International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems
Vol. 6, No. 5 (1998) 437—457
© World Scientific Publishing Company

AN INTEGRATED FRAMEWORK FOR GENERALIZED NEAREST PROTOTYPE CLASSIFIER DESIGN

LUDMILA I. KUNCHEVA[1] and JAMES C. BEZDEK[2]

[1] *School of Mathematics*
University of Wales at Bangor, Gwynedd LL57 1UT, UK
l.i.kuncheva@bangor.ac.uk

[2] *Department of Computer Science*
University of West Florida, Pensacola, FL 32514, USA
jbezdek@argo.cs.uwf.edu

Received (received date)
Revised (revised date)

We propose a *Generalized Nearest Prototype Classifier* (GNPC) as a common framework for a number of classification techniques. Specifically we consider clustering-and-relabeling; Parzen's classifier; *radial basis functions* (RBF) networks; *learning vector quantization* (LVQ) type classifiers; and nearest neighbor rules. To classify an unlabeled point $\mathbf{x}$, the GNPC combines the degrees of similarity of $\mathbf{x}$ to a set of prototypes. Five questions are addressed for these GNPC families: (1) How many prototypes do we need? (2) How are the prototypes found? (3) How are their class labels obtained? (4) How are the similarities defined? (5) How are the similarities and label information combined? The classification performance of a set of GNPCs is illustrated on two benchmark data sets: IRIS and the 2-spirals data. We study the resubstitution error of the GNPC as a function of the number of prototypes. Our conclusions are that: (a) unsupervised selection (or extraction) of prototypes followed by relabeling is inferior to the techniques that use labels to guide them towards prototypes; (b) the edited nearest neighbor rule is a viable option for GNPC design which has not received the attention it deserves.

Keywords: Pattern recognition; Prototype classifiers; Nearest neighbor; Neural networks (RBF, LVQ); Generalized nearest prototype classifier

1. Introduction

Prototype based classification is perhaps the simplest and most intuitively motivated pattern recognition paradigm [14,38]. We consider c mutually exclusive classes where c is an integer, $c \geq 2$. Let $I_{crisp} = \{\mathbf{e}_1, \ldots, \mathbf{e}_c\}$ be the canonical basis of $\Re^c$, used here as crisp class labels, i.e., objects from class i are labeled by the vector $\mathbf{e}_i = [e_{1i}, \ldots, e_{ci}]^T$, $e_{ji} = 1$ if $j = i$, and 0, otherwise. Any function $D : \Re^d \rightarrow I_{crisp}$ is called *a crisp classifier.* Let $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$, $\mathbf{v}_i \in \Re^d$ be a set of prototypes with labels $l(\mathbf{v}_i) \in I_{crisp}$. We denote by $\mathbf{L}_V$ the matrix of labels $[l(\mathbf{v}_1), \ldots, l(\mathbf{v}_p)]$. An unlabeled object $\mathbf{x}$ is subsequently assigned to the class of the closest $\mathbf{v}_i \in V$, and "closest" is usually defined in terms of a distance on $\Re^d$.

Definition 1. *The Nearest Prototype Classifier $D_{NPC} : \Re^d \rightarrow I_{crisp}$ is the triplet $(V, \mathbf{L}_V, \Delta)$, where Δ is any norm metric on $\Re^d$, such that for any $\mathbf{x} \in \Re^d$*

$$D_{NPC}(\mathbf{x}) = l(\mathbf{v}_k) \in I_{crisp} \iff$$

$$\Delta(\mathbf{x}, \mathbf{v}_k) \leq \Delta(\mathbf{x}, \mathbf{v}_i);\ \forall i \neq k. \tag{1}$$

Ties are broken randomly. Integer p may or may not equal c.

Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \Re^d$ be a set of labeled data, and let $\mathbf{L}_X = [l(\mathbf{x}_1), \ldots, l(\mathbf{x}_n)]^T$, $l(\mathbf{x}_i) \in I_{crisp}$ be the corresponding matrix of crisp labels. When $V = X$ and $p = n$, then D_{NPC} is the *nearest neighbor* (1-NN) rule. When $p = c$ and V are obtained as the sample means for each class in X, D_{NPC} is often called the *minimum-distance* classifier [14,38] (we denote it by 1-NP). Both designs can be regarded as special cases of the nearest prototype classifier at Definition 1. Our terminology is chosen so that D_{NPC} is the 1-NP *only* when V are the labeled subsample means. Definition 1 does not specify how to find V.

Along with the great variety of 1-NP and 1-NN classifiers [12] there are many other classification techniques that are based explicitly or implicitly on similarity to prototypes, for example RBF networks [23]. The prototypes are not necessarily firmly associated with crisp classes: each prototype may have a label vector $l(\mathbf{v}_i) = [l_{1i}, \ldots, l_{ci}]^T \in [0,1]^c - \{\mathbf{0}\}$ of association weights ($\{\mathbf{0}\}$ is the zero vector for $\Re^c$). The common characteristic of nearest prototype classifiers is that they calculate *similarities* $\{s(\mathbf{x}, \mathbf{v}_i)\}$ of the unlabeled point $\mathbf{x}$ to each of the prototypes $\{\mathbf{v}_i\}$ and then combine the $\{s(\mathbf{x}, \mathbf{v}_i)\}$, with $\{l(\mathbf{v}_i)\}, i = 1, \ldots, p$, to label $\mathbf{x}$. Intuitively, the closer $\mathbf{x}$ is to $\mathbf{v}_i$ *and* the higher the association weight l_{ji} is, the greater the plausibility becomes that $\mathbf{x}$ should be assigned to class j.

We seek a paradigm based on this common feature that will unify many disparate techniques and provide a common framework for comparative analysis. The *Generalized Nearest Prototype Classifier* (GNPC) defined in Section 2 provides such a framework. The classifiers that fit within it are roughly grouped into five intersecting families:

F1. Clustering-and-relabeling [4];

F2. Parzen's classifier [15];

F3. RBF networks [6,20];

F4. LVQ-type classifiers [28];

F5. Edited nearest neighbor rules [12].

Some excellent comparative studies on pattern classifiers have been published recently [7,21]. Holmström at al. [21] discuss neural and statistical classifiers and emphasize that the boundary between the two areas is not well defined. We do not provide another comparison of various methods on large data sets. Our objective is to assess classifiers within the GNPC framework in terms of how effectively they find and use prototypes, so our experiments are illustrative rather than central to this study.

Section 2 describes the GNPC framework and Section 3, the pattern classification techniques that fit within it. The classification techniques discussed are not exhaustive – the aim is to present a common framework so that techniques that fit within the representation can benefit from each other, or from the common framework itself.

In Section 4 we present experimental results with a set of GNP classifiers on the IRIS and the 2-spirals data sets. We study only the resubstitution error rate, as the data sets we use are not good choices for assessing generalization. The IRIS data are too small to be effectively split for cross-validation purposes, and the training and test sets of the 2-spirals data are virtually identical, providing almost equal resubstitution and generalization errors. We have chosen these two sets because they are extensively used as benchmarks and are widely available. Moreover, some experimental results with the same data published by other authors facilitate further comparison. Section 5 contains our comments and conclusions.

