Available online at www.sciencedirect.com
ScienceDirect
Pattern Recognition Letters 28 (2007) 1151–1155
Pattern Recognition Letters
www.elsevier.com/locate/patrec

Fast principal component analysis using fixed-point algorithm

Alok Sharma *, Kuldip K. Paliwal

Signal Processing Lab, Griffith University, Brisbane, Australia

Received 1 July 2006; received in revised form 26 January 2007
Available online 6 February 2007

Communicated by W. Zhao

Abstract

In this paper we present an efficient way of computing principal component analysis (PCA). The algorithm finds the desired number of leading eigenvectors with a computational cost that is much less than that from the eigenvalue decomposition (EVD) based PCA method. The mean squared error generated by the proposed method is very similar to the EVD based PCA method.

Keywords: Fast PCA; Eigenvalue decomposition; Mean squared error

1. Introduction

Principal component analysis (PCA) finds a linear transformation $\boldsymbol{\varphi}$ which reduces d-dimensional feature vectors to h-dimensional feature vectors (where $h < d$) in such a way that the information is maximally preserved in minimum mean squared error sense. This linear transformation is known as PCA transform or Karhunen-Loéve transform (KLT) (Fukunaga, 1990). It can also reconstruct h-dimensional feature vectors back to the d-dimensional feature vectors with some finite error known as reconstruction error. The PCA is mostly used in compression and reconstruction of high dimensional feature vectors. Since the transformation is from d-dimensional feature space to h-dimensional feature space and vice versa the size of $\boldsymbol{\varphi}$ is $d \times h$. The h column vectors are the basis vectors. The first basis vector is in the direction of maximum variance of the given feature vectors. The remaining basis vectors are mutually orthogonal and, in order, maximize the remaining variances subject to the orthogonal condition. Each basis vector represents a principal axis. These principal axes are those orthonormal axes onto which the remaining variances under projection are maximum. These orthonormal axes are given by the dominant/leading eigenvectors (i.e. those with the largest associated eigenvalues) of the measured covariance matrix. In PCA, original feature space is characterized by these basis vectors and the number of basis vectors used for characterization is usually less than the dimensionality d of the feature space (Sharma et al., 2006).

The computation of PCA requires eigenvalue decomposition (EVD) of the covariance matrix of the feature vectors. One well-known EVD method is the Cyclic Jacobi's method (Golub and van Loan, 1996). The Jacobi's method which diagonalizes a symmetric matrix requires around $O(d^3 + d^2 n)$ computations (Golub and van Loan, 1996) (where n is the number of feature vectors or samples used). This computation in many applications (e.g. fixed-point implementation) is undesirable. A number of methods have been proposed in the literature for computing the PCA transform with reduced computational complexity. Reddy and Herron (2001) proposed modification to Jacobi's method which favours fixed-point implementations. Their computational complexity is, however, still of order $O(d^3)$ for each symmetric rotation (assuming symmetric matrix of size $d \times d$ is previously computed). Basically the improvement in computational complexity is negligible. Roweis (1997) proposed expectation maximizing (EM)

* Corresponding author. Tel.: +679 323 2870/617 3875 3754; fax: +679 323 1538.
E-mail address: sharma_al@usp.ac.fj (A. Sharma).

0167-8655/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2007.01.012

algorithm for PCA, which is computationally effective than EVD method for PCA. But this technique uses EM algorithm which could be expensive in time. It also requires matrix inverse[1] computation in both the E-step and M-step for each of the iteration, which is an expensive exercise. Furthermore, EM algorithm does not converge to a global maximum; it achieves only a local maximum and thus the choice of the initial guess used in the algorithm becomes crucial. The power method (Schilling and Harris, 2000) is also used to find leading eigenvector which is less expensive method but can compute only one most leading eigenvector. Another method is snap-shot algorithm (Sirovich, 1987) which does not explicitly compute sample covariance matrix, however, requires matrix inversion and is based on the assumption that the eigenvectors being searched are linear combinations of feature vectors.

