Accelerated Kernel Feature Analysis

Xianhua Jiang †, Robert R. Snapp ††, Yuichi Motai†, and Xingquan Zhu††
† Dept. of Electrical and Computer Engineering
†† Dept. of Computer Science
University of Vermont, Burlington,VT, 05405
{xjiang, snapp, ymotai, xqzhu}@cems.uvm.edu

Abstract

A fast algorithm, Accelerated Kernel Feature Analysis (AKFA), that discovers salient features evidenced in a sample of n unclassified patterns, is presented. Like earlier kernel-based feature selection algorithms, AKFA implicitly embeds each pattern into a Hilbert space, H, induced by a Mercer kernel. An ℓ-dimensional linear subspace of H is iteratively constructed by maximizing a variance condition for the nonlinearly transformed sample. This linear subspace can then be used to define more efficient data representations and pattern classifiers. AKFA requires $\mathcal{O}(\ell n^2)$ operations, as compared to $\mathcal{O}(n^3)$ for Schölkof, Smola, and Müller's Kernel Principal Component Analysis (KPCA), and $\mathcal{O}(\ell^2 n^2)$ for Smola, Mangasarian, and Schölkopf's Sparse Kernel Feature Analysis (SKFA). Numerical experiments show that AKFA can generate more concise feature representations than both KPCA and SKFA, and demonstrate that AKFA obtains similar classification performance as KPCA for a face recognition problem.

1 Introduction

Automatic feature extraction has proven to be one of the most elusive problems in the field of pattern recognition. Although finding concise and informative representations of pattern data is an essential first step in designing a pattern recognition system, most research activity has centered on the more tractable tasks of developing and analyzing pattern classification algorithms[3]. Despite modest successes of principal component analysis [9, 7, 15, 10], self-organizing feature maps [8], and vector quantization [6] in the context of dimensionality reduction, feature selection is usually a manual process that demands an understanding of the problem domain (e.g., handwritten character or face recognition), as well as much trial and error.

Capitalizing on the success of kernel methods in pattern classification, Schölkoph, Smola and Müller [11] developed and studied a feature selection algorithm in which principal component analysis is effectively applied to a sample of n, d-dimensional patterns that are first injected into a high-dimensional Hilbert space using a nonlinear embedding. (Heuristically, injecting input patterns into a high-dimensional space may elucidate salient nonlinear features in the input distribution, in the same way that nonlinearly separable classification problems may become linearly separable in higher dimensional spaces.) Both the principal component analysis and the nonlinear embedding are facilitated by a Mercer kernel of two arguments $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, which effectively computes the inner product of the transformed arguments. This algorithm, called Kernel Principal Component Analysis (KPCA), thus avoids the problem of representing transformed vectors in the Hilbert space, and enables the computation of the inner-product of two transformed vectors of arbitrarily high dimension in constant time. Nevertheless, KPCA has two deficiencies: (*i*) the computation of the principal components involves the solution of an eigenvalue problem that requires $\mathcal{O}(n^3)$ computations, and (*ii*) each principal component in the Hilbert space depends on every one of the n input patterns, which defeats the goal of obtaining both an informative and concise representation.

Both of these deficiencies have been addressed in subsequent investigations that seek sets of salient features that only depend upon sparse subsets of transformed input patterns. Tipping [14] applied a maximum-likelihood technique to approximate the transformed covariance matrix in terms of such a sparse subset. Franc and Hlaváč [4] proposed a greedy method, which approximates the mapped space representation by selecting a subset of input data. It iteratively extracts the data in the mapped space until the reconstruction error in the mapped high-dimensional space falls below a threshold value. Its computational complexity is $\mathcal{O}(nm^3)$, where n is the number of input patterns, and m is the cardinality of the subset. Zheng [17] split the input data into M groups of similar size, and then applied KPCA to each group. A set of eigenvectors was obtained

Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)
0-7695-2597-0/06 $20.00 © 2006 IEEE

for each group. KPCA was then applied to a subset of these eigenvectors to obtain a final set of features. Although these studies propose useful approaches, none provides a method that is both computationally efficient and accurate.

To avoid the $\mathcal{O}(n^3)$ eigenvalue problem, Smola, Mangasarian and Schölkoph [13] proposed Sparse Kernel Feature Analysis (SKFA),which extracts ℓ features, one by one, using an l_1 constraint on the expansion coefficients. SKFA requires only $\mathcal{O}(\ell^2 n^2)$ operations, and is thus a significant improvement over KPCA if number of dominant features is much less than the data size. However, if $\ell > \sqrt{n}$, then the computational cost of SKFA likely exceeds that of KPCA. Schölkopf and Smola [12] introduced two efficient approximations of SKFA: (*i*) a "Probabilistic Speedup", which approximates the sums by computing only a subset of the terms, and (*ii*) a Quantile Trick, which computes a random subsample of all possible directions. Both methods reduce the accuracy of SKFA.

In the following, we propose an Accelerated Kernel Feature Analysis (or AKFA), which generates ℓ sparse features from a data set of n patterns using $\mathcal{O}(\ell n^2)$ operations. Since AKFA is based on both KPCA and SKFA, we analyze the former algorithm in Section 2, and the latter in Section 3. AKFA is then described and analyzed in Section 4. (We also describe a compressed approximation of AKFA, called ACKFA, which terminates early.) Section 5 summarizes our numerical experiments which (*i*) confirm the run-time complexity of each of the four algorithms, (*ii*) demonstrate that AKFA can generate more informative and concise features than KPCA and SKFA, and (*iii*) illustrate that AKFA obtains similar classification performance as KPCA for a face recognition problem. Our conclusions appear in Section 6.

