Enhancing Prototype Reduction Schemes With Recursion: A Method Applicable for "Large" Data Sets

Sang-Woon Kim, *Senior Member, IEEE,* and B. John Oommen, *Fellow, IEEE*

***Abstract*—Most of the prototype reduction schemes (PRS), which have been reported in the literature, process the data in its entirety to yield a subset of prototypes that are useful in nearest-neighbor-like classification. Foremost among these are the prototypes for nearest neighbor classifiers, the vector quantization technique, and the support vector machines. These methods suffer from a major disadvantage, namely, that of the excessive computational burden encountered by processing *all* the data. In this paper, we suggest a recursive and computationally superior mechanism referred to as adaptive recursive partitioning (ARP)_PRS. Rather than process all the data using a PRS, we propose that the data be recursively subdivided into smaller subsets. This recursive subdivision can be arbitrary, and need not utilize any underlying clustering philosophy. The advantage of ARP_PRS is that the PRS processes *subsets* of data points that effectively sample the entire space to yield smaller subsets of prototypes. These prototypes are then, in turn, gathered and processed by the PRS to yield more refined prototypes. In this manner, prototypes which are in the interior of the Voronoi spaces, and thus ineffective in the classification, are eliminated at the subsequent invocations of the PRS. We are unaware of any PRS that employs such a recursive philosophy. Although we marginally forfeit accuracy in return for computational efficiency, our experimental results demonstrate that the proposed recursive mechanism yields classification comparable to the *best* reported prototype condensation schemes reported to-date. Indeed, this is true for both artificial data sets and for samples involving real-life data sets. The results especially demonstrate that a fair computational advantage can be obtained by using such a recursive strategy for "*large*" data sets, such as those involved in data mining and text categorization applications.**

***Index Terms*—Prototypes for nearest neighbor classifiers (PNN), prototype reduction schemes (PRS), recursive prototype reduction, support vector machines (SVM), vector quantization (VQ).**

I. Introduction

A. Overview

THIS PAPER proposes a family of recursive Prototype Reduction Schemes relatively novel to the field of statistical pattern recognition. It can be used advantageously for medium and large data sets.

Manuscript received December 19, 2002; revised August 1, 2003. The work of S.-W. Kim was supported in part by the Korea Science and Engineering Foundation (KOSEF). The work of B. J. Oommen was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). This paper was presented in part at the 2002 International Symposium on Syntactic and Statistical Pattern Recognition (SSSPR'02), Windsor, Canada. This paper was recommended by Associate Editor D. Cook.

S.-W. Kim is with the Division of Computer Science and Engineering, Myongji University, Yongin 449-728, Korea (e-mail: kimsw@mju.ac.kr).

B. J. Oommen is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada (e-mail: oommen@scs.carleton.ca).

Digital Object Identifier 10.1109/TSMCB.2004.824524

Let $T = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \in \mathcal{R}^p$ be a set of N feature vectors in p dimensions. We assume that T is a labeled data set, so that T can be decomposed into, say, c subsets $\{T_1, \ldots, T_c\}$ such that $T = \bigcup_{k=1}^{c} T_k, T_i \cap T_j = \phi, \forall i \neq j$. Our goal is to design a classifier with this *training data* set. Specifically, we are interested in classifiers of a nearest neighbor (NN) or nearest prototype family [1]. Thus, we need one or more *prototypes* (vectors in $\mathcal{R}^p$) that will represent each T_k. The limiting case comprises using *all* of the input vectors as prototypes, but in many cases, this will impose an unacceptable computational burden on the classifier.

In nonparametric pattern classification which use the NN or the k-nearest neighbor (k-NN) rule, each class is described using a set of sample prototypes, and the class of an unknown vector is decided based on the identity of the closest neighbor(s) which are found among all the prototypes [1]. For applications involving large data sets, such as those involved in data mining, financial forecasting, retrieval of multimedia databases and biometrics, it is advantageous to reduce the number of training vectors while simultaneously insisting that the classifiers that are built on the reduced design set perform as well, or nearly as well, as the classifiers built on the original data set. Various prototype reduction schemes, which are useful in NN-like classification, have been reported in the literature—two excellent surveys are found in [2] and [3]. Bezdek *et al.* [3], who composed the second and more recent survey of the field, reported that there are "zillions!" of methods for finding prototypes (see page 1459 of [3]).

Rather than embark on yet another survey of the field, we mention here a *few* representative methods of the "zillions" that have been reported. One of the first of its kind is the condensed nearest neighbor (CNN) rule [4]. The reduced set produced by the CNN, however, customarily includes "interior" samples, which can be completely eliminated, without altering the performance of the resultant classifier. Accordingly, other methods have been proposed successively, such as the reduced nearest neighbor (RNN) rule [5], the prototypes for nearest neighbor (PNN) classifiers [6], the selective nearest neighbor (SNN) rule [7], two modifications of the CNN [8], the edited nearest neighbor (ENN) rule [9], and the nonparametric data reduction method [10], [11]. Besides these, in [12], the vector quantization (VQ) and the bootstrap [13] techniques have also been reported as being extremely effective approaches to data reduction. Recently, support vector machines (SVM) [14] have proven to possess the capability of extracting vectors that support the boundary between any two classes. Thus, they have

been used satisfactorily to represent the global distribution structure.

In designing NN classifiers, however, it seems to be intuitively true that prototypes near the separating boundary between the classes play more important roles than those which are more interior in the feature space. In creating or selecting prototypes, vectors near the boundaries between the classes have to be considered to be more significant, and the created prototypes need to be moved (or adjusted) toward the classification boundaries so as to yield a higher performance. Based on this philosophy, namely that of *selecting* and *adjusting* the reduced prototypes, we recently proposed a new hybrid approach that involved two distinct phases [15], [16]. In the first phase, initial prototypes are *selected* or *created* by any of the conventional reduction methods mentioned earlier. After this selection/creation phase, the technique in [15], [16] suggests a second phase in which the proposed reduced prototypes are migrated to their "optimal" positions by *adjusting* them by invoking an LVQ3-type *learning* scheme. The relative advantages of the scheme in [15], [16] have been demonstrated on both artificial and real-life data sets.

All the PRS methods reported in the literature, [2], [3], (including the one proposed in [15] and [16]) are practical as long as the size of the data set is not "too large." The applicability of these schemes for large-sized data sets is limited because they all suffer from a major disadvantage—they incur an excessive computational burden encountered by processing *all the data points*. It should be noted, however, that points in the interior of the Voronoi space[1] of each class are usually processed for no reason—typically, they do not play any significant role in NN-like classification methods. Indeed, it is not unfair to state that processing the points in the "interior" of the Voronoi space becomes crucial only for "smaller" instantiations of the problem.

To overcome this disadvantage for large-sized[2] data sets, in this paper, we suggest a recursive mechanism. Rather than process all the data using a PRS, we propose that the data be recursively subdivided into smaller subsets. As will be explained presently, we emphasize that the smaller subsets need **not** *be obtained as the result of invoking a clustering operation on the original data sets*. After this recursive subdivision, the smaller subsets are reduced with any traditional PRS. The resultant sets of prototypes obtained are, in turn, gathered and processed at the higher level of the recursion to yield more refined prototypes. This sequence of divide-reduce-coalesce is invoked recursively to ultimately yield the desired reduced prototypes. We refer to the algorithm presented here as the adaptive recursive partitioning (or ARP) PRS.

The paper is organized as follows. In Section II, we briefly review a *few* representative PRSs. A complete survey is impossible here—the interested reader would find more comprehensive surveys in [2], [3]. Section III presents ARP_PRS, our recursively enhanced PRS, and it is followed by a description of two recently-reported methods. Both of them are distinct from our present scheme, but, in terms of philosophy, are probably, the most related reported papers. This is followed by a short section concerning complexity issues. Experimental results for small, medium, and "large-sized" data sets, and related discussions are provided in Sections IV and V. Finally, Section VI concludes the paper.

B. Contributions of the Paper

The main contribution of this paper is the demonstration that the speed of data condensation schemes can be increased by recursive computations—which is crucial in large-sized data sets. This has been done by introducing the ARP_PRS, and by demonstrating its power in both speed and accuracy.

The first main advantage of this mechanism, when compared to the corresponding nonrecursive versions, is that *this* PRS can select more refined prototypes in significantly less time. This is achieved without sacrificing either the classification accuracy or the prototype reduction rate (which is the percentage of the number of retained prototypes) in any significant manner. This is primarily because the new scheme does **not** process all the data points at *any* level of the recursion. The leaf-levels of the recursion process the original data points, but not in their entirety. Each leaf-level recursion processes only a *subset* of the original points, the outputs of which are merged to constitute the higher levels of the recursion.

A second advantage of this scheme is that the recursive subdivision can be arbitrary. *It need not be obtained as a result of utilizing any clustering philosophy*. In this manner, prototypes, which are in the interior of the final Voronoi spaces, and which are thus ineffective in the classification, are typically eliminated at the *leaf-level* invocations of the PRS, and do not participate in any higher level of the PRS. Furthermore, *all* subsequent recursive partitionings do not have to necessarily and explicitly involve (or invoke) a clustering operation. Finally, the higher level PRS invocations, typically, do not involve any points interior to the Voronoi space because they are eliminated at the leaf levels.

We believe that, to the best of our knowledge, there is currently no reported PRS which possesses all of these properties.

The reader should observe that this philosophy is *quite distinct* from the partitioning that uses prior clustering methods, such as those which have been recently proposed in the literature to solve the travelling salesman problem (TSP) [21]. The work of [21] first divides the set of cities into subsets by adapting a *clustering* mechanism. Hamiltonian paths through these clustered cities are determined, and the solution to the original TSP is computed by merging the relevant Hamiltonian paths. Observe that in our present solution, we do not require any such clustering phase, because the partitioning of the cities into subsets can be quite arbitrary. Indeed, ironically, it is advantageous

[1]Typically, the Voronoi hyperplane between two classes is an equi-bisector of the space, partitioning the points of each class on either side. Classification is achieved by assigning a class index to a sample being tested, and in our context, this is done by computing the location of the tested sample in the Voronoi space, for example, by determining the class of *its* NN using any well-established metric.