In this paper we present a computationally-fast technique for finding the desired number of leading eigenvectors without diagonalizing any symmetric matrix. Thus it avoids Cyclic Jacobi's method for eigendecomposition. We have used the fixed-point algorithm (Hyvärinen and Oja, 1997) to find all the h leading eigenvectors which converges in just a few iterations without the need of any initial setting. Furthermore, it is free from matrix inverse computations. As a result, the presented algorithm is computationally efficient, consumes very small amount of computation time and is very easy to implement. For brevity we call this method Fast PCA. This Fast PCA generates approximately the same amount of mean squared error as that generated by EVD based PCA method.

2. PCA revisited

In this section PCA has been revisited to obtain a fixed-point PCA transform. The PCA transform can be found by minimizing mean squared error. To see this, let the feature vector be $\boldsymbol{x} \in \boldsymbol{R}^d$ (d-dimensional space), reduced dimensional feature vector be $\boldsymbol{y} \in \boldsymbol{R}^h$ and reconstructed feature vector be $\hat{\boldsymbol{x}} \in \boldsymbol{R}^d$. Then the mean squared error can be represented as

$$\mathrm{MSE} = E[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] \tag{1}$$

where $E[\bullet]$ is the expectation operation with respect to $\boldsymbol{x}$ and $\|\bullet\|^2$ is the norm squared value. We know that PCA transformation $\boldsymbol{\varphi}$ is of size $d \times h$ and it is used to do dimensionality reduction from d-dimensional space to h-dimensional feature space, i.e. $\boldsymbol{\varphi}:\boldsymbol{x} \to \boldsymbol{y}$ or $\boldsymbol{y} = \boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{x}$. It should be noted that in this transformation, $\boldsymbol{x}$ is assumed to be zero mean, if mean is not zero then $\boldsymbol{y}$ can be represented as

$$\boldsymbol{y} = \boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{\mu}) \tag{2}$$

where $\boldsymbol{\mu} = E[\boldsymbol{x}]$. Given $\boldsymbol{y}$(from Eq. (2)) one can simply transform it back to the original feature space with some finite reconstruction error. Applying back transformation we get reconstructed vector $\hat{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{x}} = \boldsymbol{\varphi}\boldsymbol{y} + \boldsymbol{\mu} \tag{3}$$

Substituting Eq. (2) in Eq. (3) we get

$$\hat{\boldsymbol{x}} = \boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}}(\boldsymbol{x} - \boldsymbol{\mu}) + \boldsymbol{\mu} \tag{4}$$

Eq. (1) can be rewritten by utilizing Eq. (4) as

$$\mathrm{MSE} = E[\|(\boldsymbol{I}_{d\times d} - \boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}})(\boldsymbol{x} - \boldsymbol{\mu})\|^2] \tag{5}$$

The desired h basis vectors of $\boldsymbol{\varphi}$ span the d-dimensional subspace and are mutually orthonormal, i.e. $\boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{\varphi} = \boldsymbol{I}_{h\times h}$. Using the orthonormality condition Eq. (5) can be further simplified as

$$\mathrm{MSE} = E[g(\boldsymbol{\varphi}, \boldsymbol{x})]$$

where $g(\boldsymbol{\varphi}, \boldsymbol{x})$ is a scalar function and is given by

$$g(\boldsymbol{\varphi}, \boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}}(\boldsymbol{I}_{d\times d} - \boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}})(\boldsymbol{x} - \boldsymbol{\mu})$$

From Appendix I we can write the derivative of $E[g(\boldsymbol{\varphi}, \boldsymbol{x})]$ with respect to $\boldsymbol{\varphi}$ as

$$\frac{\partial}{\partial \boldsymbol{\varphi}} E[g(\boldsymbol{\varphi}, \boldsymbol{x})] = -2E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\varphi}] \tag{6}$$