2 Kernel PCA (KPCA)

Using Mercer's theorem (cf., [2]), a nonlinear, positive-definite kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ of an integral operator, e.g., $k(x, y) = \exp\{-\|x - y\|^2\}$, computes the inner product of the transformed vectors $\langle \Phi(x), \Phi(y) \rangle$, where $\Phi : \mathbb{R}^d \to H$ denotes a nonlinear embedding (induced by k) into a possibly infinite dimensional Hilbert space, H. Given n points $\mathcal{X}_n = \{x_i \in \mathbb{R}^d | i = 1, \ldots, n\}$, the image $\mathcal{Y}_n = \{\Phi(x_i) \mid i = 1, \ldots, n\}$ of $\mathcal{X}_n$ spans a linear subspace of at most $(n-1)$ dimensions. Heuristically, the dominant linear correlations in the distribution of the image may elucidate important nonlinear dependencies in the original data sample $\mathcal{X}_n$.

Kernel Principal Component Analysis (KPCA) uses a Mercer kernel to perform a linear principal component analysis of this transformed image. Without loss of generality, we assume that the image of the data has been centered so that its scatter matrix in H is given by $S = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^{\mathrm{T}}$. Eigenvalues λ_j and eigenvectors e_j are obtained by solving

$$\lambda_j e_j = S e_j = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^{\mathrm{T}} e_j = \sum_{i=1}^{n} \langle e_j, \Phi(x_i) \rangle \Phi(x_i), \tag{1}$$

for $j = 1, \ldots, n$. Since Φ is not known, (1) must be solved indirectly. Letting, $a_{ji} = \frac{1}{\lambda_j} \langle e_j, \Phi(x_i) \rangle$, (1) becomes

$$e_j = \sum_{i=1}^{n} a_{ji} \Phi(x_i). \tag{2}$$

Multiplying by $\Phi(x_q)^{\mathrm{T}}$ on the left, for $q = 1, \cdots, n$, yields

$$\lambda_j \langle \Phi(x_q), e_j \rangle = \sum_{i=1}^{n} \langle e_j, \Phi(x_i) \rangle \langle \Phi(x_q), \Phi(x_i) \rangle. \tag{3}$$

Substitution of (2) into (3) produces

$$\lambda_j \Big\langle \Phi(x_q), \sum_{i=1}^{n} a_{ji} \Phi(x_i) \Big\rangle = \sum_{i=1}^{n} \left(\Big\langle \sum_{k=1}^{n} a_{jk} \Phi(x_k), \Phi(x_i) \Big\rangle \langle \Phi(x_q), \Phi(x_i) \rangle \right),$$

which can be rewritten as, $\lambda_j K a_j = K^2 a_j$,where K is a $n \times n$ Gram matrix, with the element $k_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle$, and $a_j = [a_{j1}\ a_{j2}\ \cdots\ a_{jn}]^{\mathrm{T}}$. The latter is a dual eigenvalue problem equivalent to the problem

$$\lambda_j a_j = K a_j. \tag{4}$$

Since $\|e_j\|^2 = \Big\langle \sum_{i=1}^{n} a_{ji} \Phi(x_i), \sum_{i=1}^{n} a_{ji} \Phi(x_i) \Big\rangle = \langle a_j, K a_j \rangle$ $= \lambda_j \|a_j\|^2$, the normalization of each eigenvector ($\|e_j\| = 1$) requires that $\|a_j\|^2 = 1/\lambda_j$.

In the following, we choose a Gaussian kernel, i.e.,

$$k_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = \exp\left(-\frac{1}{2\sigma^2} \|x_i - x_j\|^2 \right). \tag{5}$$

If the image of $\mathcal{X}_n$ is not centered in the Hilbert space, we need to use the centered Gram Matrix deduced by Smola, Mangasarian and Schölkoph [13]:

$$\hat{K} = K - KT - TK + TKT$$

where K is the Gram Matrix of uncentered data, and

$$T = \begin{bmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \cdots & \cdots & \cdots \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix}_{n \times n}$$

Keeping the ℓ eigenvectors associated with the ℓ largest eigenvalues, we can reconstruct data in the mapped space:

$$\Phi_i' = \sum_{j=1}^{\ell} \langle \Phi_i, e_j \rangle e_j = \sum_{j=1}^{\ell} \beta_{ji} e_j,$$

where $\beta_{ji} = \langle \Phi_i, \sum_{k=1}^{n} a_{jk}\Phi_k \rangle = \sum_{k=1}^{n} a_{jk}k_{ik}$. The reconstruction square error of each data $\Phi_i, i = 1, \cdots, n$, is

$$\text{Err}_i = \|\Phi_i - \Phi_i^{'}\|^2 = k_{ii} - \sum_{j=1}^{\ell} \beta_{ji}^2.$$

The mean square error is $\text{MErr} = \frac{1}{n}\sum_{i=1}^{n} \text{Err}_i$. Using (4), $\beta_{ji} = \lambda_j a_{ji}$. Therefore, the mean square reconstruction error is $\text{MErr} = \frac{1}{n}\sum_{i=1}^{n}(k_{ii} - \sum_{j=1}^{\ell} \lambda_j^2 a_{ji}^2)$. Since $\sum_{i=1}^{n} k_{ii} = \sum_{i=1}^{n} \lambda_i$, and $\sum_{i=1}^{n} a_{ji}^2 = \|a_j\|^2 = 1/\lambda_j$, $\text{MErr} = \frac{1}{n}\sum_{i=\ell+1}^{n} \lambda_i$.

