN listing with test examples is given in [13]. We shall refer to it here as PROC.FCM; its primary input is X_d and its outputs are ($W_d = W_{fcm}$, $v = v_{fcm}$) where W_{fcm} is a fuzzy labelling of X_d and the v_i's are fuzzy prototypes for X_d. Consult [7] for a more expansive discussion of theoretical aspects of FCM. The FCM/k-NNR and FCM/1-NPR classifier designs are illustrated schematically in Figure 1. Observe that the * has been dropped from the notation for the optimal number of neighbors ($k^*_{fcm} \rightarrow k_{fcm}$); and we have suppressed the functional dependency of $\bar{E}$ on $D^*_{k_{fcm}}$ and $D_{v_{fcm}}$ to simplify the notation.

As an aside, generalized NNR's of Types (iii) and (iv) may or may not be 'parametric' in the sense discussed above. For example, the FCM/k-NNR as defined here is not *statistically parametric, but* W_{fcm} is certainly an estimate –

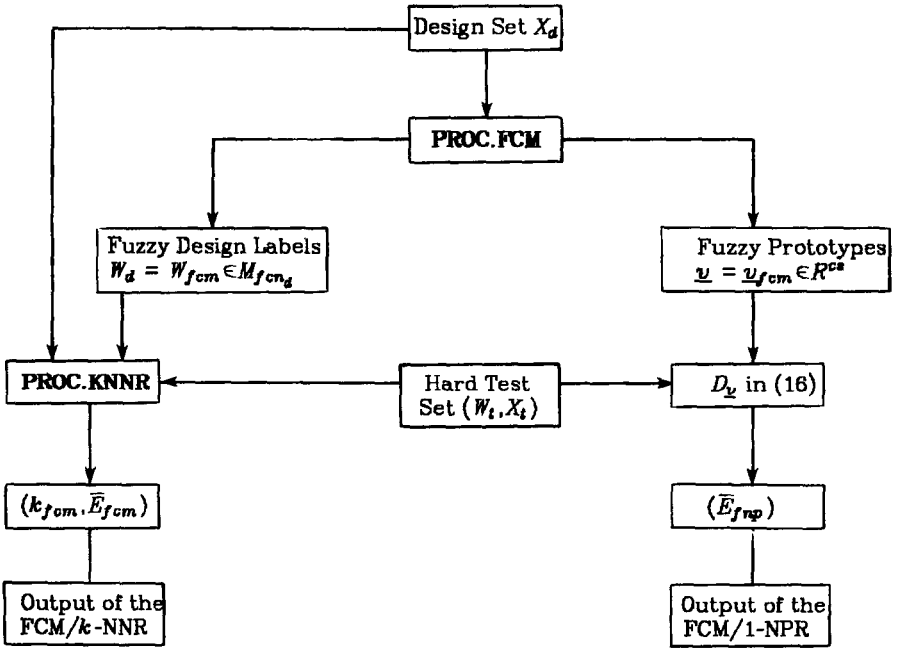

Fig. 1. The FCM/k-NNR and FCM/1-NPR classifiers.

produced algorithmically with X_d of parameters (viz., W_{fcm}) needed to define the (presumably) non-parametric rule defined by $(X_d,\ W_d = W_{fcm},\ k^* = k_{fcm})$!

4. Jóźwik's fuzzy NNR

The FCM/k-NNR is a Type (iii) rule which simply discards (if known) W_d, the hard labels accompanying X_d, and processes X_d with FCM to produce the fuzzy design labels W_{fcm} in (18a). Jóźwik's algorithm [5] is also a Type (iii) method, but in his scheme the given labels W_d of X_d are used to initialize an iterative method which ultimately produces a set of fuzzy design labels, called W_J below, for X_d. Jóźwik's algorithm is complicated to describe (c.f. [5, 9]); we briefly summarize the main steps here so that readers will understand the numerical examples of Section 6.

First, a division of data (X, W) as shown at (9) is made, but for this discussion it is convenient to replace the subscript (t) by (2); then (X_2, W_2) is temporarily laid aside. In order to describe Jóźwik's method, we add the subscript h to W_d: W_{dh} denotes the h-th iterate in a sequence of labellings of X_d, $h = 0, 1, 2, \ldots$, and we take $W_d = W_{d0}$, i.e., the given hard labels of X_d are its initial labels.

Phase 1 of Jóźwik's Method.

(J1.1) Send (X_d, W_{d0}) to PROC.KNNR with $k_{\max} = (n_d - 1)$ and $(X_t, W_t) = (X_d, W_{d0})$. Label the outputs $(k^*, \bar{E}(D_{k^*})) = (k_0, \bar{E}_0)$.

