International Journal of Information Technology & Decision Making
Vol. 10, No. 3 (2011) 479–495
© World Scientific Publishing Company
DOI: 10.1142/S0219622011004415

EVALUATE DISSIMILARITY OF SAMPLES IN FEATURE SPACE FOR IMPROVING KPCA

YONG XU*,§, DAVID ZHANG†,¶, JIAN YANG‡, ZHONG JIN‡ and JINGYU YANG‡

*Shenzhen Graduate School, Harbin Institute of Technology
Shenzhen, China

†Biometrics Research Centre, Department of Computing
The Hong Kong Polytechnic University, Kowloon, Hong Kong

‡School of Computer Science & Technology
Nanjing University of Science & Technology, Nanjing, China

§laterfall2@yahoo.com.cn
¶csdzhang@comp.polyu.edu.hk

Since in the feature space the eigenvector is a linear combination of all the samples from the training sample set, the computational efficiency of KPCA-based feature extraction falls as the training sample set grows. In this paper, we propose a novel KPCA-based feature extraction method that assumes that an eigenvector can be expressed approximately as a linear combination of a subset of the training sample set ("nodes"). The new method selects maximally dissimilar samples as nodes. This allows the eigenvector to contain the maximum amount of information of the training sample set. By using the distance metric of training samples in the feature space to evaluate their dissimilarity, we devised a very simple and quite efficient algorithm to identify the nodes and to produce the sparse KPCA. The experimental result shows that the proposed method also obtains a high classification accuracy.

Keywords: Feature extraction; kernel methods; kernel PCA.

1. Introduction

Principal component analysis (PCA) is a linear feature extraction technique that has proven itself effective because of its ability to capture the most variable components of sample data. PCA utilizes a number of eigenvectors of the generation matrix (the covariance) of the sample data as transforming axes to transform the sample data into a new space in which different components of the data are statistically uncorrelated. A number of studies[1–7] have shown that transforming the sample into the new space allows PCA to use lower dimensional data to effectively represent the characteristics of the original sample data. However, PCA does not usually do so well in representing samples using low-dimensional features when sample data exhibits a complex distribution.[1] In such cases, nonlinear feature extraction techniques will be more appropriate than linear techniques.[1]

Kernel PCA (KPCA) can be viewed as a nonlinear feature extraction method derived from PCA.[8,9] It has been shown that KPCA can perform well in extracting features from samples whose components have nonlinear relation. This is because KPCA owns the advantages of general nonlinear learning methods.[10,11] KPCA can be regarded as a combination of two processes, a first process that implicitly transforms the input space into a new space, i.e. the so-called feature space, and a second process that implements PCA in the feature space. The use of kernel functions makes KPCA more computationally tractable than a general nonlinear feature extraction method. However, KPCA-based feature extraction might be still quite inefficient and even impractical in real-world applications that have a large number of training samples. Indeed, this is a common difficulty of each kernel-method-based feature extraction procedure.[12–14] The reason for this is as follows. As the reproducing kernel theory[8,9] tells us, a transforming axis in the feature space induced by KPCA can be expanded as a linear combination of all training samples in the feature space. Thus, when projecting one sample in the feature space onto the transforming axis, we should compute the dot products of the sample and each of all the training samples. The kernel trick allows us to replace the dot product of the sample and a training sample with a kernel function, so we should calculate as many kernel functions as there are training samples. This also exposes the weak point of KPCA-based feature extraction, that the larger the size of the training sample set, the lower its computational efficiency.

Two approaches have been proposed to improve the computational efficiency of KPCA on large-scale training sample sets. The first approach is based on the supposition that in the feature space one or more training samples can be (approximately) expressed as a linear combination of the others.[15] These methods commonly identify a so-called sparse subset of the training sample set, and a linear combination of the sparse subset is used to approximately expand the eigenvector. They are also referred to as sparse KPCA methods. Hereafter we refer to the elements of this sparse subset as nodes. The second approach assumes that one eigenvector in the feature space can be approximately expanded with reference to a small number of synthetic vectors.[15] Examples of this approach can be found in Ref. 16. In either case, these approaches are more efficient than naïve KPCA because when extracting features from a sample, they calculate fewer kernel functions. Typical examples of the sparse KPCA method would include the methods shown below. Tipping[17] exploited a maximum-likelihood technique to approximate the transformed covariance matrix of a sparse subset. Franc *et al.*[18] proposed a greedy method to obtain the approximate representation of the feature space. It iteratively extracts features from the data in the feature space and terminates the feature extraction procedure when the approximation error in the feature space falls below a threshold. Smola *et al.*[19] proposed the sparse kernel feature analysis (SKFA) method. If the number of extracted features is much smaller than the size of the training sample set, SKFA will have a lower computational cost than naïve KPCA; otherwise, SKFA will have a higher computational cost. García-Osorio *et al.* recently implemented SKFA using the bootstrapping and bagging method, and Jiang *et al.*[20] proposed the accelerated kernel feature analysis (AKFA) algorithm to further reduce the computational cost of SKFA. Scholkopf *et al.*[15] also presented two methods to produce a sparse kernel method. These two methods selected nodes from the training sample set and attempted to approximate the genuine solution from the viewpoint of numeral approximation. The first method used an iterative algorithm and identified one node at a time. The second method was formulated as a quadratic programming problem. Both of these two methods, in particular the first method, identified the nodes at a high computational cost. We have also proposed an improved KPCA (IKPCA) method in Ref. 22. We assumed that in the feature space an eigenvector can be well approximated in terms of nodes and reformulated the KPCA generation matrix from the viewpoint of the PCA methodology. The eigenvector produced from this generation matrix was used to identify the nodes and to determine a linear combination of the nodes that can best approximate the eigenvector. Using the obtained eigenvector, we can efficiently implement KPCA-based feature extraction. However, this method also identified the nodes at a high computational cost.

