On Using Prototype Reduction Schemes to Optimize Kernel-Based Nonlinear Subspace Methods[*]

Sang-Woon Kim[†] and B. John Oommen[‡]

Abstract

The subspace method of pattern recognition is a classification technique in which pattern classes are specified in terms of linear subspaces spanned by their respective class-based basis vectors. To overcome the limitations of the linear methods, Kernel based Nonlinear Subspace (KNS) methods have been recently proposed in the literature. In KNS, the kernel Principal Component Analysis (kPCA) has been employed to get principal components, not in an input space, but in a high-dimensional space, where the components of the space are non-linearly related to the input variables. The length of projections onto the basis vectors in the kPCA are computed using a kernel matrix K, whose dimension is equivalent to the number of sample data points. Clearly this is problematic, especially, for large data sets.

In this paper, we suggest a computationally superior mechanism to solve the problem. Rather than define the matrix K with the whole data set and compute the principal components, we propose that the data be reduced into a smaller representative subset using a Prototype Reduction Scheme (PRS). Since a PRS has the capability of extracting vectors that satisfactorily represent the global distribution structure, we demonstrate that data points which are ineffective in the classification can be eliminated to obtain a reduced kernel

[*]The work of the first author was done while visiting at Carleton University, Ottawa, Canada. The first author was partially supported by KOSEF, the Korea Science and Engineering Foundation, and the second author was partially supported by NSERC, the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at AI'02, the 2002 AI Conference in December 2002 in Canberra, Australia.

[†]*Member IEEE.* This author can be contacted at: Dept. of Computer Science and Engineering, Myongji University, Yongin, 449-728 Korea. e-mail address : kimsw@mju.ac.kr. This work was done while visiting with the School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6.

[‡]*Fellow of the IEEE.* This author can be contacted at: School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6. e-mail address : oommen@scs.carleton.ca.

matrix, K, without degrading the performance. Our experimental results demonstrate that the proposed mechanism *dramatically* reduces the computation time without sacrificing the classification accuracy for samples involving real-life data sets as well as artificial data sets. The results especially demonstrate the computational advantage for *large* data sets, such as those involved in data mining and text categorization applications.

Keywords : *Principal Component Analysis (PCA), Linear Subspace Method (LSM), Kernel Principal Component Analysis (kPCA), Kernel based Nonlinear Subspace (KNS) Method, Prototype Reduction Schemes (PRS).*

1 Introduction

The subspace method of pattern recognition is a technique in which the pattern classes are not primarily defined as bounded regions or zones in a feature space, but rather given in terms of linear subspaces defined by the basis vectors, one for each class [1]. The length of a vector projected onto a class subspace, or the distance of the vector from the class subspace, is a measure of similarity or degree of membership for that particular class, and serves as the discriminant function of these methods. More specifically, the following steps are necessary for the methods: First, we compute the covariance matrix or scatter matrix. Then, we compute its eigenvectors and normalize them by the principal components analysis or the Karhuen-Loève (KL) expansion technique. Finally, we compute the projections (or distances) of a test pattern vector onto (or from) the class subspaces spanned by the subset of principal eigenvectors.

Various subspace classifiers, such as CLAFIC (CLAss Feature Compression) [1], the learning subspace method [1], and the mutual subspace method [2], have been proposed in the literature. All of the subspace classifiers have employed the Principal Components Analysis (PCA) to compute the basis vectors by which the class subspaces are spanned. The basic idea of the PCA is to represent N-dimensional data by a set of orthogonal directions - capturing most of the variance in the data. In practice, we describe the data with reduced dimensionality by extracting a few meaningful principal components, implying that we retain the most significant aspects of the structure in the data. Since the PCA is a linear algorithm, it is clearly beyond its capabilities to extract nonlinear structures from the data. This constitutes a primary limitation of linear subspace classifiers.

To overcome the above limitation, a kernel Principal Components Analysis (kPCA) has been proposed in [3] and [4]. The kPCA provides an elegant way of dealing with nonlinear problems in an input space R^N by mapping them to linear ones in a feature space, F. That is, a dot product in space R^N corresponds to mapping the data into a possibly high-dimensional dot product space F by a nonlinear map $\Phi : R^N \rightarrow F$, and taking the dot product in the latter space [3].

Recently, using the kPCA, kernel-based subspace methods, such as the Subspace method in Hilbert Space (SHS) [5], Kernel based Nonlinear Subspace (KNS) method [6, 7], Kernel Based Learning (KBL) algorithms [8], and Kernel Mutual Subspace (KMS) method [9], have been proposed. All of them utilize the *kernel trick* to obtain the kernel PCA components by solving a similar linear eigenvalue problem as explained above for the linear PCA. The only difference is that the size of the problem is decided by the *number* of data points, and not by the dimension. The above studies report that the performance of nonlinear subspace methods is better than that of linear methods possessing the same number of principal components.

On the other hand, in kPCA, to map the data set $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$, (where each $\boldsymbol{x}_i \in R^N$) into a feature space F, we have to define an $n \times n$ matrix, K, the so-called kernel matrix. This corresponds to the $N \times N$ covariance matrix of the linear PCA. Observe that the dimension of K is equivalent to the number of data points, n. The situation is, unfortunately, ironic. To get a good classifier with good training estimates, one typically requires a large training set. But, when a large number of observations are available, the kernel matrix has a large dimensionality. Furthermore, the same problem with a small training set yields a poorer classifier!

To solve the computational problem, a number of methods, such as the techniques proposed by Achlioptas and his co-authors [10, 11] (detailed presently), the power method with deflation [4], the method of estimating K with a subset of the data [4], the Sparse Greedy matrix Approximation (SGA) [12], the Nystom method [13], and the sparse kernel PCA method based on the probabilistic feature-space PCA concept [14], have been proposed. In [4], a method of estimating the matrix K from a subset of $n'(< n)$ data points, while still extracting principal components from all the n data points, was considered. Also, in [12], an approximation technique to construct a compressed kernel matrix K' such that the norm of the residual matrix $K - K'$ is minimized, was proposed. Although it is clearly impossible to review all of these papers in this present work, we mention that we shall use this latter method as a basis for comparison with our newly proposed schemes.

Pioneering to the area of reducing the complexity of KNS are the works of Achlioptas [11] and his co-authors. Although we shall go through them in greater detail later, we mention that the techniques proposed by them can be classified into two main categories. The first category includes the strategy of artificially introducing sparseness into the kernel matrix, which, in turn, is achieved by *physically* setting some randomly chosen values to *zero*. The other alternative, as suggested in [10], proposes the elimination of the underlying data points themselves. This is the strategy we advocate. However, rather than eliminate the "unnecessary" underlying data points in a randomized manner, we propose that this be done in a systematic manner, namely in a way by which the distribution of the points is maintained, albeit, approximately.

To be more specific, we propose to solve the problem by reducing the size of the design set without sacrificing the performance, where the latter is achieved by using a Prototype Reduction Scheme (PRS). The PRS is a way of reducing the number of training vectors while simultaneously insisting that the classifiers built on the reduced design set perform as well, or nearly as well, as the classifiers built on the original design set.

The PRS has been explored for various purposes, and has resulted in the development of many algorithms [15, 16]. One of the first of its kind, was a method that led to a smaller prototype set, the Condensed Nearest Neighbor (CNN) rule [17]. Since the development of the CNN, other methods have been proposed successively, such as the Prototypes for Nearest Neighbor (PNN) classifiers [18] (including a modified Chang's method proposed by Bezdek [19]), the Vector Quantization (VQ) technique [20] etc[1].

