and how this differs from identifying barriers, the use of leverage points in structuring problem-solving activities, and the judgment of solvability and how this relates to the detection of leverage points.

## VII. Conclusions

We have presented the concept of leverage points as a means by which new courses of action are generated in a situation requiring problem solving. The use of leverage points is a constructive strategy in which experience is needed to identify opportunities that can be deepened and elaborated. This view contrasts with computational approaches in which mechanical strategies are used to generate a problem space; algorithmic and heuristic methods are then applied to search through the problem space.

## Acknowledgment

The authors would like to thank R. Lipshitz, P. Smith, and G. Smith for their helpful comments on earlier drafts. M. Thordsen, H. Klein, D. Avison, R. Pliske, J. Flach, and T. Miller all made useful criticisms and suggestions.

## References

[1] K. Duncker, "On problem solving," *Psychol. Monographs,* vol. 58, ch. 5, p. 270, 1945.
[2] W. Köhler, *Gestalt Psychology.* New York: Liveright, 1929.
[3] A. S. Luchins, "Mechanization in problem solving: The effect of Einstelung," *Psychol. Monographs,* vol. 54, p. 248, 1942.
[4] M. Wertheimer, *Productive Thinking.* New York: Harper & Row, 1959.
[5] J. J. Gibson, *The Ecological Approach to Visual Perception.* Boston, MA: Houghton Mifflin, 1979.
[6] P. Hancock, J. Flach, J. Caird, and K. Vicente, Eds., *Local Applications of the Ecological Approach to Human-Machine Systems,* vol. 2. Hillsdale, NJ: Lawrence Erlbaum, 1995.
[7] G. A. Klein and B. W. Crandall, "The role of mental simulation in naturalistic decision making," in *Local Applications of the Ecological Approach to Human-Machine System,* vol. 2, P. Hancock, J. Flach, J. Caird, and K. Vicente, Eds. Hillsdale, NJ: Lawrence Erlbaum, 1995, pp. 324–358.
[8] J. Dewey, *How We Think.* Boston, MA: Heath, 1910.
[9] R. Lipshitz and O. Bar-Ilan, *How Problems are Solved: Retrospective Reports of Successful and Unsuccessful Problem Solving in Organizational Settings.* Haifa, Israel: Univ. of Haifa Press, to be published.
[10] G. A. Klein and J. Weitzenfeld, "Improvement of skills for solving ill-defined problems," *Educ. Psych.,* vol. 13, pp. 31–41, 1979.
[11] G. F. Smith, "Defining managerial problems: A framework for prescriptive theorizing," *Manage. Sci.,* vol. 35, no. 8, pp. 963–981, 1989.
[12] K. E. Weick, "Managerial thought in the context of action," in *The Executive Mind.* S. Srivasta, Ed. San Francisco, CA: Jossey-Bass, 1983, pp. 221–242.
[13] A. Newell and H. A. Simon, *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall, 1972.
[14] A. Newell, J. C. Shaw, and H. A. Simon, "Elements of a theory of human problem solving," *Psychol. Rev.,* vol. 65, pp. 151–166, 1958.
[15] A. Newell, *Unified Theories of Cognition.* Cambridge, MA: Harvard Univ. Press, 1990.
[16] J. R. Anderson, "Problem solving and learning," *Amer. Psych.,* vol. 48, pp. 35–44, 1993.
[17] D. Klahr and K. Dunbar, "Dual space search during scientific reasoning," *Cogn. Sci.,* vol. 12, pp. 1–48, 1988.
[18] J. Sweller, "Cognitive load during problem solving: Effects on learning," *Cogn. Sci.,* vol. 12, pp. 257–285, 1988.
[19] J. F. Voss, T. R. Greene, T. A. Post, and B. C. Penner, "Problem solving skill in the social sciences," in *The Psychology of Learning and Motivation: Advances in Research Theory,* vol. 17, G. H. Bower, Ed. New York: Academic, 1983.
[20] W. G. Chase and H. A. Simon, "The mind's eye in chess," in *Visual Information Processing,* W. G. Chase, Ed. New York: Academic, 1973.
[21] D. H. Holding, *The Psychology of Chess Skill.* Hillsdale, NJ: Lawrence Erlbaum, 1985.
[22] L. R. Keller and J. L. Ho, "Decision problem structuring: Generating options," *IEEE Trans. Syst., Man, Cybern.,* vol. 18, Sept. 1988.
[23] R. Sternberg, *Wisdom: Its Nature, Origins, and Development.* Cambridge, MA: Cambridge Univ. Press, 1990.
[24] A. VanGundy, *Techniques of Structured Problems Solving,* 2nd ed. New York: Reinhold, 1988.
[25] R. A. Brooks, "New approaches to robotics," *Science,* vol. 253, pp. 1227–1232.

