Some Notes on Twenty One (21) Nearest Prototype Classifiers

James C. Bezdek[1*] and Ludmila I. Kuncheva[2]

[1]Computer Science Department, University of West Florida,
Pensacola, FL, 32514, USA
jbezdek@uwf.edu
[2]School of Informatics, University of Wales
LL57 1UT Bangor, UK
mas00a@bangor.ac.uk

Abstract. Comparisons made in two studies of 21 methods for finding prototypes upon which to base the nearest prototype classifier are discussed. The criteria used to compare the methods are by whether they: (i) select or extract point prototypes; (ii) employ pre- or post-supervision; and (iii) specify the number of prototypes a priori, or obtain this number "automatically". Numerical experiments with 5 data sets suggest that pre-supervised, extraction methods offer a better chance for success to the casual user than post-supervised, selection schemes. Our calculations also suggest that methods which find the "best" number of prototypes "automatically" are not superior to user specification of this parameter.

Keywords. Data condensation and editing, Nearest neighbor classifiers, Nearest prototype classifiers, Post-supervision, Pre-supervision

1 Introduction

The methods discussed begin with a crisply labeled set of training data $X_{tr}=\{\mathbf{x}_1,\ldots,\mathbf{x}_n\} \subset \Re^p$ that contains at least one point with class label i, $1 \bullet i \bullet c$. Let $\mathbf{x} \subset \Re^p$ be a vector that we wish to label as belonging to one of the c classes. The standard nearest prototype (1-np) classification rule assigns x to the class of the "most similar" prototyle in a set of labeled prototypes (or reference set), say $V=\{v_1,\ldots,v_{n_p}\}$. $| V | = n_p$ may be less than, equal to, or greater than c [1].

We use two performance measures to compare 1-np designs. $E_{np}(X_{tr};\mathbf{V})$ is the *resubstitution* (or training) error committed by the 1-np rule that uses **V** when applied to the training data; $E_{np}(X_{test};\mathbf{V})$ is the *generalization* (testing) error of the same classifier when applied to a test set $X_{test} \subset \Re^p$.

*Research supported by ONR grant 00014-96-1-0642.

Good prototypes for 1-np classification have two desirable properties: *minimal cardinality* (minimum n_p) and *maximum classification accuracy* (minimum $E_{np}(X_{test}; \mathbf{V})$). However, these two goals naturally conflict. Increasing n_p up to some experimentally determined upper limit (usually with $n_p > c$) almost always results in a decreasing trend in $E_{np}(X_{test}; \mathbf{V})$, and conversely. One goal of our research is to study this conflict - how to find the smallest set of prototypes that provide an acceptable generalization error.

There are four types of class labels - crisp, fuzzy, probabilistic and possibilistic and three sets of label vectors in $\Re^c$:

$$N_{pc} = \left\{\mathbf{y} \in \Re^c : y_i \in [0,\ 1]\ \ \forall\ \ i,\ y_i > 0\ \ \exists\ \ i\right\} = [0,1]^c - \{\mathbf{0}\} \text{ (possibilistic); } \quad (1)$$

$$N_{fc} = \left\{\mathbf{y} \in N_{pc} : \sum_{i=1}^{c} y_i = 1\right\} \qquad \text{(fuzzy or probabilistic); } \quad (2)$$

$$N_{hc} = \left\{\mathbf{y} \in N_{fc} : y_i \in \{0,1\} \forall\ i\right\} = \left\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_c\right\} \qquad \text{(crisp).} \quad (3)$$

For convenience we call all non-crisp labels *soft* labels. An example of soft labeling is diagnosing ischemic heart disease, where occlusion of the three main coronary arteries can be expressed by such a label, each entry being the degree of occlusion of a particular vessel.

A useful framework for most of the methods we discuss is the *generalized nearest prototype classifier* (GNPC). If $\mathbf{x}$ and the $\mathbf{v}_i$'s are represented by feature vectors in $\Re^p$, prototypical similarity is almost always based on some function of pairwise distances between $\mathbf{x}$ and the $\mathbf{v}_i$'s. Specifically, let $\mathbf{x} \in \Re^p$ be an input vector. The GNPC is defined by the 5-tuple [2, 3] :

1. A set of prototypes $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_{n_p}\} \subset \Re^p$; (GNPC1)
2. A $c \times n_p$ prototype label matrix $L(\mathbf{V}) = [l(\mathbf{v}_1), \ldots, l(\mathbf{v}_{n_p})] \in \Re^c \times \Re^{n_p}$; (GNPC2)
3. A similarity function $S(\mathbf{x}_k, \mathbf{v}_i) = \Theta(\|\mathbf{x}_k - \mathbf{v}_i\|)$ valued in $[0,1]$. ; (GNPC3)
4. A T-norm to fuse $\{(l_i(\mathbf{v}_j), S(\mathbf{x}, \mathbf{v}_j)) : 1 \le i \le c; 1 \le j \le n_p\}$. ; (GNPC4)
5. An aggregation operator A which, for class i, $i = 1$ to c, combines $\{T(l_i(\mathbf{v}_j), S(\mathbf{x}, \mathbf{v}_j)) : 1 \le j \le n_p\}$ as $l_i(\mathbf{x}) = A[\{T(l_i(\mathbf{v}_j), S(\mathbf{x}, \mathbf{v}_j)) : 1 \le j \le n_p\}]$, the i-th element of the overall soft label for $\mathbf{x}$. . (GNPC5)

Figure 1 shows some of the many groups of classifiers that belong to the GNPC family. Abbreviations in Figure 1: *hard c-means* (HCM), *nearest neighbor* (1-nn), *learning vector quantization* (LVQ) and *radial basis function* (RBF). We use other abbreviations, each of which is defined in the sequel. Many 1-np and other classifiers can be realized by different choices of the parameters in (GNPC1-GNPC5). When the prototypes have soft labels, each prototype may "vote" with varying assurance for all c classes. For example, if $\mathbf{v}_i$ has the soft label [0.2, 0.4, 0.7], it is a fairly typical example of class 3, but is also related (less strongly) to classes 1 and 2.