Recently, Support Vector Machines (SVM) [21, 22] have proven to possess the capability of extracting vectors that support the boundary between the two classes. Thus, they have been used satisfactorily to represent the global distribution structure. As opposed to this, the Learning Vector Quantization (LVQ) [23] has a capability of adjusting the prototypes (code-book vectors) so that the decision boundaries for every pair of neighboring classes satisfy the approximation of the class distribution. Thus, apart from the above PRS methods, the SVM can also be used as a means of selecting prototype vectors. Observe that these new vectors can be subsequently

[1]Bezdek *et al* [15], who have composed an excellent survey of the field, report that there are "zillions!" of methods for finding prototypes (see page 1459 of [15]).

adjusted by means of an LVQ3-type algorithm. Based on this idea, a new prototype reduction method (referred to here as HYB) of hybridizing the SVM and the LVQ3 was introduced in [24].

In this paper we propose to utilize PRS algorithms to devise a method of reducing the computational burden of the kernel based nonlinear subspace methods. The paper is organized as follows: After providing a brief introduction to linear and kernel-based nonlinear subspace methods, in Section 3 we show how large data sets of training patterns can be reduced into smaller prototype sets by invoking various PRS. The inclusion of these in kPCA follows in Section 4, followed by a section (Section 5) highlighting the theoretical basis for our work. Experiments and discussions for artificial and real-life benchmark data sets are provided in Sections 6 and 7 respectively. Section 8 concludes the paper.

1.1 Contributions of the Paper

PRS have typically been used as methods by which the prototypes have been selected or created in the design of pattern recognition systems. In this paper, we, first of all, show that PRS can also be used as a *tool* to achieve an intermediate goal, namely, to minimize the number of samples that are *subsequently* used in a subspace based pattern recognition system, which, *in turn* is used to design the classifier. This, in itself, is novel to the field of the applications of PRS.

The second contribution of this paper is the method by which we have reduced the computational burden of a kernel based nonlinear subspace method. The latter methods, though innovative, are computationally expensive because of the associated matrix (eigenvalue/eigenvector) operations required in obtaining the basis vectors spanning the subspaces. The methods that have been used to minimize this burden are described in more detail in [12, 13, 14]. These typically utilize particular features (for example, the sparseness) of the $n \times n$ kernel matrix, where n is the number of sample points. In this paper, we show that the burden can be reduced significantly by not considering the kernel matrix *at all*. Instead, we rather define a reduced-kernel matrix by first preprocessing the training points with a PRS scheme. Further, the PRS scheme does not necessarily have to *select* a reduced set of data points. Indeed, it can rather *create* a reduced set of prototypes from which, in turn, the reduced-kernel matrix is determined. All of these concepts are novel to the field of designing kernel-based nonlinear subspace methods.

2 Kernel based Nonlinear Subspace (KNS) Method

In this section, we provide a *brief* overview to the kernel based nonlinear subspace method[2]. This is introduced by an explanation of the linear subspace methods.

2.1 Linear Subspace Methods

It is a well known fact that in most cases, a significant portion of the information content of a feature vector is concentrated on a relatively small subset of features, which can serve as principal axes. The subspace methods of pattern recognition are based on the principle of linear orthogonal expansions, where pattern classes are given in terms of linear subspaces defined by their respective basis vectors. Therefore, the output of the subspace classifier is not the class membership or the bounded region, but a transformed description of the data item given by its vector projection on the class subspace.

The subspace methods handle the feature extraction and classification in a single unified step. Also, the classification is done based on a small number of scalar products. Thus subspace classifiers are both fast and efficient.

A subspace of R^N for an N-dimensional pattern classification problem is spanned by a set of linearly independent vectors (a basis), denoted by L, of dimension p. Assuming that $L_i, i = 1, \cdots, C$, is given in terms of a set of orthogonal basis vectors, $\boldsymbol{u}$= $(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_{p_i})$, with $\boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij}$, the Kroneker-*delta* function, the length of the projection of a new pattern $\boldsymbol{x}$ onto L_i is

$$\|P_i\|^2 = \sum_{k=1}^{p_i} \left(\boldsymbol{x}^T \boldsymbol{u}_k\right)^2 = \|\boldsymbol{x}^T \boldsymbol{u}\|^2. \tag{1}$$

To compute the basis vectors, the Principal Components Analysis (PCA) or Karhuen-Loève (KL) expansion technique is employed. For each class, j, the scatter matrix of the N-dimensional data set, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n\}$, where each $\boldsymbol{x}_i \in R^N$, is estimated by $\Sigma_j = \frac{1}{n}\sum_{i=1}^n \boldsymbol{x}_i \boldsymbol{x}_i^T, \boldsymbol{x}_i \in \omega_j$. Subsequently, the first $p_j (\leq N)$ eigenvectors, $\left(\boldsymbol{e}_1, \cdots, \boldsymbol{e}_{p_j}\right)$, of the matrix Σ_j, in the order of decreasing eigenvalues, $\lambda_1 \geq \cdots \geq \lambda_{p_j}$, are used as the basis vectors.

[2]This section is included only for the sake of the reader who is new to the field. It was recommended by the Referees.

An essential concept in the subspace methods is the projection of a vector $\boldsymbol{x}$ on each class subspace L_i. Using this, the decision rule classifies each $\boldsymbol{x}$ into the class on whose class-subspace it has the longest projection. This is achieved as follows:

$$\forall j \neq i, \ \|P_i\|^2 > \|P_j\|^2 \Rightarrow \boldsymbol{x} \in \omega_i. \tag{2}$$

Since the eigenvectors are computed linearly by the PCA, classifiers based on the subspaces spanned by the basis vector matrix, $\boldsymbol{u}$, are very effective for linear problems, but not good for non-linear distributions of features. This has naturally motivated various developments of nonlinear PCA's, including the kernel PCA.

2.2 Kernel Principal Components Analysis (kPCA)

In the previous section, the eigenvalue problem was solved in the *input* space R^N. To extend this, we repeat the same computation in another space F, which is referred to as the feature space. The space F is related to the input space by a possibly nonlinear map

$$\Phi : R^N \rightarrow F, \tag{3}$$

which takes a vector $\boldsymbol{x}$ into $\Phi(\boldsymbol{x})$, where $\Phi(\boldsymbol{x})$ could have an arbitrarily large, (and possibly infinite), dimensionality. For each class, j, the Φ-based scatter matrix is estimated by $\Sigma'_j = \frac{1}{n}\sum_{i=1}^{n} \Phi(\boldsymbol{x}_i)\Phi(\boldsymbol{x}_i)^T, \Phi(\boldsymbol{x}_i) \in \omega_j$. The first $p'_j(\leq n)$ eigenvectors, $\boldsymbol{U} = \Big(\boldsymbol{U}_1, \cdots, \boldsymbol{U}_{p'_j}\Big)$, of the matrix Σ'_j, in the order of decreasing eigenvalues $\lambda'_1 \geq \cdots \geq \lambda'_{p'_j}$, are computed by

$$\lambda' \boldsymbol{U} = \Sigma'_j \boldsymbol{U} = \frac{1}{n}\sum_{i=1}^{n} \left(\Phi(\boldsymbol{x}_i) \cdot \boldsymbol{U}\right) \Phi(\boldsymbol{x}_i). \tag{4}$$