Other researchers have sought to improve KPCA by integrating it with other techniques. For example, Chin and Suter[23] devised an incremental KPCA method that can be viewed as a combination of incremental learning technique and the sparse KPCA method based on the constructed sparse subset of the training sample set. Works from Refs. 24, 25, and 26, respectively exploited the prototype reduction scheme (PRS), the gradient descent algorithm, and the Bootstrapping method to produce a sparse kernel method. Kernel methods have also been applied to many problems such as face recognition and data clustering.[31–34] We also note that recent studies on kernel methods have obtained some noticeable achievements.[31–37] For example, Gnecco *et al.* analyzed the upper bounds on the accuracy in approximating the optimal solution of KPCA.[10,31] Georgiev *et al.*[33] rigorously defined the sparse component analysis (SCA) problem of sparse signals and presented sufficient conditions for its solution.

The previous sparse KPCA methods have the following characteristic: though these sparse KPCA methods are able to extract features computationally more efficiently than naïve KPCA, they all achieve this capability at an extra computational cost. Indeed, the extra cost might be very high and is usually much higher than the cost of solving the eigenvectors of naïve KPCA. In this sense the complete procedure to implement these improvements may be still computationally inefficient.

In this paper, we propose a novel method to obtain sparse KPCA (namely, efficient sparse KPCA (ESKPCA)). The proposed method applies a dedicated algorithm to select maximally dissimilar training samples as nodes (elements of the corresponding sparse subset of the training sample set). The algorithm exploits the kernel trick to convert the dissimilar metric of samples in the feature space into an expression that contains only kernel functions. Provided that a number of nodes have been identified, the proposed method will reformulate KPCA as a new eigenvalue equation. Once the eigenvectors of the eigenvalue equation are solved, we can obtain the eigenvectors of ESKPCA.

Compared with previous sparse KPCA methods, the proposed method has the following remarkable advantages: first, differing from previous methods, the proposed method provides a novel viewpoint and algorithm to produce sparse KPCA. Second, the algorithm of the proposed method is much simpler and computationally more efficient than those of previous methods, which usually employ some elaborated criterions and procedures with a high complexity to obtain sparse KPCA. Experimental results also show that ESKPCA classifies very accurately.

The remainder of the paper is organized as follows: In Sec. 2 we show that KPCA is equivalent to the implementation of PCA in the feature space and present KPCA-based feature extraction. In Sec. 3 we show how to obtain the sparse KPCA. In Sec. 4 we compare computational costs of our approach and several other sparse KPCA methods. In Sec. 5 we test our approach and the other methods. In Sec. 6 we offer our conclusion. Our contributions mainly appear in Secs. 3 and 4.

2. KPCA and KPCA-Based Feature Extraction

In this section we describe KPCA and KPCA-based feature extraction. KPCA is a nonlinear PCA method. The implementation of KPCA seems to be equivalent to the implementation of the following process: all the samples are first transformed into a new space by using a nonlinear mapping. Then PCA is performed in the new space and extracts the lower dimensional features of samples in the new space. However, KPCA indeed does not need to explicitly perform the nonlinear mapping. Instead, KPCA implicitly obtains the nonlinear mapping by exploiting the kernel trick. This enables KPCA to have a promising computational cost in comparison with a general nonlinear feature extraction method.

We also say that KPCA is an equivalent implementation of PCA in the feature space (i.e. the new space mentioned above). We briefly present KPCA as follows. Let $x_1, x_2, \ldots, x_N$ be N training samples in the original space. Suppose that each sample from the training sample set has been transformed into the feature space by a nonlinear function ϕ. As a result, we can use $\phi(x_1) \cdots \phi(x_N)$ to denote the training samples in the feature space. If the samples in the feature space have zero mean, then the covariance matrix is $\Sigma_\phi = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)(\phi(x_i))^T$. We also refer to Σ_ϕ as the generation matrix of the feature space. According to the PCA methodology, the most useful eigenvectors of the feature space should be the eigenvectors corresponding to large eigenvalues of Σ_ϕ. That is, the most useful eigenvectors should be the solutions u_i corresponding to large λ_i of $\Sigma_\phi u_i = \lambda_i u_i$. By exploiting the kernel function $k(x_i, x_j)$ to denote the dot product, i.e. $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$, we can derive the following eigenvalue equation[9]:

$$K\alpha = \lambda\alpha, \tag{1}$$

where K is the so-called Gram matrix that has the entry $(K)_{ij} = k(x_i, x_j)$. The principal component analysis method based on the eigenvalue equation (1) is referred to as KPCA.

We formulate KPCA-based feature extraction as follows. We denote the m eigenvectors, corresponding to the first m largest eigenvalues $\lambda_1^\alpha, \lambda_2^\alpha, \ldots, \lambda_m^\alpha$ of (1), by $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(m)}$, respectively. Based on the reproducing kernel theory, we know that the ith eigenvector u_i of Σ_ϕ (the ith eigenvector in the feature space) is $u_i = \sum_{j=1}^{N} \alpha_j^{(i)} \phi(x_j)$, where $\alpha_j^{(i)}$ stands for the jth element of the vector $\alpha^{(i)}$. As a result, the ith feature of a sample $\phi(x)$ from the feature space can be expressed as $\sum_{j=1}^{N} \alpha_j^{(i)} k(x_j, x)/\sqrt{\lambda_i^\alpha}$ and the most representative m-dimensional features obtained using KPCA form the following vector[22]:

$$Y = \left[\sum_{j=1}^{N} \alpha_j^{(1)} k(x_j, x) \Big/ \sqrt{\lambda_1^\alpha} \; \sum_{j=1}^{N} \alpha_j^{(2)} k(x_j, x) \Big/ \sqrt{\lambda_2^\alpha} \cdots \sum_{j=1}^{N} \alpha_j^{(m)} k(x_j, x) \Big/ \sqrt{\lambda_m^\alpha}\right]^T . \quad (2)$$

As can be seen, calculation of each component of the m-dimensional features for the sample $\phi(x)$ depends on the N kernel functions defined in terms of x and all the training samples. Therefore, we can conclude that the larger the size of the training sample set, the lower the computational efficiency of KPCA-based feature extraction.