Among the many characteristics of prototype extraction methods for 1-np classifier design that can be discussed, we consider the following three to be most important:

(C1) *Selection* ($\mathbf{V}_s \subseteq X$) versus *replacement* ($\mathbf{V}_r \not\subset X$). When talking about prototypes in general, we use the symbol **V**. When emphasis is needed, we use subscripts ($\mathbf{V}_s$ for selection of *S-prototypes* from X, $\mathbf{V}_r$ for replacement of X by *R-prototypes*). Replacement seeks n_p points in $\Re^p$, so the search space is infinite. Selection is limited to searching in $X \subset \Re^p$, so solutions in this case can be sought by combinatorial optimization. When the prototypes are selected points from the training data, a 1-np classifier based on them is called a *nearest neighbor* (1-nn) rule. When *all* of the training data are used as prototypes, it is *the* 1-nn rule; and when multiple votes (say k of them) are allowed and aggregated, we have the well known k-nn rule classifier.

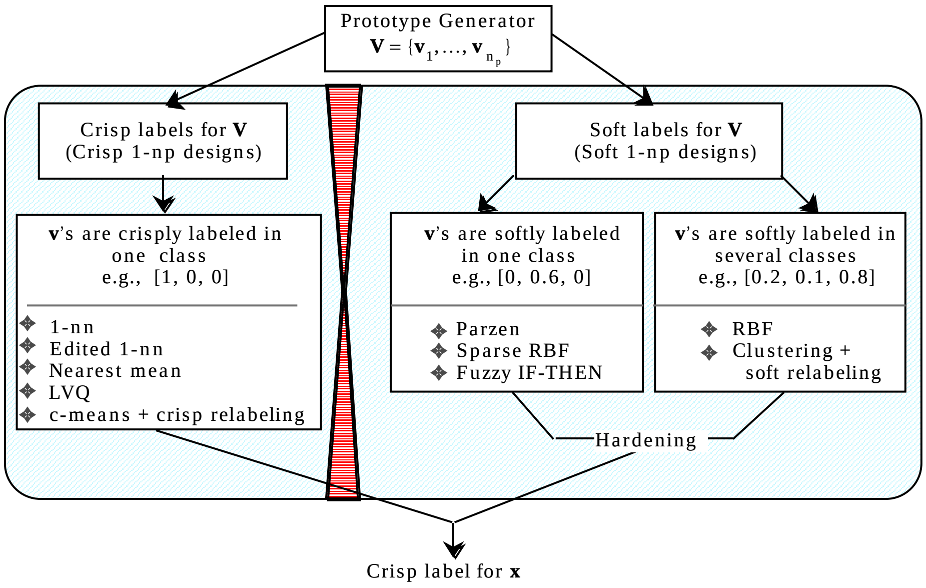

Fig. 1. A few models that are generalized nearest prototype classifiers

(C2) *Pre-supervised* versus *post-supervised* designs [4]. Pre-supervised methods use the data *and* the class labels to locate the prototypes. Post-supervised methods first find prototypes without regard to the training data labels, and then assign a class label to (*relabel*) each prototype. Selection methods are naturally pre-supervised, because each prototype *is* a data point and already has its (presumably true) label.

(C3) *User-defined* n_p versus *algorithmically defined* n_p. Most prototype generators require advance specification of n_p (e.g., classical clustering and competitive learning methods). Some models have "adaptive" variants where an initially specified n_p can increase or decrease, i.e., prototypes are added or deleted during training under the guidance of a mathematical criterion of prototype "quality". A third group of methods do not specify n_p at all, instead obtaining it as an output at the termination of training. For example, condensation methods which search for a minimal possible consistent set belong to this category. Genetic algorithms and tabu search methods have a trade-off parameter which pits the weight of a misclassification against an increase in the cardinality of **V** by 1. Thus, methods based on these types of search deliver the number of prototypes at termination of training. A method that finds or alters n_p during training will be called an *auto-*n_p method; otherwise, the method is *user-*n_p.

Table 1 lists 21 methods for prototype generation reviewed here. Eleven of the methods are discussed in [5] ; 16 of the methods are discussed in [6]; and six methods are discussed in both [5, 6]. A pertinent reference for each method is given, along with its classification by the criteria C1, C2 and C3 : selection = **(S)**, replacement = **(R)**, pre-supervised = **[PRE]**, post-supervised = **(post)**, auto-n_p = **(A)** and user-n_p = **(U)**. The notation **(A)** $\neg$ means that the algorithm can only *decrease* n_p.

Table 1. Twenty one (among zillions of!) methods for finding prototypes

Ref	Acronym	See	C1 S or R	C2 Pre/Post	C3 n_p
[5]	W+H	[5]	**(S)**	**[PRE]**	**(A)**
[9]	Tabu	[5]	**(S)**	**[PRE]**	**(A)**
[21]	LVQ1	[5]	**(R)**	**[PRE]**	**(U)**
[22]	DSM	[5]	**(R)**	**[PRE]**	**(U)**
[23]	LVQTC	[5]	**(R)**	**[PRE]**	**(A)** $\neg$
[3]	GA	[5, 6]	**(S)**	**[PRE]**	**(A)**
[14]	RND = RS	[5, 6]	**(S)**	**[PRE]**	**(U)**
[20]	BTS(3) = BS	[5, 6]	**(R)**	**[PRE]**	**(U)**
[25]	VQ	[5, 6]	**(R)**	**(post)**	**(U)**
[26]	GLVQ-F	[5, 6]	**(R)**	**(post)**	**(U)**
[30]	HCM	[5, 6]	**(R)**	**(post)**	**(A)** $\neg$
[18]	Chang	[6]	**(R)**	**[PRE]**	**(A)** $\neg$
[19]	MCA	[6]	**(R)**	**[PRE]**	**(A)** $\neg$
[10]	MCS	[6]	**(S)**	**[PRE]**	**(A)**
na	Sample Means	[6]	**(R)**	**[PRE]**	**(A)**
[27]	DR	[6]	**(R)**	**(post)**	**(U)**
[31]	MFCM(3)	[6]	**(R)**	**(post)**	**(U)**
[29]	FLVQ	[6]	**(R)**	**(post)**	**(U)**
[28]	SCS	[6]	**(R)**	**(post)**	**(U)**
[21]	SOM	[6]	**(R)**	**(post)**	**(U)**
[1]	FCM	[6]	**(R)**	**(post)**	**(U)**