Here, if we do not know the function Φ explicitly, we can not compute the basis vectors directly from (4). Fortunately, however, we can compute projections of an arbitrary pattern onto the directions of the basis vectors. Since all eigenvectors with nonzero eigenvalues must be in the span of the mapped data, $\Phi(\boldsymbol{x}_i), i = 1, \cdots, n$,

$$\boldsymbol{U} = \sum_{i=1}^{n} \boldsymbol{\alpha}_i \Phi(\boldsymbol{x}_i). \tag{5}$$

By multiplying with $\Phi(\boldsymbol{x}_i)$, (4) becomes:

$$\lambda'(\Phi(\boldsymbol{x}_i) \cdot \boldsymbol{U}) = (\Phi(\boldsymbol{x}_i) \cdot \Sigma_j' \boldsymbol{U}), i = 1, \cdots, n. \tag{6}$$

Here, to compute the expansion coefficients, $\boldsymbol{\alpha}_i, i = 1, \cdots, n$, we define an $n \times n$ matrix, K, whose elements are

$$K_{ij} = (\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)) = k(\boldsymbol{x}_i, \boldsymbol{x}_j), 1 \le i, j \le n. \tag{7}$$

Using this formulation, an eigenvalue problem for the $\boldsymbol{\alpha}_i$ is obtained as follows:

$$\lambda \boldsymbol{\alpha} = K \boldsymbol{\alpha}, \tag{8}$$

where the $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1 \cdots \boldsymbol{\alpha}_{p'})^T$ denotes the complete set of the first $p'(\le n)$ eigenvectors of K which correspond to the largest p' non-zero eigenvalues, $\lambda_1 \ge \cdots \ge \lambda_{p'}$.

The solutions of $\boldsymbol{\alpha}$ further need to be normalized by imposing $(\boldsymbol{U}_k \cdot \boldsymbol{U}_k) = 1, k = 1, \cdots, p'$ in F. From (5), (7), and (8), we observe that $(\boldsymbol{U}_k \cdot \boldsymbol{U}_k)$=$\sum_{i,j=1}^{n} \alpha_{ki}\alpha_{kj} \left(\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j)\right)$ =$\lambda_k \left(\boldsymbol{\alpha}_k \cdot \boldsymbol{\alpha}_k\right)$. After we perform a normalization, we see that $(\boldsymbol{U}_i \cdot \boldsymbol{U}_j)$ is δ_{ij}, the Kroneker-*delta* function, since :

$$\begin{aligned} (\boldsymbol{U}_k \cdot \boldsymbol{U}_k) &= \lambda_k \left(\boldsymbol{\alpha}_k \cdot \boldsymbol{\alpha}_k\right) = \lambda_k \left(\frac{\boldsymbol{U}_k}{\sqrt{\lambda_k}} \cdot \frac{\boldsymbol{U}_k}{\sqrt{\lambda_k}} \right) = 1, & (9) \\ (\boldsymbol{U}_i \cdot \boldsymbol{U}_j) &= 0, \ (whenever \ i \ \neq \ j). & (10) \end{aligned}$$

Thus, to extract principle components of a new pattern $\boldsymbol{z}$ in F, we simply project the mapped pattern $\Phi(\boldsymbol{z})$ onto a direction of $\boldsymbol{U}_j$ as follows:

$$(\boldsymbol{U}_j \cdot \Phi(\boldsymbol{z})) = \sum_{i=1}^{n} \alpha_{ji} (\Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{z})) = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{n} U_{ji} k(\boldsymbol{x}_i, \boldsymbol{z}). \tag{11}$$

Although it is nearly impossible to compute the basis vectors corresponding to the eigenvectors of (4), it is possible from (11), to compute the projections onto the directions of basis vectors. The dot product for the principal components extraction can be *implicitly* computed in F, without explicitly using the mapping Φ. Such a computational methodology is called the *"kernel trick"*, using which, nonlinear subspace classifiers can be designed.

2.3 Nonlinear Subspace Methods

The Kernel-based Nonlinear Subspace (KNS) method [6, 7] is a subspace classifier, where the kPCA is employed as the specific nonlinear feature extractor. Given a set of n N-dimensional data samples $X = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n)^T \in \omega_i$ and a kernel function $k(\cdot, \cdot)$, the column matrix of the orthogonal eigenvectors, $\boldsymbol{U}= \left(\boldsymbol{U}_1, \cdots, \boldsymbol{U}_{p'_i}\right)$, and the diagonal matrix of the eigenvalues, $\boldsymbol{\Lambda}$, can be computed from the kernel matrix $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j), 1 \leq i, j \leq n$. Then, from (9), the length of the projection of a new test pattern $\boldsymbol{z}$ onto the principal directions of $\boldsymbol{U}$ in F is

$$\|P'_i\|^2 = \sum_{j=1}^{p'_i} \sum_{i=1}^{n} (\alpha_{ji} k(\boldsymbol{x}_i, \boldsymbol{z}))^2 = \| \frac{1}{\sqrt{\boldsymbol{\Lambda}}} \boldsymbol{U}^T k(X, \boldsymbol{z}) \|^2. \tag{12}$$

Thus, the analogous decision rule as that of linear subspace classifiers, classifies each $\boldsymbol{z}$ into the class on whose "class subspace" it has the longest projection as follows:

$$\forall j \neq i, \ \|P'_i\|^2 > \|P'_j\|^2 \Rightarrow \boldsymbol{z} \in \omega_i. \tag{13}$$

In KNS, the subspace dimensions, $p'_i, i = 1, \cdots, C$, have a strong influence on the performance of subspace classifiers. In order to get a high classification accuracy, we have to select a large dimension. However, designing with dimensions which are too large leads to low performances due to the overlapping of the resultant subspaces. Also, since the speed of computation for the discriminant function of subspace classifiers is inversely proportional to the dimension, the latter should be kept small.

Various dimension selection methods have been reported in the literature [1, 25]. The simplest approach is to select a global dimension to be used for all the classes. A more popular method is that of selecting different dimensions for the various classes based on the cumulative proportion, α, which is defined as follow:

$$\alpha(p'_i) = \sum_{i=1}^{p'_i} \lambda_i \Bigg/ \sum_{i=1}^{n} \lambda_i \ . \tag{14}$$

In this method, the p'_i of each class for the data sets is determined by considering the cumulative proportion, $\alpha(p'_i)$. For each class, the kernel matrix K is computed using the available training samples for that class. The eigenvectors and eigenvalues are computed, and the cumulative sum of the eigenvalues, normalized by the trace of K, is compared to a fixed number.

The overall procedure for kernel based nonlinear subspace classifiers is summarized as follows:

1. For every class, j, compute the kernel matrix, K, using the given training data set $\{\boldsymbol{x}_i\}_{i=1}^{n}$ and the kernel function $k(\cdot, \cdot)$;

2. For every class, j, compute the normalized eigenvectors of K in F, and select the subspace dimension, p'_j;

3. For all the test patterns, we compute the projections of the pattern onto the eigenvectors, and classify it by the decision rule of (13).

In the above, the computation of principal component projections for a pattern requires evaluation of the kernel function with respect to *all* the training patterns. This is unfortunate, because, in practice, the computation is excessive for large data sets. We overcome this limitation by reducing the training set using a PRS, and the theoretical basis for such a reduction will be argued later.