# Nearest Prototype Classification: Clustering, Genetic Algorithms, or Random Search?

Ludmila I. Kuncheva and James C. Bezdek

***Abstract*— Three questions related to the *nearest prototype classifier* (NPC) are addressed: 1) when is it better to construct the prototypes instead of selecting them as a subset of the given labeled data, 2) how can we trade classification accuracy for a reduction in the number of prototypes, and 3) how good is pure *random search* (RS) for selection of prototypes from the data? We compare the resubstitution performance of the NPC on the IRIS data set, where the prototypes are either extracted by "replacement" ($R$-prototypes) or by "selection" ($S$-prototypes). Results for the $R$-prototypes are taken from a previous study and are contrasted with $S$-prototype results obtained by a *genetic algorithm* (GA) or by RS. The best results reached by both algorithms (GA and RS), followed by resubstitution NPC, are two errors with sets of three $S$-prototypes. This compares favorably to the best result found with $R$-prototypes, viz., three errors with five $R$-prototypes. Based on our results, we recommend GA selection for the NPC. A by-product of this research is a counter example to minimality of a recently published "minimal consistent set selection" procedure.**

***Index Terms*— Classifier design, genetic algorithms (GA's), nearest prototypes, random search (RS)**

## I. Introduction

The *Nearest Prototype Classifier* (NPC) is perhaps the simplest classifier in pattern recognition. Let the integer $\hat{c}$ denote the number of classes $\hat{c} > 1$ in a labeled data set. Let $\mathbf{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_c\} \subset \Re^p$ be a set of $c \geq \hat{c}$ prototypes, with each $\mathbf{v}_i \in \mathbf{V}$ labeled as one of the $\hat{c}$ classes. The NP classifier assigns any unlabeled object $\mathbf{x} \in \Re^p$ to the class of its nearest prototype. This scheme uses at least one prototype per class. The NPC can be regarded as the *one-nearest-neighbor* (1-NN) rule where the reference set is $\mathbf{V}$. A subtle difference between the two techniques is that NPC assumes a much smaller number of prototypes than the 1-NN rule.

The problem here is to design a good prototype set that will ideally be of minimal cardinality and will allow for the lowest possible error rate of the NPC. There are two strategies that are illustrated in Fig. 1.

- *Selection*, Fig. 1(a). We retain a limited number of points from the original data set $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \Re^p$, while ruling

Manuscript received January 6, 1997; revised August 31, 1997. This work was supported by the NRC COBASE program and ONR Grant 00014-96-1-0642.

L. I. Kuncheva is with the School of Mathematics, University of Wales, LL57 1UT Bangor, U.K. (e-mail: mas00a@bangor.ac.uk).

J. C. Bezdek is with the Department of Computer Science, University of West Florida, Pensacola, FL 32514 USA (e-mail: jbezdek@ai.uwf.edu).

Publisher Item Identifier S 1094-6977(98)01536-3.