Figure 2 depicts the 21 methods in Table 1 schematically. The Wilson/Hart (W+H) method is Wilson's method followed by Hart's *condensed nearest-neighbor* (C-nn), so it does not fit nicely into the tree in Figure 2. The three methods bracketed by < > are not used in our numerical experiments, but are included here for completeness.

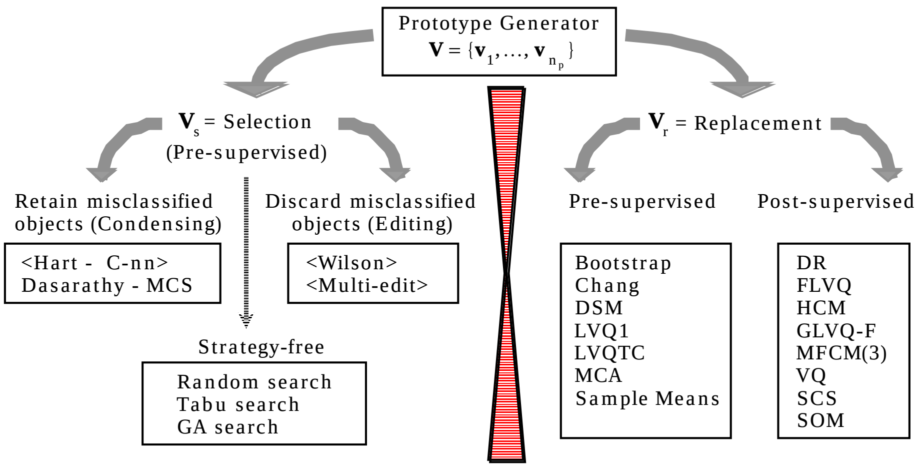

Fig. 2. Methods for finding prototypes

2 The 21 Methods

We cannot give useful descriptions of the 21 models and algorithms shown in Table 1 and Figure 2 here, so we briefly characterize the methods, and refer readers to [5, 6] and/or the original papers for more details. For convenience we drop the subscript of X_{tr}, and refer to the training data simply as X unless otherwise indicated.

Selection by condensation. Condensation seeks a *consistent* reference set $\mathbf{V}_s \wp$ X such that $E_{np}(X_{tr}; \mathbf{V}_s) = 0$. All condensation methods enforce the zero resubstitution error requirement, so trade-off between test error rate and the cardinality of $\mathbf{V}_s$ is impossible. The original condensation method is Hart's C-nn [7]. Many modifications of and algorithms similar to C-nn are known [8]. The output of C-nn depends on the order of presentation of the elements in X. Cerveron and Ferri [9] suggest running C-nn multiple times, beginning with different permutations of X, and terminating the current run as soon as it produces a set with the same cardinality as the smallest $\mathbf{V}_s$ found so far. This speeds the algorithm towards its destination and seems to produce good sets of consistent prototypes. A *minimal consistent set* algorithm (MCS) for condensation was proposed by Dasarathy [10]. Dasarathy's MCS decides which elements to retain after a pass through all of X, so unlike C-nn, MCS does not depend on the order in which the elements of X are processed. MCS, however, does not necessarily find $\mathbf{V}_s$ with the true *minimal* cardinality [3].

Selection by editing. Error-editing assumes that points from different classes that are close to decision boundaries should be discarded. Error-editing methods have no explicit connection to either the resubstitution or generalization error rate performance of the 1-np classifier based on the resultant $\mathbf{V}_s$. This group of methods include Wilson's method [11] and Multiedit [12]. Both schemes are based on deleting misclassified objects. In Wilson's method, the 3-nn algorithm is run once on X, and all misclassified objects are deleted from X after the run, leaving prototype set $\mathbf{V}_s$. Multiedit is asymptotically Bayes optimal, but is not suitable for small data sets with overlapping clusters, whereas Wilson's method works well in these cases. We have found that the methods of Hart, Wilson, and Randomized Hart are not very effective in terms of either accuracy or data set reduction. The W+H (Wilson + Hart) method introduced in [5] is just Wilson's method followed by Hart's C-nn .

Editing techniques are often followed by condensation. Editing "cleans up" the input data, yielding a $\mathbf{V}_{s,\text{initial}}$ that supposedly contains only "easy" points in it. Then a condensation method reduces $\mathbf{V}_{s,\text{initial}}$ to a possibly smaller number of relevant final prototypes, say $\mathbf{V}_{s,\text{final}}$. Ferri [13] proposes a third step : Multiedit is used for phase 1 "clean up" ; Hart's C-nn for phase 2 condensation; $\mathbf{V}_{s,\text{final}}$ is then used to reclassify all the original points in X, and the newly labeled data set, say X', is used with the *decision surface method* (DSM) to further refine the classification boundary.

Selection by search. A third group of methods for prototype selection attempt to find the smallest possible $\mathbf{V}_s$ with the highest possible 1-np accuracy through criterion-driven combinatorial optimization. These methods are *strategy free* in the sense that the decision to retain prototypes is based entirely on optimizing the criterion function. The basic combinatorial optimization problem to be solved is:

$$\underbrace{\max}_{\mathbf{V}_s \in P(X)} \{J(\mathbf{V}_s)\} = \underbrace{\max}_{\mathbf{V}_s \in P(X)} \left\{ \left(1 - E_{np}(X_{tr}; \mathbf{V}_s)\right) - \alpha \frac{|\mathbf{V}_s|}{|X_{tr}|} \right\} , \qquad (4)$$

where P(X) is the power set of X and α is a positive constant which determines the trade-off between accuracy and cardinality [3, 9] . We use three methods from this third group that all make use of (4) to evaluate potential solutions to the selection problem: random selection, GA-based search, and Tabu search.