Given Eq. (6) and the orthonormality condition of $\boldsymbol{\varphi}$ we can apply fixed-point algorithm (Hyvärinen and Oja, 1997) to solve for the values of $\boldsymbol{\varphi}$ i.e.

$$\begin{aligned} &\boldsymbol{\varphi} \leftarrow E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}}]\boldsymbol{\varphi} \\ &\boldsymbol{\varphi} \leftarrow \mathit{orthonormalize}\ (\boldsymbol{\varphi}) \end{aligned} \tag{7}$$

Note that the negative sign is removed from Eq. (7) since $\boldsymbol{\varphi}$ and $-\boldsymbol{\varphi}$ define the same direction. Moreover, since the expectation is with respect to $\boldsymbol{x}$, transformation $\boldsymbol{\varphi}$ is taken out of it. The algorithm can also be represented as

$$\begin{aligned} &\boldsymbol{\varphi} \leftarrow \Sigma_x\boldsymbol{\varphi} \\ &\boldsymbol{\varphi} \leftarrow \mathit{orthonormalize}\ (\boldsymbol{\varphi}) \end{aligned}$$

where $\Sigma_x = E[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^{\mathrm{T}}]$ is the covariance of $\boldsymbol{x}$. The orthonormalization of $\boldsymbol{\varphi}$ can be done by using Gram-Schmidt orthonormalization procedure. Thus the computation of h eigenvectors does not require eigenvalue decomposition procedure. This means that the Cyclic Jacobi's method can be altogether avoided for eigendecomposition. This would certainly reduce the computation complexity in finding eigenvectors for PCA. Furthermore, use of fixed-point algorithm converges the algorithm very fast in just a few iterations without the need of any prior knowledge of the learning rate/step size or initial settings as per required by the gradient descent based and/or EM based methods. The next section depicts the iterative algorithm for computing eigenvectors for PCA.

3. Fast PCA algorithm

One way to compute all the h orthonormal basis vectors is to use Gram-Schmidt method. The most dominating/

[1] Note that the matrix inversion scales as the cube of the matrix size.

Table 1
Fast PCA algorithm for computing leading eigenvectors

1. Choose h, the number of principal axes or eigenvectors required to estimate. Compute covariance Σ_x and set $p \leftarrow 1$
2. Initialize eigenvector $\boldsymbol{\varphi}_p$ of size $d \times 1$ e.g. randomly
3. Update $\boldsymbol{\varphi}_p$ as $\boldsymbol{\varphi}_p \leftarrow \Sigma_x \boldsymbol{\varphi}_p$
4. Do the Gram-Schmidt orthogonalization process

$$\boldsymbol{\varphi}_p \leftarrow \boldsymbol{\varphi}_p - \sum_{j=1}^{p-1} (\boldsymbol{\varphi}_p^{\mathrm{T}} \boldsymbol{\varphi}_j) \boldsymbol{\varphi}_j$$

5. Normalize $\boldsymbol{\varphi}_p$ by dividing it by its norm: $\boldsymbol{\varphi}_p \leftarrow \boldsymbol{\varphi}_p / \|\boldsymbol{\varphi}_p\|$
6. If $\boldsymbol{\varphi}_p$ has not converged, go back to step 3
7. Increment counter $p \leftarrow p + 1$ and go to step 2 until p equals h

leading eigenvector or principal axis will be measured first. Similarly, all the remaining $h - 1$ basis vectors (orthonormal to the previously measured basis vectors) will be measured one by one in a reducing order of dominance. The previously measured $(p - 1)$th basis vectors will be utilized for finding the pth basis vector. The algorithm for pth basis vector will converge when the new and old values $\boldsymbol{\varphi}_p$ point in the same direction i.e. $\boldsymbol{\varphi}_p^{+\mathrm{T}} \boldsymbol{\varphi}_p \approx 1$ (where $\boldsymbol{\varphi}_p^+$ is the new value of $\boldsymbol{\varphi}_p$). It is usually economical to use a finite tolerance error to satisfy convergence criteria

$$abs(\boldsymbol{\varphi}_p^{+\mathrm{T}} \boldsymbol{\varphi}_p - 1) < \varepsilon \tag{8}$$

where ε is a predefined tolerance or threshold and $abs(\bullet)$ is the absolute value. The Fast PCA algorithm is illustrated in Table 1.