[2]First of all, we request the freedom to use this "non-English" phrase. More importantly, it is our opinion that the issue of when a data set can be perceived to be of "large size" is a subjective one. In certain domains, Huber [30] says that a data set is "large" if its cardinality is of the order of 10^6. However, data sets of these sizes are not usually encountered in pattern recognition, where it is often hard to get an adequate number of training samples. In this paper, we shall call the data set "medium-sized" if it involves *thousands* of data points, and "large" if it involves *tens of thousands* of data points.

to have the Voronoi spaces of the various subsets to be *maximally overlapping*. This will be seen presently.

The experimental results on synthetic and real-life data prove the power of these enhancements. The real-life experiments include three "medium-size" data sets, and two "large" data sets with a fairly high dimensionality. The results we present seem to be convincing.

II. Prototype Reduction Schemes

As mentioned previously, various data reduction methods have been proposed in the literature—two excellent surveys are found in [2], [3]. To put the results available in the field in the right context, we mention, in detail, the contents of the latter. The survey of [3] contains a comparison of eleven conventional PRS methods. This comparison has been performed from the view of error rates, and of the resultant number of prototypes that are obtained. The experiments were conducted with four experimental data sets which are both artificial and real.

In summary, the eleven methods surveyed areas follows: a combination of Wilson's ENN and Hart's CNN (W+H), the random selection (RS) method, genetic algorithms (GA), a tabu search (TS) scheme, a vector quantization-based method (LVQ1), decision surface mapping (DSM), a scheme which involves LVQ with training counters (LVQTC), a bootstrap (BTS) method, a vector quantization (VQ) method, a generalized LVQ-fuzzy (GLVQ-F) scheme, and a hard C-means clustering (HCM) procedure. Among these, the W+H, RS, GA, and TS can be seen to be selective PRS schemes, and the others fall into the category of being creative. Additionally, the VQ, GLVQ-F, and HCM are post-supervised approaches in which the methods first find prototypes without regard to the training data labels, and then assign a class label to each prototype, while the remaining are pre-supervised ones that use the data and the class labels together to find the prototypes. Finally, the RS, LVQ1, DSM, BT, VQ, and GLVQ-F are capable of permitting the user to define the number of prototypes, while the rest of the schemes force the algorithm to decide this number.

The claim of [3], from the experimental results, is very easily stated. Based on the experimental results obtained, the authors of [3] claim that there seems to be no clear scheme that is *uniformly* superior to all the other PRSs. Indeed, different methods were found to be superior for different data sets. However, the experiments showed that the *creative* methods can be superior to the selective methods, but are, typically, computationally more difficult to execute. Also, the experimental results revealed that *pre*-supervised methods are generally better than the *post*-supervised ones. Furthermore, there seems to be no reason to believe that the auto-defined approaches are superior to the user-defined ones.

The most pertinent methods are reviewed here[3] by two groups: two conventional methods and a newly proposed hybrid method. Among the conventional methods, the CNN and the SVM are chosen as representative schemes of *selecting* methods. The former is one of first methods proposed, and the latter is more recent. As opposed to these, the PNN and VQ (or SOM) are considered to fall within the family of prototype-*creating* algorithms. The reviews of these methods here are necessarily brief.

A. Conventional Methods

1) The CNN Rule: The CNN [4] is suggested as a rule which reduces the size of the design set, and is largely based on statistical considerations. However, the rule does not, in general, lead to a minimal consistent set, i.e., a set which contains a minimum number of samples that are sufficient to optimally classify all the remaining samples in the given set. The procedure can be formalized as follows, where the training set is given by T, and the reduced prototypes constitute the set $T_{\rm cnn}$.

1) The first sample is copied from T to $T_{\rm cnn}$.
2) Do the following: increase i by unity from 1 to the number of samples in T per epoch:
   a) classify each pattern $x_i \in T$ using $T_{\rm cnn}$ as the prototype set;
   b) if a pattern x_i is classified incorrectly then add the pattern to $T_{\rm cnn}$, and go to 3.
3) If i is not equal to the number of samples in T, then go to 2.
4) Else the process terminates.

2) PNN Classifiers: Chang's algorithm for finding PNNs [6] can be summarized as follows: Given a training set T, the algorithm starts with every point in T as a prototype. Initially, the set[4] A is assigned to be empty, and the set B is T itself. The algorithm selects an arbitrary point from B, and initially assigns it to A. After this, the two closest prototypes p from A and q from B of the same class are merged, successively, into a new prototype, p^*, if the merging will not degrade the classification of the patterns in T, where p^* is the weighted average of p and q. For example, if p and q are associated with weights W_p and W_q, respectively, p^* is defined as $(W_p \cdot p + W_q \cdot q)/(W_p + W_q)$, and is assigned a weight, $W_p + W_q$. Initially, every prototype has an associated weight of unity. The PNN procedure is sketched below.

1) Copy T to B.
2) For all $q \in B$, set the weight $W_q = 1$.
3) Select a point in B, and move it from B to A.
4) $\mathrm{MERGE} = 0$.
5) While B is not empty:
   a) find the closest prototypes p and q from A and B, respectively;
   b) if p's class is not equal to q's class then insert q to A and delete it from B;
   c) else merge p of weight W_p, and q of weight W_q, to yield p^*, where $p^* = (W_p \cdot p + W_q \cdot q)/(W_p + W_q)$.

[3]We note that the intention here is not to survey the field, but to merely specify a few of the representative PRS. In this light, it should be emphasized that our new recursive philosophy can be invoked using any one of the PRSs explained below, or for that matter, any of the PRS methods surveyed in [2], [3]. This should soon be clear to the reader. This brief survey section has been included on the recommendation of the referees.

[4]In the spirit of the above notation, the set A should be perceived to be the set $T_{\rm pnn}$. To keep things simple, we use this notation to be consistent with the PNN-related literature.

Let the classification error rate of this new set of prototypes be ε:

- if the ε is increased then insert q to A, and delete it from B;
- else delete p and q from A and B, insert p^* with weight W_p+W_q to A, and MERGE++.

6) If MERGE is equal to 0 then output A as the set of trained code-book vectors, and the process terminates;
7) Copy A into B and go to 3.

Bezdek *et al.*, proposed a modification of Chang's PNN in [23]. First of all, instead of using the *weighted* mean of the PNN to merge prototypes, they utilized the simple arithmetic mean. Secondly, the process for searching for the candidates to be merged was modified by partitioning the distance matrix into submatrices "blocked" by common labels. This modification eliminated the consideration of candidate pairs with different labels. Based on the results obtained from experiments conducted on the Iris data set, the authors of [23] asserted that their modified form of the PNN yielded a better consistent reduced set for designing multiple-prototype classifiers.[5]

3) Vector Quantization and the Self-Organizing Map: The foundational ideas motivating VQ and the SOM are the classical concepts that have been applied in the estimation of probability density functions.[6] Traditionally, distributions have been represented either parametrically or nonparametrically. In the former, the user generally assumes the form of the probability density function, and the parameters of the function are learnt using the available data points. In pattern recognition (classification), these estimated distributions are subsequently utilized to generate the discriminant hyper-planes or hyper-quadratics, whence the classification is achieved.

As opposed to the former, in nonparametric methods, the practitioner assumes that the data must be processed in its entirety and not just by using a functional form to represent the data. The corresponding resulting pattern recognition (classification) algorithms are generally of the NN (or k-NN) philosophy, and are thus computationally expensive.

The concept of VQ [17] can be perceived as one of the earliest compromises between the above two schools of thought. Rather than represent the entire data in a compressed form using only the estimates, VQ opts to represent the data in the actual feature space. However, as opposed to the nonparametric methods, which use all or a subset of the data in the training and testing phases of classification, VQ compresses the information by representing it using a "small" set of vectors called the code-book vectors. The location of these code-book vectors are obtained by migration in the **feature** domain so that they collectively represent the distribution under consideration. We shall refer to this phase as the *intra-regional polarizing* phase [20] explained below.

In both VQ and the SOM, the polarizing algorithm is repeatedly presented with a point x_i from the set of points of a particular class. The neurons which represent the code-book vectors attempt to incorporate the topological information present in x_i. This is done as follows. First, the closest neuron to x_i, Y_{j^*}, is determined. This neuron, and a group of neurons in its neighborhood, B_{j^*}, are now moved in the direction of x_i. The set B_{j^*} is called the "Activation Bubble" or the "Update Neighborhood" in the display lattice. We shall presently specify how this is determined. The actual migration of the neurons is achieved by rendering the new Y_j to be a convex combination of the current Y_j and the data point x_i, for all $j \in B_{j^*}$. More explicitly, the updating algorithm proceeds as follow :

$$Y_j(t+1) = \begin{cases} (1-\alpha(t))Y_j(t) + \alpha(t)x_i, & \text{if } j \in B_{j^*}(t) \\ Y_j(t), & \text{otherwise} \end{cases} \quad (1)$$

where t is the discretized (synchronized) time index.

This basic algorithm has two fundamental parameters: $\alpha(t)$ and the size of the bubble $B_{j^*}(t)$, where $\alpha(t)$ is the adaptation constant and satisfies $0 < \alpha(t) < 1$. Kohonen and others [18], [19] recommend steadily decrementing $\alpha(t)$ linearly from unity for the initial learning phase, and then switching it to small values which decrease linearly from 0.2 for the fine-tuning phase.