(J1.2) For any integer $h \geq 0$, define the column vectors

$$\boldsymbol{W}_{d,h+1}^{(i)} = (k_h \boldsymbol{L}_{k_h}(\boldsymbol{x}_i) + \boldsymbol{W}_{dh}^{(i)})/(k_h + 1), \tag{19}$$

where $i = 1, 2, \ldots, n_d$ and $\boldsymbol{L}_k(\boldsymbol{x})$ is defined at (10e). In particular, the *fuzzy* labelling W_{d1} of X_d is computed with (W_{d0}, k_0) and (19).

(J1.3) For $h = 1, 2, \ldots, h_{\max}$:

(a) Send design data (X_d, W_{dh}) and test data (X_d, W_{d0}) to PROC.KNNR, and call its outputs $(k_h, \bar{E}_h)$. Compute $W_{d,h+1}$ per (19).

(b) If $\bar{E}_h \leq \bar{E}_{h+1}$ set $(W_{dh}, k_h, \bar{E}_h) = (W_1^*, k_1^*, \bar{E}_1^*)$ and stop. Otherwise, next h.

Phase 2 of Jóźwik's Method.

(J2.1) Compute fuzzy labels for the n_2 vectors in X_2 using (W_1^*, k_1^*) in formula (10e). Since each $\boldsymbol{L}_k(\boldsymbol{x}) \in N_{fc}$, the augmented matrix $W_{d0} = [W_1^* \mid \boldsymbol{L}_{k_1^*}(\boldsymbol{x}_1), \ldots, \boldsymbol{L}_{k_1^*}(\boldsymbol{x}_{n_2})] \in M_{fcn}$ is a fuzzy label matrix for $X = X_d \cup X_2$.

(J2.2) Send design data (X, W_{d0}) and test data (X, W) to PROC.KNNR, and call its outputs $(k_0, \bar{E}_0)$. Compute W_{d1} via (19). Note that W_{d1} is now a fuzzy c-partition of X, i.e., is a $(c \times n)$ matrix in M_{fcn}.

(J2.3) For $h = 1, 2, \ldots, h_{\max}$:

(a) Send design data (X, W_{dh}) and test data (X, W) to PROC.KNNR, and call its outputs $(k_h, \bar{E}_h)$. Compute $W_{d,h+1}$ per (19).

(b) If $\bar{E}_h \leq \bar{E}_{h+1}$ set $(W_{dh}, k_h, \bar{E}_h) = (W_2^*, k_2^*, \bar{E}_2^*)$ and stop. Otherwise, next h.

Phase 3 of Jóźwik's Method.
(J3.1) Select the final set of NNR parameters (X_d, W_d, k) for use in (11) as follows:

$$\text{if } \bar{E}_1^* \leq \bar{E}_2^*: \quad (X_d, W_d, k) = (X_d, W_1^*, k_1^*), \tag{20a}$$

$$\text{if } \bar{E}_1^* > \bar{E}_2^*: \quad (X_d, W_d, k) = (X, W_2^*, k_2^*). \tag{20b}$$

(J3.2) Denote the final design set chosen at (20) by (X_{dJ}, W_{dJ}, k_{dJ}), and call the error rate $(\min(\bar{E}_1^*, \bar{E}_2^*)) = \bar{E}_J$.

$\bar{E}_J$ in (J3.2) has essentially been computed via (15) using the hold-one-out method. As stated earlier, it is not our purpose here to argue the merits of $\bar{E}_J$ as an estimate of E_B; rather, we view it as an indication of the expected performance of (11) using the design parameters chosen with (20).

The two-phase approach leading to (20) complicates direct comparisons with, e.g., the FCM/k-NNR and FCM/1-NPR, because it is not known in advance whether X_{dJ} will be all of X (20b) or a subset of X (20a). Moreover, it is clear that the *partition ratio* $(n_d : n_2)$ affects not only the final choice of X_{dJ} in (J3.2); but also the values of $\bar{E}_{fcm}$ and $\bar{E}_{fnp}$. In particular, when (20b) is selected, a direct comparison with the FCM classifiers would necessitate regarding X_d in (18) as *all* of X; then $W_{fcm} \in M_{fcn}$, and there would be no independent test data left with which to compute the empirical error rates of D_{fcm} or D_{fnp}. There are several possibilities for ameliorating this dilemma. For example, one might adopt the hold-one-out strategy by taking $(n_d : n_2) = (n - 1 : 1)$ in (18); and then averaging the results of n estimates of $\bar{E}_{fcm}$, $\bar{E}_{fnp}$ by running (18) n times, once for each deleted point in X. This technique would produce an estimate that seems, for large enough n, 'pretty comparable' to $\bar{E}_J$ when (20b) is used. A different strategy is to simply 'hide' some fraction of X from all the classifiers, and use this hidden data as a new test set: this latter strategy is adopted below.