3.1. Computational complexity of the algorithm

Let L be the number of iterations used in converging the algorithm for $\boldsymbol{\varphi}_p$ and n be the number of samples used. Then the estimated computational complexity is given in Table 2.

The value of L is quite small (usually 2–5) and therefore the computational complexity can be estimated to be $O(d^2h + d^2n)$. If dimension d is large compared to h and n then computational complexity can be estimated to be $O(d^2)$.

Table 2
Computational complexity of the algorithm

Major processing steps involved in the algorithm	Computational complexity
Covariance Σ_x (step 1)	$O(d^2n)$
Gram-Schmidt orthogonalization for $\boldsymbol{\varphi}_p$ (pth basis vector)	$O(dpL)$
Gram-Schmidt orthogonalization for all $p = 1 \ldots h$ basis vectors	$O(dh^2L)$
Updating process for all $p = 1 \ldots h$ basis vectors (step 3)	$O(d^2hL)$
Total estimated	$O(d^2hL + d^2n) \approx O(d^2h + d^2n)$

4. Simulation results

To verify the performance of the Fast PCA method in terms of processing time and mean squared error we have compared it with the performance of EVD based method for PCA. For the processing time, the cputime[2] is evaluated while increasing the data dimensionality. Similarly, mean squared error is evaluated while increasing the data dimensionality. We have generated uniformly distributed random vectors of dimensions starting from 100 and going up to 4000. The number of samples or feature vectors is fixed and is 100. The dimension is reduced to $h = 10$ in all the cases and the tolerance (Eq. (8)) is set to 0.01 in the verification of the algorithm.

Fig. 1 is illustrating mean squared error for both the methods as a function of data dimensionality. In the figure '+' sign indicates PCA using EVD method and '.' sign indicates Fast PCA method. It can be observed that the errors generated by both the methods are very close to each other for all the data dimensions. From this it can be inferred that the difference in performance of these methods in terms of the mean squared error is negligible.

Fig. 2 depicts the cputime consumed in finding the h leading eigenvectors for both the methods as a function of data dimensionality. The data dimensions from 100 to 1000 are illustrated in Fig. 2 and dimensions from 2000 to 4000 are illustrated in Table 3. It is evident from Fig. 2 that cputime curve for EVD based PCA increases exponentially as the data dimensionality is increased. Moreover, from Table 3 cputime for dimensions above 2000 are very expensive for EVD based PCA method and may restrict practical applications which involve such high dimensions. For dimension 4000 the EVD based PCA method is consuming around 1413 cputime in seconds to find 10 leading eigenvectors. On the other hand, it can be observed from Fig. 2 as well as from Table 3 that the cputime for Fast PCA method is very economical for all the dimensions. Even for data dimensionality 4000, cputime is only 7.53 s, which is extremely low as compared to EVD based PCA method. Thus Fast PCA can also be efficiently used for high dimensional applications.

It appears from Fig. 2 that the complexity curve for Fast PCA is approximately linear. However, it is the presence of the complexity curve of EVD based PCA method that increases relatively rapidly which creates this appearance. The complexity curve for Fast PCA has been shown in Fig. 3 separately to depict that it indeed approximately follows $O(d^2)$ complexity.

It can be concluded from Fig. 2 and Table 3 that the Fast PCA method is highly efficient in finding the leading eigenvectors in terms of time due to its reduced computational complexity. We have also tested (not shown in this paper) the Fast PCA algorithm for $h = 100$ and we found

[2] cputime is a MATLAB keyword that returns the CPU time in seconds for any process.