$B_{j^*}(t)$ is the parameter which makes the VQ differ from the SOM. Indeed, the VQ is a special case of the SOM for which $B_{j^*}(t) = \{Y_j(t)\}$, i.e., if the size of the bubble is always zero. However, in the SOM, the nearest neuron and the neurons within a bubble of activation are also migrated, and hence, *this* widened migration process permits the algorithm to be both *topology preserving* and self-organizing. The size of the bubble is initially assigned to be fairly large to allow a global ordering to develop. Consequently, all the neurons tend to tie themselves into a knot for a value of $\alpha(t)$ that is close to unity; they subsequently quickly disperse. Once this coarse spatial resolution is achieved, the size of the bubble is steadily decreased. Consequently, only those neurons which are most relevant to the processed input point will be effected by it. Thus, the ordering, which has been achieved by the coarse resolution, is not disturbed, but the fine tuning on this ordering, is permitted. Of course, in the limit, the SOM reduces to VQ when $B_{j^*}(t) = \{Y_j(t)\}$.

After the location of the code-book vectors of each class have been computed by the process described above, the inter-class polarization is optimized by a process which Kohonen refers to as LVQ3 [19]. This is a fairly straightforward process, and we omit its details here in the interest of brevity.

4) Support Vector Machines: The SVM [14] is a fairly new and very promising classification technique developed at the AT&T Bell Laboratories. The main motivating criterion is to separate the various classes in the training set with a surface that maximizes the margin between them. It is an approximate implementation of the structural risk minimization induction principle that aims to minimize a bound on the generalization error of a model, rather than minimizing the mean square error over the training data set, which is the philosophy that empirical risk minimization methods often use.

[5]We believe that the fundamental recursive enhancement that we propose in this paper, can also be utilized to enhance the scheme proposed in [23]. We also believe that a similar enhancement can be used for the clustering-based, genetic, and random search methods proposed in [24]. This is currently being investigated. The authors are grateful to Professor Jim Bezdek for the instructive discussions we had in Spain in April 2002.

[6]This is, of course, arguable. Many would argue that the *foundational* idea of the VQ and SOM is to imitate the way arrays of neurons seem to reorganize their structure to solve a problem. Indeed, their ability to estimate probability density functions (pdfs) was a *subsequent* discovery. We are grateful to the anonymous referee who pointed this out to us.

Training an SVM requires a set of N samples. Each sample consists of an input vector x_i and its label y_i. The SVM function that has to be trained with the samples contains N free parameters, the so-called positive Lagrange multipliers $\alpha_i, i = 1, \ldots, N$. Each α_i is a measure of how much the corresponding training sample influences the function. Most of the samples do not affect the function, and consequently, most of the α_i are 0. To find these parameters, we have to solve a quadratic programming (QP) problem like

$$\text{Minimize } \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i Q_{ij}\alpha_j - \sum_{i=1}^{N}\alpha_i \tag{2}$$

$$\text{Subject to } 0 \le \alpha_i \le C, \quad \sum_{i=1}^{N} y_i\alpha_i = 0 \tag{3}$$

where Q is an $N \times N$ matrix that depends on x_i, y_i and the functional form of the SVM, and C is a constant to be chosen by the user. A larger value of C corresponds to assigning a higher penalty to the errors.

Solving the QP problem provides the support vectors of the two classes, which correspond to the samples of $\alpha_i \neq 0$. Using these, we get a hyper-plane decision function $w^T x + b = 0$, which separates the positive samples having $+1's$ as their labels from the negative samples whose labels are all -1. The weight vector and the threshold are $w = \sum_{i=1}^{N_s}\alpha_i y_i x_i$ and $b = (1/2)(w^T x_p + w^T x_n)$, respectively, where N_s is the number of support vectors, w^T is the transpose of w, and x_p and x_n are support vectors of the positive and the negative classes, respectively.

Usually, to permit much more general nonlinear decision functions, we have to first nonlinearly transform the input vectors into a high-dimensional feature space by a map ϕ, and then invoke a linear separation in *that* space. In this case, minimizing (2) requires the computation of dot products $\phi(x)\cdot\phi(y)$ in the higher-dimensional space. The expensive calculations, however, can be avoided by using a kernel function K obeying $K(x, y) = \phi(x)\cdot\phi(y)$, that can be evaluated efficiently. The kernel, K, includes functions such as polynomials, radial basis functions or sigmoidal functions. Details of the SVM can be found in [14], [22], and [25].

B. A Relatively-New Hybridized Technique

In designing NN classifiers, prototypes near the inter-class boundaries play more important roles than those which, for each class, are more interior in the feature space. In creating or selecting the prototypes, therefore, the points near the class boundaries are most significant, and the created prototypes need to be moved or adjusted toward the classification boundaries so as to yield higher performance. The approach that we proposed in [15] and [16] is based on this philosophy, namely that of invoking *creating* and *adjusting* phases. First, a reduced set of initial prototypes or code-book vectors is chosen by any of the known methods, and then their "optimal" positions are learned with an LVQ3-type algorithm, thus, minimizing the average classification error.

In LVQ3, two *generic* code-book vectors m_i and m_j, which are the two NNs to a sample of known identity x, are simultaneously updated, where x and m_j belong to the same class, and x and m_i belong to different classes. Moreover, x must fall into a zone of values called the "window," which is defined around the mid-plane of m_i and m_j. Assume that d_i and d_j are the Euclidean distances of x from m_i and m_j, respectively. Then x is defined to fall in a window of relative width w if

$$\min\left(\frac{di}{dj}, \frac{dj}{di}\right) > \left(\frac{1-w}{1+w}\right). \tag{4}$$

The updating rules for m_i and m_j ensure that the code-book vectors continue to approximate the respective class distributions, and simultaneously enhance the quality of the classification boundary. These rules are

$$\begin{aligned} m_i(t+1) &= m_i(t) - \alpha(t)[x(t) - m_i(t)] \\ m_j(t+1) &= m_j(t) + \alpha(t)[x(t) - m_j(t)]. \end{aligned} \tag{5}$$

Additionally, even when x, m_i, and m_j belong to the same class, the code-book vectors are adjusted to enhance the improvement as follows for $k = i, j$

$$m_k(t+1) = m_k(t) - \epsilon(t)\alpha(t)[x(t) - m_k(t)]. \tag{6}$$

In (5) and (6), t is the discretized (synchronized) time index, and $\alpha(t)$ and $\epsilon(t)$ are called the *learning rate* and *relative learning rate*, respectively. We shall presently demonstrate how this is done.

1) Specifying the Relevant LVQ3 Criteria: The heart of the algorithm involves post-processing the results of any conventional pre-supervised data reduction method using an LVQ3-type algorithm. However, the more crucial issue is that of determining the parameters of the LVQ3-type algorithm. This is accomplished in [15], [16] by partitioning the given data sets into two subsets, which are, in turn, utilized to optimize the corresponding LVQ3 parameters.

Let us suppose that the initial data for class i is the *training* set, $T_{i,t}$, and *validation* set, $T_{i,V}$. The *training* set $T_{i,t}$ is further partitioned into two subsets, the *placement* set, $T_{i,P}$ and the *optimizing* set, $T_{i,O}$, where $T_{i,t} = T_{i,P} \cup T_{i,O}$. The intention is that the union of the *placement* sets, $T_P = \bigcup_{i=1}^{c} T_{i,P}$, is used to *position* the condensed prototypes using an LVQ3-type algorithm, and the parameters of the LVQ3-type algorithm are, in turn, optimized by testing the classification efficiency of the *current* placement, on the union of the *optimizing* sets, T_O, where $T_O = \bigcup_{i=1}^{c} T_{i,O}$. Thus, the training set plays a triple role: 1) it is used to obtain the condensed vectors; 2) one portion of this set is used by the LVQ3-type algorithm to migrate the condensed vectors; and 3) the other portion of the training set serves the purpose of "pseudotesting", so as to obtain the best parameters for the LVQ3-type algorithm.

2) The Hybrid_PRS Algorithm: The Hybrid_PRS Algorithm essentially consists of two steps. First, initial prototypes are *selected* or *created* by any of the available conventional pre-supervised reduction methods. After this selection/creation phase, we invoke a phase in which the optimal positions (i.e., with regard to classification) are learned with an LVQ3-type scheme. The procedure is formalized below for each class.

1) For every class, j, select an initial condensed prototype set $Y_{j,\text{Test}}$ by using any of the available conventional pre-

supervised reduction methods, and the entire training sets, $T_{j,t}$.

2) Using $Y_{j,\text{Test}}$ as the set of condensed prototype vectors for class j, for all the classes, do the following using the union of the *placement* and *optimizing* sets T_P and T_O, respectively:
   a) Perform LVQ3 using the points in the overall *placement* set, T_P. The parameters of the LVQ3 are obtained by spanning the parameter space in increasing values of w from 0.0 to 0.5, in steps of Δw. The sets $Y_{j,\text{Test}}$ (for all j) and Y_{Test} are updated in the process. Select the best value w_0 after evaluating the accuracy of the classification rule on T_O, where the NN-classification is achieved by the adjusted values of Y_{Test}.
   b) Perform LVQ3 using the points in the overall *placement* set, T_P. The parameters of the LVQ3 are obtained by spanning the parameter space in increasing values of ϵ from 0.0 to 0.5, in steps of $\Delta\epsilon$. The sets $Y_{j,\text{Test}}$ (for all j) and Y_{Test} are updated in the process. Select the best value ϵ_0 after evaluating the accuracy of the classification rule on T_O, where the NN-classification is achieved by the adjusted values of Y_{Test}.
   c) Repeat the above steps with the current w_0 and ϵ_0 until the best values w^* and ϵ^* are obtained.
3) Determine the best prototype set Y_{final} by invoking LVQ3, η times, with the data in T_P, and where the parameters are w^* and ϵ^*. Again, the "pseudotesting" is achieved using the {optimizing} set, T_O.

An estimate of the classification accuracy is obtained by testing the classifier using the final values Y_{final} and the original testing (validation) data points, $T_V = \bigcup_{i=1}^{c} T_{i,V}$. Other experimental details of the scheme can be found in [15] and [16], which also reports the results of the scheme for both artificial and real-life data sets.

III. Recursive Invocations of PRSs

A. The Rationale of the Recursive Algorithm

For designing NN classifiers, prototypes near the decision boundaries play more important roles than prototypes in the interior of each labeled class. In all the currently reported PRS, however, points in the interior of the respective Voronoi spaces are processed for, apparently, no reason. Consequently, all reported PRS suffer from an excessive computational burden encountered by processing all the data, which becomes very prominent in "large" data sets.