3 Prototype Reduction Schemes (PRS)

Various data reduction methods have been proposed in the literature - two excellent surveys are found in [15, 16]. To put the results available in the field in the right context, we mention, in detail, the contents of the former. The survey of [15] 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.

The claim of [15], based on the experimental results obtained using eleven distinct methods, is very easily stated. The authors of [15] claim that there seems to be no clear scheme that is *uniformly* superior to all the other PRS. Indeed, different methods were found to be superior for different data sets. However, the experiments showed that the *creative* methods are superior to the selective methods, but are, typically, computationally more difficult to determine. Also, the experimental results revealed that the Pre-supervised methods are better than the Post-supervised ones.

The most pertinent ones are mentioned here[3] by two groups: the *selecting* and the *creating* methods. The CNN [17], which is one of first methods proposed, is chosen as a selecting method. On the other hand, the PNN, the VQ (or SOM), and the HYB are considered to fall within the family of prototype-*creating* algorithms.

The algorithm of finding PNNs [18], can be summarized as follows: Given a training set T, the algorithm starts with every point in T as a prototype. Initially, set A is empty and 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.

Bezdek and his co-authors, proposed a modification of the PNN in [19]. Based on the results obtained from experiments conducted on the Iris data set, the authors of [19] asserted that their modified form of the PNN yielded the best consistent reduced set for designing multiple-prototype classifiers.

The concept of VQ or SOM can be perceived as one of the prototype *creation* approaches. Rather than represent the entire data in a compressed form using only the estimates, VQ and

[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 philosophy of "pre-processing" a KNS is applicable by using any of the PRS methods surveyed in [15, 16] and in the existing literature.

SOM opt to represent the data in the actual feature space by creating codebook vectors which are migrated so as to be optimally placed in this space. Details of the VQ and SOM can be found in [20, 23, 26, 27].

In designing NN classifiers, prototypes near the boundary play more important roles than those which are more interior in the feature space. In creating or selecting the prototypes, therefore, the points near the boundary have to be considered to be more significant, and the created prototypes need to be moved or adjusted towards the classification boundary so as to yield higher performance. The HYB hybridized approach that we proposed in [24] is based on this philosophy, namely that of invoking *selecting/creating*, and *adjusting* phases. First, a reduced set of initial prototypes or code-book vectors is chosen by any of the previously known methods, and then their optimal positions are learned with an LVQ3-type algorithm, thus, minimizing the average classification error. The more crucial issue is that of determining the parameters of the LVQ3-type algorithm. This is accomplished in [24] by partitioning the given data sets into two subsets, which are, in turn, utilized to optimize the corresponding LVQ3 parameters. The details of the scheme are omitted here and can be found in [24]. A brief taxonomy and a ranking of creative PRS can also be found in [28].

4 Optimizing the KNS : Motivation and Proposed Solution

The fundamental problem we encounter is that of computing the solution to the kernel subspace classifier from the set of training samples. This, in turn, involves four essential phases, namely that of computing the kernel matrix, computing *its* eigenvalues and eigenvectors, extracting the principal components of the kernel matrix from among these eigenvectors, and finally, projecting the samples to be processed onto the reduced basis vectors. We observe, first of all, that all of these phases depend on the size of the data set. In particular, the most time consuming phase involves computing the eigenvalues and eigenvectors of the kernel matrix.

There is an underlying irony in the method, in itself. For a good and effective training phase, we, typically, desire a large data set. But if the training set is large, the dimensionality of the kernel matrix is correspondingly large, making the computations extremely expensive. On the other hand, a smaller training set makes the learning less accurate, although computations are correspondingly less.

There are a few ways by which the computational burden of the kernel subspace method can be reduced. Most of the reported schemes [12, 13, 14] resort to using the specific properties of the underlying kernel matrix, for example, its sparseness. Our technique is different.

The method we propose is by reducing the size of the training set. However, we do this, by not significantly reducing the accuracy of the resultant classifier. This is achieved by resorting to a PRS. The theoretical basis for this will be explained in a subsequent section.

The question now is essentially one of determining which of the training points we should retain. Rather than deciding to discard or retain the training points, we permit the user the choice of either *selecting* some of the training samples using methods such as the CNN, or *creating* a smaller set of samples using the methods such as those advocated in the PNN, VQ, and HYB. This reduced set effectively represents the new "training" set. Additionally, we also permit the user to migrate the resultant set by an LVQ3-type method to further enhance the quality of the reduced samples.

The PRS serves as a preprocessor to the n N-dimensional training samples to yield a subset of n' potentially new points, where $n' << n$. The "kernel" is now computed using this reduced set of points to yield the so-called *reduced-kernel* matrix. The eigenvalues and eigenvectors of *this* matrix are now computed, and the principal components of the kernel matrix are extracted from among *these* eigenvectors of smaller dimension. Notice now that the samples to be tested are projected onto the reduced basis directions represented by *these* vectors.

To investigate the computational advantage gained by resorting to such a PRS preprocessing phase, we observe, first of all, that the time used in determining the reduced prototypes is *fractional* compared to the time required for the expensive matrix-related operations. Once the reduced prototypes are obtained, the eigenvalue/eigenvector computations are significantly smaller since these computations are now done for a much smaller set, and thus for an $n' \times n'$ matrix. The net result of these two reductions is reflected in the time savings we report in a later section in which we discuss the experimental results obtained for artificial and real-life data sets. However, before we present the experimental verification of our claims, we briefly submit the theoretical basis for our proposed solution.

5 Theoretical Basis for the Proposed Solution

The theoretical basis for our research is essentially founded on the excellent research works of Achlioptas and his co-authors [10, 11]. If we merely resort to the properties of the kernel matrix, K, the question of enhancing the speed completely depends on *its* sparseness. This, in turn, means that any speed enhancements are going to be entirely data dependant.

To overcome this situation, where our hands are basically tied, the research in [11] presented a novel method by which sparseness can be *artificially* introduced into the kernel matrix. The authors suggested that this be achieved by randomly setting the elements of the matrix to zero with a certain probability, and modifying it to have another "constant" value, otherwise. It turns out that while this rather simple (but ingenious) scheme increases the sparseness of K, if the random modifications are done in a controlled manner, the kernel is replaced by an approximation that is more easily computed. Furthermore, this approximation can be made arbitrarily close to the true value. In the interest of readability, the formal details are omitted here - they can be found in [11].

The problem with resorting to the methods of [11] is essentially one of computation. Although the matrix becomes sparse, the problems associated with the eigenvalue computations of the "large" matrix persist. To resolve this, we consider another theoretical avenue, also introduced by Achlioptas and his co-authors [10]. In the case of the kPCA, the method in [11] does not delete entire samples, namely, the data points. Rather, the method deletes single inner products of the Gram matrix, or in other words, it deletes a random fraction of all the kernel evaluations. As opposed to this, in [10], which was the precursor to [11], the authors propose an alternate strategy. There, they explain how one can do "adaptive" sampling so that the *more important samples* are more likely to be maintained. As explained in [10], all the mathematical formal results go through, rendering the adaptive scheme to work much better in practice. The analogous result readily applies also to the kPCA setting[4].

[4]We are extremely grateful to Dr. Achlioptas for his e-mails, and for providing us with the insight into this.

The question that a practitioner faces, is indeed, one of determining which of the samples are the *more important* ones, and of developing a computational mechanism by which they are more likely to be maintained. Clearly, we have attempted to do this by using a PRS.

