Kernel Eigen Space Merging

Gaurav Sharma[1] Santanu Chaudhury[2] J. B. Srivastava[1]
[1]Dept. of Mathematics, [2] Dept. of Electrical Engg.
Indian Institute of Technology Delhi, India
grvsharma@gmail.com

In this document we describe our novel method to merge kernel eigen spaces in feature space using the *kernel trick*. The algorithm builds upon the one proposed by Hall et al. (2000). The reader is requested to refer to that for a rigorous background.

Recently there have been many attempts at bringing down the complexity of computing KPCA. Meltzer et al. (2004) used an approximate version of KPCA. They used approximate pre-images of a representative set of feature space vectors to have a compact representation of descriptors, in the context of robot localization. An iterative algorithm based on residual error minimization was used to get a reduced set of vectors for the original basis in feature space. Whereas, Kimura et al. (2005) have given a method of computing the new eigen space adding exactly *one* more input vector. However, we are interested in a more general block update to the current eigen space.

The problem of incremental KSVD, Chin et al. (2006), and then incremental KPCA, Chin and Suter (2007), have been recently solved. In solving the IKPCA, Chin and Suter (2007) have used their IKSVD, Chin et al. (2006), formulation for updating. While the approach of Chin and Suter (2007) is based totally on SVDs we work extensively with EVDs. Both approaches are kernel extensions of incremental PCA in input space, Hall et al. (2000); Lim et al. (2004). Our algorithm also differs in an intermediate step of orthogonalizing a set of feature space vectors. While Chin and Suter (2007) use SVD again, we adapt a QR algorithm, given in a different context of kernel method for learning over set proposed by Wolf and Shashua (2003), to effect the same.

1 Eigen space merging

1.1 Notation

We denote the i^{th} row of a matrix P as $P_{i.}$ and j^{th} column as $P_{.j}$. Φ_m is used for matrix with $\phi(m_r)$ as columns. The transpose of matrix P is denoted as P^T. 1_k denotes column matrix of length k with each entry having the value $1/k$.

1.2 Defining eigen space

Eigen space model for a data matrix, with vectors as columns, is defined as $\Omega = (\mu, U, \Lambda, N)$ i.e. the mean, the column matrix of the principal components, associated principal (eigen) values and the number of input vectors. It may be obtained by either diagonalizing the input covariance matrix or using the lower dimensional SVD techniquesHall et al. (2000).

Eigen space merging algorithm takes two eigen spaces (distinct linear subspaces) and returns a new eigen space which spans the two input subspaces.

1.3 Outline of eigen space merging in input space

The eigen space merging problem is defined as follows.

Given input eigen spaces, $\Omega = (\overline{x}, U, \Lambda, N)$ *and* $\Xi = (\overline{y}, V, \Delta, M)$*, construct a merged eigen space,* $\Psi = (\overline{z}, W, \Pi, P)$*, such that the column span of* W *equals the column span of* $[U, V]$*,* $P \times \overline{z} = N \times \overline{x} + M \times \overline{y}$ *and* $P = N + M$*.*

The eigen space merging algorithm in the input space is given in Algorithm 1,

Algorithm 1 Merge two eigen spaces in input space.

1: Construct an orthonormal basis set Υ, that spans both eigen space models and $(\overline{x} - \overline{y})$.
2: Formulate a new eigen value problem using Υ, to get the principal values for the merged eigen space model and also the rotation R used to transform Υ into the required basis.
3: Use R to obtain the required merged eigen space model, $W = \Upsilon R$, and keep the higher eigen value components.

This algorithm is the subject of Hall et al. (2000), the interested reader is encouraged to read the full derivations.

We start off from the last equation used in Hall et al. (2000) and then show that it can be kernalized to make the computations feasible in feature space.