To overcome this disadvantage, we propose a recursive mechanism, whereby the data set is subdivided recursively into smaller subsets to filter out the less useful internal points. Subsequently, a conventional PRS processes the smaller subsets of data points that effectively sample the entire space to yield *subsets* of prototypes—one set of prototypes for each subset. The prototypes which result from each subset are then coalesced, and processed again by the PRS to yield more refined prototypes. In this manner, prototypes which are in the interior to the Voronoi spaces, and are thus ineffective in the classification, are eliminated at subsequent invocations of the PRS. A direct consequence of eliminating the "redundant" samples in the PRS computations, is that the processing time of the PRS is *significantly* reduced. This will be clarified by the example below.

B. An Example

In order to illustrate the functioning of the recursive process, prior to presenting the recursive algorithm formally, we present an example for the two-dimensional data set referred to as "random." Two data sets, namely the training and test sets, are generated randomly with a uniform distribution, but with irregular decision boundaries. In this case, the points are generated uniformly, and the assignment of the points to the respective classes is achieved by *artificially* assigning them to the region they fall into, as per the manually created "irregular decision boundary." The training set of 200 sample vectors is used for computing the prototypes, and the test set of 200 sample vectors is used for evaluating the quality of the extracted prototypes.

To demonstrate the properties of the mechanism, we first select prototypes from the whole training set using the CNN method. To highlight the difference between the latter and a recursive mechanism, this is also repeated after randomly[7] dividing the training set into two subsets of equal size—each with 100 vectors. Fig. 1 shows the whole set and the divided subsets of the "Random" training data set, and Fig. 2 shows the prototypes selected with the CNN and the recursive PRS methods, respectively. In each case, we have also shown the NN separating classification boundaries. The reader should observe that the separating classification boundary obtained from the *entire set* [see Fig. 1(a)] is quite similar to the one obtained from the subsets of half their sizes [see Fig. 1(b) and (c)]. This observation is also true about the *prototypes* obtained from the entire sets (see the corresponding figures in Fig. 2), and the *prototypes* obtained from their subdivided sets.

Observe that in this example, we have invoked the recursive procedure *only* twice. This is, of course, only for the sake of the example. In general, however, the recursion can be invoked at any depth, whenever the size of the processed set is larger than permitted.

Fig. 1(d) shows the "marginal" difference between the boundaries of subset1 (dotted line) and subset2 (solid line). In Fig. 2, the set of prototypes of (a), which is extracted from the whole set of Fig. 1(a), consists of 36 points and has a classification accuracy of 96.25%. The prototypes of (b) and (c), selected from the subsets of Fig. 1(b) and (c), *both* consist of 21 vectors, and have accuracies of 96.00% and 94.50% respectively.

On the other hand, the set of prototypes of (d), which is created by combining the prototype sets of Fig. 2(b) and (c), consists of 27 points, and has an accuracy of 97.00%. Moreover, it should be pointed out that the time involved in the prototype selection of (d), is much less than that of (a), because the number of sample vectors of the combined sets of (b) and (c) together, is smaller than that of the whole set of Fig. 1(a).

[7]This is done by a simple sequential procedure. As the points in "random" are generated, each of them is sequentially assigned to be either in the first subset or the second.

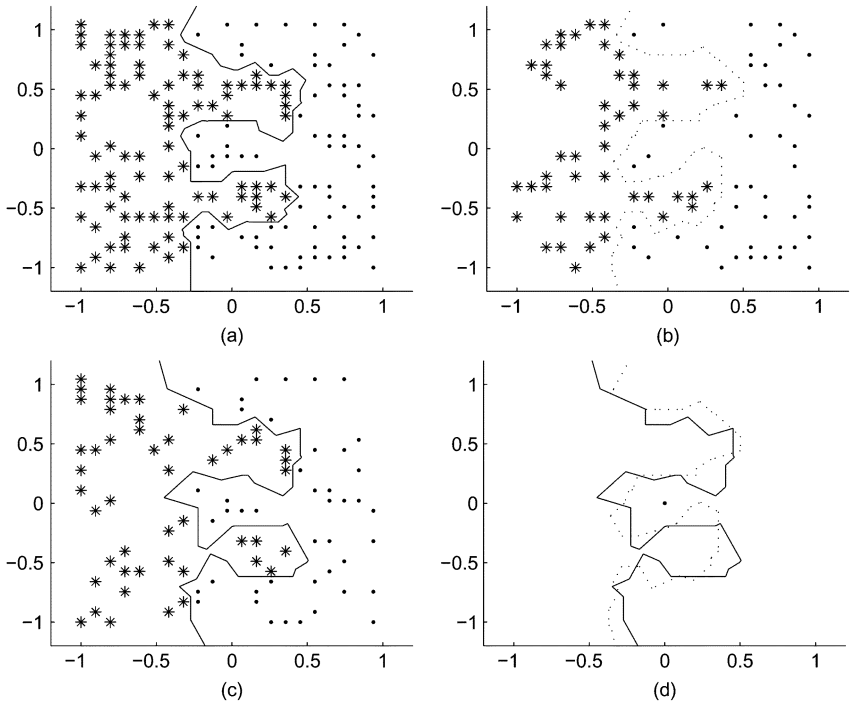

Fig. 1. The entire set and the divided subsets of the "Random" data set, where the vectors of each class are represented by "$*$" and "$\cdot$,"respectively. (a) The entire set of 200 points. (b) Subset1 containing 100 points. (c) Subset2 containing 100 points. (d) The boundary difference between subset1 (dotted line) and subset2 (solid line). Both subsets of (b) and (c) are obtained by randomly subdividing the whole set of (a).

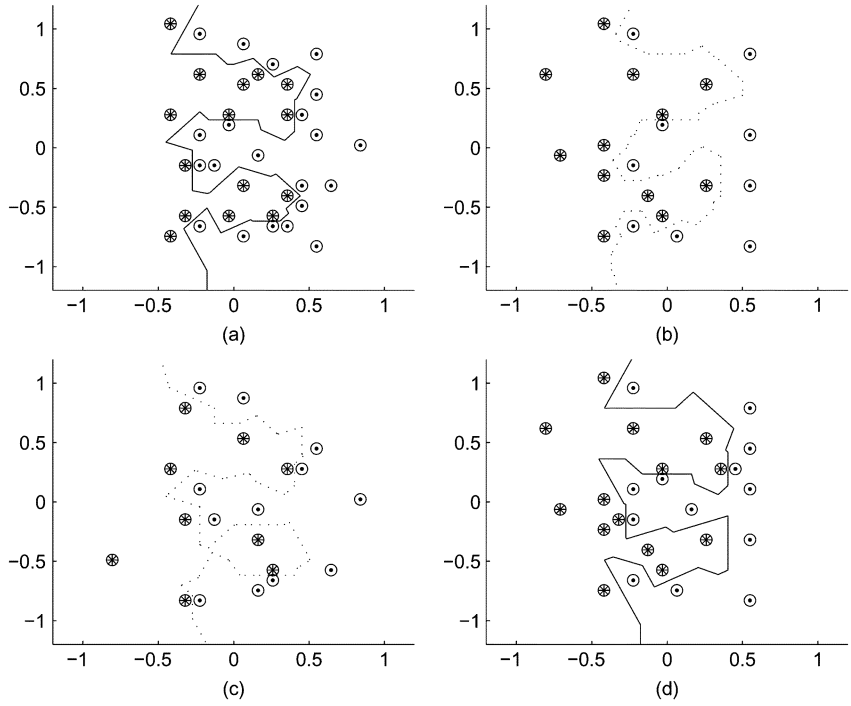

Fig. 2. Prototypes selected with the CNN and the recursive PRS methods from the data sets shown in Fig. 1. In the pictures, the selected vectors are indicated by the circled "$*$" and "$\cdot$," respectively. (a) The prototypes selected by the CNN from the whole set of Fig. 1(a). (b) The prototypes selected by the CNN from the subset1 of Fig. 1(b). (c) The prototypes selected by the CNN from the subset2 of Fig. 1(c). (d) The final prototypes selected by the recursive PRS method from a combined data set of the prototypes of (b) and (c).

From these considerations, we observe that the prototypes can also be selected from the subdivided data sets more efficiently, rather than selecting them from the original data set. We hope that this simple example adequately clarifies the motivation and advantages behind our reasoning.

C. The ARP_PRS Algorithm

The algorithm that implements the ARP_PRS can be formalized as follows, where the training set is given by T, and the reduced prototypes are found in Y_{final}.

An informal explanation of the algorithm follows. If the cardinality of the original data set is smaller than K, a traditional PRS is invoked to get the reduced prototypes. Otherwise, the original data set is recursively subdivided into J subsets, and the process continues down toward the leaf of the recursive tree. Observe that a traditional PRS is invoked only when the corresponding input set is "small enough." Finally, at the tail end of the recursion, the resultant output sets are merged, and if the size of *this* merged set is greater than K, the procedure is again recursively invoked. Observe that this is executed separately for each class.

It should be also noted that the traditional PRS, which can be otherwise time consuming for large data sets, is never invoked for *any* sets of cardinality larger than K. It is called *only* at the leaf levels when the sizes of the sets are "small," rendering the entire computation very efficient.

D. Methods With a Comparable Philosophy

Although there are "zillions" of methods which achieve prototype reduction, to the best of our knowledge, the fundamental strategy of utilizing recursion to subdivide the data points, has not been used. The reason for this is probably because if the subdivision of the points is done by means of a clustering philosophy, the resultant scheme need not be too advantageous. Indeed, it will then reflect the properties of the clustering model used.

This does not, by any means, imply that clustering is unimportant. Indeed, a fair body of literature which deals with clustering in the data mining of "large" data sets is available [30]–[34]. The application of clustering to information retrieval from "very large" data bases has also been reported.[8] But what we do observe is that it is not too expedient to apply clustering as a pre-processing module to a PRS scheme.