Analyzing our current philosophy for any arbitrary PRS is an open problem. Although this is currently being investigated, we do *not* believe that there will be a *single* analytic argument for *all* the PRS schemes that can be used to include the *more important* data points, and to consequently reduce the dimensionality of the kernel matrix. Rather, we believe that any analysis will be intricately dependent on the analytic characteristics of the particular PRS that is used.

Before one is too ambitious, it should be also mentioned that it is quite likely that, in many cases, such a theoretical analysis will, quite simply, not be possible. This is because the theoretical foundations of the corresponding PRS are not currently available, and are, probably, infeasible. Observe that there is currently no formal derivation for the analytic properties of the PNN, the CNN, and indeed, of most of the “zillions” of PRS schemes that are reported in the literature.

The methods that we have used to test our strategy include the PNN, the CNN, the VQ and a new hybrid method with enhances the SVM with an LVQ3-type algorithm. As mentioned above, unfortunately, no formal analysis is available for the PNN and the CNN. While the formal theory for the SVM [21, 22] has been extensively developed, its LVQ3-enhancement has not been analyzed. Thus, the analysis in most of these cases is, unfortunately, open. However, rather than side-step this topic, we shall present the arguments as to why our philosophy works for PRSs which base their reduction on the theory of Self Organizing Maps (SOMs) or Vector Quantization (VQ).

We believe that analogous arguments can also be made for PRS schemes which utilize a Bootstrap philosophy. This is because the Bootstrap samples are chosen in such a way that the distribution of the retained Bootstrap samples attempt to follow the distribution of the overall training set. But this is yet to be investigated.

5.1 Foundations for Using SOMs and Vector Quantization

Although the SOM is described and implemented in such an elementary manner, it has extremely elegant properties. A collection of these is found in [23, 29].

First of all Kohonen showed that the resulting map forms a Voronoi tessellation of a given input space which is a kind of VQ partitioning of the input space into non-overlapping regions [23]. Furthermore, apart from these tessellations, the neurons also *ultimately* collectively possess the stochastic properties of the underlying data points. Indeed, if the width of the neighborhood is fixed, and the update Kohonen parameter[5] $\alpha(t)$ is very small, Ritter [26] showed that for the scalar case, the density of weight vectors $G(Y)$ is proportional to $F(X)^\gamma$ where $F(X)$ is the distribution of the input vectors, $\mathcal{P}$, and

$$\gamma = ((W+1)^2/3)/((W+1)^2 + W^2), \tag{15}$$

where again, $[-W, W]$ specifies the activation bubble. To the best of our knowledge, the analogous results for vectors of higher dimensions are presently unknown.

The situation is more hopeful in the case of VQ in its purest form. Zador [30] showed that the Linde, Buzo and Gray algorithm [31] for VQ (i.e., the Kohonen map for $W = 0$) produces neurons with asymptotic density $G(Y)$ proportional to $F(X)^{n/(n+d)}$ where n is the dimension of the input space and the L_d-norm is used to measure the distortion between the neurons and the input vectors. Moreover, when $n = 1$ and $d = 2$, these two asymptotic distributions become the same, namely, $G(Y) \propto F(X)^{1/3}$.

By considering the effects of the SOM in the transformed domain Wong [29] argues that

$$\overline{Y}(t+1)(\omega) = (1-\alpha)\overline{Y}(t)(\omega) + \alpha X(t)\delta(\omega), \tag{16}$$

where $\delta(\cdot)$ is the delta function and $\overline{Y}$ is the Fourier transform of Y. Thus, all frequency components in $Y(t)$ are scaled by $1-\alpha$ while its d.c. component is increased by $\alpha X(t)$. If the same updating is repeatedly applied to the weights in the activation bubble all the weights would be equal. Of course, since the input vectors are presented randomly, the net effect is that the neighboring neurons have their weights as similar as possible. Thus, the Kohonen updating rule also smooths the weights. Indeed, the dynamics of the Kohonen map can be seen to consist of two components - the nearest-neighbor component achieved in a non-parametric manner, and a smoothing effect

[5]To be consistent with the notation used in the literature, we request the freedom to use the same notation for the Kohonen update parameter, α, as is customarily done. Of course, there is no confusion with the α used earlier in this paper.

obtained as a consequence of the windowing effect. Whereas the clustering component tends to gather the neurons' weights, the smoothing keeps the spacing among the weights. Notice that as the neighborhood $[-W, W]$ decreases, the width of the smoothing effect decreases and the clustering effect increases. Fine tuning of the map is a result of the changing balance between these two effects. However, if W is progressively decreased [23], initially all the neurons tend to coalesce close to the mean of the overall distribution when W is large, and subsequently unwind to find their relative places statistically and topologically as W is progressively decreased.

In summary, if we use a VQ-based PRS, the resultant prototypes will form a smaller subset which essentially has the same distribution in the feature space. This, informally, implies that the principal directions of the eigenvectors will be preserved (albeit approximately), which in turn will ensure the approximation of the resultant KNS. Our experimental results validate this.

6 Experimental Results : Artificial Data Sets

6.1 Experimental Data

The proposed method has been rigorously tested and compared with many conventional KNS. This was first done by performing experiments on a number of "artificial" data sets as summarized in Table 1.

Table 1: The artificial data sets used in the experiments. The vectors are divided into two subsets of equal size, and used for training and validation, alternately.

Data Set Names	# of Patterns	# of Features	# of Classes
Non_normal 1	100 (50; 50)	8	2
Non_normal 2	1,000 (500; 500)	8	2
Non_normal 3	10,000 (5,000; 5,000)	8	2

The data set named "Non_normal", which has also been employed as a benchmark experimental data set [20, 32, 33], for numerous experimental set-ups (including those involving the Bootstrap method [33]) was generated from a mixture of four 8-dimensional Gaussian distributions as follows:

1. $p_1(x) = \frac{1}{2}N(\mu_{11}, I_8) + \frac{1}{2}N(\mu_{12}, I_8)$ and

2. $p_2(x) = \frac{1}{2}N(\mu_{21}, I_8) + \frac{1}{2}N(\mu_{22}, I_8)$,

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

In our experiments, three data sets, "Non_normal 1,2,3", were generated with different sizes of testing and training sets of cardinality 100, 1,000, and 10,000 respectively. In the data sets, all of the vectors were normalized within the range $[-1, 1]$ using their standard deviations. Also, for every class j, the data set for the class was randomly split into two subsets of equal size. One of them was used for training the classifiers as explained above, and the other subset was used in the validation (or testing) of the classifiers. The roles of these sets were later interchanged.

6.2 Experimental Parameters

As in all learning algorithms, choosing the parameters[6] of the PRS and KNS play an important role in determining the quality of the solution. The parameters for the PRS are summarized as follows:

1. **Parameters for the CNN and the PNN**
   (a) None.
2. **Parameters for the VQ**
   (a) The code book size, M, is selected as one of the quantities $2^1, 2^2, \cdots, 2^p$ ($2^p \leq N$, N is the number of samples), after considering the classification accuracy,
   (b) The maximum number of iterations is 50,
   (c) The stopping and splitting criteria are both 0.01.
3. **Parameters for the HYB**
   (a) The initial code book size is determined by the SVM (in this experiment, we hybridized the SVM and an LVQ3- type algorithm),

[6]The same parameters were used for both the artificial data sets studied in this section, and for the real-life data sets studied in the next section.