2. GNPC as a common framework

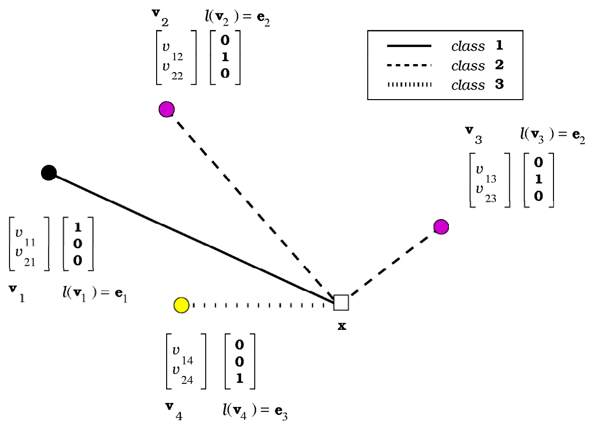
We use real-valued prototypes $\mathbf{v}_i \in \Re^d, i = 1, \ldots, p$. More complex prototype structures such as hyperplanes [5] and higher order hypersurfaces [30] are not considered, nor are special techniques for handling categorical or mixed variables. Our model is quite flexible however, because associated with each prototype $\mathbf{v}_i$ is a set of parameters θ_i which are used when calculating the *similarity* of $\mathbf{x}$ to $\mathbf{v}_i$. For example, θ_i might be a covariance matrix used to calculate Mahalanobis distances, or it might be a single value called a "smoothing parameter".

The number of prototypes p can be less than, equal to, or greater than the number of classes c. Figure 1 shows an example with $p = 4$ prototypes in $\Re^2$ that are associated with $c = 3$ classes by the *crisp* label vectors $\mathbf{L}_V = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4]$. The solid, dashed, and dotted lines indicate the crisp relationship of prototypes to classes 1, 2, and 3, respectively.

Using Euclidean distance (i.e., $d_E(\mathbf{x}, \mathbf{v}_i) = \sqrt{(\mathbf{x} - \mathbf{v}_i)^T (\mathbf{x} - \mathbf{v}_i)}$), the classical 1-NP classifier [38] will assign label 2 to the vector $\mathbf{x}$ in Figure 1 because $d(\mathbf{x}, \mathbf{v}_3) < d(\mathbf{x}, \mathbf{v}_i)$, $i = 1, 2, 4$, and $l(\mathbf{v}_3) = \mathbf{e}_2$. Since $p > c$ in Figure 1, some authors call this a nearest multiple prototype (1-nmp) design [4,38].

It is natural to assume that the similarity s between $\mathbf{x}$ and $\mathbf{v}_i$ has maximal value when $\mathbf{x} = \mathbf{v}_i$ and decreases with the distance between the two vectors in $\Re^d$. In general, similarity measures are required to be only reflexive and symmetric. In this paper we restrict similarity a little more than this.

Definition 2. *A norm-induced similarity function* $s(\mathbf{x}, \mathbf{v}_i; \theta) \equiv s(\Delta(\mathbf{x}, \mathbf{v}_i); \theta)$, *where* θ *is a set of parameters of* s, *is any monotonically decreasing function* $s : \Re^+ \to [0, 1]$ *of any norm metric* Δ *on* $\Re^d$.



class 1
class 2
class 3
$\mathbf{v}_1$ $l(\mathbf{v}_1) = \mathbf{e}_1$
$\mathbf{v}_2$ $l(\mathbf{v}_2) = \mathbf{e}_2$
$\mathbf{v}_3$ $l(\mathbf{v}_3) = \mathbf{e}_2$
$\mathbf{v}_4$ $l(\mathbf{v}_4) = \mathbf{e}_3$
$\mathbf{x}$

Figure 1: Nearest prototype classifier with crisp labels

For example, s might be

$$s(\mathbf{x}, \mathbf{v}_i; \theta) \equiv s\left(d_E(\mathbf{x}, \mathbf{v}_i); \phi\right) = \exp\left(-\frac{1}{2} d_E(\mathbf{x}, \mathbf{v}_i)^2\right). \tag{2}$$

Assigning to $\mathbf{x}$ the label of the *most similar* prototype is equivalent to assigning to $\mathbf{x}$ the label of the *nearest* prototype. Another widely used example is based on the Mahalanobis norm,

$$s(\mathbf{x}, \mathbf{v}_i; \theta) \equiv s\left(d_M(\mathbf{x}, \mathbf{v}_i); \{S\}\right) = \exp\left(-\frac{1}{2} d_M(\mathbf{x}, \mathbf{v}_i)^2\right). \tag{3}$$

where

$$d_M(\mathbf{x}, \mathbf{v}_i) = \sqrt{(\mathbf{x} - \mathbf{v}_i)^T S^{-1} (\mathbf{x} - \mathbf{v}_i)}, \tag{4}$$

where S is a covariance matrix. Using d_M and $h_i \in \Re^+$ as a smoothing parameter associated with prototype $\mathbf{v}_i$, s can be defined as the Gaussian kernel centered at $\mathbf{v}_i$ [15],

$$s(\mathbf{x}, \mathbf{v}_i; \theta_i) \equiv s\left(d_M(\mathbf{x}, \mathbf{v}_i); \{h_i, S\}\right) =$$

$$= \frac{1}{(2\pi)^{d/2} h_i^d \sqrt{|S|}} \exp\left(-\frac{1}{2h_i^2} d_M(\mathbf{x}, \mathbf{v}_i)^2\right). \tag{5}$$

To simplify notation we write $s(\mathbf{x}, \mathbf{v}_i; \theta_i) = s_i$ and call s a similarity function, meaning always the *norm-induced similarity* as at Definition 2.

Instead of using crisp labels we may let l_{ji} vary in the interval [0,1], thereby providing soft connections between the prototypes and the classes. When we restrict the sum of labels to one this is called either *fuzzy* or *probabilistic* labeling [4]. An example of such a label vector for prototype $\mathbf{v}_i$ and $c = 4$ is $l(\mathbf{v}_i) = [0.1, 0.7, 0.0, 0.2]^T$.

An even looser connection between prototypes and classes is provided by the *possibilistic* label [19], where the sum of elements of $l(\mathbf{v}_i)$ is not restricted to one.

Figure 2 shows the example in Figure 1 but with possibilistic labels, and leads you to ask: what crisp label should be assigned to $\mathbf{x}$? (i.e., how are the $\{s(\mathbf{x}, \mathbf{v}_i)\}$ combined with $\{l(\mathbf{v}_i)\}$?). The soft labeling of prototypes is indicated in Figure 2 by using all three types of lines (solid, dashed, and dotted) to link each prototype to $\mathbf{x}$.

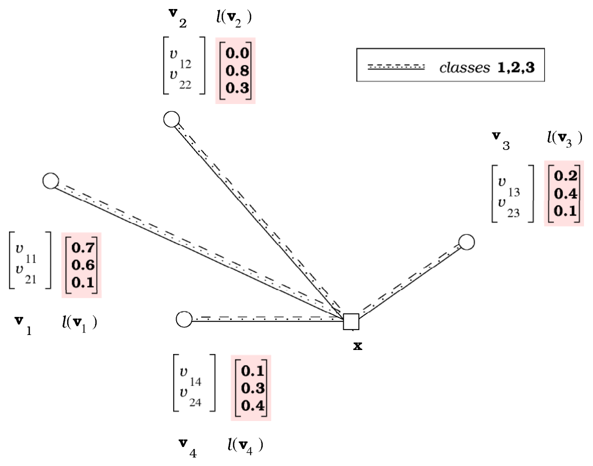


Figure 2: Nearest prototype classifier with possibilistic labels

Finally, we need to compute the membership of $\mathbf{x}$ in each class. This is done below with an aggregation function $\mathcal{A}$ that is generalized matrix multiplication using an $\mathcal{S}$ operator instead of summation and a $\mathcal{T}$ operator instead of multiplication. Let $Q = [q_{ij}]$ and $R = [r_{ij}]$ be two matrices of size $n \times m$ and $m \times k$, respectively. The matrix $\mathcal{A}(Q, R) = [a_{ij}]_{n\times k}$ is defined by

$$a_{ij} = \underset{t=1}{\overset{m}{\mathcal{S}}} \left(\mathcal{T}(q_{it}, r_{tj})\right) \tag{6}$$

We assume that $\mathcal{T}$ is a monotonic on each argument, commutative and associative binary operator that maps $[0,1] \times [0,1] \to [0,1]$. A natural choice for T are the t-norms defined over fuzzy sets, e.g. minimum $\equiv \min\{z_1, z_2\}$; product $\equiv z_1.z_2$; bold union $\equiv \max\{0, z_1 + z_2 - 1\}$, $z_1, z_2 \in [0,1]$ [41].