3. Deriving Sparse KPCA

3.1. *Idea of our approach*

Our approach to deriving sparse KPCA assumes that one can use a linear combination of nodes to express one eigenvector in the feature space. If the nodes are much fewer than training samples, sparse KPCA-based feature extraction will be much faster than KPCA-based feature extraction.

Our approach, i.e. ESKPCA consists of two procedures. The first procedure identifies the nodes and the second procedure determines the linear combination of the nodes that can best express the eigenvector. The first procedure selects as nodes training samples in the feature space that are maximally dissimilar. The rationales are as follows: in expressing the eigenvector, if one sample is very similar or correlated to another sample, the simultaneous use of the two samples is indeed almost equivalent to the use of either of them. The maximization of the dissimilarity also allows the linear combination of a fixed number of nodes to contain the maximum amount of information of the training sample set. To determine the linear combination of the nodes, which can best approximate the eigenvector, the second procedure first reformulates KPCA as a new eigenvalue equation based on the nodes. Then by solving the eigenvalue equation, the second procedure obtains the coefficients of the optimal linear combination. The main innovative idea of our approach is to use maximally dissimilar samples as nodes, which is implemented in the first procedure.

3.2. *Reformulate KPCA using nodes*

In this subsection, provided that nodes have been identified, we show below how to reformulate KPCA as a new eigenvalue equation and how to implement the second procedure to produce the sparse KPCA. We assume that in the feature space an eigenvector u_i generated from the eigenvalue equation $\Sigma_\phi u_i = \lambda_i u_i$ can be approximated using

$$u_i \approx \tilde{u}_i = \sum_{j=1}^{s} \gamma_j^{(i)} \phi(x_j^0), \quad s < N, \tag{3}$$

where $x_1^0, \ldots, x_s^0$ denote s nodes selected from the set of the training samples. We refer to $\tilde{u}_i$ as the approximation eigenvector. We can obtain the feature by projecting a sample in the feature space onto the approximation eigenvector $\tilde{u}_i$. For sample x_n from the training sample set, its feature generated from $\tilde{u}_i$ is

$$f_n = \phi(x_n)^T \tilde{u}_i = \sum_{j=1}^{s} \gamma_j^{(i)} k(x_n, x_j^0), \tag{4}$$

where $n = 1, 2, \ldots, N$. The PCA methodology requires that the variance of all f_n should be maximized. Thus, if the features have zero mean, then

$$\gamma = [\gamma_1^{(i)} \quad \gamma_2^{(i)} \cdots \gamma_s^{(i)}]^T \tag{5}$$

should be the eigenvector corresponding to the maximum eigenvalue of the following eigenvalue equation[28]:

$$K' K'^T \gamma = \lambda \gamma, \tag{6}$$

where K' is defined as

$$K' = \begin{bmatrix} k(x_1, x_1^0) & \ldots & k(x_n, x_1^0) \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ k(x_1, x_s^0) & \ldots & k(x_n, x_s^0) \end{bmatrix} \cdot \gamma$$

is also referred to as a coefficient vector since its elements are used as the coefficients of the linear combination in (3). The second procedure of our approach to deriving sparse KPCA should solve (6) and obtain the coefficient vector.

From (3), we know that the feature extraction result, with regard to $\tilde{u}_i$ of a sample x should be $\sum_{j=1}^{s} \gamma_j^{(i)} k(x_j^0, x)$. This means that we should compute only s kernel functions for producing the feature, whereas KPCA-based feature extraction should calculate N kernel functions. Since it is usually that $s \ll N$, the proposed feature extraction process will be computationally much more efficient than KPCA-based feature extraction. We refer to the sparse kernel PCA based on the eigenvalue equation (6) as efficient sparse KPCA (ESKPCA).

3.3. *The first procedure of our approach to deriving sparse KPCA*

In this subsection we present the first procedure of our approach to deriving sparse KPCA. The first procedure selects as nodes training samples that are maximally dissimilar. We assess the dissimilarity between two samples in the feature space using the squared distance between them. In the feature space, the squared distance between two samples $\phi(x_i)$ and $\phi(x_j)$ can be expressed by

$$d_f^2(x_i, x_j) = \|\phi(x_i) - \phi(x_j)\|^2 = (\phi(x_i) - \phi(x_j))^T(\phi(x_i) - \phi(x_j)). \tag{7}$$

Using the definition of the kernel function, we can convert (7) into

$$d_f^2(x_i, x_j) = k(x_i, x_i) + k(x_j, x_j) - 2k(x_i, x_j). \tag{8}$$

The first procedure of our approach (i.e. the algorithm to identify nodes) works as follows:

Step 1. Calculate $\bar{x}$ using $\bar{x} = \frac{\sum_i x_i}{N}$. Take $\bar{x}$ as the first node and denote it by x_1^0. We represent the original training sample set by T_1.

Step 2. Identify the second node.
First, we calculate the squared distance between each sample $\phi(x_i)$ from the set T_1 and the first node, i.e. $d_f^2(x_i, x_1^0)$, $i = 1, 2, \ldots, N$. Then we select the sample that has the maximum distance value as the second node. We denote the second node by x_2^0. We remove x_2^0 from T_1. The renewed T_1 is represented by T_2.