For *random selection* (RS), the desired cardinality n_p of $\mathbf{V}_s$ and the number of trials T are specified in advance. Then T random subsets of X of cardinality n_p are generated, and the one with the smallest error rate is retained as $\mathbf{V}_s$. Skalak calls this method a Monte Carlo simulation [14]. Random search works unexpectedly well for moderate sized data sets [3], [14] .

Editing training data with GAs has been discussed by Chang and Lippmann [15]; Kuncheva, [16, 17]; and Kuncheva and Bezdek [3]. Our GA model is close to random selection, and our computational experience is that a few runs of this simple scheme can lead to a reasonably good solution. An even simpler evolutionary algorithm for data editing called "random mutation hill climbing" was proposed by Skalak [14]. Instead of evolving a population of chromosomes simultaneously, only one chromosome evolves (should we call it *survival of the only* ?), and only mutation is performed on it. The best set in T mutations is returned as $\mathbf{V}_s$. The evolutionary schemes in [3] and [14] are both heuristic. GA conducts a larger search by keeping different subsets of candidates in its early stages. On the other hand, the random mutation method is *really* simple, and, like the GA in [3], outperforms RS.

Tabu search is an interesting alternative to randomized methods [9]. In this scheme the number of iterations T is fixed but the cardinality n_p is not. Similar to random mutation hill climbing, TS operates on only the current solution S. A tabu vector of length $|X|$ is set up with all entries initially zero. An entry of 0 in the k-th place in the Tabu vector indicates that $\mathbf{x}_k$ can be added or deleted from S, while an entry greater than 0 prohibits a change in the status of $\mathbf{x}_k$. A parameter T_t called *tabu tenure* specifies the number of iterations before a change of any previously altered bit is allowed. An initial subset is picked as S, stored as the initial approximation of $\mathbf{V}_s$, and evaluated by J($\mathbf{V}_s$). All neighboring subsets to S are evaluated by J. The neighbor subset $\hat{S}$ that yields the highest value of J is called the *winning neighbor*. If $J(\mathbf{V}_{\hat{S}}) > J(\mathbf{V}_s)$, $\hat{S}$ replaces S, regardless of the tabu vector, and $\mathbf{V}_s$ and J($\mathbf{V}_s$) are updated. If $J(\mathbf{V}_{\hat{S}}) \leq J(\mathbf{V}_s)$, the tabu vector is checked. If the move from S to $\hat{S}$ is allowed, it is made anyway, and the corresponding slot of the tabu vector is set to T_t . Thus, tabu search does not necessarily have the ascent property. All other non-zero values in the tabu vector are then decreased by one. Different criteria can be applied for terminating the algorithm. Cerveron and Ferri's constructive initialization was used in [5], but we did not wait until a consistent set was obtained. Instead, the initial incremental phase was terminated at a prespecified number of iterations.

Pre-supervised replacement. The oldest method in this group (maybe 200 years old) replaces crisp subset X_i in X with its sample mean $\bar{\mathbf{v}}_i = \sum_{\mathbf{x} \in X_i} \mathbf{x} / n_i$, where $n_i = |X_i|$, i = 1,…,c. Chang [18] gave one of the earliest pre-supervised algorithms for extracting R-prototypes. Chang's algorithm features sequential updating based on a criterion that has a graph-theoretic flavor. Bezdek et al. [19] proposed a modification of Chang's algorithm that they called the *modified Chang algorithm* (MCA). Hamamoto et al. [20] gave a number of bootstrap methods for the generation of R-prototypes. The Hamamoto method we used (called BTS(3)), requires choosing three parameters: the number of nearest neighbors k, the desired number of prototypes n_p and the number of random trials T. A random sample of size n_p is drawn from X. Each data point is replaced by the mean of its k-nearest neighbors with *the same class label*. The 1-np classifier is run on X using the new labeled prototypes. The best set from T runs is returned as the final $\mathbf{V}_r$. In our experience, Hamamoto et al.'s method gives nice results. BTS(3) is a simple, fast, and unashamedly random way to get pre-supervised R-prototypes that often yield low 1-np error rates.

Another basic design that can be used for prototype generation is the LVQ1 algorithm [21]. An initial set of $n_p \bullet c$ labeled prototypes are randomly selected from X as initial prototypes so that each class is represented by at least one prototype. LVQ1 has three additional user-specified parameters: the *learning rate* α_k (0,1), a constant η (0,1) and the terminal number of iterations T. The standard competitive learning update equation is then used to alter the prototype set. Geva and Sitte's DSM [22] is a variant of LVQ1 which they assert better approximates classification boundaries of the training data than LVQ1 does. These authors say the price for better classification rates is that DSM is somewhat less stable than standard LVQ's. In LVQ1 the winning prototype is either punished or rewarded, depending on the outcome of the 1-np label match to the input. In DSM, when the 1-np rule produces the correct label, no update is made, but when misclassification occurs, the winner (from the wrong class) is punished, and the nearest prototype from the same class as the current input is identified and rewarded.

Both LVQ1 and DSM operate with a fixed number of prototypes chosen by the user, so are user-n_p methods. An auto-n_p modification of LVQ that can prune and relabel prototypes was proposed by Odorico [23], who called it LVQTC. In LVQTC the winning prototype is updated depending on the distance to input $\mathbf{x}_k$ and the history of the prototype. A prototype's historical importance is determined by the number of times it has been the winner, and the learning rate used for this prototype decreases as its hit rate increases. The rationale for this treatment of the learning rate is that prototypes which have been modified many times have already found a good place in the feature space and should be less affected by subsequent inputs. This strategy is very similar to one of the earliest competitive learning models, viz., *sequential hard c-means* [24]. Odorico may or may not have recognized this, but in any case adds some novel heuristics to the original algorithm which seem both justifiable and useful.