Jóźwik's method invites several interesting questions. For example, the termination criterion in both phases is to stop when $\bar{E}_h \leq \bar{E}_{h+1}$. $\bar{E}_h$ is defined on the (possibly infinite) set $\{0, 1, 2, \ldots\}$. One wonders about the 'global' minimum of $\bar{E}_h$: does one always exist? Is it unique? Is it the first 'local' minimum? A second line of inquiry: under what circumstances can we expect (20a) to hold, i.e., when will adding more (fuzzy) labels *degrade* the predicted performance of the fuzzy J/k-NNR? And finally, is the investment of computation time required to find (W_J, k_J) sufficiently rewarding to justify the extra Phase 2 calculations? The experiments below are designed with these questions in mind.

5. Computing protocols

All three NNR's were tested using equation (12). In particular, the (hard) label corresponding to the class having maximum membership in $N_k(\boldsymbol{x})$ was assigned to each $\boldsymbol{x} \in X_r$. The Euclidean norm and the inner product were used for all distances: for $\boldsymbol{x}, \boldsymbol{y} \in R^s$,

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{s} x_i y_i, \tag{21a}$$

$$\Delta^2(x, y) = \|x - y\|^2 = \langle x - y, x - y \rangle. \tag{21b}$$

This choice for Δ is admittedly uninspired (albeit convenient, and we suspect, not greatly injurious to our conclusions). We are content here to remark that the choice for Δ in NN and NP designs is obviously an important facet of the expected performance of D; and that Fukunaga and his co-workers have pioneered several ingenious methods for determining an optimal Δ for use in (hard) k-NNR's [14]. The value of the weighting exponent (m) for the FCM algorithm was fixed at $m = 2.00$. Ties were broken in all algorithms by randomly selecting an alternative from the ones available. Three data sets were used for the experiments:

Data Set A. Anderson's Iris Data [15]. $c = 3$ classes; $s = 4$ features; $n = 150$ samples, 50 from each class.

Data Set B. (Synthetic) univariate normal data drawn from mixture (1a) with $p_1 = p_2 = 0.5$; and $c = 2$ classes; $s = 1$ feature; $n = 1000$ samples, 500 from each class; $g(x \mid 1) \sim \mathrm{N}(1, 1)$, $g(x \mid 2) \sim \mathrm{N}(2, 1)$; $E_B = 0.31 \sim$ the 'true' optimal Bayes error rate.

Data Set C. (Synthetic) bivariate normal data drawn from mixture (1a) with $p_1 = p_2 = 0.5$, and $c = 2$ classes; $s = 2$ features; $n = 1000$ samples, 500 from each class;

$$g(x \mid 1) \sim N(\mu_1, I). \qquad \mu_1 = (1, 1)^T, \qquad I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$g(x \mid 2) \sim N(\mu_2, I), \qquad \mu_2 = (2, 2)^T;$$

$E_B = 0.24 \sim$ the 'true' optimal Bayes error rate.

These data sets are henceforth referred to collectively as (X, W), X being the pooled samples, and W being the observed (hard) labels attached to X. Partitioning of the data was done by dividing each (X, W) into three subpairs:

$$W = [W_d \mid W_2 \mid W_t], \tag{22a}$$

$$X = \{X_d \mid X_2 \mid X_t\}, \tag{22b}$$

$$n = n_d + n_2 + n_t. \tag{22c}$$

Our division in (22) reserves (X_t, W_t) as a 'final' test set for all algorithms against the possibility that Jóźwik's scheme opts for (20b) and uses $(X_2 \cup X_d, W_2)$ as discussed in Section 4 above. The fraction of each data set laid aside for final testing was as follows: $n_{tA} = 36$ samples, 12 from each class; and $n_{tB} = n_{tC} = 100$ samples, 50 from each class.