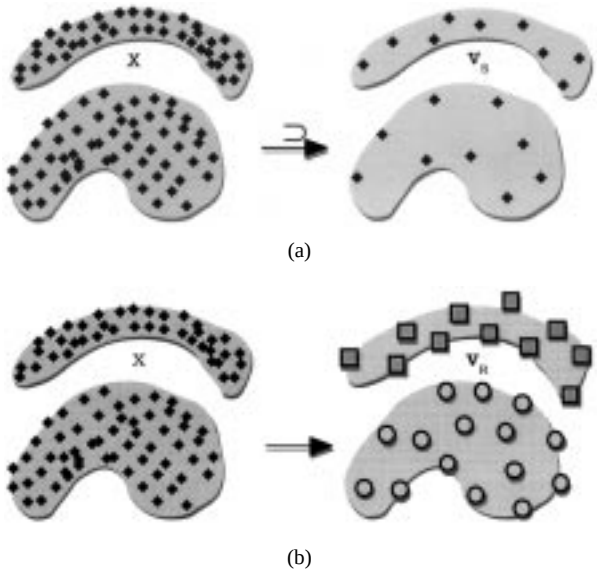


Fig. 1. (a) Editing by selection of $S$-protoypes in $\mathbf{X}$. (b) Replacement of $\mathbf{X}$ by $R$-prototypes.

out those that do not contribute significantly to the classification accuracy [6]. Editing techniques that find subsets guaranteeing zero errors when the original data set $\mathbf{X}$ is submitted to the NPC are called *condensation techniques*, and the resultant set is said to be *consistent* with $\mathbf{X}$.

- *Replacement*, Fig. 1(b). The original data set is replaced by a number of labeled prototypes that do not necessarily coincide with any points in $\mathbf{X}$. The simplest example of this is to replace each labeled subset by its sample mean vector.

To distinguish between the two types, prototypes obtained by selection will be referred to as $S$-prototypes, while those obtained by replacement will be called $R$-prototypes. When something applies to both $S$- and $R$-prototypes, the single word "prototypes" will be used.

In this paper we try to answer three questions.

1) Is it better to extract the prototype set or to select it? ($R$-prototypes versus $S$-prototypes.)
2) How can we trade classification accuracy against a reduction in the number of prototypes?
3) How good is pure *random search* (RS) for selection of $S$-prototypes from data?

It has been claimed recently that *genetic algorithms* (GA's) are a good editing technique for the selection of $S$-prototypes [4], [12], [13]. Contrary to expectations, it appears that RS competes surprisingly well when the number of $S$-prototypes to be kept is small. The GA and RS algorithms for $S$-prototype selection are described in Section II. The methods and results for $R$-prototypes taken from a previous study by Bezdek *et al.* [3] are briefly presented in Section III. We compare the resubstitution classification error of NPC on IRIS data with $R$- and $S$-prototypes in Section IV. Section V contains our conclusions and discussion.

## II. Techniques for Selecting $S$-Prototypes

Since we are tackling only the set $\mathbf{X}$ (without any split into training/test subsets), we confine the study to *condensation* techniques for editing the sample set. Condensation selects a subset of the original set that "preserves" the classification boundary, i.e., it guarantees zero resubstitution errors if used as the reference for the 1-NN rule. A collection of papers on this subject can be found in [6].

The aim is to find the consistent set with the smallest possible cardinality, called a *minimal consistent subset* (MCS). One of the earliest papers on this topic is by Hart [10], whose elegant method has been used as a basis for many subsequent modifications. Hart's iterative procedure selects elements sequentially and ends up with a relatively large consistent set. A recent study by Dasarathy [7] presented a technique for finding a consistent set that he believed to be minimal. On the IRIS data set, however, Dasarathy's technique finds a 15-element consistent subset, whereas our GA method resulted in a 12-element consistent set. Thus, we provide a counterexample to Dasarathy's conjecture that his MCS method is truly minimal.

Further, we wish to be able to select a small subset of $S$-prototypes that restricts the resubstitution error to a certain predefined upper limit. In the condensation techniques mentioned above, this task is not dealt with, and their modification for this problem is not at all straightforward. This difficulty motivated us to use GA's and even brute-force RS for solving this problem.

### A. GA

GA's are evolutionary optimization techniques with a wide scope of applications [8], [9]. They are basically a guided RS and are deemed to work well in some large search spaces.