Algorithm 1 ARP_PRS

Input: The original Training Set, T.

Output: The set of reduced prototypes, Y_{Final}.

Method:

Call Recursive_PRS$(T, Y_{\mathrm{Final}}, K, J)$

End Algorithm ARP_PRS

Procedure $\mathrm{Recursive\_PRS}(\mathrm{InSet}, \mathrm{OutSet}, K, J)$

Input: The subset of the training set, InSet, and a parameter, K, which specifies the size of smallest set for which the procedure is not invoked recursively. This is the "basis" case of the recursion. InSet is not recursively sub divided if $|\mathrm{InSet}| \leq K$. In this case, we invoke a conventional PRS (referred to by PRS) that yields the extracted prototypes. Also, at every level, we opt to partition the original set into J subsets, where J is a user-specified parameter.

If $|\mathrm{InSet}| \leq K$

Call PRS(*InSet, OutSet*)

Return *OutSet*

Else

Partition *InSet* into J mutually exclusive Sets $\mathrm{InSet}_1 \ldots \mathrm{InSet}_J$

For $i \leftarrow 1$ to J **Do**

Call $\mathrm{Recursive\_PRS}(\mathrm{InSet}_i, \mathrm{OutSet}_i)$

$\mathrm{TempSet} \leftarrow \mathrm{OutSet}_1 \cup \mathrm{OutSet}_2 \cdots \cup \mathrm{OutSet}_J$

Call $\mathrm{Recursive\_PRS}(\mathrm{TempSet}, \mathrm{OutSet})$

End Procedure Recursive_PRS

However, to present our work in the overall perspective, we shall compare our work with two recent PR schemes which, in one sense, adapt the concept of subdividing the available data points [27], [28].

The work of [27] can be explained as follows. It is well known that iterative techniques that use the k-means approach, or the expectation maximization (EM) perspective, are sensitive to the initial starting conditions of the algorithm. The authors of [27] propose a refinement procedure in which initial points (for both discrete and continuous data sets) are specified in such a way that the corresponding iterative algorithms converge to "better" solutions.

The refinement algorithm, which is iterative and not recursive, initially chooses J small random subsamples of the data, $S_i, i = 1, \ldots J$. These subsamples are clustered using a k-means algorithm with the condition that on termination, empty clusters will have their initial centers re-assigned followed by a re-clustering of the subsamples. The sets $\mathrm{CM}_i, i = 1, \ldots J$ are the result of clustering over the subsamples, which collectively constitute the set CM. These are then, in turn, clustered using a k-means strategy initialized with the sets CM_i, producing the final solution FM_i. The refined initial point is then chosen as the FM_i, which yields the minimal distortion over the set CM.

Computational results on small real-world data sets indicate that the k-means solution from the refined points provides twice as much information as the solution from the random initial points. Furthermore, the average distortion is decreased by 9%. The computational results on a large-scale data set in 300 dimensions demonstrated a drop in distortion by about 20%, and the information gain improved by a factor of 4.13.

We clarify the difference between this work and our current method. First of all, we do not invoke any k-means-type algorithm. More importantly, we emphasize, that in our case, the subdivision of the data points is not achieved by invoking a clustering strategy. This is because a clustering philosophy will subdivide the points in terms of their "closeness" to a particular class. This, unfortunately, includes all the points in the interior of the various Voronoi spaces, which are irrelevant when computing the classification discriminant function. Thus, we save time by "short-circuiting" the entire clustering phase.

Another very interesting strategy, which we applaud, was proposed in [28], where the authors presented a new algorithm, the sequential minimal optimization (SMO), for training SVMs. Typically, the process of training a SVM requires the solution of a large QP optimization problem. The authors of [28] introduce SMO by subdividing this large QP problem into a series of smaller QP problems. In that sense, the philosophy is similar to the principles which we have introduced. These reduced QP problems are, in turn, solved *analytically*, thus avoiding

[8]The literature we cite here is not exhaustive. We mention these papers to demonstrate that we are by no means the pioneers in the field of processing large data sets. We are grateful to the anonymous referee for providing us with these references.

the computation of time-consuming numerical QP optimization routines.

Unlike the previous methods proposed for SVMs, SMO introduces two Lagrange multipliers to *jointly* optimize the QP criterion function, and computes the optimal values for these multipliers. It then updates the SVM to reflect the new optimal values for each of the multipliers.

The advantage of the SMO lies in the fact that it allows for *two* Lagrange multipliers, which can both be solved for analytically. Furthermore, the SMO can be used when the user does not have access to a QP package, and/or does not wish to utilize it. Experimentally, the SMO performs well for SVMs with sparse inputs, and even for nonlinear SVMs. This is because the kernel computation time can be reduced, thus directly enhancing its performance. The SMO performs well for large problems, because it "scales" well with the size of the training set. Indeed, it appears that it is superior to methods that utilize "chunking." The authors of [28] assert that the SMO is a strong candidate for becoming the standard SVM training algorithm.

It is fitting to mention the differences between the SMO and our present technique. First of all, unlike the SMO, the strategy that we propose can be used even if the underlying "primitive" PRS (invoked by the recursive mechanism) is *not* the SVM. Thus, it is not a scheme that is particular to any specific mechanism. Secondly, in achieving the recursive decomposition, we do not require any QP solution, or optimization using Lagrangians. Indeed, as the pseudocode of the algorithm demonstrates, no optimization of any sort is mandatory or recommended. The reason for this is because, in the recursive subdivision, we are not attempting to get the best classifier. Rather, we attempt to "discard" the points in the interior of the classification space, namely, those which are not of importance in determining the final classifier. Our algorithm is thus much faster at every level of the recursion. Finally, when the classifier is ultimately constructed, it is achieved using the final subset of points which are close to the boundaries of the respective spaces, further enhancing the computations.

E. Some Thoughts on Complexity Analysis

A formal complexity analysis for the ARP_PRS is not easy. This is because it depends on the specific PRS method that is utilized on each recursive call. Rather than embark on a general formal analysis, we shall now consider two pertinent avenues of study. In the first case, we shall consider the number of prototypes that a traditional PRS (say, *trad*) would lead to, and compare it to the number of prototypes that ARP_PRS, the recursive scheme, would yield if it were utilizing *trad* as its basic "building block". In the second case, we shall consider the time involved in a serialized computation, and compare the performance of *trad* with the performance of ARP_PRS, again with the premise that the latter uses *trad* as its basic PRS module.

To render the analysis tractable, we have to make some simple but reasonable assumptions. In the first case, we shall assume a reasonable functional form for the *number of prototypes* generated on any specific call of *trad*. Clearly, this functional form cannot be assumed to be valid for any arbitrary depth of the recursive tree. But we shall assume that this functional form is valid at least for the number of levels at which the question of resorting to recursion is pertinent. In the second case, we shall assume a reasonable functional form for the *time* required for any specific call of *trad*. Again, clearly, this functional form may not be valid for any arbitrary depth of the recursive tree. But here, too, we shall assume that this explicit form is valid as long as resorting to recursion is meaningful.

In both of these cases, we shall follow the analysis essentially for the scenario when we resort to recursively calling *trad* only for two subsets, and for at most two levels. From the analysis, it will be clear that the analogous results will also be valid if the depth of the recursive tree is greater than two.

1) Analysis on the Number of Prototypes Generated: Consider a system which uses a traditional PRS, referred to as *trad*. We assume that the number of prototypes generated by *trad* depends on the number of input sample data points, and that this relationship obeys the following equation:

$$N_{\text{out}} \approx K_1 (N_{\text{in}})^a. \tag{7}$$

Let the original number of points to be processed, (i.e., the number at the top-level) be N. If each set is split into *two* subsets prior to calling *trad*, the cardinality of the sets after a single pass, and after two passes, can be evaluated using (7), and are given by the two (8) and (9) respectively:

$$K_1 (N)^a \tag{8}$$

$$K_1 \left\{ 2K_1 \left(\frac{N}{2} \right)^a \right\}^a \tag{9}$$

where $a < 1$, and K_1 is the constant determining the efficiency of *trad*, as given by (7). We now consider the condition for the number of prototypes being generated after a single pass, being greater than the number of prototypes being generated after two passes. Indeed, this results in the following inequality:

$$K_1 (N)^a \geq K_1^{1+a} 2^{(1-a)a} N^{a^2} \Rightarrow (N)^{a-a^2} \geq K_1^a 2^{a-a^2} \tag{10}$$

which, in turn, leads to the inequality $N \geq 2K_1^{1/(1-a)}$, which is *always* true for a sufficiently large value of N.

To clarify issues, consider the simple example when $a = 0.5$. This is, of course, the scenario encountered in traditional "geometry" involving compact spaces, where the number of points on the perimeter of a region is of the order of the square root of the points in the interior of the region. Going through the same steps as above, we see that if there are N points at the top level, the number of points after a single pass is $K_1\sqrt{N}$, and the number of points remaining after two passes obtained by partitioning the original set into two subsets is $K_1\sqrt{N/2} + K_1\sqrt{N/2} = 2K_1\sqrt{N/2}$. Consequently, the inequality

$$K_1\sqrt{N} \geq K_1 (2N)^a \Rightarrow N^{0.25} \geq K_1^{0.5} 2^{0.25} \tag{11}$$

which is true whenever $N \geq 2K_1^2$.