(b) The parameters for the LVQ3 learning, such as the α, the ϵ, the window length, w, and the iteration length, η, are specified as described in [24].

The parameters for the Kernel based Nonlinear Subspace Method (KNS) and the Sparse Greedy Matrix Approximation (SGA), are summarized as follows:

1. **Parameters for the KNS**

   (a) To select the subspace dimensions, we guess several initial values based on the cumulative proportion, α, and select the best one by observing how the classification errors are distributed among the classes.

   (b) The kernel function employed in the experiments is the polynomial kernel with the intercept and the degree of the polynomial given as, $k(i,j) = (1 + x_i' x_j)^2$.

2. **Parameters for the SGA**

   (a) The maximum number of iterations is 10^3, and the error bound is $\epsilon = 1.0 \times 10^{-5}$.

6.3 Experimental Results

We report below the run-time characteristics of the proposed algorithm for the artificial data sets, "Non_normal" as shown in Table 2[7]. In this table, KNS represents the method of calculating the kernel matrix with the entire data set. The KNS-Sub1 (60%) and KNS-Sub2 (14%) are methods which pertain to calculating the kernel matrix with a ramdomly selected subset of the data points. On the other hand, the SGA-KNS[8], CNN-KNS, PNN-KNS, VQ-KNS, and HYB-KNS are obtained by calculating the kernel matrix using the prototypes obtained with each respective method.

The proposed methods, the CNN-KNS, PNN-KNS, VQ-KNS, and HYB-KNS can be compared with the traditional versions, such as the KNS (which utilizes the entire set), the subset KNSs, and

[7]From the Table, we see that the computation time of the PNN is greater than that of the others. However, in [18], it has been suggested that the time could be significantly reduced by employing an algorithm similar to the minimal spanning tree, and using a method of associating with every sample point in the data set, two distances w_i and b_i. Also, in [19], Bezdek and his co-authors proposed a modification of the PNN as mentioned in Section 3.

[8]In the implementation of this method, we, at first, reduced the $n \times n$ kernel matrix into a matrix of dimensions $n' \times n'$ ($n' << n$) by using the SGA technique described in [12]. Subsequently, we employed the linear subspace method to compute the eigenvectors and projections of the *latter* matrix. The program code for the SGA algorithm was graciously provided by Dr. Alex Smola, of the Australian National University. The authors are very grateful to him for his assistance in this study.

the SGA-KNS, using two criteria, namely, the classification accuracy (%), *Acc*, and the processing CPU-time (in seconds), T_1. We report below a summary of the results obtained for the case when one subset was used for training and the second for testing. The results when the roles of the sets are interchanged are almost identical. In the tables, each result is the averaged value of the training and the test sets, respectively.

In KNS, the subspace dimensions have a strong influence on the performance of subspace classifiers. Therefore, the subspace dimensions, the number of prototypes employed for constructing the kernel matrix, and their extraction times in the PRSs and the SGA, should be investigated. To assist in this task, we display the results of the *Acc* and T_1, the subspace dimensions for the two data sets, $(p_{11}, p_{12}), (p_{21}, p_{22})$, the number of prototypes finally utilized, $n' = \frac{1}{2}(n'_1 + n'_2)$, and their extraction times, $t' = \frac{1}{2}(t'_1 + t'_2)$.

From Table 2, we can see that the processing CPU-time of the CNN-KNS, PNN-KNS, VQ-KNS, and HYB-KNS methods can be reduced significantly by employing a PRS such as the CNN, PNN, VQ, and HYB. Indeed, this is achieved without sacrificing the accuracy so much. It should be mentioned that the reduction of processing time increased dramatically when the size of the data sets was increased.

Consider the CNN-KNS method for the three "Non_normal" data sets. If the entire sets of sizes 50, 500, and 5,000 respectively, were processed, the times taken for the KNS are 0.63, 57.50, and 12,464.0 seconds, respectively. However, if the same sets were first preprocessed by the CNN, to yield the CNN-KNS procedure, the processing times are 0.11, 5.09, and 360.29 seconds respectively[9]. Notice that the classification accuracy of the two methods is almost the same. Identical comments can also be made about the PNN-KNS, VQ-KNS, and HYB-KNS schemes.

In fact, if we reckon the actual CPU processing time as the summation of the classification time, T_1, and the prototype extraction time, t', the table demonstrates that the processing time of the SGA-KNS is longer than those of the currently proposed methods.

It is not so easy to crown any one PRS to be superior to the others in the context of optimizing KNS methods. This re-iterates the basic proposition found in the literature, that the suitability of

[9]It should be mentioned that the learning and recognition time taken by the conventional method SGA-KNS for the set *Non_3* is only 7.9 seconds. However, the time required to approximate the $n \times n$ kernel matrix using the smaller $n' \times n'$ matrix obtained by using the SGA technique is 1,305.35 seconds! Thus, it is meaningless to merely compare the methods on any *one single* statistic.

any PRS depends on the properties of the data set itself [15, 24]. However, the reader should easily observe that the computational advantage gained is *significant*, and the accuracy lost is *marginal*. But as a matter of comparison, it is clear that with regard to the time involved in generating the reduced problem, the proposed methods are computationally almost *always* less intensive than the SGA methods used to optimize the KNS.

7 Experimental Results : Real-life Data Sets

7.1 Experimental Data

In order to further investigate the advantage gained by utilizing the proposed methods, we conducted experiments on "real-life" data sets, which we refer to as the "Ionosphere", "Sonar", "Arrhythmia", and "Adult4" sets. The information about these data sets is summarized in Table 3. These data sets are real benchmark data sets, cited from the UCI Machine Learning Repository[10].

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 or absence of cardiac arrhythmia, and to classify the feature into one of the 16 groups. In our case, in the interest of uniformity, we merely attempted to classify the total instances into two category, namely, "normal" and "abnormal".

The "Adult4" data set was extracted from a census bureau database[11]. 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 this case, the total number of sample vectors is 33,330. Among them, we randomly selected 8,336 samples due to the time considerations, which is approximately 25 % of the whole set.

In the above data sets, all of the vectors were normalized to be within the range $[-1, 1]$ using their standard deviations, and the data set for class j was randomly split into two subsets, $T_{j,t}$ and $T_{j,V}$, of equal size. One of them was used for choosing the initial prototypes and training the

[10]http://www.ics.uci.edu/mlearn/MLRepository.html
[11]http://www.census.gov/ftp/pub/DES/www/welcome.html

classifiers, and the other one was used in their validation (or testing). Later, the role of these sets were interchanged.

The experimental results for the "Ionos","Sonar", "Arrhy" and "Adult4" data sets are shown in Table 4.

Consider the results of Table 4 for the data captioned *Ionos*. Processing the entire data set of 176 samples required a CPU-time of 14.32, 5.12, 2.28 and 6.61 seconds, and yielded the accuracies of 84.38, 72.73, 70.46 and 88.92 % with the KNS, KNS-Sub1, KNS-Sub2 and SGA-KNS methods. However, if the samples were pre-processed by a PRS, to yield the CNN-KNS, PNN-KNS, VQ-KNS or HYB-KNS respectively, the times taken were 2.79, 2.24, 0.34 and 2.67 seconds respectively - which represented much smaller computation times. In this case, the accuracies actually *increased* to the values of 89.20, 91.20, 79.55 and 85.80 respectively[12].