The objective is to find a set of $S$-prototypes $S^*$ that satisfies

$$S^* = \arg \max_{S \subset \mathbf{X}} \mathcal{F}(S) \tag{1}$$

where $\mathcal{F}(S)$ is the *fitness function*. In our study, we chose $\mathcal{F}$ to comprise two terms

$$\mathcal{F}(S) = \hat{P}(S) - \alpha f(|S|). \tag{2}$$

The first term $\hat{P}(S)$ denotes the apparent classification accuracy of the NPC when using $S$ as the reference set, i.e., it is the ratio of correctly classified vectors from $\mathbf{X}$ to the overall number of vectors $n$, $n = |\mathbf{X}|$. The second (penalty) term is a function of the cardinality of $S$ weighted by the coefficient $\alpha > 0$. Generally, the higher the cardinality, the higher the penalty. In this study, we experimented with two penalty functions

$$f_1(|S|) = |S| \tag{3}$$

and

$$f_2(|S|) = (|S| - T)^2 \tag{4}$$

where $T$ is a positive integer. $f_2$ forces the GA to converge to a predefined number of prototypes, $|S^*| = T$. This was needed to enable comparisons with other techniques.

Selection of a subset of a given set is perhaps one of the problems that is most amenable to the GA approach because the natural encoding is to make every chromosome a binary string of length $n$. We represent $S \subset \mathbf{X}$ by a chromosome whose $i$th bit is one if $\mathbf{x}_i \in S$ and zero, otherwise. The population consists of a prespecified number of chromosomes (sets of $S$-prototypes), which are subsequently evolved while trying to maximize the fitness function (1) (for more detail, see [12] and [13]).

The version of the algorithm implemented here proceeds in the following steps.

1) *Initialization.* A set of $N_{pop}$ randomly generated chromosomes is the initial "population set" $\Pi = \{S_1, \cdots, S_{N_{pop}}\}$. Each bit in a chromosome takes the value one with a prespecified probability $P_{ini}$. By introducing this parameter, we can generate sparse or dense chromosomes for initializing the search. For example, when we wish to start a search with $S$-prototypes assuring very low error rates, we can fix $P_{ini} = 0.85$. Alternatively, we may initialize the chromosomes with

$P_{ini} = 0.1$, which means that the search will start from sets containing about 10% of the data points, so the initial error will probably be much higher than when $P_{ini} \approx 1$. GA parameters are initialized: maximal number of generations $M_{gen}$, mutation probability $P_m$, the weighting coefficient $\alpha$ for the penalty term of (1), and the specific form of $f$ in (2). The chromosomes in $\Pi$ are then evaluated by the fitness function.

2) *Forming of the mating set* $\mathbf{M}$. Classically, the mating set $\mathbf{M}$ is formed on a "roulette" principle: chromosomes are sampled from $\Pi$ with probability proportional to their fitness values. In the current implementation, $\mathbf{M}$ coincides with $\Pi$.
3) *Crossover*. Parent couples are randomly selected out of the elements of $\mathbf{M}$. Every couple produces two offspring chromosomes by exchanging parts of their codes. Here we adopted uniform crossover as recommended in [1] and [2]. The parent chromosomes swap their $i$th bits with a certain probability (we set it to 0.5 here), and $i$ goes from one to $n$. Each pair of parents produces two offspring chromosomes, thereby constituting the offspring set $\mathbf{O}$.
4) *Mutation*. Each bit of each offspring chromosome alternates (mutates) with a predefined probability (mutation rate $P_m$). All elements of $\mathbf{O}$ are then evaluated by the fitness function.
5) *Selection*. $\Pi$ and $\mathbf{O}$ are pooled and the best $N_{pop}$ individuals survive, i.e. they stand as the new $\Pi$ (elitist strategy).

Steps 2)–5) are executed $M_{gen}$ times.

### B. RS

We paraphrase an interesting situation described in Fogel [8]. An opponent of evolutionary techniques was cited, who claimed that evolutionary algorithms are almost as bad as pure RS, which is "... the most inefficient method for problem solving." Contrary to this view, our experiments show that for reasonably small tasks, RS outperforms all of its sophisticated competitors.