Step 3. Identify the third node.
We identify the third node as follows: We first calculate the squared distance between each sample from the set T_2 and each of x_1^0, x_2^0. For each of x_1^0, x_2^0, we calculate the sum of the squared distance values of every sample from the set T_2 and refer to this sum as the distance sum. Then we select the sample that has the maximum distance sum as the third node. We denote the third node by x_3^0. We remove x_3^0 from T_2. The renewed T_2 is denoted by T_3.

Step 4. Identify the qth node.
After the previous $q-1$ steps have identified the $q-1$ nodes $x_1^0, x_2^0, \ldots, x_{q-1}^0$, we identify the qth node as follows: we first calculate the squared distance between each sample from the set T_{q-1} and each of $x_1^0, x_2^0, \ldots, x_{q-1}^0$. For each of $x_1^0, x_2^0, \ldots, x_{q-1}^0$, we calculate the sum of the squared distance values of every sample from the set T_{q-1} and refer to this sum as the distance sum. Then we select the sample that has the maximum distance sum as the qth node. We denote the qth node by x_q^0.

We explain the reasonability of the above algorithm as follows: first, $\bar{x}$ can be regarded as the center of the original sample space and can well represent the average information of the original sample data. Step 1 of our algorithm selects it as the first node, considering that $\phi(\bar{x})$ can behave like one sample in the feature space. Second, other steps of our algorithm allow the nodes identified later to have maximal dissimilarities with previously identified nodes. This enables the nodes to contain the maximum amount of information of the training sample set.

The proposed algorithm can be terminated by setting the number of nodes or some other condition such as by setting the ratio of the number of nodes to the total number of the training samples. For a real-world application, the proper number of nodes can be determined empirically. In the experimental section of this paper presented later, in order to observe the performance of the proposed method, we will show the variation with the number of nodes of the classification accuracy of ESKPCA.

Once the nodes have been identified, the sample $\phi(x)$ in the feature space can be represented by

$$f = \left[\sum_{j=1}^{s} \gamma_j^{(1)} k(x_j^0, x) \Big/ \sqrt{\lambda_1} \quad \sum_{j=1}^{s} \gamma_j^{(2)} k(x_j^0, x) \Big/ \sqrt{\lambda_2} \cdots \right.$$
$$\left. \sum_{j=1}^{s} \gamma_j^{(m)} k(x_j^0, x) \Big/ \sqrt{\lambda_m} \right]^T,$$

where $\gamma^{(i)} = [\gamma_1^{(i)} \quad \gamma_2^{(i)} \cdots \gamma_s^{(i)}]^T$. $\gamma^{(1)}, \gamma^{(2)}, \ldots, \gamma^{(m)}$ respectively stand for the first m eigenvectors corresponding to the first m largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ of the eigenvalue equation (6). To obtain the features of a sample, ESKPCA-based feature extraction should calculate s kernel functions in advance, whereas KPCA-based feature extraction should calculate N kernel functions.

It should be pointed out that when the center of a data distribution happens to lie within the distribution, our first step that takes the center of the samples as the first node is very reasonable. However, if the center is out of the data distribution, the first step seems to be problematic. As a result, the following strategy seems to be more reasonable: for a simple data distribution whose center is within the distribution, we choose the center of the samples as the first node. However, for arbitrary distributions we take the sample that is closest to the center as the first node.

4. Comparison Between Different Sparse KPCA Methods

In this section, we compare computational efficiencies of the feature extraction procedure and the training phase to produce the sparse KPCA of ESKPCA and other sparse KPCA methods. The training phase of a sparse KPCA method is the procedures to produce the sparse KPCA. The training phase of ESKPCA indeed consists of the first and second procedures to produce the sparse KPCA shown in Sec. 3. The feature extraction procedure of ESKPCA has a similar computational cost to those of other sparse KPCA such as IKPCA in Ref. 22. This is because the computational cost of feature extraction depends on the number of nodes, and both ESKPCA and IKPCA can produce a similar number of nodes that are much fewer than the samples of the training sample set. If both ESKPCA and IKPCA use s nodes, both of them should calculate s kernel functions, which need $O(s)$ operations, to perform feature extraction for a sample. In other words, they do this at the same computational cost. On the other hand, as shown later, ESKPCA has a much more computationally efficient training phase than other methods such as the methods proposed in Refs. 18–20.

We first analyze computational cost of the training phase of the IKPCA method proposed in Ref. 22 as follows: Most of the computational cost of the training phase of IKPCA comes from its procedure of identifying nodes. IKPCA identifies one node by assessing each candidate from the training sample set. For a certain candidate, IKPCA first produces an eigenvalue equation using the current candidate, the previously identified nodes, and all the training samples. Then IKPCA assesses the candidate by the eigenvalues of the produced eigenvalue equation. In the $(r+1)$th step of the procedure of identifying nodes, IKPCA solves the eigenvalues at the computational cost of $O((r+1)^3)$ (r is the number of the previously identified nodes) for assessing a certain candidate. This is also the major computational burden when identifying nodes. Since there are $N-r$ candidates, the computational cost of assessing all the candidates is $O((N-r)(r+1)^3)$. The computational cost of solving the eigenvalues dramatically increases with the increase of r. Moreover, when the procedure of identifying the $(r+1)$th node obtains the eigenvalue equations, it should perform the matrix operations $K_1(K_1)^T$ in advance at the computational cost of $O((N-r)r^2N)$, $r = 1, 2, \dots, s$.