$\mathcal{S}$ is an aggregation operator, $\mathcal{S} : [0,1]^m \to [0,1]$. We can choose $\mathcal{S}$ to be a *mean aggregation operator* [41] which is commutative, monotonically nondecreasing on each argument, and idempotent. Examples of such operators are the simple average $\equiv (z_1 + z_2 + \ldots + z_m)/m$; and the maximum $\equiv \max\{z_1, \ldots, z_m\}$, $z_i \in [0,1]$. The aggregation we use here results in the vector $\mathcal{A}(\mathbf{L}_V, \mathbf{s}) = \mathbf{a} = [a_1 \ldots a_c]^T$.

Definition 3. *The Generalized Nearest Prototypes Classifier (GNPC) is the 5-tuple* $(V, \mathbf{L}_V, s, \mathcal{T}, \mathcal{S})$ *where*

- $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_p\}$, $\mathbf{v}_i \in \Re^d$ *is the set of prototypes*;

- $\mathbf{L}_V \in [0,1]^{c\times p}$ *is the label matrix for the prototypes in* c *classes*;
- $s(\Delta(\mathbf{x},\mathbf{v}_i);\theta)$ *is a similarity function as at Definition 2*;
- $\mathcal{T}$ *is any t-norm defined over fuzzy sets, and* $\mathcal{S}$ *is an aggregation operator* [41]. *Thus the function* $\mathcal{A}$ *at (6) is completely specified.*

For an unlabeled vector $\mathbf{x} \in \Re^d$, *the GNPC calculates the similarity vector* $\mathbf{s} = [s_1,\ldots,s_p]^T$, *produces the label vector* $\mu(\mathbf{x}) = \mathcal{A}(\mathbf{L}_V,\mathbf{s})$, *and assigns the crisp class label* $\mathbf{e}_k \in I_{crisp}$ *to* $\mathbf{x}$ *if*

$$\mu_k(\mathbf{x}) = \max_{i=1,\ldots,c}\{\mu_i(\mathbf{x})\}. \tag{7}$$

Ties are broken randomly.□

We will show that the only operation types that are needed for the five families of classifiers studied here, are:

- $\mathcal{T}$ operations: **product**
- $\mathcal{S}$ operations: **maximum, average**

Note especially that the GNPC is a crisp classifier, no matter what type of labels are in the columns of $\mathbf{L}_V$. Generally, a GNPC representation can be built by answering five basic questions:

Q1. *How many* prototypes do we need? (p = ?)

Q2. Given (X,$\mathbf{L}_X$), how do we *find* the prototypes $\{\mathbf{v}_i\}$?

Q3. How do we *create the prototype label matrix* $\mathbf{L}_V$?

Q4. How do we *choose the similarity function* s and how do we obtain θ_i?

Q5. How do we *combine* the similarities $\mathbf{s}$ with the labels (specify $\mathcal{S}$ and $\mathcal{T}$) to find $\mathcal{A}(\mathbf{L}_V,\mathbf{s})$?

For the examples in Figures 1 and 2 we can design the following GNPC: ($V = \{\mathbf{v}_1,\mathbf{v}_2,\mathbf{v}_3,\mathbf{v}_4\}; \mathbf{L}_V = [\mathbf{e}_1,\mathbf{e}_2,\mathbf{e}_2,\mathbf{e}_3]; s(\mathbf{x},\mathbf{v}_i;\phi) = \exp\left(-(\mathbf{x}-\mathbf{v}_i)^T(\mathbf{x}-\mathbf{v}_i)\right)$; $\mathcal{T}$ is **product**; $\mathcal{S}$ is **max**). For the example in Figure 1 we obtain

$$\mathbf{a} = \mathcal{A}(\mathbf{L}_V,\mathbf{s}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.0002 \\ 0.0011 \\ 0.0369 \\ 0.0111 \end{bmatrix} = \begin{bmatrix} 0.0002 \\ 0.0369 \\ 0.0111 \end{bmatrix}.$$

Consequently, the GNPC label assigned to $\mathbf{x}$ by (7) is class 2. For the example in Figure 2, we obtain

$$\mathbf{a} = \mathcal{A}(\mathbf{L}_V, \mathbf{s}) = \begin{bmatrix} 0.7 & 0.0 & 0.2 & 0.1 \\ 0.6 & 0.8 & 0.4 & 0.3 \\ 0.1 & 0.3 & 0.1 & 0.4 \end{bmatrix} \begin{bmatrix} 0.0002 \\ 0.0011 \\ 0.0369 \\ 0.0111 \end{bmatrix} = \begin{bmatrix} 0.0074 \\ 0.0148 \\ 0.0044 \end{bmatrix}.$$

According to this result $\mathbf{x}$ will be given label 2 by our GNPC at (7).

There are many ways to generate prototypes: see (Bezdek and coauthors, 1999)[1] for extensive coverage of methods based on batch models (c-means), sequential methods (learning vector quantization), neural networks (aggregation networks) and heuristic methods (mountain clustering); see (Dasarathy, 1990)[12] for nearest neighbor methods, and (Duda and Hart, 1973)[14] for probabilistic methods.

3. The five families

Figure 3 represents the five groups of classifiers discussed in this section.We will exhibit relationships between each group and the GNPC (7). In what follows **A1-A5** stand for the *answers* to questions **Q1-Q5** shown in Section 2.

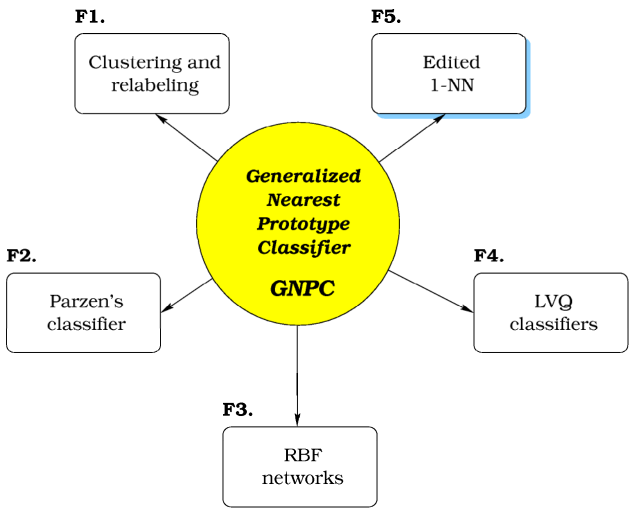


Figure 3: Groups of GNPCs

3.1. *Baseline designs*

Table 1 gives the **GNPC** representation of three *baseline* designs: 1-NN, 1-NP, and *linear discriminant analysis* (LDA) (equiprobable classes) [14]. Notation $\bar{V}$ in Table 1 stands for the sample means $\bar{V} = \{\bar{\mathbf{v}}_1, \ldots, \bar{\mathbf{v}}_c\}$

$$\bar{\mathbf{v}}_i = \frac{1}{n_i} \sum_{l(\mathbf{x}_j)=\mathbf{e}_i} \mathbf{x}_j, \quad 1 \le i \le c,$$

where n_i is the number of elements of X with class label $\mathbf{e}_i$.