2 Kernalizing eigen space merging

The final approximated eigen value problem Hall et al. (2000) to be solved for the merging in input space is:

$$\frac{N}{P}\begin{bmatrix} \Lambda & 0 \\ 0 & 0 \end{bmatrix} + \frac{M}{P}\begin{bmatrix} G\Delta G^T & G\Delta\Gamma^T \\ \Gamma\Delta G^T & \Gamma\Delta\Gamma^T \end{bmatrix} + \frac{NM}{P^2}\begin{bmatrix} gg^T & g\gamma^T \\ \gamma g^T & \gamma\gamma^T \end{bmatrix} = R\Pi R^T \tag{1}$$

Where, $G := U^T V$, $\Gamma := \nu^T V$, $g := U^T(\overline{x} - \overline{y})$ and $\gamma := \nu^T(\overline{x} - \overline{y})$ (ν comes as in equation (5) below). Observe that in the eigen value equation (1), all the terms are products of the column vectors in the input space.

Now, we wish to carry out our computations in feature space instead of the input space. Let us assume we have the implicit non linear map

$$\phi : \mathbb{R}^n \to \mathcal{F} \tag{2}$$

to transform input space to feature space, and the associated kernel, K defined as

$$K(x_i, x_j) := \phi(x_i)^T \phi(x_j) \tag{3}$$

To carry out the computations in feature space, we transform the input space vectors with the map ϕ. Upon doing so, we get, in the equation (1), U and V are now in higher dimension space and $g := U^T(\overline{\phi(x)} - \overline{\phi(y)})$ and $\gamma := \nu^T(\overline{\phi(x)} - \overline{\phi(y)})$, where the mean is now taken in the feature space and not in the input space.

We can see clearly here that all of the quantities in equation (1) can be expressed in terms of products of input vectors transformed to feature space i.e. in the form of $\phi(x_i)^T\phi(x_j)$, except possibly those involving ν. We show here that they can also be computed using the kernel K, without using the map ϕ explictly, by computing ν as a linear combination of feature space mapped input space vectors.

2.1 Basis spanning both subspaces in feature space

The way we get ν is as follows. We are interested in finding a basis that spans both the input subspaces. A sufficient basis can be taken as:

$$\Upsilon := [U, \nu] \tag{4}$$

where ν is a basis spanning components of V and $(\overline{\phi(x)} - \overline{\phi(y)})$ orthogonal to U i.e. residues of columns of V and $(\overline{\phi(x)} - \overline{\phi(y)})$ on U.

$$\nu := Orthonormalize([H, h]) \tag{5}$$
$$H := V - U(U^T V) \tag{6}$$
$$h := (\overline{\phi(x)} - \overline{\phi(y)}) - U(U^T(\overline{\phi(x)} - \overline{\phi(y)})) \tag{7}$$

We can reduce these equations in terms of function of $\phi(x_i)$ and $\phi(y_i)$ as follows,

$$u_i = \textstyle\sum_{k=1}^{N} \alpha_{ki}\phi(x_k) \Rightarrow u_i = \Phi_x \alpha_{.i} \tag{8}$$
$$\Phi_x := [\phi(x_1) \dots \phi(x_N)] \tag{9}$$
$$\Rightarrow U = [\Phi_x\alpha_{.1}, \Phi_x\alpha_{.2}, \dots, \Phi_x\alpha_{.p}] \tag{10}$$

So we express U, and similarly V, as the product of two matrices

$$U := \Phi_x \alpha \qquad V := \Phi_y \beta \tag{11}$$

where $\alpha = [\alpha_{.1}, \alpha_{.2}, \ldots, \alpha_{.p}]$, and thus we get the following expression for H

$$\Rightarrow H = \Phi_y \beta - (\Phi_x \alpha)(U^T V) \tag{12}$$

$$H = \Phi_y \beta - (\Phi_x \alpha)(\alpha^T \Phi_x^T \Phi_y \beta) \tag{13}$$

Say the number of principal components of the two eigen spaces be p and q. Then α is $N \times p$ and β is $M \times q$, and the expression of H reduces to

$$H = \Phi_y \beta - \Phi_x \delta \tag{14}$$

$$\delta := \alpha \alpha^T \Phi_x^T \Phi_y \beta \tag{15}$$

and δ is thus an $N \times q$ dimensional matrix. Similarly, for the residue of the means we first write the mean vectors as

$$\overline{\phi(x)} = \tfrac{1}{N} \textstyle\sum_{i=1}^{p} \phi(x_i) \Rightarrow \overline{\phi(x)} = \tfrac{1}{N} \Phi_x \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{16}$$

$$\overline{\phi(x)} = \Phi_x 1_N \quad \overline{\phi(y)} = \Phi_y 1_M \tag{17}$$