K_1 and K_2 are defined as

$$K_1 = \begin{bmatrix} k(x_1, x_1^{ikpca}) \dots k(x_n, x_1^{ikpca}) \\ \cdots \quad \cdots \quad \cdots \\ k(x_1, x_r^{ikpca}) \dots k(x_n, x_r^{ikpca}) \end{bmatrix}$$

and

$$K_2 = \begin{bmatrix} k(x_1^{ikpca}, x_1^{ikpca}) \dots k(x_1^{ikpca}, x_r^{ikpca}) \\ \cdots \quad \cdots \quad \cdots \\ k(x_r^{ikpca}, x_1^{ikpca}) \dots k(x_r^{ikpca}, x_r^{ikpca}) \end{bmatrix},$$

respectively. x_j^{ikpca} stands for the jth node obtained using IKPCA. To obtain the K_1 and K_2 for all the candidates, IKPCA also should calculate a large number of kernel functions. We note that, for the computational costs shown above, $O((N-r)r^2N)$ is the largest. Thus, we can conclude that when IKPCA identifies the $(r+1)$th node, the needed computational cost is $O((N-r)r^2N)$. This shows that as r increases, the computational cost of identifying the next node also increases dramatically. As a result, the computational cost of the entire training phase will be quite high. For simplicity, we can consider that IKPCA completes the entire training phase at the computational cost of $O((N - s)s^2N)$, where s is the total number of the nodes identified.

For the method proposed by Franc,[18] when identifying the $(r + 1)$th node, the method will perform the matrix inversion operation at a computational cost of $O(N(r + 1)^3)$. Thus, the method identifies the $(r + 1)$th node at a computational cost of not less than $O(N(r+1)^3)$, $r = 1, 2, \ldots, s-1$. For simplicity, we can regard $O(Ns^3)$ as the computational cost of the training phase of this method, where s is the number of the identified nodes.

We analyze the computational costs of SKFA, AKFA, and naïve KPCA as follows. As we know, most of the computational cost of the training phase of naïve KPCA comes from its procedure of solving the eigenvalue equation. Naïve KPCA needs the computational cost of $O(N^3)$ to implement this procedure. When extracting features from a sample, naïve KPCA requires $O(N)$ operations to calculate the N kernel functions. We note that both SKFA and AKFA use an integrated process to work out the training phase and the consequent feature extraction. SKFA proposed by Smola *et al.*[19] extracts m features using $O(m^2N^2)$ operations. If the number of extracted features, m, is much smaller than the size of the training sample set, N, the computational cost of SKFA will be lower than naïve KPCA. On the other hand, once if $m > N^{1/2}$, the computational cost of SKFA will be greater than $O(N^3)$. AKFA proposed by Jiang *et al.*[20] aims to improve SKFA and extracts m features for a sample at the computational cost of $O(mN^2)$.

As for our approach, when identifying the node, we should calculate only the kernel functions of the candidate and each of the previous identified nodes, and it does not need to solve any eigenvalue equation. There is also not any matrix operation. When identifying the $(r+1)$th node, our approach should calculate only $3(N - r + 1)r$ kernel functions. As a result, compared to IKPCA and the method in Ref. 18, our approach identifies nodes at a much lower computational cost.

After all the nodes have been identified, our approach needs to solve only one eigenvalue equation in the form of (6), which allows the eigenvectors to be finally obtained. Our approach completes this at only a computational cost of $O(s^3)$, where s is the number of the identified nodes. In addition, our approach needs an extra computational cost of $O(s^2N)$ to produce the eigenvalue equation. Thus, we conclude that the entire training phase of our approach produces the sparse KPCA at a computational cost of $O(s^2N)$. As shown in the experimental section, s can be much smaller than N. As a result, the training phase of our approach usually needs a quite low computational cost and will be much more computationally efficient than SKFA, AKFA, and IKPCA. We summarize the computational costs of the node identifying procedure and the entire training phase of each method using Table 1.

5. Experiments

We tested ESKPCA and other methods using several benchmark datasets.[38] The dataset “Image” includes 20 subsets. Each of the other datasets includes 100 subsets.

Table 1. Computational costs of the node identifying procedure and the entire training phase of each method.

Methods	Procedure of Identifying the $(r+1)$th Node	Entire Training Phase
Naïve KPCA	/	$O(N^3)$
SKFA	/	$O(m^2N^2)$
AKFA	/	$O(mN^2)$
Method in Ref. 18	$O((r+1)^3N)$	$O(s^3N)$
IKPCA	$O((N-r)r^2N)$	$O((N-s)s^2N)$
ESKPCA	$3(N-r+1)r$ kernel functions	$O(s^2N)$

Each subset of these datasets consists of one training subset and one test subset. We adopted the Gaussian kernel function

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right).$$

σ^2 was set to the square of Frobenius norm of the covariance matrix of the first training subset. For every dataset, the first training subset was used to obtain the sparse KPCA and the samples of all the test subsets were used as test samples in the classification experiment. When using each method to conduct experiments, we extracted features of each sample from the first training subset and all the test subsets. We then exploited the nearest neighbor classifier to classify the test samples. For each dataset, since every test subset has a classification error rate, we show both the mean of the classification error rates of all the test subsets from a dataset and the standard deviation of the classification error rates using Tables 2–7. In each table, the number in the bracket denotes the standard deviation and the number before the bracket stands for the mean of the classification error rates. Hereafter "mean" and "standard deviation" represent the average value and standard variance of the classification error rates, respectively. The meaning of the column "Number of features (nodes)" in these tables is as follows: For naïve KPCA, it represents the number of the features extracted. For other methods, it simultaneously stands for the number of the exploited nodes and the extracted features. In other words, if s nodes were identified and employed by one sparse KPCA method, we

Table 2. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset "Image."