Classifier $\rightarrow$	1-NN	1-NP	LDA
A1. (p)	$p = n$	$p = c$	$p = c$
A2. (V)	$V = X$	$V = \bar{V}$	$V = \bar{V}$
A3. $(\mathbf{L}_V)$	$\mathbf{L}_V = \mathbf{L}_X$	$\mathbf{L}_V = [\mathbf{e}_1, \ldots, \mathbf{e}_c]$	$\mathbf{L}_V = [\mathbf{e}_1, \ldots, \mathbf{e}_c]$
A4. (s, θ)	(2), d_E, ϕ	(2), d_E, ϕ,	(3), $d_M, \{S\}$
A5. $(\mathcal{T}, \mathcal{S})$	(prod, max)	(prod, max)	(prod, max)

Table 1: Baseline GNPC designs

For nonequiprobable classes, the only component of the GNPC representation of LDA that will change is the label matrix: $\mathbf{L}_V = [p_1\mathbf{e}_1, \ldots, p_c\mathbf{e}_c]$, where p_i is the a priori probability for class i. An example of a prototype label matrix for $c = 3$ nonequiprobable classes is

$$\mathbf{L}_V = \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.7 \end{bmatrix}.$$

Note that the representation of a classifier or a family of classifiers in terms of our GNPC *is not unique.* We may construct $\mathbf{L}_V$ in many ways, and choose proper $s, \mathcal{T}$, and $\mathcal{S}$ accordingly. Therefore, the rest of this section provides *a* GNPC representation of each of the five families of classifiers.

3.2. *Clustering and relabeling (F1)*

An easy scheme to build a GNPC is clustering followed by relabeling. First, data vectors are pooled disregarding the labels $\mathbf{L}_X$ and clustered into $p \geq c$ clusters. We call this *unsupervised design.* Each cluster is represented by a single prototype. Data vectors in the same cluster are supposed to possess much more similarity to each other than to the vectors in other groups.

The set of prototypes is then *relabeled* and used with D_{NPC} at Definition 1 [4]. A relabeling scheme that minimizes the overall number of resubstitution misclassifications is to label each cluster prototype with the class held by the majority of the vectors within that cluster. Let $C_i \subset X$ be the crisp cluster corresponding to prototype $\mathbf{v}_i$. The relabeling procedure assigns crisp label $l(\mathbf{v}_i) = \mathbf{e}_l \in I_{crisp}$ to prototype $\mathbf{v}_i$ if

$$\begin{aligned} &\mid \{\mathbf{x}_j | \mathbf{x}_j \in C_i \cap X, l(\mathbf{x}_j) = \mathbf{e}_l\} \mid = \\ &= \max_{k=1,\ldots,c} \mid \{\mathbf{x}_t | \mathbf{x}_t \in C_i \cap X, l(\mathbf{x}_t) = \mathbf{e}_k\} \mid \end{aligned} \tag{8}$$

GNPC representation:

A1. $1 \leq p \leq n$, the number of clusters p can be either fixed or decided empirically, e.g., by a cluster validity measure (usually $p \geq c$);

A2. V is found by clustering: any procedure that produces a set of point prototypes using a measure of similarity between $\mathbf{x}$ and $\mathbf{v}_i$ can be used.

A3. $\mathbf{L}_V$ is crisp and the columns are obtained by relabeling procedure (8).

A4. s is a similarity function (it may or may not require $\{\theta_i\}$);

A5. $\mathcal{T}$ is **product**, $\mathcal{S}$ is **max**.

Some authors use this type of GNPC to compare clustering procedures in terms of their classification accuracy [4,24,25]. Generally, the classification accuracy of GNPCs in this family will not be as high as if we used the class labels $\mathbf{L}_X$ to guide the procedure towards prototypes $\mathbf{V}$. Our experiments in Section 4 confirm this.

An advantage can be gained by making a soft relabeling, i.e., $l(\mathbf{v}_i) \in [0,1]^c - \{\mathbf{0}\}$, thereby accounting for all of the class labels $\{l(\mathbf{x}_j)\}$ represented within the i-th crisp cluster, $i = 1, \ldots, p$. For example, let C be a cluster containing m_1 samples from class 1, m_2 samples from class 2, and m_3 samples from class 3, $m_1 + m_2 + m_3 = m$. Then the prototype of cluster C can be labeled by the vector $\left[\frac{m_1}{m}, \frac{m_2}{m}, \frac{m3}{m}\right]^T$.

Alternatively, we may cluster separately the data vectors from X – one run for each class (*supervised design*). A special case of this design is the 1-NP classifier where we assume that the points of a class *are* one single cluster, and hence, the prototypes are computed as the sample means. Generally, more than one prototype per class can be used, the number of prototypes may differ from class to class, and the labels can be either crisp or fuzzy.

3.3. *Parzen's classifier (F2)*

Parzen's classifier is based on the statistical mixture model, and is thus built by approximating assumed class-conditional *probability density functions* (p.d.f.s) and plugging them in the Bayes classification rule [11,15]. Each data point is used as a prototype, i.e. $V = X, \mathbf{L}_V = \mathbf{L}_X$. A *kernel* function $K(\mathbf{x}, \mathbf{v}_i)$ is defined, usually taking the form $K\left(\frac{(\mathbf{x}-\mathbf{v}_i)}{h}\right)$, where h a smoothing parameter.

Using the kernel estimates of the class-conditional p.d.f.s (e.g., the Gaussian kernel (5) with identity covariance matrix) and the expression n_i/n as an estimate of the prior probability for class i. There are many kernel functions [15]. We can pick K to be valued in [0,1]. Using this formulation, the following set of discriminant functions can be derived,

$$g_k(\mathbf{x}) = \frac{1}{n\, h_k^d} \sum_{l(\mathbf{v}_j)=\mathbf{e}_k} K\left(\frac{(\mathbf{x}-\mathbf{v}_j)}{h_k}\right), k = 1, \ldots, c. \tag{9}$$

Parzen's classifier assigns to $\mathbf{x}$ the crisp label indexing the largest $g_i(\mathbf{x})$.

GNPC representation:

A1. $p = n$;

A2. $V = X$;

A3. $\mathbf{L}_V = \mathbf{L}_X$;

A4.

$$s_i = \frac{1}{n\theta_i^d} K\left(\frac{(\mathbf{x} - \mathbf{v}_i)}{\theta_i}\right),$$

where θ_i is the smoothing parameter for class k such that $l(\mathbf{v}_j) = \mathbf{e}_k$

A5. $\mathcal{T}$ is **product**, $\mathcal{S}$ is **average**.

A possible choice for h_k is $h_k = n_k^{-\frac{t}{d}}$, where t is a user-defined constant between 0 and 1 ([15], p. 174). This selection guarantees that the value of s is in [0,1].

In the classical Parzen's model the smoothing parameter h_k is the same for all n_k prototypes of class k. If we attach a specific h_i to each prototype, $i = 1, \ldots, p$, and use some training procedure to tune the value, the model is called *Kernel Discriminant Analysis* (KDA) [21]. Parzen's model is also isomorphic and isofunctional to *Probabilistic Neural Nets* (PNN) [35].