We implemented RS by suspending the evolutionary operators 2)–4) from the algorithm above. That is, we implemented only *elitist selection* by keeping the best scoring chromosome. Thus, we generate $N_{pop}$ $S$-prototype sets with a fixed number of elements (number of 1 s in the chromosome), evaluate the fitness function for them, and store the best solution encountered so far. No memory is kept from generation to generation, i.e., every set of $N_{pop}$-candidate solutions is drawn from the uniform random distribution. The algorithm ends up with a single best solution found out through $N_{pop}(M_{gen}+1)$ evaluations of the fitness function, the same number of total candidates that the GA method considers.

## III. Techniques for Designing $R$-Prototypes

Bezdek *et al.* [3] present seven methods for generating $R$-prototypes from labeled data, based on sequential competitive learning, sample means, modified fuzzy $c$-means, Chang's method [5], and an original modification of Chang's method. These methods are compared on the IRIS data set. The methods are listed and described in more detail below.

- Sample Means: This is perhaps the most inaccurate technique. It relies on the hypothesis that the classes are hyperspherical with equal volume and equal prior probabilities. The $R$-prototypes $\mathbf{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_{\hat{c}}\}$ are found as the sample means of every class.

The second group of methods presented in [3] is based on sequential competitive learning. $R$-prototypes are computed with sequential updating for every point in the data set. The new value of prototype $\mathbf{v}_i$ at step $t$ is denoted as $\mathbf{v}_{i,t}$ and is obtained as

$$\mathbf{v}_{i,t} = \mathbf{v}_{i,t-1} + a_{ik,t}(\mathbf{x}_k - \mathbf{v}_{i,t-1}), \qquad i = 1, \cdots, c \tag{5}$$

where $\mathbf{x}_k$ is the vector in $\mathbf{X}$ 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. The following three are studied in [3]; none uses the class labels of the data in the course of $P$-prototype tuning.

- LVQ: Only the winner (nearest $P$-prototype) is updated and $a$ varies only with $t$.
- GLVQ-F: This algorithm belongs to a family of fuzzy LVQ models. All $cR$-prototypes are updated at each step with $a$ being a function of $t$, $\mathbf{V_{t-1}}$ and $\mathbf{x}_k$.
- DR: This is the so-called dog-rabbit algorithm in which the learning rate distribution $a$ is determined heuristically at each step.

Since none of the three algorithms uses class labels during training, the $R$-prototypes must be given class labels at termination of learning. The authors use a "relabeling" procedure, whereby each $P$-prototype gets the label of the most widely represented class amongst the data points associated with the prototype. Therefore, the learning is performed irrespective of classification accuracy, and it is always possible that clusters in the input space do not match the physical class labels very well.

The next two techniques overcome this difficulty. They are extraction techniques that guarantee a consistent set of $R$-prototypes.

- Chang's Method [5]: Starting with all of $\mathbf{X}$ by fixing $\mathbf{V} \equiv \mathbf{X}$, the $R$-prototypes are obtained by merging two at each step whose resultant $R$-prototype will assure zero errors if used instead of the original pair.
- Modified Chang method: The authors of [3] propose a modification that consists of changing the merging formula to a simpler one and improving the search scope for candidates for merging at each step.

The last algorithm considered in [3] follows.

- MFCM-3: This is a modification of fuzzy $c$-means clustering, which allows for more than one prototype per class to be retained.

## IV. Experimental Result

We used the popular IRIS data set comprising 150 vectors in $\Re^4$: 50 each from three physically labeled subspecies of IRIS flowers [11]. Our experiments with the GA and RS searches are only illustrative because of the small number of runs. The purpose here is to show the capacity of the algorithms and not to investigate their characteristics (e.g., robustness, convergence, etc.).

In one of the experiments, we constrain the GA to zero-error chromosomes only (consistent sets of $S$-prototypes). The restriction is implemented in the form of a genetic operator, which practically eliminates the first term of the fitness function (2). This operator "kills" all the offspring producing errors and keeps the reproduction of a pair of parents going (by crossover and mutation) until two survivors are created. The number of evaluations of the fitness function in this version can be significantly higher than in the other one.