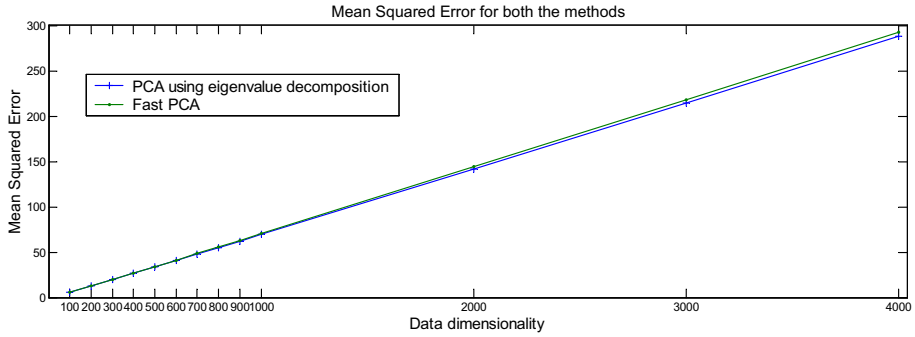

Fig. 1. Mean squared error for PCA using EVD and Fast PCA as a function of data dimensionality.

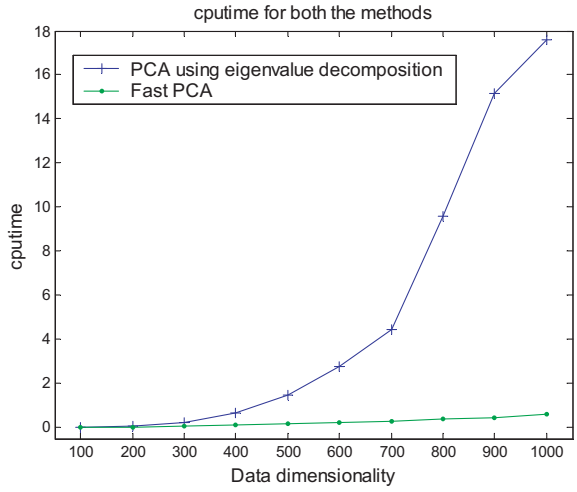

Fig. 2. cputime for PCA using EVD method and Fast PCA method as a function of data dimensionality from 100 to 1000.

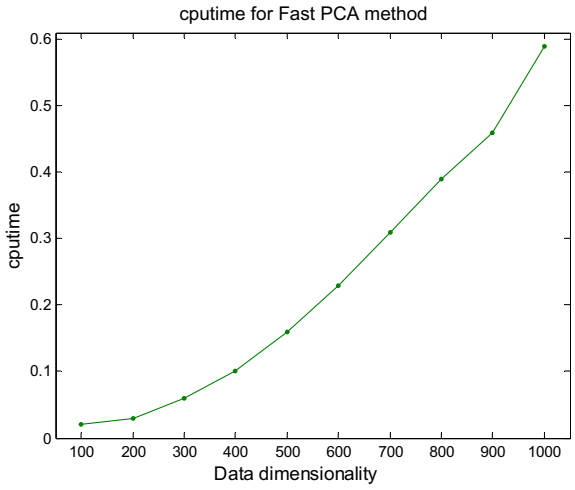

Fig. 3. cputime for Fast PCA method as a function of data dimensionality from 100 to 1000.

Table 3
cputime for PCA using EVD method and Fast PCA method as a function of data dimensionality from 2000 to 4000

Data dimensionality	EVD based PCA method (cputime in seconds)	Fast PCA method (cputime in seconds)
2000	153.26	2.28
3000	531.56	5.30
4000	1413.72	7.53

that it is still very economical than EVD based PCA method.