The remaining samples in each case were divided into X_d and X_2 in five different *apparent* partition ratios as follows: $(n_d : n_2) = (1:2)$, $(1:1)$, $(2:1)$, $(1:9)$, and $(9:1)$. These would be 'real' partition ratios as defined in (9) if Jóźwik's method always chose (20a), and the artifice of using the 'fresh' test set (X_t, W_t) could be

Table 1. Summary of classifiers

Classifier, D	Design labels, W_d	Optimal no. of neighbors, k^*	Empirical error rate, $\bar{E}(D)$
Hard/k-NNR	W_{nn}	k_{nn}	$\bar{E}_{nn}$
Fuzzy J/k-NNR	W_J	k_J	$\bar{E}_J$
FCM/k-NNR	W_{fcm}	k_{fcm}	$\bar{E}_{fcm}$
FCM/1-NPR	W_{fnp}	-	$\bar{E}_{fnp}$

eliminated. As it turned out, eleven of the fifteen runs made on data sets A, B, and C using the five partition ratios above resulted in $\bar{E}_J = \bar{E}_1^*$. For the Iris data, Phase 2 processing using X_2 resulted in lowering $\bar{E}_J$ to $\bar{E}_2^* < \bar{E}_1^*$ in all but the (1:9) case; in this latter instance $n_d = 12$ points resulted in no error, whereas $n_d + n_2 = 114$ points had a 7% error rate. We mention that in the four instances where $\bar{E}_J = \bar{E}_2^*$, the improvement on passing to Phase 2 only ranged from 0.2% to 2.5%. In view of this it seems that for 'reasonable' ratios of $(n_d : n_2)$, Phase 2 of Jóźwik's algorithm may not be very cost effective. Consequently, the four classifiers are compared below on common data sets which, in the notation of equation (22), have the common labels (X_d, W_d) and (X_t, W_t). In other words, our discussion follow the notation in (9); the partition ratios shown in the tables are real ones – they look a bit bizarre because data sets labelled X_2 in (22) have simply been ignored (the ratios $(n_d : n_2)$ *were* integer valued!).

Finally, we simplify the notation for the discussion below by letting the output of PROC.KNNR be called $(k_{nn}, \bar{E}_{nn})$ when $(k^*, \bar{E}(D_{k^*}))$ are the result of using PROC.KNNR in the crisp mode (i.e., when the labels for the design data are hard, $W_d = W_{nn} \in M_{cn_d}$). Summarizing, we will compare four classifiers with notations for common parameters as shown in Table 1.

6. Numerical examples

Tables 2–4 summarize the data processing of A, B, and C with the four classifiers under examination. The columns are arranged (left to right) in ascending order of the partition ratio n_d/n_t. Consequently, one generally expects $\bar{E}(D)$ to decrease (moving right in these tables) as the number of design samples increases (the number of *test* samples are fixed in each table). This is not always the case, nor is it true that the optimal number of neighbors always decreases with an increase in the number of training samples. These anomalies are probably due to the vagaries of random sampling. Comparing Tables 2–4 further, there is no apparent trend between (optimal) numbers of neighbors (k^*) and the various algorithms. Indeed, the Fuzzy J/k-NNR, e.g., uses *on average* the most (4.6), intermediate (5.2), and least (2.4) number of neighbors compared to the other NNR's to achieve minimal $\bar{E}$ rates over the three sets A, B, and C, respectively. In terms of computing then, there seems little hope in being able to predict *a priori* that any particular generalized k-NNR will be more efficient (use a fewer number of neighbors) than a competitor; and moreover, increasing the size of the

Table 2. Empirical error rates (in percent) for data set A: The Iris data

		Partition ratio n_d/n_t with $n_t = 36$					
		0.33	1.00	1.58	2.16	2.83	Row-wise average
k^*	k_{nn}	1	1	3	4	1	2
	k_J	1	1	1	10	10	4.6
	k_{fcm}	1	6	1	1	1	2
$\bar{E}$	$\bar{E}_{nn}$	0.0	2.8	0.0	0.0	2.8	1.12
	$\bar{E}_J$	0.0	2.8	2.8	2.8	5.6	2.8
	$\bar{E}_{fcm}$	38.9	33.3	5.6	5.6	5.6	17.8
	$\bar{E}_{fnp}$	38.9	38.9	5.6	5.6	11.1	20.0

design set does *not* necessarily decrease either the optimal number of neighbors k^* or the corresponding empirical error rate $\bar{E}(D_{k^*})$.

Turning to the row-wise averages of E we note from Table 2 that the Hard and Fuzzy J/k-NNR's both achieved very good results with data set A. Both FCM-based rules were on average quite poor, the worst performance occurring when $n_d/n_t = 0.33$ where both FCM designs had 38.9% error rates. The high error rates for both FCM-based designs at $n_d \leq n_t = 36$ are in all likelihood due to the fact that (W_{fcm}, v_{fcm}) are being estimated in *unsupervised* fashion for (3) classes in 4-dimensional feature space using either 4 or 12 *unidentified* samples per class! Statistically, this is quite obviously the cause of poor performance here. On the whole, Table 2 suggests that when n is small relative to c and k, it is probably a bad idea to ignore the information reposited in (given) hard labels: the Hard and Fuzzy J/k-NNR's don't, whereas the FCM-based classifiers do.