Post-supervised replacement. Methods in this category disregard class labels during training, and use X without its labels to find a set $\mathbf{V}_r$ of algorithmically labeled prototypes. The prototypes are then *relabeled* using the training data labels. To assign physical labels to the prototypes, the 1-np rule is applied to X using the extracted prototypes. The number of winners for each prototype from all c classes are counted. Finally, the most represented class label is assigned to each prototype. This relabeling strategy guarantees the smallest number of misclassifications of the resultant 1-np classifier on X, and is used in all of our post-supervised designs.

Vector quantization (VQ) is one of the standard sequential models that has been used for many years [25]. We adhered to the basic algorithm and applied it to each data set in the post-supervised mode. VQ starts with the user randomly selecting an initial set of n_p *unlabeled* prototypes from X. The closest prototype is always rewarded according to the update equation $\mathbf{v}_{i,\,new} = \mathbf{v}_{i,old} + \alpha_t(\mathbf{x}_k - \mathbf{v}_{i,old})$. The learning rate is indexed on t, the iteration counter (one iteration is one pass through X). We also used the closely related *self-organizing map* (SOM), which reduces to unsupervised VQ under circumstances laid out in [21]. Our runs with the SOM are discussed in more detail in [6], and some results using the SOM are traced out on Figure 5.

Generalized Learning Vector Quantization - Fuzzy (GLVQ-F) is an unsupervised sequential learning method for finding prototypes in which *all c* prototypes are updated after each input is processed. The update formula for the special case of weighting exponent m = 2 is [26]

$$\mathbf{v}_{i,new} = \mathbf{v}_{i,old} + u_i \alpha_t (\mathbf{x}_k - \mathbf{v}_{i,old}) \text{ , with } u_i = \sum_{j=1}^{c} \left(\frac{\|\mathbf{x} - \mathbf{v}_i\|^2}{\|\mathbf{x} - \mathbf{v}_j\|^2} \right), 1 \le i \le c. \quad (5)$$

The rest of the GLVQ-F algorithm is the same as VQ. Limit analysis in [26] shows that GLVQ-F reduces to VQ under certain conditions. Yet another sequential competitive learning model used in [6] is the deterministic "*dog-rabbit*" (DR) algorithm [27]. Like GLVQ-F, the DR algorithm may update all c prototypes for each input. Unlike GLVQ-F, the DR algorithm is not based on an optimization problem. Rather, its authors use intuitive arguments to establish the learning rate distribution that is used by the DR model.

The *soft competition scheme* (SCS) is a probabilistic sequential learning model that bears much similarity to algorithms for the estimation of the components of certain mixtures of normal distributions. Updates in SCS are made to all c prototypes, instead of just the winner [28]. The fuzzy learning vector quantization (FLVQ) model shares many of the same characteristics as SCS, and these two models are compared in [29].

Three unsupervised batch learning models were also used in [6]. If we disregard the labels of X, we can cluster it with *any* clustering algorithm that generates point prototypes, relabel the prototypes, and take them as $\mathbf{V}_r$. Good candidates include the various the c-means methods [1]. Our experiments in [5] used only classical hard c-means [30], and we plot a point on Figure 5 that came from the modified fuzzy c-means (MFCM-3) algorithm of Yen and Chang [31].

3 The Data Sets and Numerical Experiments in [5]

This sections summarizes the main points about 11 of the 21 methods (see Table 1), which used the four data sets shown in Table 2; see [5] for better descriptions of the methods, data and computational protocols.

Four figures in [5] plot $E_{np} \equiv E_{np}(X_{test}; \mathbf{V})$ versus $|\mathbf{V}| = n_p$ for the eleven methods and four data sets in Table 2. The closer a point (n_p, E_{np}) is to the origin, the better the 1-np classifier, because such a classifier will have a smaller number of prototypes and also a smaller test error rate than classifiers that plot further from the origin. For example, Figure 3 (Figure 5 in [5]) has coordinates for 10 of the 11 methods. The GA approach resulted in 77 prototypes for the cone-torus data, so we decided not to plot this point, to keep the scale so that the other 10 methods could be seen clearly. The same thing occurred with the other three data sets; the number of prototypes chosen by GA was much larger than those found by the other 10 methods, and this also occurred for the (W+H) classifier with two of the four data sets.

Table 2. Characteristics of the four data sets used in [5]

Name	p # of features	c # of classes	$\lvert X_{tr} \rvert$	$\lvert X_{test} \rvert$	Electronic Access
Cone-Torus	2	3	400	400	http://www.bangor.ac.uk/~mas00a/Z.txt
Normal Mixture	2	2	250	1000	http://www.stats.ox.ac.uk/~ripley/PRNN/
Phoneme	5	2	500	4904	ftp.dice.ucl.ac.be, directory pub/neural-nets/ELENA/
Satimage	36 (4 used)	6	500	5935	ftp.dice.ucl.ac.be, directory pub/neural-nets/ELENA/

LVQTC and BTS(3) are the *Pareto optimal* [PO, 32] designs in Figure 3, i.e., the ones that are better than all methods in some dimension, and not dominated by any other method in other dimensions. The tradeoff between minimal representation and maximal classification accuracy is evident from the fact that none of the classifiers studied in [5] had smallest coordinates in both dimensions.