2) Analysis on the Computational Time Required: As opposed to the above analysis, where we considered the number of prototypes that are generated, we shall now analyze the computational *time* involved in individual and recursive computations involving *trad*. Consider a system which uses *trad* as its primitive PRS. We assume that the time required by *trad* depends on

the number of input sample data points, and that this relationship obeys the following equation:

$$K_2(N)^A, \quad A \geq 1 \tag{12}$$

where K_2 is a constant. Let the original number of points to be processed, (i.e., the number at the top-level) be N. If each set is split into *two* subsets prior to calling *trad*, the time after a single pass, and after two passes, can be evaluated using (12), and are given by the two equations (13) and (14), respectively, as follows:

$$K_2(N)^A \tag{13}$$

$$2K_2\left(\frac{N}{2}\right)^A. \tag{14}$$

Comparing the respective times involved we can see that:

$$K_2N^A \geq K_2 2\left(\frac{N}{2}\right)^A \Rightarrow N^A \geq 2\frac{N^A}{2^A} \tag{15}$$

which, in turn, leads to the inequality $1 \geq 2^{1-A}$, which is *always* true whenever $A \geq 1$. This implies that it is expedient to split the set of points into two subsets and then invoke *trad*, as opposed to allowing the latter to process the entire set of points.

IV. Experimental Results: Small and Medium-Sized Data Sets

A. Experimental Data

The ARP_PRS has been tested fairly extensively, and compared with many conventional PRS. This was first done by performing experiments on a number of small and "medium-sized" data sets, both real and artificial, as summarized in Table I.

The data set named "Non_normal (Medium-size)," which has been also employed in [11], [12], and [13] as a benchmark experimental data set, was generated from a mixture of four 8-dimensional Gaussian distributions as follows:

1) $p_1(x) = (1/2)N(\mu_{11}, I_8) + (1/2)N(\mu_{12}, I_8)$;
2) $p_2(x) = (1/2)N(\mu_{21}, I_8) + (1/2)N(\mu_{22}, I_8)$;

where $\mu_{11} = [0, 0, \ldots, 0], \mu_{12} = [6.58, 0, \ldots, 0], \mu_{21} = [3.29, 0, \ldots, 0]$, and $\mu_{22} = [9.87, 0, \ldots, 0]$. In these expressions, I_8 is the 8-dimensional *Identity* matrix.

The "Sonar" data set contains 208 vectors. Each sample vector, of two classes, has sixty attributes which are all continuous numerical values. The "Arrhythmia" data set contains 279 attributes, 206 of which are real-valued, and the rest are nominal. In our experiments, the nominal features were replaced by zeros. The aim of the pattern recognition exercise was to distinguish between the presence and absence of cardiac arrhythmia, and to classify the feature into one of the 16 groups. In our case, in the interest of simplicity, we merely attempted to classify the total instances into one of two categories, namely, "normal" and "abnormal."[9]

[9]In one sense, the results presented here are not as strong as they appear. Of course, the most conclusive proof of our scheme would have been if we were able to *inter*-classify all the 16 groups of data. But we opted to classify them into the "normal" and "abnormal" subgroups, so that each class would have a reasonably "large" number of data points.

TABLE I
Small and Medium-Sized Benchmark Data Sets Used in the Comparative Experiments. The Vectors Are Divided Into Two Subsets of Equal Size, and Used for Training and Validation, Alternatively

Data Set Names	Total Patterns	No. of Features	No. of Classes
Non_normal	1000 (500; 500)	8	2
Sonar	208 (104; 104)	60	2
Arrhythmia	452 (226; 226)	279	16

The data set "Non_normal (Medium-size)", was generated randomly with the normal distribution. However, the data sets "Sonar" and "Arrhythmia", which are real benchmark data sets, are cited from the UCI Machine Learning Repository [26].

In the above data sets, all of the vectors were normalized using their standard deviations. Also, for every class j, the data set for the class was randomly split into two subsets, $T_{j,t}$ and $T_{j,V}$, of equal size. One of them was used for choosing initial code-book vectors and training the classifiers as explained earlier, and the other subset was used in the validation (or testing) of the classifiers. The roles of these sets were later interchanged.

In this case, because the size of the sets was not excessively large, the recursive versions of CNN,[10] PNN, VQ, and SVM[11] were all invoked only for a depth of *two*.

B. Experimental Results

We report below the run-time characteristics of the ARP_PRS algorithm for the "Medium-sized" data sets. The experimental results of the CNN, PNN, VQ, and SVM methods implemented with the recursive mechanism, for the "Non_normal (Medium-size),", "Sonar," and "Arrhythmia" data sets are shown in Tables II, III, and IV, respectively.

The ARP_PRS can be compared with the nonrecursive versions using three criteria, namely, the processing CPU-time (CT), the classification accuracy rate (Acc), and the prototype reduction rate (Re). The reduction rates on the data sets $\mathrm{Re}(\cdot)$ were computed as

$$\mathrm{Re}(\cdot) = \frac{|\mathrm{Dataset}| - |\mathrm{Prototypes}|}{|\mathrm{Dataset}|} \times 100(\%)$$

where $|\cdot|$ is the cardinality of the corresponding set.

We report below a summary of the results obtained for the case when one subset was used for training and the second for

[10]It appears from the literature that the CNN method by Hart is not the *best* competitor for prototype selection in terms of both accuracy *and* effectiveness. We have chosen this method over the methods surveyed in [2], [3] because of its relative simplicity and ease of implementation. Of course, the intent is to demonstrate that any "primitive" method can be enhanced for "large" data sets by a recursive application. One referee suggested that an alternate candidate for comparison would be the Minimum Consistent Set algorithm by Dasarathy [29]. We are grateful for this pointer, and are currently investigating how the latter would perform if called recursively.

[11]As mentioned earlier, the SVM does reduce the set of prototypes, but not for the NN method. This means that the set of prototypes, which are also the *support vectors* obtained through the SVM method could be absolutely useless with 1-NN. All the other methods considered in this paper (including those surveyed) are supposed to select a reference set suitable for the 1-NN method. Thus, from this perspective, SVM belongs to a completely different group! Thus, although it is, in one sense, inappropriate for testing it as a basic PRS method, it has advantages if it is used recursively. This is the rationale for including it in our test suite. We are grateful to the anonymous referee who commented on this in his review.

TABLE II
COMPARISON OF THE NON-RECURSIVE AND THE *RECURSIVE* CNN, PNN, VQ, AND SVM METHODS FOR THE "NON_NORMAL (MEDIUM-SIZE)" DATA SET. HERE, DS, CT, NP, AND Acc ARE THE DATA SET SIZE (THE NUMBER OF SAMPLE VECTORS), THE PROCESSING CPU-TIME (IN SECONDS), THE NUMBER OF PROTOTYPES, AND THE CLASSIFICATION ACCURACY RATE (%), RESPECTIVELY. FOR EACH TECHNIQUE, THE RESULTS FOR THE NON-RECURSIVE VERSION ARE WRITTEN IN THE FIRST ROW, AND HIGHLIGHTED IN **BOLD** FONTS. ALSO, THE EXPERIMENTS WITH INDEX "2" REFER TO THE CASES WHEN THE TRAINING AND TESTING SETS ARE THE INTERCHANGED VERSIONS OF THE CORRESPONDING SETS USED IN THE EXPERIMENTS WITH INDEX "1"

Methods	*DS1*	*CT1*	*NP1*	*Acc1*	*DS2*	*CT2*	*NP2*	*Acc2*
CNN	**500**	**0.61**	**64**	**92.60**	**500**	**0.63**	**66**	**91.20**
	250	0.13	34		250	0.15	37	
	250	0.15	34		250	0.12	29	
	68	0.11	54	91.60	66	0.11	54	90.00
PNN	**500**	**81.74**	**56**	**92.40**	**500**	**208.55**	**380**	**91.60**
	250	7.58	31		250	7.55	34	
	250	7.54	29		250	7.54	26	
	60	0.22	46	89.20	60	0.26	50	88.80
VQ	**500**	**0.23**	**4**	**95.60**	**500**	**0.41**	**4**	**94.80**
	250	0.11	4		250	0.15	4	
	250	0.16	4		250	0.13	4	
	8	0.06	4	95.60	8	0.07	4	94.40
SVM	**500**	**0.86**	**62**	**95.40**	**500**	**0.97**	**57**	**94.40**
	250	0.24	32		250	0.25	35	
	250	0.23	30		250	0.24	24	
	62	0.07	60	95.60	59	0.06	57	94.40

TABLE III
EXPERIMENTAL RESULTS OF THE RECURSIVE CNN, PNN, VQ, AND SVM METHODS FOR THE "SONAR" DATA SET. HERE, DS, CT, NP, AND Acc ARE THE DATA SET SIZE (THE NUMBER OF SAMPLE VECTORS), THE PROCESSING CPU-TIME (IN SECONDS), THE NUMBER OF PROTOTYPES, AND THE CLASSIFICATION ACCURACY RATE (%), RESPECTIVELY. FOR EACH TECHNIQUE, THE RESULTS FOR THE NON-RECURSIVE VERSION ARE WRITTEN IN THE FIRST ROW, AND HIGHLIGHTED IN **BOLD** FONTS. ALSO, THE EXPERIMENTS WITH INDEX "2" REFER TO THE CASES WHEN THE TRAINING AND TESTING SETS ARE THE INTERCHANGED VERSIONS OF THE CORRESPONDING SETS USED IN THE EXPERIMENTS WITH INDEX "1"

Methods	*DS1*	*CT1*	*NP1*	*Acc1*	*DS2*	*CT2*	*NP2*	*Acc2*
CNN	**104**	**0.83**	**52**	**78.85**	**104**	**0.95**	**53**	**80.77**
	72	0.28	34		72	0.28	34	
	72	0.31	34		72	0.30	33	
	68	0.40	44	80.77	67	0.38	43	77.88
PNN	**104**	**4.32**	**34**	**82.69**	**104**	**4.75**	**33**	**82.69**
	72	1.45	22		72	1.43	20	
	72	1.49	20		72	1.35	17	
	42	0.31	16	72.12	37	0.20	19	78.85
VQ	**104**	**4.96**	**32**	**78.85**	**104**	**5.08**	**32**	**77.88**
	72	2.14	32		72	2.24	32	
	72	2.20	32		72	2.13	32	
	64	2.24	32	83.65	64	2.12	32	76.92
SVM	**104**	**0.48**	**53**	**82.69**	**104**	**0.75**	**59**	**85.58**
	72	0.12	34		72	0.10	32	
	72	0.24	33		72	0.15	34	
	67	0.56	50	83.65	66	0.44	66	86.54

testing. The results when the roles of the sets are interchanged are almost identical. From Tables II, III, and IV, we can see that the CT index (the processing CPU-time) of the pure CNN, PNN, VQ and SVM methods can be reduced significantly by merely employing the recursive philosophy. Indeed, this is achieved without sacrificing the accuracy, Acc, so much. It should be mentioned that in the case of the SVM, the Acc always increased when the recursive philosophy was employed.