Turning to Table 3, one finds quite a different situation. Data set B has a calculable Bayes error rate of $E_B = 31\%$, so *in the limit* a lower rate is theoretically impossible. However, the row-wise averages of $\bar{E}_{fcm}$ and $\bar{E}_{fnp}$ are, respectively, 29.4% and 29.8%. This re-emphasizes a point made in an earlier comparison of $\bar{E}_{nn}$ to $\bar{E}_{fnp}$ [10], viz., that there is nothing in the theory to preclude

Table 3. Empirical error rates (in Percent) for data set B: A univariate normal mixture

		Partition ratio n_d/n_t with $n_t = 100$					
		0.90	3.00	4.50	6.00	8.10	Row-wise average
k^*	k_{nn}	9	15	15	18	16	15
	k_J	1	9	8	3	5	5.2
	k_{fcm}	8	2	1	1	1	2.6
$\bar{E}$	$\bar{E}_{nn}$	40	29	34	34	27	32.8
	$\bar{E}_J$	32	32	36	40	31	34.2
	$\bar{E}_{fcm}$	31	29	29	29	29	29.4
	$\bar{E}_{fnp}$	33	29	29	29	29	29.8
E_B		31	31	31	31	31	31.0

better performance than E_B with finite sets of samples. Indeed, the values in Table 3 reinforce our remarks above; it is hard to interpret $\bar{E}(D)$ in (14) as more than a very rough estimate of E_B, not is it safe to assume that $\bar{E}(D)$ is necessarily greater than the 'true but unknown' Bayes error rate. Our use of $\bar{E}(D)$ for a given D as a predictor of future performance *with samples of the same type and size* seems uncontrovertible. In this context Table 3 suggests that the FCM/k-NNR is the best choice for data such as B, predicting an average error rate for D_{fcm} which is about 4.8% lower than $\bar{E}_J = 34.2\%$. Note further that on average only 0.4% separate the FCM/k-NNR and FCM/1-NPR error rates. For these data, the most cost-efficient classifier is D_{fnp} because the storage and calculations involved in (18b) + (16) for the FCM/1-NPR are less than for (18a) + (12) for the FCM/k-NNR. Finally, we point out two differences between Tables 2 and 3 and might further account for the reversal in rankings of the two FCM designs with the Hard and Fuzzy J/k-NNR's; (i) the data in B are *normally distributed,* and (ii), training sample sizes run from $n_{di} = 45$ to $n_{di} = 405$ (unlabelled) samples per class in Table 3. Both of these facts seem to improve the quality of FCM-based designs.

Using the same method of ranking the four designs in Table 4 again yields the FCM/k-NNR as the classifier of choice; and again, the Fuzzy J/k-NNR is ranked last, with a row-wise average, $\bar{E}_J = 28.6\%$, some 3.8% higher than $\bar{E}_{fcm}$. Note here that none of the four designs predict (on average) an $\bar{E}(D)$ lower than $E_B = 0.24$. All four are again good 'ballpark' estimates (within 5% in all cases for both sets B and C) of E_B.

Another aspect of our study concerned the termination criterion of Jóźwik, step (b) of (J1.3). To investigate this stopping rule we ignored the criterion and ran Phase 1 of Jóźwik's rule out to $h = 50$ and then plotted $\bar{E}_h$ as a function of h. The results, shown in Figure 2, are quite interesting because three radically different types of graphs were observed. Figure 2(a) involves the IRIS data while Figures 2(b) and 2(c) involves data distributed as two-class mixtures of $N(\mu_i, 1)$ normals. For, view 2(a), $\bar{E}_h$ oscillates from a 'local' minimum of 2.8% to a local maximum of 4.0%, having five aperiodic cycles over the integers 0 to 50. Since

Table 4. Empirical error rates (in percent) for data set C: A bivariate normal mixture

		Partition ratio n_d/n_t with $n_t = 100$					
		0.90	3.00	4.50	6.00	8.10	Row-wise average
k^*	k_{nn}	6	8	16	17	11	11.6
	k_J	3	1	1	2	5	2.4
	k_{fcm}	7	9	13	22	2	10.6
$\bar{E}$	$\bar{E}_{nn}$	24	24	27	27	27	25.8
	$\bar{E}_J$	28	26	28	32	29	28.6
	$\bar{E}_{fcm}$	23	25	25	25	26	24.8
	$\bar{E}_{fnp}$	24	26	27	27	27	26.2
E_B		24	24	24	24	24	24.0