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
95	1.31 (0.57)	1.44 (0.61)	5.24 (2.00)
85	1.31 (0.57)	1.15 (0.49)	5.85 (2.15)
75	1.35 (0.59)	1.19 (0.52)	5.13 (1.83)
65	1.35 (0.59)	1.31 (0.62)	4.23 (1.61)

Note: In this table and other tables the nodes used for ESKPCA and IKPCA are as many as the features extracted.

Table 3. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset “Cancer.”

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
56	8.53 (3.00)	8.78 (3.14)	8.75 (3.33)
48	9.01 (3.11)	8.78 (3.14)	9.44 (3.50)
40	9.82 (3.32)	9.58 (3.40)	8.97 (3.34)
32	10.17 (3.38)	8.44 (3.04)	9.70 (3.48)
24	10.17 (3.38)	8.08 (3.01)	9.82 (3.53)

Table 4. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset “Heart.”

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
40	7.26 (2.48)	8.78 (2.93)	9.15 (3.00)
35	8.03 (2.62)	8.15 (2.75)	10.05 (3.08)
30	8.03 (2.62)	8.96 (3.06)	9.42 (2.83)
25	8.03 (2.62)	8.70 (3.04)	9.15 (2.95)
20	7.35 (2.47)	8.69 (3.02)	10.09 (3.05)

Table 5. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset “Banana.”

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
40	13.80 (0.20)	13.80 (0.20)	13.92 (0.20)
30	13.80 (0.20)	13.68 (0.20)	13.51 (0.20)
20	1.3.8 (0.20)	13.87 (0.20)	13.48 (0.20)
10	13.7 (0.20)	13.94 (0.22)	13.69 (0.19)

Table 6. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset “Thyroid.”

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
20	0.99 (0.84)	1.44 (1.14)	1.95 (1.20)
15	0.99 (0.84)	0.51 (0.65)	2.29 (1.41)
10	0.99 (0.84)	0.99 (0.92)	1.47 (1.06)

also extracted s features using this method and then classified the testing samples using the obtained features.

Tables 2–7 show that for all the databases except for “Banana” and “Heart,” ESKPCA can obtain a lower classification error rate than IKPCA. Especially, the experimental result of the dataset “Image” shows that ESKPCA can classify much more accurately than IKPCA. ESKPCA also has a similar classification

Table 7. Means and standard deviations of the error rates (%) of three feature extraction methods on the dataset "German."

Number of Features	Naïve KPCA-Based Feature Extraction	ESKPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
75	9.00 (3.15)	8.33 (2.73)	10.13 (3.54)
65	9.50 (3.25)	9.09 (3.10)	9.89 (3.35)
55	9.46 (3.28)	8.07 (2.80)	10.91 (3.64)
45	9.26 (3.23)	8.43 (2.89)	10.35 (3.55)
35	9.38 (3.19)	9.67 (3.33)	10.39 (3.63)
25	9.64 (3.28)	9.78 (3.35)	9.80 (3.40)

Table 8. The ratio of the sum of variances of the first feature, the first five features, and the first ten features to the sum of variances of all the features.

	ESKPCA-Based Feature Extraction	Naïve KPCA-Based Feature Extraction	IKPCA-Based Feature Extraction
Image	96.49%; 99.77%; 99.97%	78.32%; 93.09%; 97.44%	78.42%; 93.20%; 97.55%
Cancer	61.62%; 86.05%; 93.88%	26.62%; 51.36%; 65.36%	32.11%; 61.46%; 77.53%
Heart	78.56%; 92.11%; 96.65%	31.80%; 52.31%; 65.06%	39.92%; 64.84%; 79.21%
Banana	54.51%; 98.02%; 99.97%	45.15%; 89.80%; 98.25%	45.18%; 89.84%; 98.29%
Thyroid	89.68%; 99.07%; 99.93%	76.32%; 92.53%; 97.57%	76.96%; 93.24%; 98.15%
German	50.89%; 61.73%; 70.07%	11.21%; 20.55%; 28.28%	24.13%; 42.95%; 57.73%

performance to naïve KPCA. Most importantly, as presented earlier, ESKPCA needs a much lower computational cost than other sparse KPCA methods.

We use Table 8 to show the ratio of the sum of variances of the first number of features to the sum of variances of all the features (referred to as variance sum). For each method, we show the ratios on the first feature, the first five, and the first ten features, respectively. For example, in Table 8, 96.49%; 99.77%; 99.97% mean that in our approach the first feature, the first five, and the first ten features can capture the 96.49%, 99.77%, and 99.97% variance sum of all the features of the training samples in dataset "Image," respectively. We also say that the approach can use first feature, the first five, and the first ten features to capture the 96.49%, 99.77%, and 99.97% "energy" of the samples in dataset "Image," respectively. Moreover, we see that using the same number of features, our approach can capture more energy than naïve KPCA and IKPCA. For each dataset, the number of nodes exploited in both our approach and IKPCA is the largest number of nodes shown in Tables 2–7. For instance, both our approach and IKPCA used 95 nodes to perform the experiment on dataset "Image."

To visually illustrate the distribution of training samples as shown in Ref. 39, we use Fig. 1 to show the 2D distribution of samples of a two-class toy dataset (the red and blue circles represent samples from the two classes, respectively). Figures 2 and 3, respectively show the distributions of the first 2D features of the samples obtained using naïve KPCA and ESKPCA. It is clear that ESKPCA and naïve KPCA can produce a similar data distribution.