Kernel PCA can now be summarized: First, calculate the Gram matrix using (5), which contains the inner products between pairs of image vectors. Then use (4) to get the coefficient vectors a_j, for $j = 1, \ldots, n$. The projection of a test point $x \in \mathbb{R}^d$ along the j-th eignevector is

$$\langle e_j, \Phi(x) \rangle = \sum_{i=1}^{n} a_{ji} \langle \Phi(x_i), \Phi(x) \rangle = \sum_{i=1}^{n} a_{ji} k(x, x_i).$$

Because the above implicitly contains an eigenvalue problem of rank n, the computational complexity of Kernel PCA is $\mathcal{O}(n^3)$. In addition, each resulting eigenvector is represented as a linear combination of n terms. Thus, all data contained in $\mathcal{X}_n$ must be retained, which is computationally cumbersome, and unacceptable for incremental learning.

3 Sparse Kernel Feature Analysis (SKFA)

As mentioned in the introduction, several investigations have addressed the computational costs of KPCA, associated with both time complexity and data retention requirements. In particular, Sparse Kernel Feature Analysis (SKFA) [13] extracts the features one by one in order of decreasing projection variance. Another advantage is that the ℓ features of SKFA only depend on ℓ elements of $\mathcal{X}_n$, which is extremely useful for on-line learning.

In the following, we let $v_i \in H$, for $i = 1, \ldots, \ell$ denote the features selected by SKFA. (We intentionally avoid using e_i, in order to distinguish these features from the eigenvectors obtained using KPCA.) Again, we analyze the scatter matrix of the image data. Following (1), with e_j replaced by v_j, we obtain $v_j^{\mathrm{T}} \lambda_j v_j = v_j^{\mathrm{T}} (\sum_{i=1}^{n} \langle v_j, \Phi(x_i) \rangle \Phi(x_i))$, where v_j is the jth feature with unit length. Thus, $\lambda_j = \sum_{i=1}^{n} \langle v_j, \Phi(x_i) \rangle^2$. Therefore, the first feature, corresponding to the maximum eigenvalue, is chosen as the direction with the maximum projected variance:

$$v_1 = \underset{\|v\|^2=1}{\arg\max} \frac{1}{n} \sum_{i=1}^{n} |\langle v, \Phi(x_i) \rangle|^2. \tag{6}$$

The global solution of (6), v_1, needs to satisfy an l_2 normalization constraint, i.e., unit Euclidian length. Changing the l_2 constraint to an l_1 constraint leads to a vertex solution. The l_1 constraint assumed by SKFA is

$$V_{l_1} = \Big\{ \sum_{j=1}^{n} a_j \Phi_j \;\Big|\; \sum_{j=1}^{n} |a_j| \leq 1 \Big\}. \tag{7}$$

The first feature selected by SKFA satisfies

$$v_1 = \underset{v \in V_{l_1}}{\arg\max} \frac{1}{n} \sum_{i=1}^{n} |\langle v, \Phi(x_i) \rangle|^2.$$

Smola et al. showed that this feature corresponds to an element of the image $\mathcal{Y}_n$, whence

$$v_1 = \underset{\Phi(x_j) \in V_{l_1}}{\arg\max} \frac{1}{n} \sum_{i=1}^{n} |\langle \Phi(x_j), \Phi(x_i) \rangle|^2. \tag{8}$$

Subsequent features are obtained iteratively. Suppose, $i-1$ features $\{v_t \in H | t = 1, \ldots, i-1\}$ have been found, then project each image $\Phi_j = \Phi(x_j)$ into the orthogonal subspace to obtain

$$\Phi_j^i = \Phi_j - \sum_{t=1}^{i-1} v_t \frac{\langle \Phi_j, v_t \rangle}{\|v_t\|^2} = \Phi_j - \sum_{t=1}^{i-1} \frac{a_{tj} v_t}{\|v_t\|^2}, \tag{9}$$

where $a_{tj} = \langle \Phi_j, v_t \rangle$. Then normalize Φ_j^i by the l_1 constraint in (7). The projection variance of the normalized Φ_j^i with all $\Phi_k, k = 1, \cdots, n$ is then calculated. Finally one identifies the maximum projection variance and selects the corresponding Φ_j^i as the ith feature, v_i.

Based on the ℓ features extracted by SKFA, each training data in the mapped space can be reconstructed by

$$\Phi_i^{'} = \sum_{j=1}^{\ell} \frac{\langle \Phi_i, v_j \rangle v_j}{\|v_j\|^2} = \sum_{j=1}^{\ell} \frac{a_{ji} v_j}{\|v_j\|^2},$$

where a_{ji}, the projection of i-th training data on j-th feature, is stored after extracting the j-th feature. According to (9), the feature set $\{v_1, \ldots, v_\ell\}$ only depends upon the set of ℓ image vectors, $\{\Phi_{\text{idx}(i)}, i = 1, \cdots, \ell\}$, where $\text{idx}(i)$ denotes the subscript of the projected image Φ_j^i that was selected when constructing v_i. Therefore, after training, we only need to retain the ℓ input vectors $\{x_{\text{idx}(i)} \in \mathbb{R}^d | i = 1, \ldots, \ell\}$, where ℓ is the number of features extracted.

In this way, SKFA extracts ℓ features, where one assumes $\ell \ll n$. As $\mathcal{O}(in^2)$ operations are required to extract the i-th feature, the total computational cost for ℓ features is $\mathcal{O}(\ell^2 n^2)$, which is an improvement over the $\mathcal{O}(n^3)$ operations required by kernel PCA.