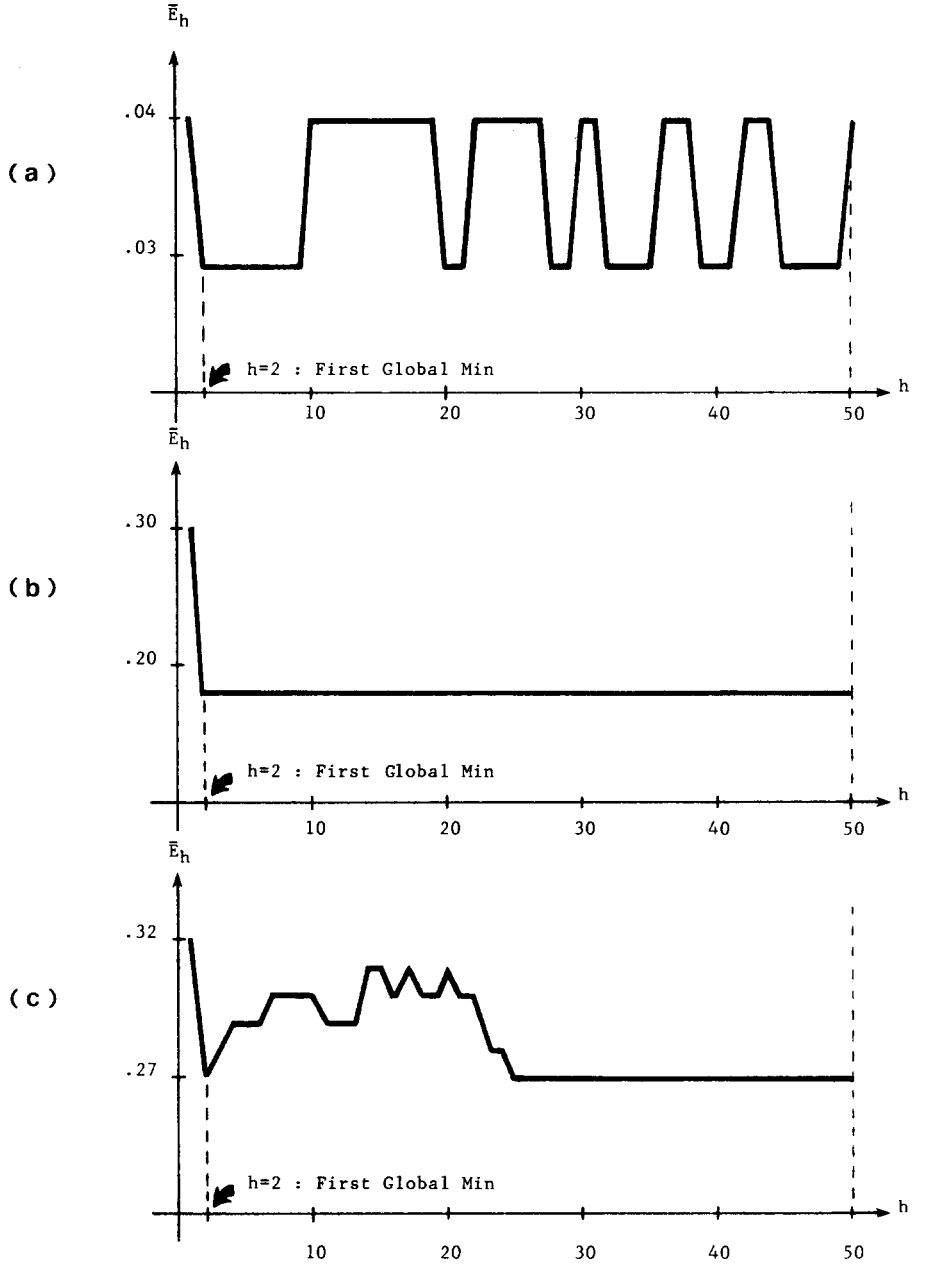

Fig. 2. Optimal (Phase 1) error rates for the J/k-NNR as a function of iterate h.