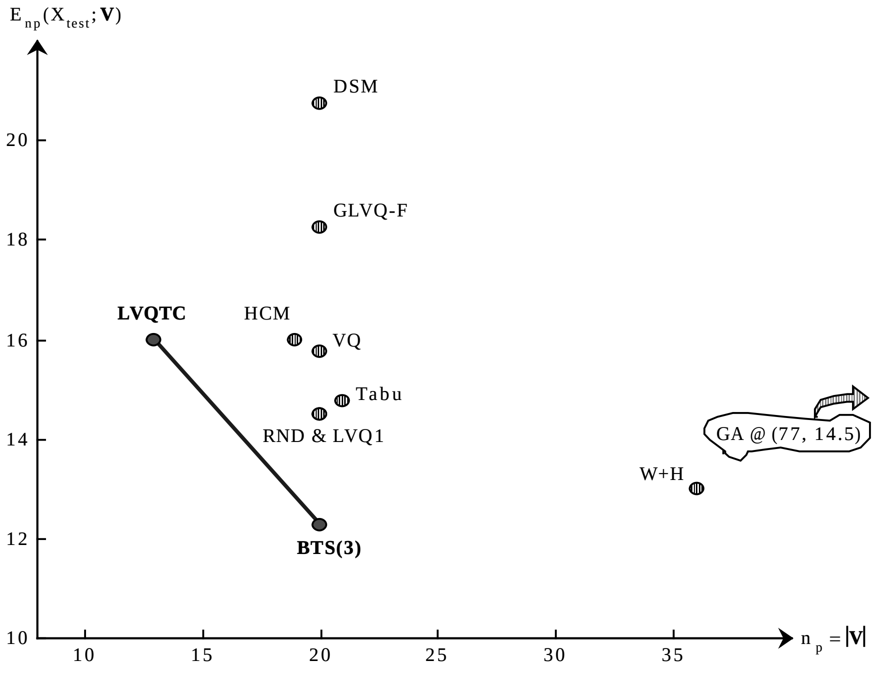

Fig. 3. $\boxed{n_p = |\mathbf{V}|}$ versus $\boxed{E_{np}(X_{test}; \mathbf{V})}$ for the Cone-Torus data [5]

Figure 4 addresses the relevance of (C1)-(C3) to 1-np classifier design by associating each of the 11 classifiers reviewed in [5] with one of the eight possible coordinate triples that are possible in (C1, C2, C3) space. The horizontal axis is the average *ratio* of n_p, the number of prototypes found or used, to the cardinality of the training set X_{tr}, $\bar{n}_p = \frac{1}{4}\sum_{i=1}^{4}\left(|\mathbf{V}_i| / |X_{tr,i}|\right)$. The vertical axis is the average training error, $\bar{E}_{np} = \sum_{i=1}^{4} E_{np}(X_{tr,i};\mathbf{V}_i)/4$. The "best" designs are again closest to the origin, and the four Pareto-optimal designs for averages over the four data sets are captured by the shaded region in Figure 4. The coordinates of these four designs (LVQTC, Tabu, LVQ1, BTS(3)) show ratios of: 3:1 for replacement vs. selection, 4:0 for pre-supervised designs vs. post-supervised designs, and 2:2 for auto-n_p vs. user-n_p selection of the number of prototypes.. Thus, averaging over the four sets of data changes only one ratio: the 3:1 best case ratio changes to 4:0 in the comparison of pre- to post-supervised designs.

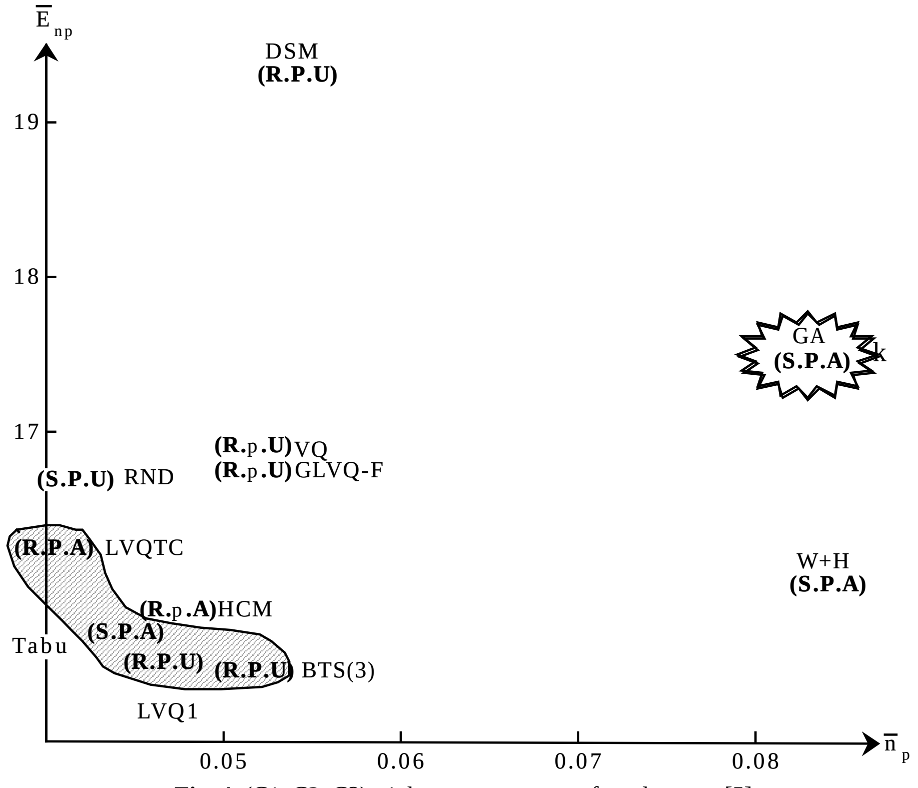

Fig. 4. (C1, C2, C3) triples : averages over four data sets [5]

4 The Data and Numerical Experiments in [6]

Resubstitution errors $E_{np}(X_{tr}; \mathbf{V})$ computed with 1-np classifiers built from runs using all 150 points in the Iris data [33] as X_{tr} are used to compare the subset of sixteen classifiers shown in Table 1 referenced as [6]. Using training error as the standard of comparison enables us to compare different *consistent* classifiers.