4 Accelerated Kernel Feature Analysis (AKFA)

To improve the efficiency and accuracy of SKFA, we propose an *Accelerated Kernel Feature Analysis* (AKFA)

that (*i*) saves computation time by iteratively updating the Gram Matrix, (*ii*) normalizes the images with the l_2 constraint before the l_1 constraint is applied, and (*iii*) optionally discards data that falls below a threshold magnitude δ during updates.

First, instead of extracting features directly from the original mapped space, AKFA extracts the i-th feature based on the i-th updated Gram matrix K^i, where each element $k^i_{jk} = \langle \Phi^i_j, \Phi^i_k \rangle$. Since $\Phi^i_j = \Phi^{i-1}_j - v_{i-1}\langle \Phi^{i-1}_j, v_{i-1} \rangle$,

$$\begin{aligned} k^i_{jk} &= \langle \Phi^{i-1}_j, \Phi^{i-1}_k \rangle - \langle \Phi^{i-1}_j, v_{i-1} \rangle \langle \Phi^{i-1}_k, v_{i-1} \rangle \\ &= k^{i-1}_{jk} - \frac{\langle \Phi^{i-1}_j, \Phi^{i-1}_{\mathrm{idx}(i-1)} \rangle \langle \Phi^{i-1}_k, \Phi^{i-1}_{\mathrm{idx}(i-1)} \rangle}{\|\Phi^{i-1}_{\mathrm{idx}(i-1)}\|^2} \\ &= k^{i-1}_{jk} - \frac{k^{i-1}_{j,\mathrm{idx}(i-1)} k^{i-1}_{k,\mathrm{idx}(i-1)}}{k^{i-1}_{\mathrm{idx}(i-1),\mathrm{idx}(i-1)}}. \end{aligned} \tag{10}$$

By updating the Gram Matrix, we don't need to save the projection of each individual data on all previous features. The computational cost for extracting i-th feature becomes $\mathcal{O}(n^2)$, instead of $\mathcal{O}(in^2)$ as in SKFA.

The second improvement is to revise the l_1 constraint. As shown in (8), SKFA treats each individual sample data as a possible direction, and computes the projection variances with all data. Since SKFA includes its length in its projection variance calculation, it is biased to select vectors with larger magnitude. From the objective function (6), we know that we are actually looking for a direction with unit length. When we choose a image vector as a possible direction, we only consider its direction ignoring the length, which improves the accuracy of the features. Therefore, in our AKFA algorithm, we replace the l_1 constraint (7) by

$$V^i_{l_1} = \left\{ \sum_{j=1}^{n} a_j \frac{\Phi^i_j}{\|\Phi^i_j\|} \middle| \sum_{j=1}^{n} |a_j| \le 1 \right\}.$$

The i-th feature is extracted by

$$v_i = \arg\max_{v \in V^i_{l_1}} \frac{1}{n} \sum_{j=1}^{n} |\langle v, \Phi^i_j \rangle|^2.$$

Since v_i is extracted from Φ^i space, the solution is located on one of the $\widehat{\Phi}^i_j = \Phi^i_j / \|\Phi^i_j\|$ for $j = 1, \cdots, n$. Equation (8) reduces to

$$v_i = \arg\max_{\widehat{\Phi}^i_j} \frac{1}{n} \sum_{t=1}^{n} |\langle \Phi^i_t, \widehat{\Phi}^i_j \rangle|^2 = \arg\max_{\widehat{\Phi}^i_j} \frac{1}{n\, k^i_{jj}} \sum_{t=1}^{n} {k^i_{jt}}^2. \tag{11}$$

Each $\widehat{\Phi}^i_j$ satisfies the l_1 constraint, so the normalization step that appears in SKFA is not required.

Let $\Phi^i_{\mathrm{idx}(i)}$ denote the image vector corresponding to the i-th feature. Suppose we have selected $(i-1)$ features with $\mathbf{V_{(i-1)}} = \mathbf{\Phi_{(i-1)}} \mathbf{C_{(i-1)}}$, where $\mathbf{V_{(i-1)}} = [v_1 \; v_2 \; \cdots \; v_{(i-1)}]$, $\mathbf{\Phi_{(i-1)}} = [\Phi_{\mathrm{idx}(1)} \; \Phi_{\mathrm{idx}(2)} \; \cdots \; \Phi_{\mathrm{idx}(i-1)}]$, and $\mathbf{C_{(i-1)}}$ is the coefficient matrix,which is upper-triangular. Then $\Phi^i_{\mathrm{idx}(i)} = \Phi_{\mathrm{idx}(i)} - \sum_{t=1}^{i-1} \langle \Phi_{\mathrm{idx}(i)}, v_t \rangle v_t$. Let's study the second term:

$$\begin{aligned} \sum_{t=1}^{i-1} \langle \Phi_{\mathrm{idx}(i)}, v_t \rangle v_t &= \sum_{t=1}^{i-1} v_t v_t^{\mathrm{T}} \Phi_{\mathrm{idx}(i)} \\ &= \mathbf{\Phi_{(i-1)}} \mathbf{C_{i-1}} \mathbf{C_{i-1}^{\mathrm{T}}} \mathbf{K_{idx(i)}} \end{aligned}$$

where,

$$\mathbf{K_{idx(i)}} = [k_{\mathrm{idx}(i),\mathrm{idx}(1)} k_{\mathrm{idx}(i),\mathrm{idx}(2)} \cdots k_{\mathrm{idx}(i),\mathrm{idx}(i-1)}]^{\mathrm{T}}.$$