Now looking at the equation for h we can see that,

$$h := (\overline{\phi(x)} - \overline{\phi(y)}) - U(U^T(\overline{\phi(x)} - \overline{\phi(y)})) \tag{18}$$

$$h = (\Phi_x 1_N - \Phi_y 1_M) - (\Phi_x \alpha)(\alpha^T \Phi_x^T \Phi_x 1_N - \alpha^T \Phi_x^T \Phi_y 1_M) \tag{19}$$

Thus we get a similar form for h as we had for H, given by

$$h = \Phi_x \epsilon - \Phi_y \sigma \tag{20}$$

$$\epsilon := 1_N - \alpha \alpha^T \Phi_x^T \Phi_x 1_N + \alpha \alpha^T \Phi_x^T \Phi_y 1_M \tag{21}$$

$$\sigma := 1_M \tag{22}$$

Writing H and h in block matrices form we get,

$$H = [\Phi_x \Phi_y] \begin{bmatrix} -\delta \\ \beta \end{bmatrix} \quad h = [\Phi_x \Phi_y] \begin{bmatrix} \epsilon \\ -\sigma \end{bmatrix} \tag{23}$$

This enables us to write $[H, h]$ as,

$$[H, h] = \Phi_{xy} \lambda \quad \lambda := \begin{bmatrix} -\delta & \epsilon \\ \beta & -\sigma \end{bmatrix} \tag{24}$$

After we calculate the residuals, however, we have to eliminate those which have norm zero, corresponding to overlap in the eigen spaces.

To get ν we have to orthonormalize $[H, h]$. Consider its QR factorization,

$$\Phi_{xy} \lambda = QR \tag{25}$$

by post multiplying both sides by R^{-1} we get a way of computing Q, a basis for ν,

$$Q = \Phi_{xy} \lambda R^{-1} \tag{26}$$

To obtain Q, we calculate R^{-1} by keeping the computations in the lower dimensional space only, using the kernel trick, as described in the following section.

2.2 Orthonormalizing using kernel trick

Recently Wolf and Shashua (2003) have proposed an interleaving algorithm to find the QR factorization of a feature space transformed input set of vectors, in the context of calculating similarity between sets using Kernel Principal Angles. We augment the algorithm to make it applicable here. Let $A = [\phi(a_1), \phi(a_2), \ldots, \phi(a_R)]$ be the set of vectors, and $V = [v_1, v_2, \ldots, v_R]$ be the un-normalised vectors obtained from the Gram-Schmidt orthogonalization of A and $T = S^{-1} = [t_1, t_2, \ldots, t_R]$ be the inverse of S. The algorithm works by observing that in GS process $A = VS$, since S is upper diagonal then $V = AS^{-1}$ implies,

$$v_j = \sum_{q=1}^{j} t_{qj}\phi(a_q) = At_j \tag{27}$$

all that is required to make the algorithm work is to compute the dot products $v_j^T\phi(a_i)$ and $v_j^T v_j$. In the present case we note that, $A = \Phi\lambda$ and,

$$\begin{aligned} \phi(a_j) &:= \Phi\lambda_{.j} && (28)\\ \phi(a_j)^T\phi(a_j) &= \lambda_{.j}^T\Phi^T\Phi\lambda_{.j} && (29) \end{aligned}$$