TABLE IV
EXPERIMENTAL RESULTS OF THE RECURSIVE CNN, PNN, VQ AND SVM METHODS FOR THE "ARRHYTHMIA" DATA SET. HERE, DS, CT, NP AND Acc ARE THE DATA SET SIZE (THE NUMBER OF SAMPLE VECTORS), THE PROCESSING CPU-TIME (IN SECONDS), THE NUMBER OF PROTOTYPES, AND THE CLASSIFICATION ACCURACY RATE (%), RESPECTIVELY. AS IN THE PREVIOUS TABLES, FOR EACH TECHNIQUE, THE RESULTS FOR THE NON-RECURSIVE VERSION ARE WRITTEN IN THE FIRST ROW, AND HIGHLIGHTED IN **BOLD** FONTS. ALSO, THE EXPERIMENTS WITH INDEX "2" REFER TO THE CASES WHEN THE TRAINING AND TESTING SETS ARE THE INTERCHANGED VERSIONS OF THE CORRESPONDING SETS USED IN THE EXPERIMENTS WITH INDEX "1"

Methods	*DS1*	*CT1*	*NP1*	*Acc1*	*DS2*	*CT2*	*NP2*	*Acc2*
CNN	**226**	**3.69**	**32**	**95.58**	**226**	**2.98**	**28**	**97.35**
	113	1.36	26		113	0.71	11	
	113	0.81	12		113	0.98	17	
	38	0.49	26	96.90	28	0.23	13	96.90
PNN	**226**	**112.23**	**8**	**98.67**	**226**	**121.69**	**7**	**99.56**
	113	14.54	7		113	14.96	6	
	113	14.93	6		113	16.16	6	
	13	0.12	3	82.74	12	0.11	3	94.69
VQ	**226**	**25.91**	**64**	**99.12**	**226**	**25.05**	**64**	**98.67**
	113	0.22	64		113	0.20	64	
	113	0.20	64		113	0.23	64	
	128	0.14	64	97.35	128	0.14	64	99.12
SVM	**226**	**0.31**	**65**	**99.56**	**226**	**0.33**	**69**	**99.56**
	113	0.11	49		113	0.12	56	
	113	0.12	49		113	0.09	47	
	98	0.11	63	99.56	103	0.13	71	99.56

TABLE V
"LARGE-SIZED" DATA SETS USED FOR EXPERIMENTS. THE VECTORS ARE DIVIDED INTO TWO SETS OF EQUAL SIZE, AND USED FOR TRAINING AND VALIDATION, ALTERNATELY

Data Set Names	Total Patterns	No. of Features	No. of Classes
Non_normal	20,000 (10,000; 10,000)	8	2
Adult	33,330 (16,665; 16,665)	14	2

Consider the PNN method for the "Non_normal (Medium-size)" data set. If the 500 samples were processed nonrecursively, the time taken is 81.74 s, the size of the reduced set is 56, and the resulting classification accuracy is 92.4%. However, if the 500 samples are subdivided into two sets of 250 samples each, processing each subset involves only 7.58 and 7.54 s, leading to 31 and 29 reduced prototypes, respectively. When *these* 60 prototypes are, in turn, subjected to a pure PNN method, the number of prototypes reduced to 46 in just 0.22 s and yielded an accuracy of 89.2%. If we reckon that the recursive computations can be done in parallel, the time required is only *about one-tenth* of the time which the original PNN would take. Even if the computations were done serially, the advantage is marked.

To highlight the advantage, we consider another example. Consider the VQ method for the "Arrhythmia" data set. If the 226 samples were processed nonrecursively, the time taken is 25.91 s, the size of the reduced set is 64, and the resulting classification accuracy is 99.12%. However, if the 226 samples are subdivided into two sets of 113 samples each, processing each subset involves only 0.22 and 0.20 s, leading to 64 and 64 reduced prototypes respectively. When *these* 128 prototypes are in turn subjected to a pure VQ method, the number of prototypes reduced to 64 in just 0.14 s, and yielded an accuracy of 97.35%. Again, if the recursive computations are done in parallel, the time required is only *a small fraction* of the time which

TABLE VI
EXPERIMENTAL RESULTS OF THE RECURSIVE SVM FOR THE "NON_NORMAL (LARGE-SIZE)" DATA SET. HERE, $\mathrm{Depth}(i)$ MEANS THE DEPTH AT WHICH THE DATA SET IS SUB-DIVIDED INTO *EIGHT* SUBSETS. DS, CT, SV, AND Acc ARE THE DATA SET SIZE (THE NUMBER OF SAMPLE VECTORS), THE PROCESSING CPU-TIME (IN SECONDS), THE NUMBER OF SUPPORT VECTORS, AND THE CLASSIFICATION ACCURACY RATE (%), RESPECTIVELY. AS IN THE PREVIOUS TABLES, FOR EACH TECHNIQUE, THE RESULTS FOR THE NON-RECURSIVE VERSION ARE WRITTEN IN THE FIRST ROW, AND HIGHLIGHTED IN **BOLD** FONTS. ALSO, THE EXPERIMENTS WITH INDEX "2" REFER TO THE CASES WHEN THE TRAINING AND TESTING SETS ARE THE INTERCHANGED VERSIONS OF THE CORRESPONDING SETS USED IN THE EXPERIMENTS WITH INDEX "1"

Depth(i)	*DS1*	*CT1*	*SV1*	*Acc1*	*DS2*	*CT2*	*SV2*	*Acc2*
1	**10000**	**436.96**	**1273**	**94.87**	**10000**	**246.41**	**1286**	**94.83**
	1250	5.11	158		1250	6.20	160	
	1250	5.25	160		1250	5.58	167	
	1250	5.42	157		1250	5.08	182	
	1250	5.75	166		1250	4.97	139	
2	1250	5.48	157		1250	7.17	189	
	1250	5.03	144		1250	5.45	168	
	1250	5.65	185		1250	4.69	137	
	1250	5.05	154		1250	4.29	147	
	1281	15.82	1268	94.87	1289	16.43	1279	94.79

the original VQ would take.[12] Even if the computations were done serially, the advantage is undoubtedly, quite significant.

The reader should observe that the computational advantage gained is noticeable, and the accuracy lost is marginal. It should be mentioned, however, that a reduced accuracy is not typical. In the VQ and SVM methods, the resulting accuracy is identical or slightly higher than that which the original PRS yielded, and the computation time is significantly lower. These results are also typical for the other data sets, and can be gleaned from the tables.

V. EXPERIMENTAL RESULTS: LARGE-SIZED DATA SETS

A. Experimental Data

In order to further investigate the advantage gained by utilizing the proposed recursive PRS for more computationally intensive sets,[13] we conducted experiments on "large-sized" data sets, which we refer to as the "Non_normal (Large-size)" and "Adult" sets.[14] The information about these data sets is summarized in Table V. In this case, because the size of the sets was reasonably large, the recursive version of the SVM was invoked to a depth of *two*.

As in the case of the "Non_normal (Medium-size)" data set, the data set "Non_normal (Large-size)" was generated randomly with the normal distributions.

The "Adult" data set extracted from a census bureau database[15] was also obtained from the UCI Machine Learning Repository [26]. The aim of the pattern recognition task here is to separate people by incomes into two groups: in the first group the salary is more than 50 K dollars, and in the second group the salary is less than or equal to 50 K dollars. Each sample vector has fourteen attributes. Some of the attributes, such as the age, hours-per-week, etc., are continuous numerical values. The others, such as education, race, etc., are nominal symbols. In the experiments, the nominal attributes were replaced with numeric zeros.

Due to time considerations, we experimented only with the recursive SVM method for the large-sized data sets. The results for the other methods are currently being compiled. The experimental results for the "Non_normal (Large-size)" and "Adult" data sets are shown in Tables VI and VII, respectively.

Consider the results of Table VI. At the *depth* 1, the data set of 10 000 samples required a CPU time, CT, of 436.96 s, and yielded an Acc of 94.87% with the reduction, Rl, of 87.27%. However, if the 10 000 samples are subdivided into eight sets of 1250 each at the *depth* 2, processing each of these takes only the times given in the third column, whose average is 5.34 s, leading to 158, 160, 157, 166, 157, 144, 185, and 154 reduced prototypes, respectively. When *these* 1281 samples were subjected to a pure SVM method, the number of reduced samples fell to 1268 in 15.82 s, and yielded an accuracy of 94.87%. The recursive computations were done serially, and the time required was 58.56 s, which is only 13.4% of the time which the original SVM would take. If the computations were done in parallel, the advantage is even *more* marked.

From Table VII, we also observe the same characteristics as those seen in Table VI. At the *depth* 1, the data set of 16 665 samples required a computation time of 3 825.46 s, and gave an accuracy of 82.84% with a reduction of 61.31%. However, if the 16 665 samples were first subdivided into eight subsets of 2 083 each at the *depth* 2, processing each of these involves only the times given in the third column, whose average is 18.06 s, leading to 846, 819, 825, 821, 836, 807, 853, and 814 reduced prototypes, respectively. When *these* 6621 samples are, in turn, subjected to a pure SVM method, the number of reduced samples decreases to 6270 in 256.80 s, with an accuracy of 81.28%. The recursive computations were done serially, and so the time required was 401.26 s, which is only 10.5% of the time which the original SVM would take. Hopefully, these results demonstrate the power of our new recursive philosophy.

[12]The results of recursive VQ in Table III seem to show that for this special data set, recursive invocations are not beneficial. We are unable to find a reason for this.