Therefore,

$$v_i = \frac{\Phi_{\mathrm{idx}(i)} - \mathbf{\Phi_{(i-1)}} \mathbf{C_{i-1}} \mathbf{C_{i-1}^{\mathrm{T}}} \mathbf{K_{idx(i)}}}{\sqrt{k^i_{\mathrm{idx}(i),\mathrm{idx}(i)}}}$$

Let

$$\mathbf{C}_{i,i} = \left(k^i_{\mathrm{idx}(i),\mathrm{idx}(i)} \right)^{-1/2} \tag{12}$$

and

$$\mathbf{C}_{1:(i-1),i} = -\mathbf{C}_{i,i} \mathbf{C_{(i-1)}} \mathbf{C^{\mathrm{T}}_{(i-1)}} \mathbf{K_{idx(i)}}, \tag{13}$$

then $\mathbf{V_i} = \mathbf{\Phi_i} \mathbf{C_i}$, where $\mathbf{V_i} = [\mathbf{V_{(i-1)}}, v_i]$, and

$$\begin{aligned} \mathbf{\Phi_i} &= [\mathbf{\Phi_{(i-1)}}, \Phi_{idx(i)}], \\ \mathbf{C_i} &= \begin{pmatrix} \mathbf{C_{(i-1)}} & \mathbf{C}_{1:(i-1),i} \\ \mathbf{0} & \mathbf{C}_{i,i} \end{pmatrix}. \end{aligned}$$

The third improvement is to discard negligible data and thereby eliminate unnecessary computations. In the i-th updated Gram matrix K^i, defined in (10), the diagonal elements k^i_{jj} represents the reconstruction error of the j-th data point with respect to the previous $(i-1)$ features. If k^i_{jj} is relatively small, then the previous $(i-1)$ features contain most of the information that would be acquired from this image vector. One can therefore optionally discard image points that satisfy $k^i_{jj} < \delta$, for some predetermined $\delta > 0$. This variation, which we call *Accelerated Cut-off Kernel Feature Analysis* (ACKFA), is useful if the data size is huge. It can also be used as a criteria to stop extracting features if one is unsure of the number of features that should be selected.

The entire algorithm (ACKFA) is summarized below:

Step 1 Compute the $n \times n$ Gram matrix $k_{ij} = k(x_i, x_j)$, where n is the number of input vectors. This part requires $\mathcal{O}(n^2)$ operations.

Step 2 Let ℓ denote the number of features to be extracted. Initialize the $\ell \times \ell$ coefficient matrix $\mathbf{C}$ to $\mathbf{0}$, and $\mathrm{idx}(\cdot)$ as an empty list which will ultimately store the indices of the selected image vectors. Initialize the threshold value δ for the reconstruction error. (For AKFA, set $\delta = 0$.) The overall cost is $\mathcal{O}(\ell^2)$.

Step 3 For $i = 1$ to ℓ repeat:

1. Using the i-th updated $\mathbf{K^i}$ matrix, extract the i-th feature using (11). If $K^i_{jj} < \delta$, then discard jth column and jth row vector without calculating the projection variance. Use $\mathrm{idx}(i)$ to store the index. This step requires $\mathcal{O}(n^2)$ operations.
2. Update the coefficient matrix by using (12) and (13), which requires $\mathcal{O}(i^2)$ operations.
3. Use (10) to obtain $\mathbf{K}^{i+1}$, an updated Gram matrix. Neglect all rows and columns that contain a diagonal element less than δ. This step requires $\mathcal{O}(n^2)$ operations.

The total computational complexity of ACKFA is up to $\mathcal{O}(\ell n^2)$ when no data is cut during updating, which is the same as AKFA. If we increase δ, more data will be cut, and the total computational time will be decreased. Since the number of data being cut at each step depends on δ and data set, we consider an average situation. Suppose after extracting a feature, we only keep a fraction p of all possible directions, and let there be one point left after extracting ℓ features. That means $np^\ell = 1$, where n is the data size. It is equal to $p = n^{-1/\ell}$. The total computational time is

$$CT_{ACKFA} = \sum_{k=0}^{\ell-1} (np^k)^2 = \frac{n^2 - 1}{1 - n^{-2/\ell}} \approx \frac{n^2}{1 - n^{-2/\ell}}$$

when n is large. The computational time of SKFA is $CT_{SKFA} = \mathcal{O}(\ell^2 n^2)$. Therefore,

$$\frac{CT_{SKFA}}{CT_{ACKFA}} = \ell^2 (1 - n^{-2/\ell})$$

For example, when $\ell = 50$, data size $n = 3000$, CT_{SKFA}/CT_{ACKFA} is about 684, so the speed of ACKFA is 684 times faster than SKFA! The experimental results will confirm that our features have better performance than those obtained by SKFA.

5 Experiments

We check the performance of the features based on the computational time, reconstruction error in the mapped space, projection variance, and face recognition. We use mean square error to represent the reconstruction error, and a Gaussian kernel with $\theta = 4$, for Experiments 5.1 to 5.3.

Data Size	Time for Each Method			
	KPCA	ACKFA	AKFA	SKFA
500	4.69	0.08	0.21	16.82
1000	42.68	0.46	0.95	67.57
1500	138.31	0.95	1.98	152.02
2000	352.28	1.94	3.80	269.91
2500	621.65	2.78	5.69	420.33
3000	1184.8	4.35	10.34	643.07
3500	1668.2	5.69	11.31	890.25

Table 1. Computational Time (Seconds) with Data Size

All results were obtained using *Matlab 7.0.1* on a Dell Inspiron 600M with Intel Pentium(R) M processor 1.5GHz and 512 MB of RAM. We use the *Statistical Pattern Recognition Toolbox* [5] for the Gram matrix calculation and KPCA algorithm.