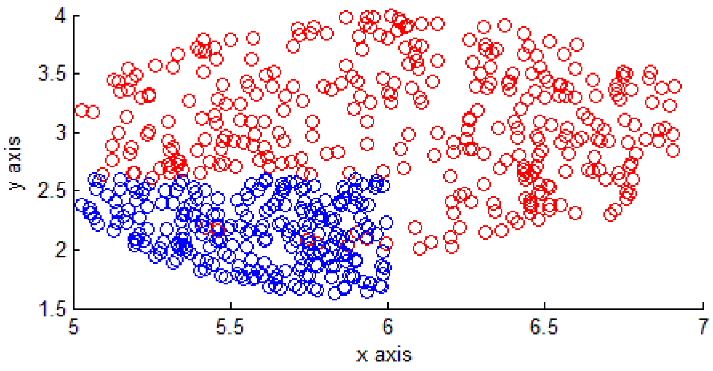

Fig. 1. 2D distribution of samples of the toy dataset.

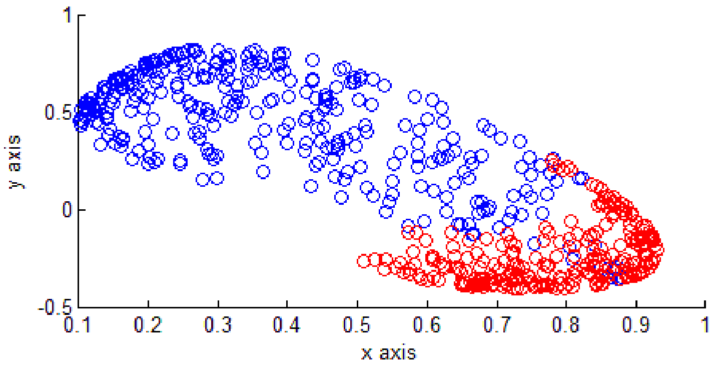

Fig. 2. The distribution of the first 2D features of the samples, obtained using naïve KPCA.

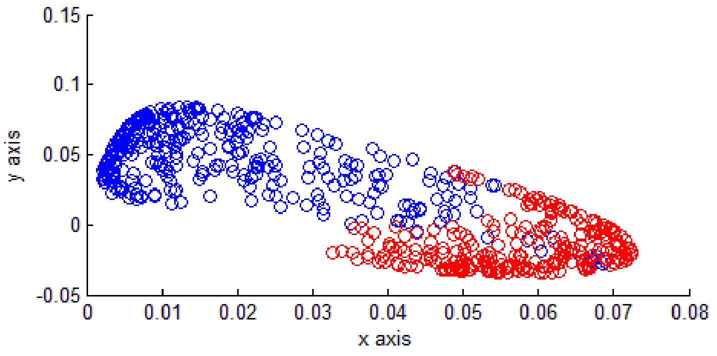

Fig. 3. The distribution of the first 2D features of the samples, obtained using ESKPCA. ESKPCA took 30% of the original samples as nodes.

6. Conclusions and Discussion

In this paper, we propose a novel computationally efficient sparse KPCA method namely ESKPCA. ESKPCA consists of two procedures. The first procedure selects a small number of nodes from the training sample set using the criterion that nodes should be dissimilar as much as possible, which actually allows the nodes to convert the maximum amount of information of the training sample set to the consequent eigenvector. This procedure first takes the mean of all the training samples as the first node. Then this procedure identifies other nodes with the requirement that it should be far from previously identified nodes as much as possible. Requiring the feature extracted using the approximate eigenvector has the maximum variance, the second procedure obtains the eigenvalue equation of ESKPCA. By solving the eigenvaule problem, the second procedure then gets the optimal approximate eigenvectors.

Since the nodes are much fewer than the training samples, the ESKPCA-based feature extraction process is much more efficient than naïve KPCA-based feature extraction. More importantly, among all known sparse KPCA methods, ESKPCA identifies nodes at the lowest computational cost. Indeed, in terms of the computational cost of the entire training phase, ESKPCA is also the most efficient among all the known sparse KPCA methods. Moreover, experimental result shows that ESKPCA obtains a high classification accuracy.

Acknowledgments

This work was partly supported by Program for New Century Excellent Talents in University (Nos. NCET-08-0156 and NCET-08-0155), NSFC under grant Nos. 61071179, 60803090, 60902099, and 61001037, as well as the Fundamental Research Funds for the Central Universities (HIT.NSRIF. 2009130).

References

1. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification* (China Machine Press, Beijing, 2004).
2. M. Kirby and L. Sirovich, Application of the KL procedure for the characterization of human faces, *IEEE Transactions on Pattern Analysis and Machine Intellegence* **12**(1) (1990) 103–108.
3. M. Turk and A. Pentland, Face recognition using eigenfaces, *Proceedings of IEEE Conference On Computer Vision and Pattern Recognition* (1991) 586–591.
4. J. Yang and J.-Y. Yang, Why can LDA be performed in PCA transformed space? *Pattern Recognition* **36**(2) (2003) 563–566.
5. C. Liu, Gabor-based kernel PCA with fractional power polynomial models for face recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**(5) (2004) 572–581.
6. Y. Xu, C.-L. Lin and W. Zhao, Producing computationlly efficient KPCA-based feature extraction for classification problems, *Electronics Letters* **46**(6) (2010).