The parameters for the GA follow.

- $N_{pop}$ was either ten or 20.
- $P_{ini}$ was set to 0.8 for the restricted GA. Dense initialization helped to quickly find an initial population. Then "evolution" was driven toward minimizing the cardinality. For the other version of the GA, $P_{ini}$ was 0.1.
- $M_{gen}$ was 500 in all the experiments.

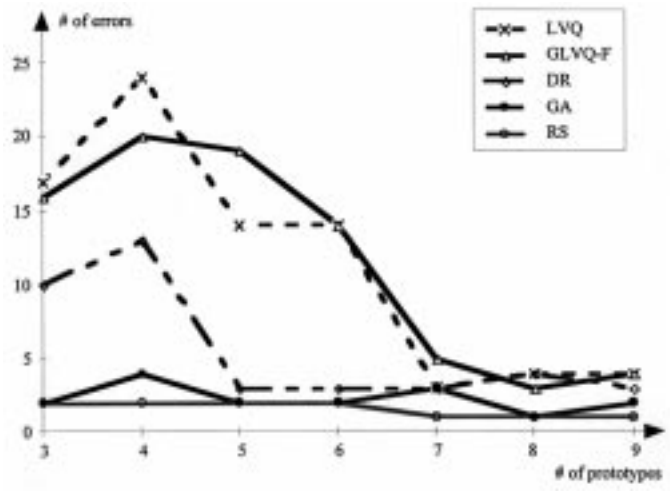


Fig. 2. Comparison of $R$- and $S$-prototype resubstitution error rates for IRIS with five methods.

TABLE I
MINIMAL NUMBER OF RESUBSTITUTION ERRORS FOR IRIS

| $c \rightarrow$ | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|
| LVQ | 17 | 24 | 14 | 14 | 3 | 4 | 4 |
| GLVQ-F | 16 | 20 | 19 | 14 | 5 | 3 | 4 |
| DR | 10 | 13 | 3 | 3 | 3 | 4 | 3 |
| GA | 2 | 4 | 2 | 2 | 3 | 1 | 2 |
| RS | 2 | 2 | 2 | 2 | 1 | 1 | 1 |

- $P_m$ was either 0.015 or 0.025.
- $\alpha$ was 0.01.
- We ran experiments with both $f_1$ and $f_2$ as the penalty term of (2).

With GA and RS we can derive as many prototypes as we like (which is not possible for the condensation techniques and for the Chang algorithm and its modification). Fig. 2 graphs the number of errors versus the number of prototypes for three $R$-prototype techniques that allow specification of the number of prototypes as well as GA and RS. First, both $S$-prototype selection methods work better than their $R$-prototype competitors. And second (however awkward it might seem to admit it), RS finds the minimal sets with the lowest error rate.

Table I shows the lowest resubstitution error rate for IRIS achieved by LVQ, GLVQ-F, DR, GA, and RS. Both RS and GA have found sets of three $S$-prototypes that produce two resubstitution errors (Table II). The best "inconsistent" result claimed by the authors in [3] is obtained by the DR algorithm, five $R$-prototypes that result in three errors.

Exhaustive search of all combinations of three prototypes found 95 sets with two resubstitution errors. The total number of tested sets was 125 000. Each of these 125 000 sets had one prototype per class: every other three-prototype set will result in at least 50 resubstitution errors.

When the limit $T$ in (4) was 10 or 11, GA resulted in subsets with one resubstitution error. The error rate of the sets of size 10 and 11 found by RS was also one error. Increasing the cardinality more did not lead to better results with RS: two errors with a 12-element set and one error with a 13-element set. Our restricted version of the GA, however, converged to consistent sets of 12, 12, and 13 elements on

TABLE II
RS (LEFT) AND GA (RIGHT) $S$-PROTOTYPES WITH TWO RESUBSTITUTION NPC ERRORS FOR IRIS

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | class |
|---|---|---|---|---|
| 4.4 | 2.9 | 1.4 | 0.2 | 1 |
| 5.9 | 3.0 | 4.2 | 1.5 | 2 |
| 5.7 | 2.5 | 5.0 | 2.0 | 3 |