Thus for the required dot products we get,

$$\begin{aligned} v_j &= \Phi\lambda t_j && (30)\\ v_j^T\phi(a_i) &= t_j^T\lambda^T\Phi^T\Phi\lambda_{.i} && (31)\\ v_j^T v_j &= t_j^T\lambda^T\Phi^T\Phi\lambda t_j && (32) \end{aligned}$$

Now s_j can be obtained using the simple GS formula, and t_j is as given by Wolf and Shashua (2003)

$$\begin{aligned} s_j &= \left[\frac{v_1^T\phi(a_j)}{v_1^T v_1}, \ldots, \frac{v_{j-1}^T\phi(a_j)}{v_{j-1}^T v_{j-1}}, 1, 0, \ldots, 0\right]^T && (33)\\ &= \left[\frac{t_1^T\lambda^T\Phi^T\Phi\lambda_{.j}}{t_1^T\lambda^T\Phi^T\Phi\lambda t_1}, \ldots, \frac{t_{j-1}^T\lambda^T\Phi^T\Phi\lambda_{.j}}{t_{j-1}^T\lambda^T\Phi^T\Phi\lambda t_{j-1}}, 1, 0, \ldots, 0\right]^T && (34)\\ t_j &= [-t_1, -t_2, \ldots, -t_{j-1}, e_j, 0, \ldots, 0]s_j && (35) \end{aligned}$$

By using the Algorithm 2, Wolf and Shashua (2003), we can now compute R^{-1} of the QR factorization of Φ.

Algorithm 2 Calculate the QR factorization of ν in feature space.

1: Initialize $s_1 = t_1 = e_1$
2: **for** $j = 2, \ldots, q+1$ **do**
3: Compute s_j using equation (34)
4: Compute t_j using equation (35)
5: **end for**
6: Compute D using equation (32)
7: $R = D[s_1, s_2, \ldots, s_{q+1}]$
8: $R^{-1} = [t_1, t_2, \ldots, t_{q+1}]D^{-1}$
9: Compute $\nu = Q = \Phi\lambda R^{-1} = \Phi\xi$

2.3 Final equations

Finally going back to equation (1) we have the terms calculated in input space as follows,

$$
\begin{aligned}
G &:= U^T V = \alpha^T \Phi_x^T \Phi_y \beta && (36)\\
\Gamma &:= \nu^T V = \xi^T \Phi^T \Phi_y \beta && (37)\\
g &:= U^T(\overline{\phi(x)} - \overline{\phi(y)}) \\
&= \alpha^T \Phi_x^T \Phi_x 1_N - \alpha^T \Phi_x^T \Phi_y 1_M && (38)\\
\gamma &:= \nu^T(\overline{\phi(x)} - \overline{\phi(y)}) \\
&= \xi^T \Phi^T \Phi_x 1_N - \xi^T \Phi^T \Phi_y 1_M && (39)
\end{aligned}
$$

2.4 Final algorithm

The final algorithm is summarized in Algorithm 3

Algorithm 3 Kernel based algorithm to merge two eigen spaces in feature space.

1: Compute the $[H, h]$ using equation (24).
2: Orthonormalize $[H, h]$ to get ν using Algorithm 2.
3: Form the eigen value equation (1) using the equations (36) to (39).
4: Solve equation (1) to get eigen values, Π, and rotation matrix, R.
5: Get the required principal components, $W = \Upsilon R$

Hence we will have W in the form of linear combination of input vectors as,

$$
\begin{aligned}
\Upsilon &:= [U, \nu] = [\Phi_x \alpha, \Phi \xi] \\
&= [[\Phi_x, \Phi_y][\alpha^T, 0]^T, \Phi \xi] && (40)\\
\Upsilon &= \Phi[\tilde{\alpha}, \xi] && (41)\\
\tilde{\alpha} &:= [\alpha^T, 0]^T && (42)\\
W &= \Upsilon R = \Phi[\tilde{\alpha}, \xi] R && (43)
\end{aligned}
$$

Note that in case of batch method (with input space dimension d), we would have to solve an EVD for an $min(d, N + M)$ square matrix but here the problem has been reduced to a square matrix of size $(p + q + 1)$. As the number of principal components will be much smaller than the actual number of data vectors, i.e. $p + q + 1 \ll min(d, N + M)$, and EVD computation is the most heavy step, our incremental method is expected to perform better. The orders of time complexity for all steps are as given in Table 1. The complexity is given as it scales with each variable while treating others as constants. N, M and p, q are the number of data vectors and number of principal components the two KESs to be merged, respectively.