$\bar{E}_2 = \bar{E}_3 = 2.8\%$, (J1.3(b)) would terminate Phase 1 at $h = 3$ with $k_1^* = 1$. The point of Figure 2(a) is to discover if some higher value of h might result in an $\bar{E}_h$ even lower than $\bar{E}_2$. This was not the case; (J1.3(b)) here stops Phase 1 at the global minimum of $\bar{E}_h$; and it occurs at the first (i.e., least) value of h where $\bar{E}_h$ minimizes with respect to $\bar{E}_{h+1}$. Figure 2(b) shows a much different behavior

using a different data set – but it supports the same conclusions. Namely, $\bar{E}_h$ achieves a global minimum as early as possible (again at $h = 2$), and never increases as h runs out to 50. Finally, a third type of behavior is manifested by the graph shown in Figure 2(c), wherein $\bar{E}_h$ has global minimum at $h = 2$ and again for all h from 35–50. Note, however, that in this third case $\bar{E}_h$ has several *local* minima at larger values of $\bar{E}_h$ than $\bar{E}_2$. In all three cases, these examples suggest that Jóźwik's stopping rule possesses the properties one would like it to have; it *does* attain a global minimum in the smallest possible number of iterations. Moreover, in all three cases, this happens at $h = 2$. One wonders if this might be the case in *every* instance!

7. Conclusions

First, some remarks about Jóźwik's design. Our calculations indicate that this algorithm is well defined in the sense that the stopping rule is consistent. A theoretical investigation supporting our empirical results concerning $\bar{E}_h$ is desirable. Our experience with J/k-NNR indicates that Phase 2 of the algorithm seldom improves the performance of the Phase 1 design. In the four cases with data set A where Phase 2 did improve the Phase 1 result, the average improvement in $\bar{E}_J$ was 1.2%. The cost of this marginal decrease in terms of computing is quite high. In general it appears that modifying the J/k-NNR by simply stopping at (J1.3) (perhaps always at $h = 2$?) usually produces the optimal result. Finally, the J/k-NNR was always within a few percent of the 'best' (in terms of $\bar{E}(D)$) design, but in our experiments never out-performed all of its competitors. However, with small data sets (such as A) we feel that utilization of the labelling informaion residing in W_d is necessary, and the J/k-NNR does so in a very interesting manner. This aspect of Jóźwik's algorithm deserves further study.

Second, what can be said about the FCM/k-NNR? This design provided the best results for both normal mixtures among all the designs compared, so it seems safe to assert that the FCM/k-NNR at least warrants more extensive field tests whenever NNR's are being used. It is too early to conclude that *normality* is an important precondition for success with the FCM/k-NNR. However, evidence that this may be true is mounting (cf. [11, 12] for additional examples in other contexts that suggest FCM is particularly effective when the $g(\boldsymbol{x} \mid j)$'s are Gaussian). A further supposition along these lines concerns the link between the norm metric Δ used in (11), the norm used to measure similarity in FCM, and the shape of level sets of the $g(\boldsymbol{x} \mid j)$'s. In examples B and C, the Euclidean norm (for Δ and FCM) is well matched with the covariance structure of the g's, because both are essentially circular, whereas the structure of the g's is unknown in example A. Although FCM makes no statistical assumptions whatsoever, it does seem to consistently yield its best performance when the data are drawn from a mixture of normals. The theoretical reason for this apparent behavior has not yet emerged.

Do W_{fcm} in (18a) and $\boldsymbol{v}_{fcm}$ in (18b) contain the same *information*? More specifically, are the FCM/k-NNR and FCM/1-NPR operationally equivalent

classifiers? On the basis of the results in Tables 2–4 it appears that, for all practical purposes, the answer is yes. The classifier D_{fcm} consistently exhibited beter performance than D_{fnp} – but how much better? Over fifteen runs, the FCM/k-NNR had an average error rate about 1.33% less than the FCM/1-NPR! Against this negligibly small improvement are three very important advantages for D_{fnp}. First, D_{fnp} needs much smaller storage: once found, only $\boldsymbol{v}_{fcm}$ need be stored (cs real numbers), as compared to D_{fcm} storing X_d and W_{fcm} ($n_d(s+c)$ real numbers). For example, the storage required for these parameters in column (5) of Table 4 is $cs = 4$ for D_{fnp} whereas $(n_d(s+c)) = 810(2+2) = 3240$ for D_{fcm}. Second, the number of operations required per decision is as follows: D_{fnp} requires c distance calculations, whereas D_{fcm} needs n_d such computations. Using the same column of Table 4 for this comparison, D_{fnp} completed its test of X_t by calculating $2(100) = 200$ distances; D_{fcm} needed $810(100) = 81\,000$! Thus, D_{fnp} is at least several orders of magnitude better than D_{fcm} in terms of both storage and CPU time. And last, D_{fnp} is more easily implemented in hardware for real-time applications. These factors may or may not outweigh the apparent reduction in $\bar{E}$ realized by the FCM/k-NNR as compared to the FCM/1-NPR – but certainly they should be kept in mind!.