5.1 Computational Time and Reconstruction Error with Data Size

In this experiment, we use an artificially generated data set in $\mathbb{R}^2$. The data form a circle centered at $(0, 0)$ with radius 8.0. The data are corrupted by adding normally distributed noise with standard variance 1.0. The data size is increased from 500 to 3000. For each algorithm AKFA, ACKFA with $\delta = 0.4$, KPCA, and SKFA, we extract ten features that describe each data set. The results are shown in Table 1, 2, and Figure 1. Table 1 indicates that the computational time of KPCA increases rapidly with the data size n. When $n = 3500$, the computational time of KPCA is a factor of 293 larger than that of ACKFA, about 147 times that of AKFA, and about 2 times of SKFA. The computational time versus data size n is plotted in Figure 1 with common-logarithm scales. The following linear fits validate the complexity analyses in Sections 2 through 4.

METHOD	$\log_{10}$(time)
KPCA	$3.03 \log_{10} n - 7.45$
SKFA	$2.05 \log_{10} n - 4.32$
AKFA	$2.00 \log_{10} n - 6.02$
ACKFA	$2.01 \log_{10} n - 6.36$

Note that SKFA consumes more time than KPCA when n is small and ℓ, the number of features is large. For example, when data size is 300, the number of features is 50, the computational time of SKFA with $\mathcal{O}(\ell^2 n^2)$ is about 8 times that of KPCA with $\mathcal{O}(n^3)$.

Table 2 shows that KPCA, which computes accurate estimates of the eigenvectors, incurs a smaller reconstruction error than the other approximate methods. In this sense, the features generated by KPCA are more accurate. The reconstruction errors appear to be stable for all methods as n increases.

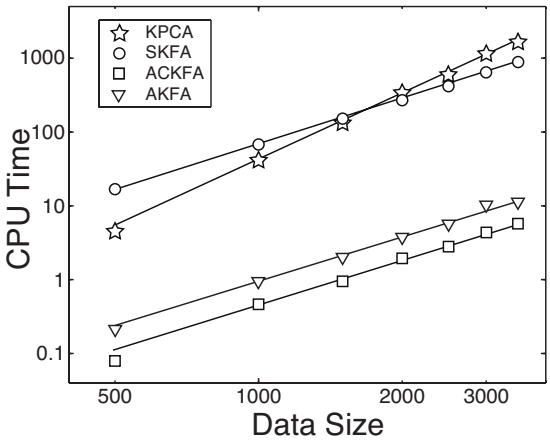

Figure 1. Logarithmic Computational Time (Seconds) with Logarithmic Data size: Features:10.

Data Size	Error of Each Method			
	KPCA	ACKFA	AKFA	SKFA
500	0.0525	0.0950	0.0702	0.1193
1000	0.0584	0.0996	0.0789	0.1305
1500	0.0553	0.0849	0.0773	0.1201
2000	0.0567	0.0866	0.0763	0.1307
2500	0.0564	0.0837	0.0772	0.1433
3000	0.0569	0.0853	0.0781	0.1328
3500	0.0553	0.0825	0.0743	0.1309

Table 2. Reconstruction Error with Data Size

5.2 ACKFA with δ

The previous experiment reveals a trade off between reconstruction error and computation time for ACKFA and AKFA. Using the same problem, with $n = 1000$, we now study the performance of ACKFA with respect to δ. With $\ell = 20$ features, we increase δ from 0.01 to 0.15. If δ is larger than 0.15, all data are cut off before we reach 20 features. Without cutting, the computational time is 1.8326 s, and the reconstruction error is 0.025. The computational time and reconstruction error of ACKFA are plotted against δ in Figure 2. The computational time decreases almost linearly with respect to δ. In contrast, the reconstruction error remains steady for $0.05 \leq \delta \leq 0.07$, then increases rapidly from $0.07 \leq \delta \leq 0.11$, and flattens out from $0.11 \leq \delta \leq 0.15$. A careful choice of δ can balance the trade off between efficiency and accuracy.

5.3 Interpreting Features using Projection Variance

This experiment uses the objective function (6) to verify the features by using KPCA, ACKFA with $\delta = 0.4$,

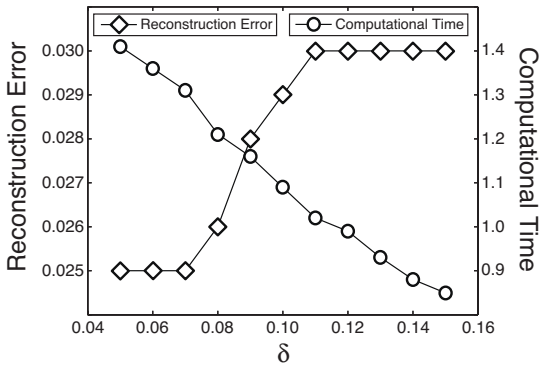

Figure 2. Computational Time (Seconds) and Reconstruction Error of ACKFA with δ: Data size:1000. Features:20.

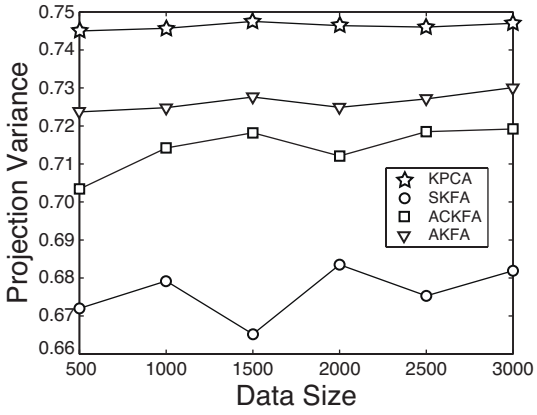

Figure 3. Projection Summation of 10 Features: Data size: 500 to 3000.