[13]The design of PRSs for such data sets was, of course, the intention in proposing the recursive enhancements in the first place. Fortunately, we can again report a marked advantage.

[14]As mentioned earlier, these data sets are not "large" according to Huber's classification [30]. But, we would like to refer to them as being "large," because training sets of such cardinalities are not typically available in pattern recognition and classification problems.

[15][Online] Available: http://www.census.gov/ftp/pub/DES/www/welcome.html.

TABLE VII
EXPERIMENTAL RESULTS OF THE RECURSIVE SVM FOR THE "ADULT" DATA SET. HERE, $\mathrm{Depth}(i)$ MEANS THE DEPTH AT WHICH THE DATA SET IS SUB-DIVIDED INTO *EIGHT* SUBSETS. DS, CT, SV, AND Acc ARE THE DATA SET SIZE (THE NUMBER OF SAMPLE VECTORS), THE PROCESSING CPU-TIME (IN SECONDS), THE NUMBER OF SUPPORT VECTORS, AND THE CLASSIFICATION ACCURACY RATE (%), RESPECTIVELY. AS IN THE PREVIOUS TABLES, FOR EACH TECHNIQUE, THE RESULTS FOR THE NON-RECURSIVE VERSION ARE WRITTEN IN THE FIRST ROW, AND HIGHLIGHTED IN **BOLD** FONTS. ALSO, THE EXPERIMENTS WITH INDEX "2" REFER TO THE CASES WHEN THE TRAINING AND TESTING SETS ARE THE INTERCHANGED VERSIONS OF THE CORRESPONDING SETS USED IN THE EXPERIMENTS WITH INDEX "1"

Depth(i)	*DS1*	*CT1*	*SV1*	*Acc1*	*DS2*	*CT2*	*SV2*	*Acc2*
1	**16665**	**3825.46**	**6448**	**82.84**	**16665**	**2365.16**	**6218**	**82.63**
2	2084	19.19	843		2084	17.07	801	
	2083	17.82	819		2083	15.04	814	
	2083	15.94	825		2083	14.01	771	
	2083	20.17	821		2083	19.32	825	
	2083	15.94	836		2083	29.01	810	
	2083	21.75	807		2083	12.10	802	
	2083	15.26	853		2083	17.01	808	
	2083	18.39	814		2083	16.20	775	
	6621	256.80	6270	81.28	6406	219.46	6059	79.49

VI. CONCLUSION

Conventional PRSs can require excessive computation because they usually process all the data, even though data in the interior of the Voronoi spaces is not useful for classifier design. In this paper we have proposed a mechanism whereby the data are recursively subdivided into smaller subsets, and the data points which are ineffective in the classification are eliminated for subsequent calls of the PRS. Our recursive PRS processes the smaller subsets of data that effectively sample the entire space to yield subsets of prototypes. These prototypes are, in turn, gathered and processed by the PRS to yield more refined prototypes.

The proposed method was tested on both artificial and real-life benchmark data sets, and compared with a few representative conventional methods. The experimental results for small, medium-sized and "large" data sets demonstrate that the proposed algorithm can improve the speed of the CNN, PNN, VQ, and SVM methods by *an order of magnitude*, while yielding almost the same classification accuracy and reduction rate, especially if recursion is resorted to many times. Apparently, the advantage of the recursive processes increases with the size of the data.

ACKNOWLEDGMENT

The authors would like to thank the anonymous referees for the trouble they took to go through the original version of this paper in great detail. The comments that both referees made were extremely helpful, and there is no doubt that they increased the quality of this revised version considerably. In particular, the authors are extremely grateful to one of the referees who took the pains to send back a thoroughly annotated version of the original manuscript.

REFERENCES

[1] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 4–37, Jan. 2000.

[2] B. V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1991.

[3] J. C. Bezdek and L. I. Kuncheva, "Nearest prototype classifier designs: An experimental study," *Int. J. Intell. Syst.*, vol. 16, no. 12, pp. 1445–1473, 2001.

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

[5] G. W. Gates, "The reduced nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 431–433, May 1972.

[6] C. L. Chang, "Finding prototypes for nearest neighbor classifiers," *IEEE Trans. Computers*, vol. C-23, no. 11, pp. 1179–1184, Nov. 1974.

[7] G. L. Hitter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, "An algorithm for a selective nearest neighbor rule," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 665–669, Nov. 1975.

[8] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-6, pp. 769–772, Nov. 1976.

[9] P. A. Devijver and J. Kittler, "On the edited nearest neighbor rule," in *Proc. 5th Int. Conf. Pattern Recognition*, Dec. 1980, pp. 72–80.

[10] K. Fukunaga and J. M. Mantock, "Nonparametric data reduction," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 115–118, Jan. 1984.

[11] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. San Diego, CA: Academic, 1990.

[12] Q. Xie, C. A. Laszlo, and R. K. Ward, "Vector quantization techniques for nonparametric classifier design," *IEEE Trans. Pattern Anal. Machine Intell*, vol. 15, pp. 1326–1330, Dec. 1993.

[13] Y. Hamamoto, S. Uchimura, and S. Tomita, "A bootstrap technique for nearest neighbor classifier design," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 73–79, Jan. 1997.

[14] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[15] S.-W. Kim and B. J. Oommen, "Enhancing prototype reduction schemes with LVQ3-type algorithms," *Pattern Recognit.*, vol. 36, no. 5, pp. 1083–1093, 2003.

[16] ——, "A brief taxonomy and ranking of creative prototype reduction schemes," *Pattern Anal. Applicat. J.*, pp. 232–244, 2003.

[17] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. COM-28, pp. 84–95, Jan. 1980.

[18] Laboratory of Computer and Information Science (CIS) [Online]. Available: http://cochlea.hut.fi/research/som_lvq_pak.shtml

[19] T. Kohonen, *Self-Oganizing Maps*, Berlin, Germany: Springer-Verlag, 1995.

[20] N. Aras, B. J. Oommen, and I. K. Altinel, "The Kohonen network incorporating explicit statistics and its application to the travelling salesman problem," *Neural Networks*, pp. 1273–1284, Dec. 1999.

[21] N. Aras, I. K. Altinel, and B. J. Oommen, "A Kohonen-like decomposition method for the traveling salesman problem—KNIES_JDECOMPOSE," in *Proc. 14th Eur. Conf. Artificial Intelligence*, Berlin, Germany, Aug. 2000, pp. 261–265.

[22] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[23] J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, "Multiple-prototype classifier design," *IEEE Trans. Syst., Man, Cybern. C*, vol. 28, pp. 67–79, Feb. 1998.

[24] L. I. Kuncheva and J. C. Bezdek, "Nearest prototype classification: Clustering, genetic algorithms or random search?," *IEEE Trans. Syst., Man, Cybern. C*, vol. 28, pp. 160–164, Feb. 1998.

[25] Kernel Machines [Online]. Available: http://svm.first.gmd.de/

[26] UCI Machine Learning Repository [Online]. Available: http://www.ics.uci.edu/mlearn/MLRepository.html
[27] P. S. Bradley and U. M. Fayyad, "Refining initial points for k-means clustering," in *Proc. 15th Int. Conf. Machine Learning*, Madison, WI, July 1998.
[28] J. C. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," Tech. Rep. Microsoft Research, MSR-TR-98-14, Apr. 1998.
[29] B. V. Dasarathy, "Minimal consistent set (MCS) identification for optimal nearest neighbor decision systems design," *IEEE Trans. Systems, Man, Cybern.*, vol. 24, pp. 511–517, Mar. 1994.
[30] P. Huber, *Massive Data Sets Workshop: The Morning After, Massive Data Sets*. Washington, DC: Nat. Acad. Press, 1996, pp. 169–184.
[31] P. Bradley, U. Fayyad, and C. Reina, "Scaling clustering algorithms to large databases," in *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining*. Menlo Park, CA, 1998, pp. 9–15.
[32] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD Int. Conf. Management of Data*. Nashville, TN, 1996, pp. 103–114.
[33] F. Farnstrom, J. Lewis, and C. Elkan, "Scalability for clustering algorithms revisited," in *SIGKKD Explorations*. Nashville, TN: ACM Press, 2000, no. 2, pp. 1–7.
[34] P. Domingos and G. Hulten, "A general method for scaling up machine learning algorithms and its application to clustering," in *Proc. 18th Int. Conf. Machine Learning*, Williamstown, MA, July 2001, pp. 106–113.

Sang-Woon Kim (S'85–M'88–SM'03) received the B.E. degree from Hankook Aviation University, Korea, in 1978 and the M.E. and Ph.D. degrees from Yonsei University, Korea, in 1980 and 1988, respectively, both in electronic engineering.

In 1989, he joined the Department of Computer Science and Engineering, Myongji University, Korea, where he is currently a Full Professor. From 1992 to 1993, he was a Visiting Scientist at the Graduate School of Electronics and Information Engineering, Hokkaido University, Hokkaido, Japan. From 2001 to 2002, he was a Visiting Professor at the School of Computer Science, Carleton University, Ottawa, ON, Canada. His research interests include pattern recognition, machine learning, and avatar communications in virtual worlds. He is the author or co-author of 20 papers and ten books.

Dr. Kim is a member of the IEICE and the IEEK.

B. John Oommen (S'79–M'83–SM'88–F'03) was born in Coonoor, India, in 1953. He received the B.Tech. degree from the Indian Institute of Technology, Madras, in 1975, the M.E. degree from the Indian Institute of Science, Bangalore, in 1977, and the M.S. and Ph.D. degrees from Purdue University, West Lafayettte, IN, in 1979 and 1982, respectively.

In 1981, he joined the School of Computer Science at Carleton University, Ottawa, ON, Canada, where he is currently a Full Professor. His research interests include automata learning, adaptive data structures, statistical and syntactic pattern recognition, stochastic algorithms, and partitioning algorithms. He is the author of more than 200 refereed journal and conference publications and is a *Fellow of the IEEE*.

Dr. Oommen is on the Editorial Board of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS and *Pattern Recognition*.