7. Z. Jin, F. Davoine, Z. Lou and J.-Y. Yang, A novel PCA-based bayes classifier and face analysis, *IAPR International Conference on Biometrics* (ICB2006), Hong Kong, 5–7 January (2006).
8. B. Scholkopf, A. Smola and K. R. Muller, Nonlinear component analysis as a kernel eigenvalue problem, *Neural Computation* **10**(5) (1998) 1299–1319.
9. B. Scholkopf, A. Smola and K. R. Muller, Kernel principal component analysis, *Artificial Neural Networks* — ICANN'97, Berlin, 1997, pp. 583–588.
10. Y. Shi, J. Wan, X. Zhang, G. Kou, Y. Peng, Z. Cao and Y. Guo, Comparison study of two kernel-based learning algorithms for predicting the distance range between antibody interface residues and antigen surface, *International Journal of Computer Mathematics* **84**(5) (2007) 697–707.
11. J. Li, Z. Chen, L. Wei, W. Xu and G. Kou, Feature selection via least squares support feature machine, *International Journal of Information Technology and Decision Making* **6**(4) (2007) 671–686.
12. Y. Xu, J.-Y. Yang, J. Lu and D.-J. Yu, An efficient renovation on kernel fisher discriminant analysis and face recognition experiments, *Pattern Recognition* **37** (2004) 2091–2094.
13. Y. Xu, J.-Y. Yang and J. Yang, A reformative kernel fisher discriminant analysis, *Pattern Recognition* **37** (2004) 1299–1302.
14. Y. Xu, D. Zhang, Z. Jin, M. Li and J.-Y. Yang, A fast kernel-based nonlinear discriminant analysis for multi-class problems, *Pattern Recognition* **39**(6) (2006) 1026–1033.
15. B. Scholkopf, S. Mika, C. Burges, P. Knirsch, K. R. Muller, G. Ratsch and A. Smola, Input space versus feature space in kernel-based methods, *IEEE Transactions on Neural Networks* **10**(5) (1999) 1000–1017.
16. C. J. C. Burges and B. Scholkopf, Improving the accuracy and speed of support vector learning machines, *Advances in Neural Information Processing Systems*, Vol. 9, eds. M. Mozer, M. Jordan and T. Petsche (MIT Press, Cambridge, MA, 1997), pp. 375–381.
17. M. E. Tipping, Sparse kernel principal component analysis, in *NIPS 2000: Neural Information Processing Systems*, eds. T. K. Leen, T. G. Dietterich and V. Tresp (MIT Press, 2000), pp. 633–639.
18. V. Franc and V. Hlavac, Greedy algorithm for a training set reduction in the kernel methods, in *CAIP 2003: Computer Analysis of Images and Patterns*, Vol. 2756 of Lecture Notes in Computer Science (Springer-Verlag, Berlin, Germany, 2003), pp. 426–433.
19. A. Smola, O. Mangasarian and B. Scholkopf, Sparse kernel feature analysis, Technical Report 99–04 (Data Mining Institute, University of Wisconsin, Madison, 1999).
20. X. Jiang, Y. Motai, R. R. Snapp and Xingquan Zhu, Accelerated kernel feature analysis, *CVPR* **1** (2006) 109–116.
21. C. J. C. Burges, Simplified support vector decision rules, *Proceedings of 13th International Conference on Machine Learning*, ed. L. Saitta (Morgan Kaufmann, San Mateo, CA, 1996), pp. 71–77.
22. Y. Xu, D. Zhang, F. Song, J.-Y. Yang, Z. Jing and M. Li, A method for speeding up feature extraction based on KPCA, *Neurocomputing* **70**(4–6) (2007) 1056–1061.
23. T.-J. Chin and D. Suter, Incremental kernel principal component analysis, *IEEE Transactions on Image Processing* **16**(6) (2007) 1662–1674.
24. S.-W. Kim and B. J. Oommen, On using prototype reduction schemes to optimize kernel-based fisher discriminant analysis, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **38**(2) (2008) 564–570.

25. K. I. Kim and Y. Kwon, Example-based learning for single-image super-resolution pattern recognition, *Proceedings of the 30th DAGM Symposium*, ed. G. Rigoll (Springer, Berlin, Germany, 2008), pp. 456–463.
26. C. García-Osorio and C. Fyfe, Regaining sparsity in kernel principal components, *Neurocomputing* **67** (2005) 398–402.
27. M. Yang, N. Ahuja and D. Kriegman, Face recognition using kernel eigenfaces, *ICIP 2000: IEEE International Conference on Image Processing*, Vol. 1, Vancouver, Canada, 2000, pp. 37–40.
28. B. Scholkopf and A. Smola, *Learning With kernels* (MIT Press, Cambridge, MA, 2002).
29. N. Cristianini and J. Shawe-Taylor, *Kernel Methods for Pattern Analysis* (Cambridge University Press, New York, 2004).
30. M. Girolami, Mercer kernel based clustering in feature space, *IEEE Transactions on Neural Networks* **13**(4) (2002) 669–688.
31. G. Gnecco and M. Sanguineti: Accuracy of suboptimal solutions to kernel principal component analysis, *Computer Optimization Applications* **42** (2009) 265–287.
32. S.-W. Kim and B. John Oommen, On optimizing kernel-based fisher discriminant analysis using prototype reduction schemes, *Lecture Notes in Computer Science on Structural, Syntactic, and Statistical Pattern Recognition* **4109/2006** (2006) 826–834.
33. P. Georgiev, P. Pardalos, F. J. Theis, A. Cichocki and H. Bakardjian, Sparse component analysis: A new tool for data mining, in *Data Mining in Biomedicine*, eds. P. Pardalos, V. Boginski and A. Vazacopoulos (Springer, Berlin, 2007), pp. 91–116.
34. G. Gnecco and M. Sanguineti, Error bounds for suboptimal solutions to kernel principal component analysis, *Optimization Letters* **4** (2010) 197–210.
35. Y. Washizawa, Subset kernel PCA for pattern recognition, in *Proceedings of the IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)* (2010), pp. 162–169.
36. Z. Hussain and J. Shawe-Taylor, Theory of matching pursuit, *NIPS (2008)* (2009), pp. 721–728.
37. D. Zhang and W. Shi, An improved kernel principal component analysis for large-scale data set, *Lecture Notes in Computer Science* **6064** (2010) 9–16.
38. http://ida.first.fraunhofer.de/projects/bench/benchmarks.htm
39. H. Hoffmann, Kernel PCA for novelty detection, *Pattern Recognition* **40**(3) (2007) 863–874.