From Table 4, we also observe the same characteristics as those seen in Table 2. The data set *Adult*4 of 4,168 samples was processed requiring a computation time of 15,558.0 seconds, and gave an accuracy of 85.78 % when the KNS method utilized the entire data set. However, if the samples were preprocessed with a PRS to yield the CNN-KNS, the computation was achieved in 536.38 seconds, and yielded an accuracy of 90.10 %.

It appears as if we can make one final concluding remark on the newly proposed KNS enhancements. From the experimental results reported, we see that the various methods, namely, the CNN-KNS, PNN-KNS, VQ-KNS, and HYB-KNS, have different measures of accuracy for the various data sets investigated. However, from the results for the high-dimensional data set of "Arrhy" and from the large data set of "Adult4", we observe that the HYB-KNS is uniformly superior to the others in terms of the classification accuracy, *Acc*, while requiring a little additional prototype extraction time, t'. We thus propose that this method is the most ideal candidate of choice for enhancing a KNS using a PRS.

[12]It should be mentioned that such an increase in accuracy is an exception, rather than a rule. The accuracy typically falls marginally when the KNS is preprocessed by a PRS.

8 Conclusions

Kernel based nonlinear subspace methods suffer from a major disadvantage for large data sets, namely that of the excessive computational burden encountered by processing all the data points. In this paper, we suggest a computationally superior mechanism to solve the problem. Rather than define the kernel matrix and compute the principal components using the entire data set, we propose that the size of the data be reduced into a smaller prototype subset using a Prototype Reduction Schemes (PRS). This PRS can be any one of the "zillions" of currently reported PRS methods. Since PRS yields a smaller subset of data points that effectively samples the entire space to yield subsets of prototypes, these prototypes can be used quite effectively for composing a "reduced-kernel" matrix, thus alleviating the computational burden significantly.

The proposed method has been tested on artificial and real-life benchmark data sets, and compared with the conventional methods. The experimental results for both large data sets demonstrate that the proposed schemes can improve the learning speed of the proposed methods by an order of magnitude, while yielding almost the same classification accuracy.

Although we have given a theoretical basis for using the VQ as the underlying PRS to optimize the KNS, the problem of analyzing our methodology for other PRSs, like the PNN, the CNN, the hybrid-SVM method etc. remains open. We believe that this will be nontrivial.

Acknowledgments

We are very grateful to Dr. Achlioptas for providing us with his papers, and for the insight we got from him through the e-mails. Thanks, Dimitris ! The first author is also grateful to the School of Computer Science at Carleton University, Ottawa, for being willing to host him during his years of sabbatical leave.

References

[1] E. Oja, *Subspace Methods of Pattern Recognition*, Research Studies Press, 1983.

[2] K. Maeda and S. Watanabe, "A pattern matching method with local structure (in Japanese)", *IEICE Trans. Information & Systems*, vol. J68-D, no. 3, pp. 345 - 352, Mar. 1985.

[3] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Müller, G. Ratsch, K. Tsuda, and A. J. Smola, "Input space versus feature space in kernel-based methods", *IEEE Trans. Neural Networks*, vol. 10, no. 5, pp. 1000 - 1016, Sept. 1999.

[4] B. Schölkopf, A. J. Smola, and K. -R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem", *Neural Comput.*, vol. 10, pp. 1299 - 1319, 1998.

[5] K. Tsuda, "Subspace method in the Hilbert space (in Japanese)", *IEICE Trans. Information & Systems*, vol. J82-D-II, no. 4, pp. 592 - 599, April 1999.

[6] E. Maeda and H. Murase, "Multi-category classification by kernel based nonlinear subspace method", in the *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 99)*, IEEE Press, 1999.

[7] E. Maeda and H. Murase, "Kernel based nonlinear subspace method for pattern recognition (in Japanese)", *IEICE Trans. Information & Systems*, vol. J82-D-II, no. 4, pp. 600 - 612, April 1999.

[8] K. R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithm", *IEEE Trans. Neural Networks*, vol. 12, no. 2, pp. 181 - 201, Mar. 2001.

[9] H. Sakano, N. Mukawa and T. Nakamura, "Kernel mutual subspace method and its application for object recognition (in Japanese)", *IEICE Trans. Information & Systems*, vol. J84-D-II, no. 8, pp. 1549 - 1556, Aug. 2001.

[10] D. Achlioptas and F. McSherry, "Fast computation of low-rank approximations", in *Proceedings of the Thirty-Third Annual ACM Symposium on the Theory of Computing*, Hersonissos, Greece, ACM Press, pp. 611 - 618, 2001.

[11] D. Achlioptas, F. McSherry and B. Schölkopf, "Sampling techniques for kernel methods", in *Advances in Neural Information Processing Systems 14*, MIT Press, Cambridge, MA, pp. 335-342, 2002.

[12] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning", *Proceedings of ICML'00*, Bochum, Germany, Morgan Kaufmann, pp. 911 - 918, 2000.

[13] C. Williams and M. Seeger, "Using the Nystrom method to speed up kernel machines", in *Advances in Neural Information Processing Systems*, vol. 13, MIT Press, Cambridge, MA, 2001.

[14] M. Tipping, "Sparse kernel principal component analysis", in *Advances in Neural Information Processing Systems 13*, MIT Press, Cambridge, MA, pp. 633 - 639, 2001.

[15] J. C. Bezdek and L. I. Kuncheva, "Nearest prototype classifier designs: An experimental study", *International Journal of Intelligent Systems*, vol. 16, no. 12, pp. 1445 - 11473, 2001.

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

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

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

[19] J. C. Bezdek, T. R. Reichherzer, G. S. Lim, and Y. Attikiouzel, "Multiple-prototype classifier design", *IEEE Trans. Systems, Man, and Cybernetics - Part C*, vol. SMC-28, no. 1, pp. 67 - 79, Feb. 1998.

[20] Q. Xie, C.A. Laszlo and R. K. Ward, "Vector quantization techniques for nonparametric classifier design", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-15, no. 12, pp. 1326 - 1330, Dec. 1993.

[21] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition", *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121 - 167, 1998.

[22] V. N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.

[23] T. Kohonen, *Self-Oganizing Maps*, Berlin, Springer - Verlag, 1995.

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

[25] J. Laaksonen and E. Oja, "Subspace dimension selection and averaged learning subspace method in handwritten digit classification", *Proceedings of ICANN*, Bochum, Germany, pp. 227 - 232, 1996.

[26] H. J. Ritter, "Asymptotic level density for a class of vector quantization processes", in *IEEE Transaction on Neural Networks*, vol. 2, pp. 173 - 175, 1991.

[27] H. J. Ritter and K. J. Schulten, "Kohonen's self-organizing maps : Exploring their computational capabilities", in *Proc. of the Int. Joint Conf. on Neural Networks*, pp. 2455 - 2460, 1988.

[28] S. -W. Kim and B. J. Oommen, "A Brief Taxonomy and Ranking of Creative Prototype Reduction Schemes". To appear in *Pattern Analysis and Applications Journal.*

[29] Y. Wong, "A comparative study of the Kohonen self-organizing map and the elastic net", *Computational Learning Theory and Natural Learning Systems*, vol. 2, pp. 401 - 413, 1996.

[30] P. L. Zador, "Asymptotic quantization of error of continuous signals and the quantization dimension", *IEEE Transaction on Information Theory*, vol. 28, pp. 139–149, 1982.

[31] Y. Linde, A. Buzo and R. M. Gray, "An algorithm for vector quantization", *IEEE Trans. Communication*, vol. COM-28, pp. 84 - 95, 1980.