(a)

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | class |
|---|---|---|---|---|
| 5.0 | 3.2 | 1.2 | 0.2 | 1 |
| 6.1 | 3.0 | 4.6 | 1.4 | 2 |
| 5.9 | 3.0 | 5.1 | 1.8 | 3 |

(b)

TABLE III
A 12-ELEMENT CONSISTENT SET OF $S$-PROTOTYPES FOR IRIS FOUND BY THE GA METHOD

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | class |
|---|---|---|---|---|
| 4.8 | 3.0 | 1.4 | 0.1 | 1 |
| 5.9 | 3.2 | 4.8 | 1.8 | 2 |
| 6.3 | 2.5 | 4.9 | 1.5 | 2 |
| 6.8 | 2.8 | 4.8 | 1.4 | 2 |
| 6.0 | 2.7 | 5.1 | 1.6 | 2 |
| 6.2 | 2.9 | 4.3 | 1.3 | 2 |
| 5.8 | 2.7 | 5.1 | 1.9 | 3 |
| 6.3 | 2.9 | 5.6 | 1.8 | 3 |
| 6.0 | 2.2 | 5.0 | 1.5 | 3 |
| 6.2 | 2.8 | 4.8 | 1.8 | 3 |
| 6.3 | 2.8 | 5.1 | 1.5 | 3 |
| 6.0 | 3.0 | 4.8 | 1.8 | 3 |

three different runs. The two 12-element sets are nearly identical. We show one of the sets in Table III. The other consistent set of 12 $S$-prototypes differs from the one in Table III only by the vector from class 1, whose components were $x_1 = 5.0$; $x_2 = 3.4$; $x_3 = 1.6$; $x_4 = 0.4$.

In Fig. 3, we plot the best results for IRIS achieved by each of the considered techniques as points in the (Errors, Prototypes) space. In general, $S$-prototype selection leads to better solutions. The Pareto-optimal set of results is

$$\{(0,\ 11,\ \text{Modified Chang}),\ (1,\ 7,\ \text{RS}),\ (2,\ 3,\ \text{RS}),\ (2,\ 3,\ \text{GA})\}$$

where the parentheses enclose (errors, prototypes, design method). This shows that for these trials, the best consistent solution is a set of $R$-prototypes, while the best inconsistent solutions are $S$-prototypes.

## V. DISCUSSION

### A. $S$-Prototypes versus $R$-Prototypes

Our experiments clearly indicate that $S$-prototypes result in a smaller resubstitution error rate. Since $S$-prototypes are found by direct minimization of the classification error, while $R$-prototypes are usually extracted by optimizing some other criterion, better results with $S$-prototypes are not surprising. Only Chang's method and its modification are designed to converge to $R$-prototypes that ensure zero error rate. Note that the Modified Chang's prototype set belongs to the Pareto-optimal set of solutions. It seems that finding

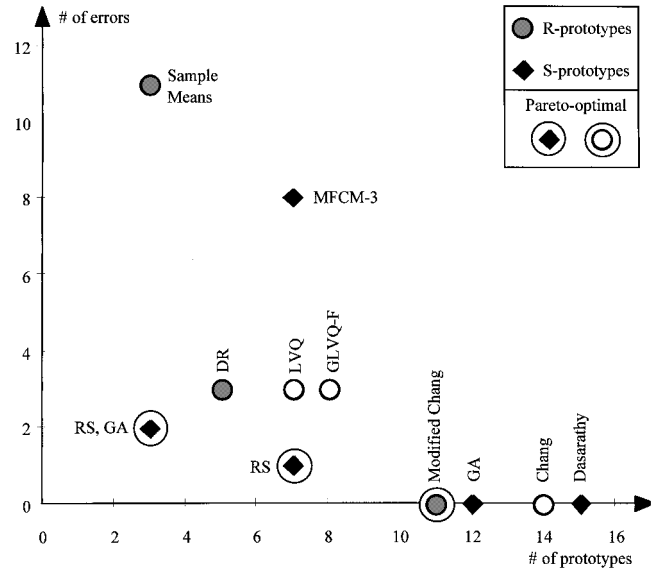


Fig. 3. Comparison of best resubstitution error rates for IRIS with all $R$- and $S$-prototype methods.

clusters in the data and subsequent relabeling does not guarantee good classification results. Since accuracy is the main goal, our preference goes to $S$-prototypes.