Do the examples above provide enough evidence to justify further study of generalized NNR's? Certainly (11) raises some interesting theoretical issues: what convergence theory can be established for decision rule (11)? What about upper and lower bounds on E_B? Are there theoretical reasons for preferring a particular form of the generalized NNR? And for the applications community, the numerical examples presented above certainly establish the *utility* of generalized k-NNR's. Several investigations are underway to corroborate some of these suppositions.

References

[1] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis (Wiley-Interscience, New York, 1974).

[2] K. Fukunaga and T.E. Flick, Density identification and risk estimation, IEEE Trans. Pattern Anal. Mach. Intell. (1985) to appear.

[3] J.M. Garnett, III and S.S. Yau, Nonparametric estimation of the Bayes error of feature extractors using ordered nearest neighbor sets, IEEE Trans. Comput. 26 (1977) 46–54.

[4] J. Kittler and P.A. DeVijver, An efficient estimator of pattern recognition system error probability, Pattern Recognition 13 (1981) 245–249.

[5] A. Jóźwik, A learning scheme for a fuzzy k-NN rule, Pattern Recognition Lett. 1 (1983) 287–289.

[6] R.P.W. Duin, The use of continuous variables for labelling objects, Pattern Recognition Lett. 1 (1982) 15–20.

[7] J. Bezdek, Pattern Recognition with Fuzzy Objective Functions (Plenum, New York, 1981).

[8] T.M. Cover and P.E. Hart, Nearest neighbor pattern classification, IEEE Trans. Inform. Theory 13 (1967) 21–27.

[9] S. Chuah and J. Bezdek, Optimal classifier design using fuzzy k-nearest neighbor rules, in: J. Kacprzyk and S. Orlovski, Eds., Soft Optimization Models Using Fuzzy Sets and Possibility Theory (1986), to appear.

[10] J. Bezdek and P. Castelaz, Prototype classification and feature selection with fuzzy sets, IEEE Trans. Systems Man Cybernet. 7 (1977) 87–92.

[11] J. Bezdek and J. Dunn, Optimal fuzzy partitions: A heuristic for estimating the parameters in a mixture of normal distributions, IEEE Trans. Comput. 24 (1975) 835–838.
[12] J. Bezdek, R. Hathaway and V. Huggins, Parametric estimation for normal mixtures, Pattern Recognition Lett. 3 (1985) 79–84.
[13] J. Bezdek, R. Ehrlich, W. Full, FCM: The fuzzy c-means clustering algorithms, Comput. Geosci. 10 (1984) 191–203.
[14] R.D. Short and K. Fukunaga, The optimal distance measure for nearest neighbor classification, IEEE Trans. Inform. Theory 27 (1981) 622–627.
[15] R.A. Fisher, The use of multiple measurements in taxanomic problems, Ann. Eugenics 7 (1936) 179–188.

Appendix. PROC.KNNR: The generalized k-NNR

Input: Δ, a metric for R^s
(X_d, W_d) a labelled design set, W_d hard or fuzzy
(X_t, W_t) a labelled test set, W_t hard
$(l_{1,\ldots,} l_c)$ a set of voting thresholds for (11)
Parameters: c, n_d, n_t, $k_{\max}$ (maximum number of neighbors tried)
Initialize: $\bar{E}(D_{k_{\max}}) = (0, 0, \ldots, 0)$
Loop
For $j = 1$ to n_t
  Choose $\boldsymbol{x}_j \in X_t$
  Find $N_k(\boldsymbol{x}_j)$ per (10c)
  Find $J_k(\boldsymbol{x}_j)$ per (10d)
  For $i = 1$ to $k_{\max}$
    Find $L_i(\boldsymbol{x}_j)$ per (10e)
    Find $D_i(\boldsymbol{x}_j)$ per (11), $D = D_i$, break ties arbitrarily
    Find $e_i(\boldsymbol{x}_j)$ per (14b), $D = D_i$
    $\bar{E}(D_i) = \bar{E}(D_i) + e_i(\boldsymbol{x}_j)/n_t$
  Next i
Next j
Find $(k^*, \bar{E}(D_{k^*}))$ per (15)
Output: $(k^*, \bar{E}(D_{k^*}))$ as in (15), the optimal number of neighbors and minimum predicted error rate for the design set (X_d, W_d).

Note 1. This procedure automatically finds the smallest number k^* of nearest neighbors to use with (X_d, W_d) that produces the minimal error rate $\bar{E}(D_{k^*})$ on (X_t, W_t).

Note 2. One can send (X_d, W_d) to this procedure twice, i.e., as both the design set and test set, provided W_d is hard. PROC.KNNR accessed in this fashion corresponds to the so-called 'leave-one-out' method of establishing $\bar{E}(D_{k^*})$ using all of the samples for design.