[32] K. Fukunaga, *Introduction to Statistical Pattern Recognition, Second Edition*, Academic Press, San Diego, 1990.

[33] Y. Hamamoto, S. Uchimura and S. Tomita, "A bootstrap technique for nearest neighbor classifier design", *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-19, no. 1, pp. 73 - 79, Jan. 1997.

Table 2: The experimental results of the proposed method for the artificial data sets, "Non_normal 1,2,3". Here, the two values, *Acc* and T_1, are the classification accuracy (%) and the processing CPU-time (in seconds). The values (p_1, p_2), n', and t' are subspace dimensions of each class, the number of prototypes, and their extraction times, respectively. The final column, $1nn$, is the classification result obtained by invoking a 1-nearest neighbor rule, where the reference set consists of the n' prototypes.

Dataset	Methods	*Acc*	T_1	(p_1, p_2)	n'	t'	$1nn$
	KNS (Whole Set)	95.00	0.63	(2,2) (1,2)	50	0	95.00
	KNS-Sub1 (60%)	93.00	0.36	(2,2) (1,2)	30	0	94.00
	KNS-Sub2 (14%)	80.00	0.10	(2,2) (1,2)	7	0	91.00
Non_1	SGA-KNS	94.00	0.12	(1,1) (2,2)	6	0.51	94.00
	CNN-KNS	96.00	0.11	(1,1) (1,2)	7	0.00	95.00
	PNN-KNS	94.00	0.19	(1,1) (1,1)	8	0.02	94.00
	VQ-KNS	95.00	0.07	(1,1) (1,1)	2	0.11	95.00
	HYB-KNS	95.00	0.15	(1,1) (1,1)	6	0.11	87.00
	KNS (Whole Set)	94.80	57.50	(1,1) (1,1)	500	0	92.50
	KNS-Sub1 (60%)	94.80	29.13	(1,1) (1,1)	300	0	92.30
	KNS-Sub2 (14%)	94.90	5.57	(1,1) (1,1)	70	0	92.00
Non_2	SGA-KNS	93.80	0.89	(1,1) (1,1)	8	14.07	91.90
	CNN-KNS	94.70	5.09	(1,1) (1,1)	65	0.14	91.90
	PNN-KNS	94.30	22.50	(1,1) (1,1)	219	41.47	92.10
	VQ-KNS	94.90	0.69	(1,1) (1,1)	4	0.06	95.20
	HYB-KNS	94.60	5.00	(1,1) (1,1)	60	14.62	66.00
	KNS (Whole Set)	94.81	12,464.00	(1,1) (1,1)	5,000	0	91.94
	KNS-Sub1 (60%)	94.78	4,099.10	(1,1) (1,1)	3,000	0	86.50
	KNS-Sub2 (14%)	94.77	546.17	(1,1) (1,1)	700	0	92.57
Non_3	SGA-KNS	92.53	7.90	(1,1) (1,1)	8	1,305.35	90.73
	CNN-KNS	94.81	360.29	(1,1) (1,1)	491	58.57	91.63
	PNN-KNS	94.76	8,534.20	(1,1) (1,1)	4,124	53,516.33	92.04
	VQ-KNS	94.51	181.21	(1,1) (1,1)	256	3.19	90.12
	HYB-KNS	94.76	488.87	(1,1) (1,1)	641	58.38	42.29

Table 3: The "real-life" 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	# of Patterns	# of Features	# of Classes
Ionosphere	351 (176; 175)	34	2
Sonar	208 (104; 104)	60	2
Arrhythmia	452 (226; 226)	279	16
Adult4	8,336 (4,168; 4,168)	14	2

Table 4: The experimental results of the proposed method for the real-life data sets "Ionos", "Sonar", "Arrhy", and "Adult4". Here, the two values, *Acc* and T_1, are the classification accuracy (%) and the processing CPU-time (in seconds). The values (p_1, p_2), n', and t' are subspace dimensions of each class, the number of prototypes, and their extraction times, respectively. The final column, $1nn$, is the classification result obtained by invoking a 1-nearest neighbor rule, where the reference set consists of the n' prototypes.

Data Sets	Methods	*Acc*	T_1	(p_1, p_2)	n'	t'	$1nn$
Ionos	KNS (Whole Set)	84.38	14.32	(45,66) (32,45)	176	0	78.70
	KNS-Sub1 (60%)	72.73	5.12	(45,66) (32,45)	106	0	82.39
	KNS-Sub2 (14%)	70.46	2.28	(18,18) (16,10)	46	0	78.41
	SGA-KNS	88.92	6.61	(9,18) (16,41)	64	44.48	76.42
	CNN-KNS	89.20	2.79	(13,36) (11,31)	46	0.13	89.20
	PNN-KNS	91.20	2.24	(7,30) (8,25)	35	2.43	91.20
	VQ-KNS	79.55	0.34	(2,2) (2,2)	4	0.19	79.55
	HYB-KNS	85.80	2.67	(13,31) (11,35)	45	0.26	85.80
Sonar	KNS (Whole Set)	87.50	3.67	(23,22) (23,23)	104	0	82.22
	KNS-Sub1 (60%)	79.33	2.37	(23,22) (23,23)	62	0	73.08
	KNS-Sub2 (14%)	74.52	2.12	(14,17) (25,24)	27	0	72.12
	SGA-KNS	85.10	3.28	(21,20) (20,20)	96	13.78	80.77
	CNN-KNS	80.29	1.71	(20,17) (22,17)	53	0.18	79.81
	PNN-KNS	84.14	1.21	(17,15) (20,13)	34	1.32	82.69
	VQ-KNS	74.52	1.03	(16,16) (16,16)	32	0.79	78.37
	HYB-KNS	84.62	1.94	(27,22) (21,17)	56	1.84	80.77
Arrhy	KNS (Whole Set)	98.24	18.29	(5,10) (13,19)	226	0	97.57
	KNS-Sub1 (60%)	97.57	9.62	(5,10) (13,19)	137	0	95.80
	KNS-Sub2 (14%)	97.57	9.62	(5,10) (13,19)	59	0	89.82
	SGA-KNS	98.46	18.83	(25,28) (28,32)	206	213.05	97.35
	CNN-KNS	95.80	2.72	(15,16) (14,14)	30	0.84	96.47
	PNN-KNS	96.02	1.28	(3,4) (3,4)	8	34.32	99.12
	VQ-KNS	98.23	4.63	(13,17) (12,15)	64	5.38	98.90
	HYB-KNS	98.90	7.12	(23,32) (21,25)	67	3.50	99.12
Adult4	KNS (Whole Set)	85.78	15,558.00	(13,12) (13,12)	4,168	0	93.38
	KNS-Sub1 (60%)	84.74	4,515.10	(13,12) (13,12)	2,501	0	92.81
	KNS-Sub2 (14%)	78.33	600.00	(13,12) (13,12)	1,084	0	92.89
	SGA-KNS	81.80	244.85	(27,18) (17,19)	301	108,135.50	81.33
	CNN-KNS	90.10	536.38	(8,8) (8,8)	755	237.69	91.56
	PNN-KNS	88.86	456.85	(9,9) (9,8)	660	16,863.64	89.35
	VQ-KNS	54.32	13.76	(7,8) (7,8)	16	1.31	78.21
	HYB-KNS	91.96	287.70	(11,11) (11,11)	440	5,453.64	92.75