Using all the vectors in X as prototypes might be computationally infeasible. Therefore some *reduced* versions have been proposed where $p < n$. The main questions then becomes how to find these p "most relevant" prototypes and how to tune the parameter values. This relates the reduced Parzen (KDA, PNN) model to RBF networks where these are the central questions.

3.4. *Radial Basis Function Networks (F3)*

Let $\Phi(\mathbf{x}, \mathbf{v}_i; \theta_i)$ denote a radial basis function centered at $\mathbf{v}_i, i = 1, \ldots, p$. Generally, Φ has behavior similar to the kernel function in the Parzen's classifier: it takes its maximal value for $\mathbf{x} = \mathbf{v}_i$ and approaches zero when $||\mathbf{x} - \mathbf{v}_i|| \to \infty$.

The network has d inputs (one per feature), p hidden nodes (one per prototype) and c outputs $[y_1, \ldots, y_c]^T$ (one per class). When $\mathbf{x}$ is submitted to the network, the i-th output renders the value

$$y_k = \sum_{i=1}^{p} w_{ki} \, \Phi(\mathbf{x}, \mathbf{v}_i; \theta_i) = \mu_k(\mathbf{x}),$$

where $\{w_{ki}\}, k = 1, \ldots, c;\ i = 1, \ldots, p$ are the weight coefficients connecting the i-th hidden node with the k-th output. The crisp class label of $\mathbf{x}$ is decided by the index of the maximal $\mu_k(\mathbf{x}), k = 1, \ldots, c$, as in equation (7).

Parzen's classifier is a special case of RBF networks. Since Parzen's classifier is asymptotically Bayes-optimal (i.e., as $n \to \infty$, the classification error approaches the Bayes error from above), a class of sparsely connected RBF networks possesses the same property [11]. The advantage of RBF networks is their flexibility in the finite-sample case.

The prototypes (centers) can be selected or extracted in advance as the first stage of training, and kept fixed during the second training stage when $\{\theta_i\}$ and $\{l_{ki}\}$ are tuned, $k = 1, \ldots, c$. The other option is to initialize the prototype locations and to tune them together with the other network parameters during a single training stage. There are many ways to initialize the prototypes [20]: we can place them onto the vertices of a large grid in $\Re^d$, choose them at random from X, select them by nearest neighbor type editing of X, use an addition/deletion scheme, find clusters in X, evolve or select them by genetic algorithms [32,39,40], etc. The sets $\{\mathbf{v}_i\}, \{\theta_i\}$, and the label matrix $\mathbf{L}_V$ can be tuned in either a conjugated or independent way, thereby giving rise to many different RBF architectures.

Because of the restriction at Definition 2, we consider only RBF networks whose radial-basis function co-domain is [0,1], and whose hidden-output connections are numbers in the interval [0,1]. Therefore the GNPC does not incorporate RBF networks based on orthogonal least-squares training ([10]) (perhaps the most successful model). The GNPC definition does embed a class of RBF networks among which are those based on "semiparametric" mixture modeling [37] and also the classical scheme of Moody and Darken [33].

GNPC representation:

A1. $1 \le p \le n$;

A2. V is either selected and fixed in advance or tuned during the training stage;

A3. $l(\mathbf{v}_i) = [w_{1i}, w_{2i}, \ldots, w_{ci}]^T$;

A4. $s_i = \Phi(\mathbf{x}, \mathbf{v}_i; \theta_i)$; θ_i consists of the parameter values of the radial basis function associated with prototype $\mathbf{v}_i$.

A5. $\mathcal{T}$ is **product**, $\mathcal{S}$ is **average**.

3.5. *LVQ-type procedures (F4)*

LVQ classifiers compute V by sequential competitive learning [20,28]. Every point in the data set sequentially determines updates for one (or more) prototypes. The new value of prototype $\mathbf{v}_i$ at step t is denoted as $\mathbf{v}_{i,t}$ and is usually obtained as

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + a_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}), \quad i = 1, \ldots, p, \tag{10}$$

where $\mathbf{x}_k$ is the vector in X submitted to the algorithm at iterate t and $\{a_{ik,t}\}$ is the learning rate distribution. Depending on how $a_{ik,t}$ is formulated a variety of techniques can be described. There is an essential difference between supervised and unsupervised LVQ techniques. In unsupervised versions the prototypes are unlabeled and need a relabeling procedure to find $\mathbf{L}_V$ (like the one in the **F1** family of GNPCs). Supervised LVQ methods tune the prototypes according to a *reward-punishment* scheme so that the closest prototype from the same class moves towards, and the closest prototype from a different class moves away from $\mathbf{x}_k$. Equation (10) is used for the "reward" part, and

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} - a_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}), \quad i = 1, \dots, p, \tag{11}$$

for the "punishment" part.

Notice that in supervised LVQ classifiers the prototypes come with their labels. The labels are fixed in advance (prior to tuning the prototypes) and depend on the initialization procedure. They can be either assigned at random (if the initial V is a set of p randomly picked points in $\Re^d$) or taken as the labels of the p points of X used as the initial V.

Although LVQ classifiers are usually regarded as neural network classifiers, their representation is closer to the (non-neural-network) **F1** family than to the RBF networks family **F3**. Unsupervised LVQ classifiers differ from **F1** only in the way of *finding* the prototypes, and therefore can be regarded as a special case of **F1**. We consider **F4** separately because of the large number of LVQ models that have been developed. In contrast to the broad GNPC representation of **F1**, here we *specify* LVQ as the prototype extraction procedure in answer **A2**.

GNPC representation:

A1. $1 \le p \le n$, the number of prototypes is usually fixed in advance;

A2. V is found by an LVQ algorithm;

A3. $\mathbf{L}_V$ is crisp, and is fixed prior to prototype tuning.

A4. s is a similarity function as at Definition 2; $\theta_i = \phi$;

A5. $\mathcal{T}$ is **product**, $\mathcal{S}$ is **max**.

There is a great variety of techniques that can be used to find **V**. The main drawbacks of competitive learning methods are that they depend heavily on the initialization and are usually terminated by $a_{ik,t} \to 0$ in (10) and (11) rather than by any criterion related to the fitness of V for representation of X.

3.6. *Edited 1-Nearest Neighbor (F5)*

In numerous papers that compare pattern classifiers the classical nearest neighbor (1-NN) method is recognized as a good competitor to many neural networks and other classification paradigms. It is usually pointed out that, although one of the best, the 1-NN method requires large memory and time resources, and therefore other classifiers are recommended.

We believe that *edited* versions of 1-NN design are excellent examples of flexible, small, and accurate GNPCs. Editing techniques (see [12]) aim at finding V as a subset of X so that the classification accuracy using GNPC with V instead of X is not much affected. Since X is finite, classification accuracy (resubstitution *and* generalization) can be the same or even better with the edited set than error rates achieved using all of X.

Most editing techniques do not specify the number of prototypes they want to retain, i.e. p is data/technique dependent. For example, the *condensing* group of techniques [12] aims at retaining the minimal $V \subset X$, (called a "consistent" subset) that produces zero errors on X with the NPC. There are some "pseudo-editing" versions that do not *select* but *extract* prototypes by a subsequent merging of elements of V so that the resultant V is consistent [4,9].