3 Kernel Eigen Space Merging Experiments

We designed our experiments to test and demonstrate the accuracy and time complexity of the proposed eigen space merging algorithm. All the results given below are averaged over 10 independent runs of the experiment in context, unless otherwise stated.

3.1 Convergence for Incrementally Created Subspace

We tested the accuracy of the proposed method against batch KPCA using synthetically generated data as well as a real life dataset. The synthetic datasets used was the famous 2D datasets, of 1000 vectors each, with x obtained uniformly from $[-1, 1]$ and y taken as $y = x^2 + \eta$ where η is normal noise with mean zero and standard deviation 0.2. The real life dataset was of ACR-SIFT, Lowe (2004); Mikolajczyk and Schmid (2004) based 128 dimensional vectors obtained from a subset of images in the Caltech-101, Fei-Fei et al. (2004) dataset. For both the datasets, the batch KPCA was done on whole dataset to get the eigen space spanning all the points, and the incremental eigen spaces were computed

with 30 randomly sampled vectors at a time. For the real life dataset, 30 principal components were kept and for the sythetic, 6 were kept. To compare the eigen spaces we used the kernel principal angles based distance similar to that used by Chin and Suter (2007), comparing 6D and 3D subspaces for ACR-SIFT and synthetic datasets, respectively. Figure 1 shows the distance between the full eigen space and those obtained by merging 30 new vectors every update, for synthetic and real data respectively. The incremental eigen spaces get closer to the full spaces with more updates. The rate and the extent of this convergence was found to depend on the various parameters.

3.2 Time Complexity

The second experiment was designed to compare the time complexity of the incremental eigen space algorithm and the batch KPCA algorithm empirically. Using both the synthetic and the standard dataset, we randomly selected different number of input vectors; for given number of data vectors, we then used the batch KPCA and IKPCA (updating with 100 new vectors at a time) to arrive at the full eigen spaces. Figure 2(left) shows the total time taken by both algorithms to arrive at the full eigen space, for different input set size. Incremental method was found to outperform batch method for bigger datasets. Figure 2(right) further shows the quadratic scaling of IKPCA with input set size, confirming the theoretical analysis.

References

T.-J. Chin and D. Suter. Incremental kernel principal component analysis. *IEEE Trans. IP*, 16(6), 2007.

T.-J. Chin, K. Schindler, and D. Suter. Incremental kernel svd for face recognition with image sets. *AFGR*, 2006.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. *WGMBV in CVPR*, 2004.

P. Hall, D. Marshall, and R. Martin. Merging and splitting eigen space models. *IEEE Trans. PAMI*, 22(9):1042–1049, 2000.

S. Kimura, S. Ozawa, and S. Abe. Incremental kernel pca for online learning of feature space. *ICCIMCA and ICIAWTIC*, 2005.

J. Lim, D. Ross, R.-S. Lin, and M.-H. Yang. Incremental learning for visual tracking. *Advances in NIPS*, 2004.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

J. Meltzer, M.-H. Yang, R. Gupta, and S. Soattol. Multiple view feature descriptors from image sequences via kernel principal component analysis. *T. Pajdla and J. Matas(Eds.): ECCV*, 1:215–227, 2004.

K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, 60(1):63–86, 2004.