Another aspect of this question is generalization, i.e., accuracy on unknown data. Overfitting the data set $\mathbf{X}$ is a risk when directly minimizing the classification error. It may turn out that NPC with $R$-prototypes found by clustering-relabeling is better at generalization. Experiments on the generalization performance of GA selection [13] suggest that with a small number of $S$-prototypes NPC can reach reasonably good generalization rates. No generalization experiments with $R$-prototypes are presented in [3].

It is also true that selection of $S$-prototypes by RS and GA is much more computationally demanding and, for large data sets, may be infeasible.

One point in favor of GA's (and $S$-prototypes) is that the *cluster-validity* problem is surmounted. RS and most of the other techniques either require specifying the number of prototypes in advance or they converge to a set whose cardinality cannot be specified or changed as desired (e.g., the Chang's methods and the condensation techniques). In GA's, the optimal number of $S$-prototypes is decided in the course of the evolutionary process and can be influenced by the value of $\alpha$.

### B. Trading-Off Accuracy for Prototype Number Reduction

We consider $S$-prototypes here. The condensation techniques guarantee only that the resultant set is consistent (zero resubstitution error). It is not clear, however, how to find a subset if we agree to accept, say, one error (or more generally, $l$ errors at most). Due to the sequential character of condensation algorithms, their modifications to meet this requirement are not straightforward. For example, it must be decided at which particular sample(s) of $\mathbf{X}$ the error(s) should be committed. In the GA implementation, we can embed the maximal admissible number of errors in the reproduction scheme and prevent survival of chromosomes that do not meet the requirement. Starting with large subsets of $S$-prototypes and penalizing the size, we can arrive at reasonably small sets with the desired error rate.

### C. RS versus GA's

Can RS really be that good? RS easily solved a problem that was difficult for more sophisticated techniques. One possibility is that for the search space considered the number of estimations of the criterion function (error rate by NPC) was large enough that RS had a relatively high probability of hitting a good solution. For example, there are 551 300 combinations of three $S$-prototypes from 150. In our experiment, we performed 10 020 evaluations (with replacement), which were enough to find an $S$-prototype set resulting in two errors. We can expect, however, that when the number of prototypes increases (e.g., in the case of more complex classification structure in the feature space), RS will be "dethroned" by GA's or by other relatives of brute-force search.

Next, when we restricted the GA to consistent solutions only, it ended up with 12 $S$-prototypes with zero resubstitution NPC error. The RS did not achieve this result because it had no chance to *evolve* the final solution based on history.

Finally, a GA has many degrees of freedom, and the requirements that we may wish to define on the solution can be embedded in it. The above considerations point to GA's as a good option for $S$-prototype selection for NPC design.

## References

[1] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, fundamentals," *Univ. Comput.*, vol. 15, pp. 58–69, 1993.

[2] ——— "An overview of genetic algorithms: Part 2, research topics," *Univ. Comput.*, vol. 15, pp. 170–181, 1993.

[3] J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, "Multiple-prototype classifier design," this issue, pp. 67–79.

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

[5] C. L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. Comput.*, vol. C-23, pp. 1179–1184, Nov. 1974.

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

[7] ———, "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 511–517, Apr. 1994.

[8] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* Piscataway, NJ: IEEE Press, 1995.

[9] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[10] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-14, pp. 515–516, Mar. 1968.

[11] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[12] L. I. Kuncheva, "Editing for the $k$-nearest neighbors rule by a genetic algorithm," *Pattern Recognit. Lett.*, vol. 16, pp. 809–814, 1995.

[13] ———, "Fitness functions in editing $k$-NN reference set by genetic algorithms," *Pattern Recognit.*, vol. 30, no. 6, pp. 1041–1049, 1997.