It is desirable to develop an editing technique that finds V of a prespecified cardinality p with the lowest possible resubstitution error rate. Alternatively, we may fix a limit on the resubstitution error rate and search for the smallest V. In this study we tried *random selection* (RS) of $V \subset X$ with $|V| = p$, where p is fixed, $c \leq p < n$ (the algorithm is shown in Figure 4). The RS technique selects p elements of X at random and evaluates the GNPC resubstitution error. When p is small, RS editing is a surprisingly viable option for finding V. Better editing results at the expense of longer running time have been achieved by genetic algorithms [31].

```
EDITING BY RANDOM SELECTION
1. Choose $p, c \leq p < n$; set the maximal number of steps $T$; set $E = \infty$
2. For i=1 to $T$ do
    2.a. Select at random $W \subset X, |W| = p$;
    2.b. Calculate the resubstitution error rate $e(W)$ of the 1-NN classifier with $W$
         as the reference set;
    2.c. If $e(W) < E$ then $V \leftarrow W; E \leftarrow e(W)$
         end do
3. Return $E, W$
```

Figure 4: The RS editing algorithm

GNPC representation:

A1. p depends on the editing technique. In the RS technique p is specified in advance;

A2. V is usually a proper subset of X (except for the pseudo-editing techniques);

A3. $\mathbf{L}_V$ is constituted from the label vectors of the selected points from X;

A4. s is a similarity function; $\theta_i = \phi$;

A5. $\mathcal{T}$ is **product**, $\mathcal{S}$ is **max**.

Two families of soft classifiers are closely related to this group, viz., the fuzzy and possibilistic k-nearest neighbor classifiers (k-NN) [3,27,44]. Instead of crisp $\mathbf{L}_X$

they use fuzzy or possiblistic labels on X, $l(\mathbf{x}_j) \in [0,1]^c - \{\mathbf{0}\}$ and combine s with the labels of the k nearest to $\mathbf{x}$ points from X. In most of these methods the relevant question is how to "fuzzify" the crisp labels.

In this study we confined $\mathcal{S}$ to be either the maximum or classical *mean aggregation operator*. It should be emphasized that the k-NN models with $1 < k$ fit within the GNPC scheme if $\mathcal{S}$ is chosen as an *Ordered Weighted Averaging* (OWA) operator [41], i.e., the operator *selects* the k largest arguments and operates on only them.

3.7. *Other related designs*

There is a group of GNPCs that are not a coherent family and therefore are not treated as such here. This group comprises various heuristic schemes whose unifying concept is the connection between the 1-NN rule and noncompetitive-learning type neural networks such as *multilayer perceprtons* (MLP) and RBF networks. Two topics have been studied:

- constructing neural networks that implement *exactly* the 1-NN rule [8,16,29,34];
- optimizing the 1-NN (GNPC) performance by designing a neural network (of MLP or RBF type) that provides a close or exact representation of the GNPC, and tuning V (and θ_i, where applicable) [13,36,42,43,45,46].

The second option looks more promising because the GNPC parameters are derived by some neural network training procedure while "exact implementation" means that we either already have the prototypes V or use all of X.

Each of the above neural network designs is very specific and includes many heuristics: clustering and/or editing for initialization, special training algorithms, $\mathbf{L}_V$ definition, etc. Therefore it is difficult to erect a *common* representation. Since each architecture and training scheme in this group stems from the 1-NN rule, *functionally* the resultant classifiers are GNPCs.

4. Experimental results

To illustrate the performance of the GNPC we use two data sets

- The *IRIS data*, consisting of 150 vectors in $\Re^4$ coming from three classes represented by 50 points each [22];
- The *2-spirals data*, consisting of 192 points in the plane, placed as two intertwined spirals that form two classes. The points along each spiral belong to one of two classes, shown as circles and pluses in Figure 5.

Both data sets are used extensively as benchmarks and are widely available. We use them to study the behavior of some GNPCs with respect to the number of prototypes p. The test set that is provided for the two-spirals data is not much different from the training set, which precludes a good generalization study. Our

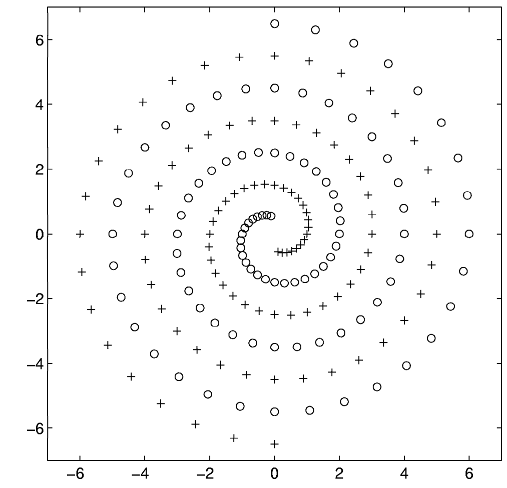


Figure 5: The two-spirals data

primary aim was to rate the classifiers within the GNPC framework in terms of how effectively they find and use prototypes. Therefore, in the experiments below we do not split the data into training and test sets, so the error rate shown is *resubstitution.* We implemented the following GNPCs in Matlab code. Whenever a norm was required for *any* model we used the Euclidean norm.

- Unsupervised prototype design
  - *Hard c-means clustering* (HCM) [22] followed by relabeling as at (8) (**F1**). HCM was run 10 times for each $p, p = 3, \ldots, 9$ with IRIS data and $p = 5, \ldots, 60$ with 2-spirals data, starting from different initializations, and the smallest error rate was stored as the result with the current p;
  - *Fuzzy c-means clustering* (FCM) [2] followed by relabeling (**F1**). The fuzzifying parameter m was set to 2 in all the experiments. As above, FCM was run 10 times for each value of p and the result reported is the lowest error rate found. The relabeling of the clusters was done in two steps. First, every $\mathbf{x}_j \in X$ was assigned to a crisp cluster corresponding to the highest membership value produced by FCM for that $\mathbf{x}_j$. Second, the crisp label for each prototype was then computed with (8);
- Supervised prototype design
  - *RBF network* (**F3**) [33]. We used HCM clustering to find V. The label matrix $\mathbf{L}_V$ was calculated by the nonnegative least-squares method. The smoothing parameter h_i was fixed at 0.3 in all experiments (for the two data sets the value 0.3 produced better classification results than the nearest-neighbor heuristic suggested in [33]). The smallest error from 10 runs is reported below.

- *LVQ1* [28] (**F4**). Kohonen's original algorithm was implemented by iterating through (10) and (11). In our experiments, a was a function only of the iterate number t, $a(t) = \eta a(t-1)$. We used $a_0 = 0.3$ and $\eta = 0.8$, as recommended in [17]. Each run of LVQ1 consisted of 40 passes through all of X. Before each pass X was randomly permuted. As with the above two procedures, we ran LVQ1 10 times and stored the lowest error rate.
- *Decision surface mapping (DSM)* (**F4**). This is an LVQ-type procedure that sometimes achieves lower resubstitution error rates than LVQ1 [17] . It differs from LVQ1 in that equations (10) and (11) are applied only if the label of the closest prototype does not match the label of $\mathbf{x}_j$. The closest ("wrong") prototype is updated by (11), and the closest "correct" prototype for $\mathbf{x}_j$ is found and updated by (10). All other parameters are the same as with LVQ1.
- *Edited 1-NN* (**F5**). We applied a simple random search algorithm (Figure 4) to select p elements of X as the prototypes V. The only parameter of the algorithm, the maximal number of steps T, was set to 10000 with the IRIS data and to 2000 with the 2-spirals data.