5. Conclusion

In this paper we have presented a method (called Fast PCA) for computing the PCA transformation with reduced computational cost. This Fast PCA method utilizes fixed-point algorithm. The proposed method is compared with the EVD based PCA method. It was seen that the Fast PCA method is computationally effective and economical in finding small number of leading eigenvectors or orthonormal axes for PCA as compared to the EVD based PCA method. The mean squared error provided by proposed method is also very close to the MSE provided by EVD based PCA method.

Appendix I

Lemma 1. *Let the scalar function* $g(\boldsymbol{\varphi},\boldsymbol{v}) = \boldsymbol{v}^{\mathrm{T}}(\boldsymbol{I}_{d\times d} - \boldsymbol{\varphi}\boldsymbol{\varphi}^{T})\boldsymbol{v}$ *be a differentiable function of a* $d\times h$ *rectangular matrix* $\boldsymbol{\varphi}$ *such that* $h < d$. *Suppose* $\boldsymbol{v}$ *is any vector of size* $d\times 1$. *Then the gradient of* $g(\boldsymbol{\varphi},\boldsymbol{v})$ *is defined as* $\nabla_{\boldsymbol{\varphi}} g(\boldsymbol{\varphi},\boldsymbol{v}) = -2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}$.

Proof 1. The scalar function $g(\boldsymbol{\varphi},\boldsymbol{v})$ can be simplified as $g(\boldsymbol{\varphi},\boldsymbol{v}) = \boldsymbol{v}^{\mathrm{T}}\boldsymbol{v} - \boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{v}$. Its derivative with respect to $\boldsymbol{\varphi}$ is given as

$$\begin{aligned}\partial g(\boldsymbol{\varphi},\boldsymbol{v}) &= \mathrm{trace}[\partial(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v})] - \mathrm{trace}[\partial(\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{v})]\\ &= -\{\mathrm{trace}[\boldsymbol{v}^{\mathrm{T}}\partial\boldsymbol{\varphi}\boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{v}] + \mathrm{trace}[\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}\partial\boldsymbol{\varphi}^{\mathrm{T}}\boldsymbol{v}]\}\\ &= -2\,\mathrm{trace}[\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}\partial\boldsymbol{\varphi}^{\mathrm{T}}]\end{aligned}$$

$\{\because \mathrm{tr}(A^{\mathrm{T}}) = \mathrm{tr}(A) \quad \text{and} \quad \mathrm{tr}(AB) = \mathrm{tr}(BA)\}$

or $\nabla_{\boldsymbol{\varphi}} g(\boldsymbol{\varphi},\boldsymbol{v}) = -2\boldsymbol{v}\boldsymbol{v}^{\mathrm{T}}\boldsymbol{\varphi}$. □

References

Fukunaga, K., 1990. Introduction to Statistical Pattern Recognition, second ed. Academic Press, New York.

Golub, G.H., van Loan, C.F., 1996. Matrix Computations, third ed. John Hopkins University Press, Baltimore.

Hyvärinen, A., Oja, E., 1997. A fast fixed-point algorithm for independent component analysis. Neural Comput. 9 (7), 1483–1492.

Reddy, K.M., Herron, T.J., 2001. Computing the eigen decomposition of a symmetric matrix in fixed-point algorithms, IEEE Bangalore Section – Tenth Annual Symposium.

Roweis, S., 1997. EM algorithms for PCA and SPCA. Neural Inf. Proces. Sys. 10, 626–632.

Schilling, R.J., Harris, S.L., 2000. Applied Numerical Methods for Engineers Using Matlab and C. Brooks/Cole Publishing Company.

Sharma, A., Paliwal, K.K., Onwubolu, G.C., 2006. Class-dependent PCA, MDC and LDA: A combined classifier for pattern classification. Pattern Recognition 39, 1215–1229.

Sirovich, L., 1987. Turbulence and the dynamics of coherent structures. Quart. Appl. Math. XLV, 561–571.