Figure 5 is typical of the results in [6]. We see from Figure 5 that four classifiers are consistent: MCA with 11 R- prototypes; GA with 12 S-prototypes; Chang with 14 R-prototypes; and MCS with 15 S-prototypes. There are two selection (GA, RS) and two replacement (MCA, BS) designs among the four consistent classifiers in Figure 5. There are three Pareto optimal points from four designs in Figure 5 : RS and GA (2 errors with 3 prototypes), BS (1 error with 5 prototypes), and MCA (no errors with 11 prototypes). We itemize the characteristics of the Pareto optimal designs in Table 3, along with their counterparts for the four data sets used in [5].

5 Conclusions

What can we conclude about the three characteristics of 1-np designs ?

(C1) *Selection* **(S)**versus *replacement* **(R)**;

(C2) *Pre-supervised* **[PRE]** versus *post-supervised* **(post)**; and

(C3) *User-*n_p **(U)** versus *auto-*n_p **(A)**.

Column 1 of Table 3 shows the winning 1-np classifier designs from four figures in [5] and one figure in [6] for the five data sets used in these two studies. Each row has a set of 3 check (🕑) marks corresponding to the three characteristics. Since there are four Pareto optimal designs for the Iris data, each pair of columns in Table 3 has a total of 12 checks. The bottom row sums the checks in each column, and each pair gives us a rough measure of the relative efficacy of 1-np designs for each of the pairs of characteristics comprising C1, C2 and C3.

So, the ratio for selection vs. replacement is 1:2; for pre- vs. post supervision is 5: 1; and for user vs. auto n_p, 1:1. This indicates that - at least for these data sets and trials - pre-supervised, replacement prototypes are the more desirable combination of (C1) and (C2), while finding the best *number* of prototypes is done equally well by user specification or "automatically". We conclude that:

1. Replacement prototypes seem to produce better 1-np designs than points selected from the training data.
2. Pre-supervision seems - overwhelmingly - to find more useful prototypes for 1-np classifiers than post-supervision does.

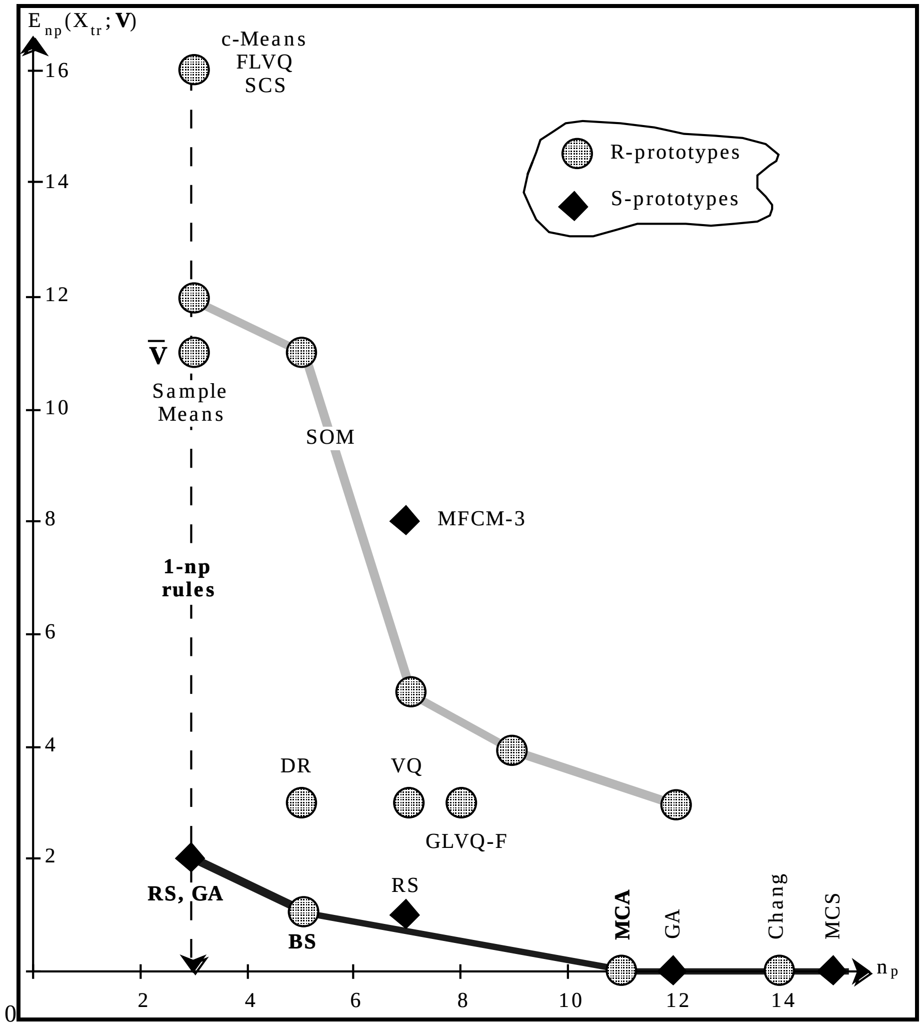

Fig. 5. Best case R- and S-Prototype classifiers for 16 methods on Iris [6]

3. Methods that "automatically" determine the best number of prototypes and methods that are largely based on user specification and trials-and-error are equally likely to yield good 1-np classifiers.
4. There is a clear tradeoff between minimal n_p and minimal error rate, and this is a data dependent issue.
5. 1-np classifiers that use $n_p > c$, with c the indicated number classes in the training data, will almost always produce lower error rates than 1-np designs that use one prototype per class.