We did not implement a GNPC from **F2** because it is mainly of theoretical importance. Practically, this group is inferior to most other classifiers. Moreover, methods for reducing the number of kernels make the **F2** family virtually equivalent to **F3** (RBF networks).

Table 2 shows the number of misclassifications with $p = 3$ prototypes with the IRIS data set. We also tabulate some results published by other authors on the same data set (each source is specified). The acronyms in the table stand for: GLVQ-F = a fuzzy version of generalized LVQ; FALVQ = fuzzy algorithms for LVQ; LP = loose-pattern clustering approach; DR = "dog-rabbit" clustering technique. In Table 2 the column labeled S/U indicates the way prototypes are found: S = supervised, U = unsupervised, U+S = hybrid. The point of Table 2 is not to identify a "most superior" technique. Rather, we show these results to emphasize how broad the GNPC framework is.

Table 3 shows the number of misclassifications for the IRIS data with $p = 3, \ldots, 9$. Again, results with our implementations of GNPCs are displayed together with results presented by other authors. The acronym RBF (GA) denotes an RBF network where the centers (prototypes) are selected by a genetic algorithm.
We separate the unsupervised (the upper part) from the supervised (the lower part) GNPC designs. Dashes mean that information is not available.

As expected, the resubstitution error rate decreases as p increases. The results in both tables favor supervised selection of prototypes. The results with the random search editing are surprisingly good.

The structure of the data in Figure 5 suggests that the 2-spirals problem is difficult for classifiers that use *points* in $\Re^2$ as prototypes. This poses an interesting test for the GNPCs. The results with five GNPCs are plotted in Figure 6 with

GNPC	Family	S/U	Errors
"Boolean neural network" [16]	**F6**	S	30
LVQ + Relabeling (R) [4]	**F4**	U	17
HCM + R	**F1**	U	16
FCM + R	**F1**	U	16
GLVQ-F + R [24]	**F4**	U	16
FALVQ + R [25]	**F4**	U	16
FCM + R (LP) [18]	**F1**	U+S	15
RBF	**F3**	U	15
1-NP	**F1**	S	11
HCM + R (LP) [18]	**F1**	U+S	10
DR + R [4]	**F1**	U	10
DSM	**F4**	S	9
LVQ1	**F4**	S	6
Edited 1-NN (Random search)	**F5**	**S**	**2**

Table 2: Misclassifications with $p = 3$ prototypes for the IRIS data.

respect to number of prototypes p ($5 \leq p \leq 60$).

Again, the GNPCs that use supervised techniques to find V are superior to unsupervised designs. As in the previous example, RS editing is one of the simplest and most effective methods. In our view the **F5** family of GNPCs (edited 1-NN) should be considered as one of the first options for classifier design. This is impeded by the lack of well developed software, which in turn is a result of the fact that editing techniques have not received the attention that they probably deserve.

5. Conclusions

We proposed an integrated framework for the generalized nearest prototype classifier (GNPC). Five large families of classifiers are shown to fit within the GNPC framework. The five families differ most importantly in the way prototypes are *obtained* and not in their formal GNPC representation. Our common model provides links between many very disparate classification paradigms, enabling them to borrow ideas, algorithms or heuristics from each other. For example, the techniques for finding prototypes V from **F3** can be used to reduce the prototype number in, say, **F2** or **F5**. On the other hand, soft relabeling used in **F3** can be applied to **F1**. By picking different combinations of $s, \mathcal{T}$, and $\mathcal{S}$ we can design many different GNPCs, some of which may be better alternatives to the models used so far.

With respect to the classifiers studied here we found that

- Supervised techniques for finding V lead to better accuracy than unsupervised ones.
- The edited 1-NN rule is a viable option for GNPC design and needs further investigation.

$p \rightarrow$	3	4	5	6	7	8	9
Unsupervised							
HCM + R (**F1**)	16	16	10	14	11	4	3
FCM + R (**F1**)	16	22	14	14	4	4	4
LVQ + R [4] (**F4**)	17	24	14	14	3	4	4
GLVQ-F + R [4] (**F4**)	16	20	19	14	5	3	4
DR + R [4] (**F1**)	10	13	3	3	3	4	3
RBF (**F3**)	15	13	11	9	6	4	4
Supervised							
LVQ1 (**F4**)	6	4	3	3	2	2	2
DSM (**F4**)	9	3	3	3	2	4	2
RBF (GA) [39] (**F3**)	-	16	-	5	-	3	-
Edited 1-NN (RS) (F5)	2	2	2	2	1	1	1

Table 3: Minimal number of resubstitution errors with the IRIS data

Considering questions **Q1-Q5**, the following comments can be made:

A1. Often $p = c$ (in most of the **F1** techniques) or $p = n$ (**F2, F3, F5**). Alternatively, p can be found empirically or systematically and fixed before training the GNPC. Another choice is to include it as a trained parameter. Algorithms that tune p are often referred to as "adaptive".

A2. The answer to **Q2** determines the vast diversity of GNPCs. Based on our results we recommend supervised techniques for finding V.

A3. The labeling of the prototypes (**Q3**) seems to play crucial role. It is desirable to encode as much information as possible in $\mathbf{L}_V$. Therefore, soft labeling seems more promising than crisp labeling. The techniques that find a good $\mathbf{L}_V$ are still in the developmental stage.

A4. The similarity measure s (**Q4**) usually gets less attention in GNPC design. This is because many choices of s turn out to be equally useful (e.g., types of the radial basis functions in **F3** [10], distances in **F5** or **F1**, etc.)

A5. The only aggregation operators (**Q5**) that were used here to describe all five families of classifiers were $\mathcal{S} \in \{\mathbf{max}, \mathbf{average}\}$ and $\mathcal{T} = \mathbf{product}$. We can build a great variety of other GNPCs using the rich palette of t-norms and aggregation operators defined for fuzzy sets.

6. Acknowledgement

This research was supported by the NRC COBASE program and ONR Grant # N 00014-96-1-0642.