L. Wolf and A. Shashua. Learning over sets using kernel principal angles. *JMLR*, 4:913–931, 2003.

Step Description	Order of complexity
Calculate $[H, h]$	$O(N^2), O(M), O(p), O(q)$
Orthonormalize $[H, h]$	$O(N^2), O(M^2), O(q^3)$
Form EVD	$O(N^2), O(M^2), O(p^2), O(q^3)$
Solve EVD	$O(N), O(M), O(p^3), O(q^3)$
Total complexity	$O(N^2), O(M^2), O(p^3), O(q^3)$

Table 1: Time complexity summary

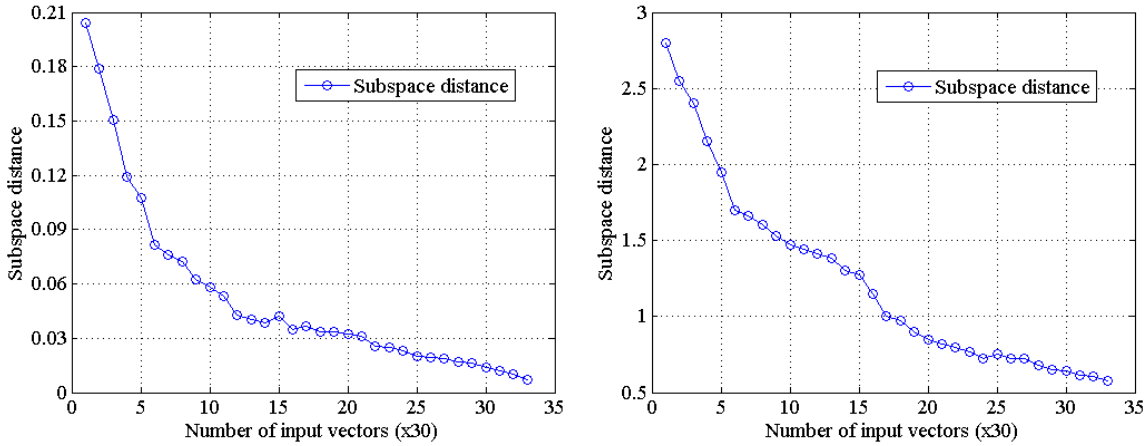

Figure 1: Subspace distance moves towards zero with more updates. The figure shows the subspace distance versus update for synthetic vectors (left) and ACR-SIFT vectors (right).

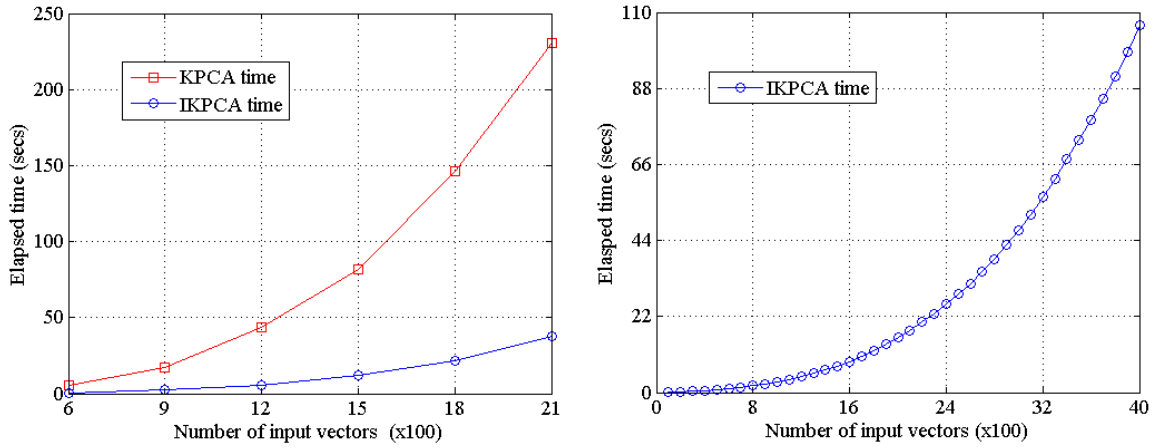

Figure 2: The time complexity of batch vs the IKPCA (left). The time complexity of IKPCA clearly shows quadratic scaling (right).