AKFA and SKFA. Using the artificial data set as before, random data sets from size $n = 500$ to $n = 3000$ are generated. For each method, we calculated the projection variance on the first 10 features $\frac{1}{n}\sum_{i=1}^{10}\sum_{j=1}^{n}\langle\Phi(x_j), v_i\rangle^2$, which measures the total information those 10 features contain. The experimental results in Figure 3 show that KPCA captures the maximum projection. AKFA and ACKFA outperform SKFA, which we attribute to the modification of (8). AKFA is better than ACKFA because the latter neglects data points.

5.4 Patterns Selected by AKFA and KFA

To further investigate the feature points selected by AKFA and SKFA, we artificially generate 1200 points in the shape of a Swiss Cross, centered at $(0, 0)$ in $\mathbb{R}^2$. Normally distributed noise with variance 1.0 is added to each vector. We extract 15 features in the image space. The corresponding data selected by AKFA and SKFA are shown as

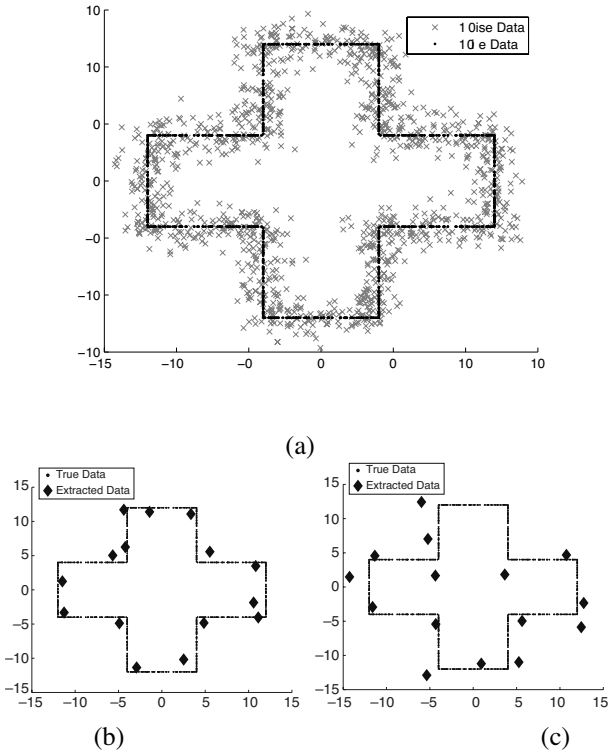

Figure 4. Comparison between AKFA (b) and SKFA (c) for the Swiss Cross (a) problem.

diamonds ($\diamondsuit$) in Figure 4. We can see that basically both methods are picking the corner points, however the features selected by AKFA lie closer to the actual vertices. This difference stems from the different way each method calculates the projection variance. This experiment suggests that AKFA extracts more informative features than SKFA.

5.5 Face Recognition

Previous research demonstrates that features obtained by kernel analysis, including KPCA, are useful for face recognition applications [16]. The following experiments demonstrate that AKFA is also useful within this problem domain. Following Yang et al., the Yale Face Database [1] is classified using a k-nearest neighbor classifier under the leave-one-out method. The data set consists of 165 gray-scale images of 15 different human subjects, each with 11 exposures: center-light, w/glasses, happy, left-light, w/o glasses, normal, right-light, sad, sleepy, surprised, and wink. Each image is down-sampled to a 26×25 pixel array, and is subsequently centered and normalized. In each trial, one of the images is left out, and the remaining 164 are analyzed by AKFA, ACKFA with $\delta = 0.16$, SKFA, and KPCA with a

METHOD	σ				TIME(s)
	14	20	24	34	
AKFA(%)	72.12	75.15	76.97	75.76	0.135
ACKFA(%)	72.12	76.36	75.15	64.24	0.085
KPCA(%)	76.36	75.76	76.36	75.15	0.185
SKFA(%)	68.48	74.55	76.36	73.33	15.705

Table 3. Classification Accuracy with the Gaussian Kernel Parameter σ

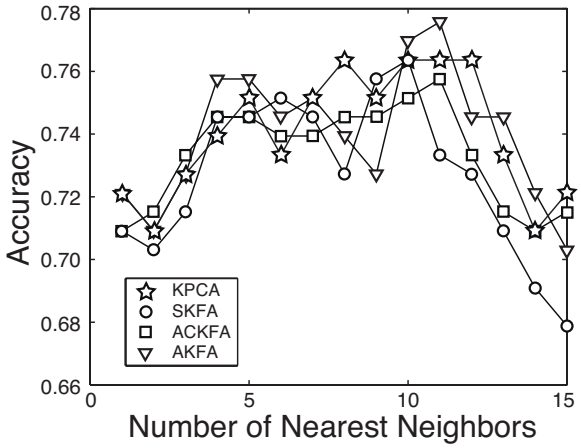

Figure 5. Accuracy with the Number of Nearest Neighbor

Gaussian kernel with width $\sigma \in \{14, 20, 24, 34\}$.

For each trial, $\ell = 80$ features are selected. In the first experiment we select $k = 10$, and compare in Table 3 the resulting classification accuracy and execution time of these four algorithms using the four different values of σ. (Since $\ell^2 \gg n$, SKFA requires significantly more time than KPCA.) Figure 5 compares the accuracies of the four algorithms for a range of k, with $\sigma = 24$. The best accuracy occurs when the k is close to 10.