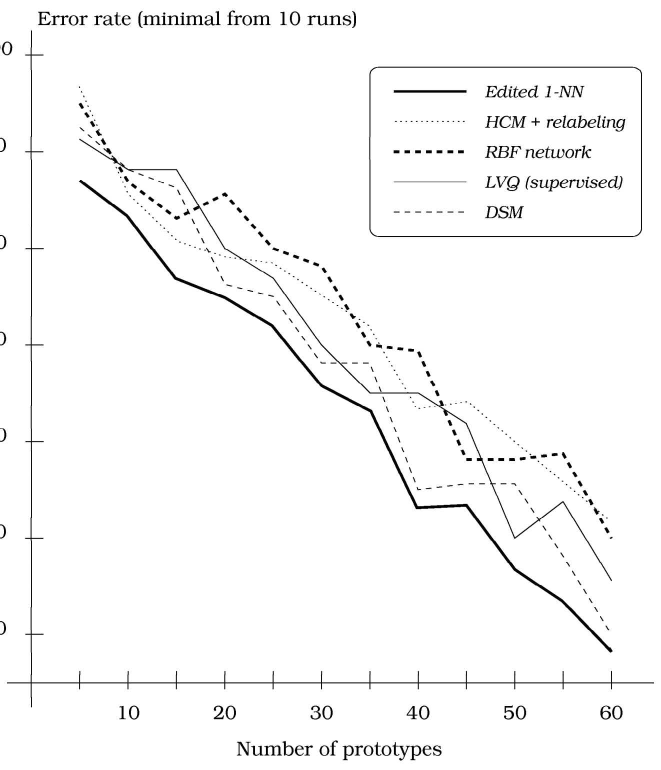


Figure 6: Number of misclassifications with the 2-spirals data

7. References

1. J.C. Bezdek, J.A. Keller, R. Krishnapuram and N.R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer Academic Publishers, 1999 (in press).
2. J.C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms.* NY: Plenum, 1981.
3. J.C. Bezdek, S.K. Chuah, and D. Leep, "Generalized k-nearest neighbor rules," *Fuzzy Sets and Systems*, vol. 18, pp. 237-256, 1985.
4. J.C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, "Classification with multiple prototypes", *Proc. Fifth International Conference on Fuzzy Systems, FUZZ/IEEE* New Orleans, pp. 626-632, 1996.
5. J.C. Bezdek, C. Coray, R. Gunderson, and J. Watson, "Detection and characterization of cluster substructure: I. Linear structure: Fuzzy C-lines," *SIAM J. Appl. Math.*, vol. 40, no. 2, pp. 339-357, 1981.
6. C. Bishop, *Neural Networks for Pattern Recognition.* Oxford: Clarendon Press, 1995.
7. J.L Blue, G.T. Candela, P.J. Grother, R. Chellappa, and C.L. Willson, "Evaluation of pattern classifiers for fingerprint and OCR applications," *Pattern Recognition*, vol. 27, pp. 485-501, 1994.
8. N.K. Bose and A.K. Garga, "Neural network design using Voronoi diagrams", *IEEE Transactions on Neural Networks*, vol. 4, pp. 778-787, 1993.

9. C.L. Chang, "Finding prototypes for nearest neighbor classifiers", *IEEE Transactions on Computers*, vol 23, No 11, pp. 1179-1184, 1974.
10. S. Chen, C.F.N. Cowan, and P.M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302-309, 1991.
11. R.L. Coultrip and R.H. Granger, "Sparse random networks with LTP learning rules approximate Bayes classifiers via Parzen's method," *Neural Networks* vol. 7, pp. 463-476, 1994.
12. B.V. Dasarathy *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* Los Alamitos, California: IEEE Computer Society Press, 1990.
13. C. Decaestecker, "NNP: A neural net classifier using prototypes," in *Proc. IEEE International Conference on Neural Networks,* San Francisco, CA, pp. 822o-824, 1993.
14. R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* NY: John Wiley & Sons, 1973.
15. K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, Inc., 1972.
16. S. Gazula and M.R. Kabuka, "Design of supervised classifiers using Boolean neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no.12, pp. 1239-1246, 1995.
17. S. Geva and J. Sitte, "Adaptive nearest neighbor pattern classification," *IEEE Transactions on Neural Networks*, vol. 2, no 2, pp. 318-322, 1991.
18. T. Gu and B. Dubuisson, "A loose-pattern process approach to clustering fuzzy data sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, pp. 366-372, 1985.
19. R.J. Hathaway and J.C. Bezdek, "Optimization of clustering criteria by reformulation," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 2, 241-245, 1995.
20. S. Haykin, *Neural Networks. A Comprehensive Foundation.* NY: Macmillan College Publishing Company, 1994.
21. L. Holmström, P. Koistinen, J. Laaksonen, and E. Oja, "Neural and statistical classifiers - taxonomy and two case-studies," *IEEE Transactions on Neural Networks*, vol. 8, pp. 5-17, 1997.
22. R.A. Johnson and D.W. Wichern, *Applied Multivariate Statistical Analysis*, 3-d edition, NJ: Prentice Hall, Englewood Cliffs, 1992.
23. R.P. Lippmann, "Pattern classification using neural networks," *IEEE Communication Magazine*, pp. 47-64, 1989.
24. N.B. Karayiannis and P.-I. Pai, "Fuzzy algorithms for learning vector quantization," *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1196-1211, 1996.
25. N.B. Karayiannis, J.C. Bezdek, N.R. Pal, R.J. Hathaway, and P.-I. Pai, "Repairs to GLVQ: A new family of competitive learning schemes," *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1062-1071, 1996.
26. T. Kaylani and S. Dasgupta. "Weight initialization of MLP classifiers using boundary preserving patterns," in *Proc. IEEE International Conference on Neural Networks, Orlando, Florida*, pp. 113-118, 1994.
27. J.M. Keller, M.R. Gray, and J.A. Givens, "A fuzzy k-nearest neighbors algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 4, pp. 580-585, 1985.
28. T. Kohonen, "Improved versions of learning vector quantization," in *Proc. Int. Joint Conference on Neural networks*, San Diego, CA, pp. I-545-550, 1990.
29. K. Koutroumbas and N. Kalouptsidis. "Nearest Neighbor pattern classification neural networks," in *Proc. IEEE International Conference on Neural Networks, Orlando, Florida*, pp. 2911-2915, 1994.

30. R. Krishnapuram, H. Frigui,and O. Nasraoui, "Fuzzy and possibilistic shell clustering algorithms and their application to boundary detection and surface approximation - Part 1," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 1, pp. 29-43, 1995.
31. L. Kuncheva, "Editing for the k-nearest neighbors rule by a genetic algorithm," *Pattern Recognition Letters*, vol. 16, pp. 809-814, 1995.
32. L. Kuncheva, "Initializing of an RBF network by a genetic algorithm," *Neurocomputing*, vol. 14, pp. 273-288, 1997.
33. J. Moody and C.J. Darken, "Fast learning in networks of locally tuned processing units," *Neural Computation*, vol. 1, pp. 281-294, 1989.
34. O.J. Murphy, "Nearest neighbor pattern classification perceptrons," in *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1595-1598, 1990.
35. D.F. Specht, "Probabilistic neural nets," *Neural Networks*, vol. 3, no. 1, pp. 109-118, 1990.
36. G.D. Tattersall and K. Yi, "Packed hyper-ellipsoid classifiers," *Electronic Letters*, vol. 30, pp. 427-428, 1994.
37. H.G.C. Traven, "A neural network approach to statistical pattern classification by "semiparametric" estimation of probability density functions," *IEEE Transaction on Neural Networks*, vol.2, no. 3, pp. 366-377, 1991.
38. J.T. Tou and R.C. Gonzalez, *Pattern Recognition Principles.* Reading, MA: Addison-Wesley, 1974.
39. B.A. Whitehead and T.D. Coathe, "Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction," *IEEE Transactions on Neural Networks*, vol. 7, no. 4, pp. 869-880, 1996.
40. B.A. Whitehead, "Genetic evolution of radial basis function coverage using orthogonal niches," *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1525-1528, 1996.
41. R.R. Yager and D.P. Filev, *Essentials of Fuzzy Modeling and Control.* NY: John Wiley & Sons, 1994.
42. H. Yan, "Handwritten digit recognition using an optimized nearest neighbor classifier," *Pattern Recognition Letters*, vol. 15, pp. 207-211, 1994.
43. H. Yan, J. Mao, Y. Zhu, and B. Chen. "Magnetic resonance image segmentation using optimized nearest neighbor classifiers," in *Proc. ICIP-94, Austin, TX,* pp. 49-52, 1994.
44. M.-S. Yang and C.-T. Chen, "On strong consistency of the fuzzy generalized nearest neighbor rule," *Fuzzy Sets and Systems*, vol. 60, no. 3, pp. 273-281, 1993.
45. H.-C. Yau and M.T. Manry. "Iterative improvement of a nearest neighbor classifier," *Neural Networks*, vol. 4, pp. 517-424, 1991.
46. Q. Zhao and T. Higuchi, "Evolutionary learning of nearest neighbor MLP," *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 762-767, 1996.