Table 3. A summary of the Pareto optimal (PO) designs for the five data sets

Method	$E_{np}(X_*;\mathbf{V})$	PO in [5]	**(S)**	**(R)**	**[PRE]**	**(post)**	**(U)**	**(A)**
LVQTC	X_{test}	Fig. 5		🕐	🕐			🕐
BTS(3)	X_{test}	Fig. 5		🕐	🕐		🕐	
LVQ1	X_{test}	Fig. 6		🕐	🕐		🕐	
RS	X_{test}	Fig. 6	🕐		🕐		🕐	
Tabu	X_{test}	Fig. 7	🕐		🕐			🕐
LVQTC	X_{test}	Fig. 7		🕐	🕐			🕐
HCM	X_{test}	Fig. 8		🕐		🕐		🕐
VQ	X_{test}	Fig. 8		🕐		🕐	🕐	
		PO in [6]						
RS	X_{train}	Fig. 6	🕐		🕐		🕐	
GA	X_{train}	Fig. 6	🕐		🕐			🕐
BTS(3)	X_{train}	Fig. 6		🕐	🕐		🕐	
MCA	X_{train}	Fig. 6		🕐	🕐			🕐
Σ(🕐)			4	8	10	2	6	6

6 References

[1] J. C. Bezdek, J. Keller, R. Krishnapuram and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer, Norwell, MA, 1999.

[2] L. I. Kuncheva and J.C. Bezdek, An integrated framework for generalized nearest prototype classifier design, *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 6 (5), 1998, 437-457.

[3] L.I. Kuncheva and J.C. Bezdek, On prototype selection: Genetic algorithms or random search?, *IEEE Trans. on Systems, Man, and Cybernetics*, C28 (1), 1998, 160-164.

[4] L.I. Kuncheva and J.C. Bezdek, Pre-supervised and post-supervised prototype classifier design, *IEEE Trans. on Neural Networks*, 10(5), 1999, 1142-1152.

[5] J. C. Bezdek and L. I. Kuncheva, Nearest prototype classifier designs: An experimental study, in review, *IEEE Trans. on Fuzzy Systems*, 2000.

[6] J. C. Bezdek and L. I. Kuncheva, Point prototype generation and classifier design, in *Kohonen Maps*, eds. E. Oja and S. Kaski, Elsevier, Amsterdam, 1999, 71-96.

[7] P. E. Hart, The condensed nearest neighbor rule, *IEEE Trans. on Information Theory*, IT-14, 1968, 515-516.

[8] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, California: IEEE Computer Society Press, 1991.

[9] V. Cerveron and F. J. Ferri, Another move towards the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule, in review, *IEEE Trans. SMC*, 2000.

[10] B.V. Dasarathy, Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design, *IEEE Trans. on Systems, Man, and Cybernetics*, 24, 1994, 511-517.

[11] D. L. Wilson, Asymptotic properties of nearest neighbor rules using edited data, *IEEE Trans. on Systems Man and Cybernetics*, SMC-2, 1972, 408-421.

[12] P.A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

[13] F. J. Ferri, Combining adaptive vector quantization and prototype selection techniques to improve nearest neighbor classifiers, *Kybernetika*, 34 (4), 1998, 405-410.

[14] D. B. Skalak, Prototype and feature selection by sampling and random mutation hill climbimg algorithms, *Proc. 11th International Conference on Machine Learning, New Brunswick, N.J.*, Morgan Kaufmann, Los Alamitos, CA, 1994, 293-301.

[15] E. I. Chang and R.P. Lippmann, Using genetic algorithms to improve pattern classification performance, in *Advances in Neural Information Processing Systems*, **3**, R.P. Lippmann, J.E. Moody and D.S. Touretzky, Eds., San Mateo, CA: Morgan Kaufmann, 1991, 797-803.

[16] L. I. Kuncheva, Editing for the k-nearest neighbors rule by a genetic algorithm, *Pattern Recognition Letters, Special Issue on Genetic Algorithms*, 16, 1995, 809-814.

[17] L. I. Kuncheva,Fitness functions in editing k-NN reference set by genetic algorithms, *Pattern Recognition*, 30, 1997, 1041-1049.

[18] C. L. Chang Finding prototypes for nearest neighbor classification, *IEEE Trans. Computer*, 23(11), 1974, 1179-1184.

[19] J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, Multiple prototype classifier design, *IEEE Trans. on Systems, Man, and Cybernetics*, C28 (1), 1998, 67-79.

[20] Y. Hamamoto, S. Uchimura and S. Tomita, A bootstrap technique for nearest neighbor classifierdesign, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19 (1), 1997, 73-79.

[21] T. Kohonen, *Self-Organizing Maps*, Springer, Germany, 1995.

[22] S. Geva and J. Sitte, Adaptive nearest neighbor pattern classifier, *IEEE Trans. on Neural Networks*, 2 (2), 1991, 318-322.

[23] R. Odorico, Learning vector quantization with training counters (LVQTC), *Neural Networks*, 10 (6), 1997, 1083-1088.

[24] J. MacQueen, Some methods for classification and analysis of multivariate observations,*Proc. Berkeley Symp. Math. Stat. and Prob.*, 1, eds. L. M. LeCam and J. Neyman, Univ. of California Press, Berkeley, 1967, 281-297.

[25] A. Gersho and R. Gray,*Vector Quantization and Signal Compression*, Kluwer, Boston, 1992.

[26] 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 Trans. on Neural Networks*, 7, 1996, 1062-1071.

[27] P. McKenzie, P. and M. Alder, Initializing the EM algorithm for use in Gaussian mixture modeling, in *Pattern Recognition in Practice IV ; Multiple Paradigms, Comparative Studies and Hybrid Systems*, eds. E.S. Gelsema and L. N. Kanal, Elsevier, NY, 1994, 91-105.

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

[29] J. C. Bezdek and N. R. Pal, Two soft relatives of learning vector quantization, *Neural Networks*, 8(5), 1995, 729-743.

[30] R. O. Duda and P.E. Hart,*Pattern Classification and Scene Analysis*, John Wiley & Sons, N.Y., 1973.

[31] J. Yen and C.W. Chang, A multi-prototype fuzzy c-means algorithm, *Proc. European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany, 1994, 539-543.
[32] W. L Winston, *Operations Research, Applications and Algorithms*, 3rd ed., Duxbury Press, Belmont, CA. , 1994.
[33] J. C. Bezdek, J. M. Keller, R. Krishnapuram, L. I. Kuncheva and N. R. Pal Will the real Iris data please stand up?, *IEEE Trans. Fuzzy Systems*, 7(3), 1999, 368-369.