With $k = 10$ and $\sigma = 24$, confusion matrix is constructed for AKFA in Tables 4, which shows the error frequencies of each class (depicted in **bold**), according to class. The confusion matrices generated by KPCA and SKFA are similar. In the event of a tie, the most frequent class with the lowest index is selected. The seventh (right-light) exposure of Subject 1 is incorrectly selected an inordinate number of times. Faces from Subjects 8 and 12, are also difficult to classify. (These images are shown in Figure 6).

For comparison, the best accuracy obtained with features selected by linear PCA, is 60.61%, which corresponds to a subspace of $\ell = 50$ dimensions and $k = 5$. Consequently, kernel features can improve the performance of face recognition. Although the features of AKFA have a larger reconstruction error than KPCA, the two methods have similar

True Pattern Class	Class Determined by k-NN Classifier														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	10	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0
2	**2**	9	0	0	0	0	0	0	0	0	0	0	0	0	0
3	**2**	0	7	0	0	0	0	0	0	0	0	0	**1**	**1**	0
4	**2**	0	0	9	0	0	0	0	0	0	0	0	0	0	0
5	**2**	0	0	0	9	0	0	0	0	0	0	0	0	0	0
6	**2**	0	0	0	0	9	0	0	0	0	0	0	0	0	0
7	**1**	**1**	0	0	0	0	9	0	0	0	0	0	0	0	0
8	**2**	0	0	0	**1**	0	0	6	0	0	0	0	0	0	**2**
9	**2**	**1**	0	0	0	0	0	0	8	0	0	0	0	0	0
10	**2**	0	0	0	0	0	**1**	0	0	8	0	0	0	0	0
11	0	0	0	**1**	0	0	0	0	0	0	10	0	0	0	0
12	**2**	**1**	0	**1**	0	0	0	0	0	0	0	7	0	0	0
13	**1**	0	0	0	0	0	0	0	**1**	0	0	0	9	0	0
14	**2**	0	0	0	0	0	0	0	0	0	0	0	0	9	0
15	**2**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	8

Table 4. Confusion Matrix of AKFA

Figure 6. Top Layer: Subject 1; Middle Layer: Subject 8; Bottom Layer: Subject 12.

performance for face recognition, which further confirms that the features of AKFA are useful for computer vision and pattern recognition.

6 Conclusion

This paper describes, analyzes, and demonstrates AKFA, a new feature extraction algorithm derived from the Sparse Kernel Feature Analysis (SKFA) of Smola et al [13]. The time complexity of AKFA is $\mathcal{O}(\ell n^2)$ which is more efficient than the $\mathcal{O}(\ell^2 n^2)$ time complexity of SKFA, and the complexity $\mathcal{O}(n^3)$ of a more systematic principal component analysis (KPCA). The face recognition experiment shows that AKFA has the similar classification performance as KPCA using k-nearest neighbor classifier, which demonstrates that the features extracted by AKFA are practically useful.

We hope this work will inspire future studies that reveal how AKFA can be used to elucidate hidden structures in practical vision problems, such as image retrieval. We intend to study how the accuracy of AKFA can be further improved by representing features as more general linear combinations of image vectors.

References

[1] P. N. Bellhumer, J. Hespanha, and D. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Special Issue on Face Recognition*, 17(7):711–720, 1997.

[2] R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1, pages 138–140. Wiley, New York, 1966.

[3] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley & Sons, New York, second edition, 2001.

[4] V. Franc and V. Hlaváč. Greedy algorithm for a training set reduction in the kernel methods. In *CAIP 2003: Computer Analysis of Images and Patterns*, volume 2756 of *Lecture Notes in Computer Science*, pages 426–433, Berlin, Germany, 2003. Springer-Verlag.

[5] V. Franc, V. Hlavac, and M.I.Schlesinger. Statistical pattern recognition toolbox, 2005.

[6] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer, Boston, 1991.

[7] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR 2004: Computer Vision and Pattern Recognition*, pages 506–503, 2004.

[8] T. Kohonen. *Self-Organization and Associative Memory*. Springer Verlag, New York, third edition, 1989.

[9] F. D. la Torre and M. J. Black. Dynamic coupled component analysis. In *CVPR 2001: Computer Vision and Pattern Recognition*, pages 643 – 650, Kauai, Hawaii, 2001.

[10] F. D. la Torre and M. J. Black. Robust principal component analysis for computer vision. In *ICCV 2001: International Conference on Computer Vision*, pages 362–369, 2001.

[11] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. In *Neural Computation*, volume 10, pages 1299–1319, 1998.

[12] B. Schölkopf and A. J. Smola. *Learning with Kernel*, chapter 14. MIT Press, London, England, 2002.

[13] A. Smola, O. Mangasarian, and B. Schölkopf. Sparse kernel feature analysis. In *Technical Report 99–04*. University of Wisconsin, Data Mining Institute, Madison, 1999.

[14] M. E. Tipping. Sparse kernel principal component analysis. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS 2000: Neural Information Processing Systems*, pages 633–639. MIT Press, 2000.

[15] M. Turk and A. Pentland. Face recognition using eigenfaces. In *CVPR 1991: Computer Vision and Pattern Recognition*, pages 586–590, Maui, Hawaii, 1991.

[16] M. Yang, N. Ahuja, and D. Kriegman. Face recognition using kernel eigenfaces. In *ICIP 2000: IEEE International Conference on Image Processing*, volume 1, pages 37–40, Vancouver, Canada, 2000.

[17] W. Zheng, C. Zou, and L. Zhao. An improved algorithm for kernel principal component analysis. *Neural Processing Letters*, 22(1):49–56, 2005.