Fast Kernel Principal Component Analysis(KPCA) for the Polynomial and Gaussian kernels

Vikas Chandrakant Raykar, Ankur Ankur
Perceptual Interfaces and Reality Laboratory
Institute for Advanced Computer Studies
University of Maryland, College Park

February 1, 2003

1. Principal Component Analysis(PCA)

- PCA is a statistical dimensionality reduction technique. Given N points in d dimensions PCA essentially projects the data points onto p, directions($p < d$)which capture the maximum variance of the data.
- These directions correspond to the eigen vectors of the covariance matrix of the training data points.
- Intuitively PCA fits an ellipsoid in d dimensions and uses the projections of the data points on the first p major axes of the ellipsoid.

Let α_{nm} be the projection of the n^{th} data point on the m^{th} basis e_m. Using these projections and the basis functions we can reconstruct a c^{th} order approximation of x_n as

$$\widehat{x}_n = \sum_{m=1}^{c} \alpha_{nm} e_m$$

$$E = [e_1 e_2 \ldots\ldots e_c]$$

$$\alpha_n^T = [\alpha_{n1} \alpha_{n2} \ldots\ldots \alpha_{nc}]$$

Then $\widehat{x}_n$ can be written as

$$\widehat{x}_n = E\alpha_n$$

$$\min_{E,\alpha_n} \frac{1}{2}\sum_{n=1}^{N} \|x_n - E\alpha_n\|^2$$

subject to the constraint that $E^T E = I$. This function can be minimized by forming using the method of Lagrange multipliers.

$$\alpha_k = E^T x_k \tag{1}$$

and E is got by solving the eigen value problem

$$SE = E\Lambda \tag{2}$$

where S is the covariance matrix

$$S = \sum_{n=1}^{N} x_n x_n^T \tag{3}$$

1. Subtract the mean from all the data points $x_n \leftarrow x_n - \frac{1}{N}\sum_{i=1}^{N} x_i$
2. Compute the covariance matrix $S = \sum_{n=1}^{N} x_n x_n^T$
3. Diagonalize S to get its eigen values and eigen vectors i.e get E and Λ.
4. Retain c eigen vectors corresponding to c largest eigen values such that $\frac{\sum_{j=1}^{c}\lambda_j}{\sum_{j=1}^{N}\lambda_j}$ equals the desired variance to be captured.
5. Project the data points on the eigen vectors $\alpha_n = E^T(x_k - \frac{1}{N}\sum_{i=1}^{N} x_i)$ and use the projections instead of the data points.

PCA-Alternate approach There are some applications where we have a few samples of very high dimensions(Face recognition). In such cases where $N < d$ we can formulate the problem in terms of an $N \times N$ matrix called as the dot matrix.

$$X = [x_1 x_2 ......x_N]$$

According to our previous formulation for one eigen value we have to solve

$$\begin{aligned} Se &= \lambda e \\ XX^T e &= \lambda e \\ X^T XX^T e &= \lambda X^T e \\ K\alpha &= \lambda\alpha \end{aligned} \tag{4}$$

where $K = X^T X$ is called as the dot product matrix and $\alpha = X^T e$.

$$\begin{aligned} X^T e &= \alpha \\ XX^T e &= X\alpha \\ Se &= X\alpha \\ \lambda e &= X\alpha \\ e &= \frac{1}{\lambda} X\alpha \end{aligned} \tag{5}$$

Therefore e lies in the span of X the data points. Normalizing e

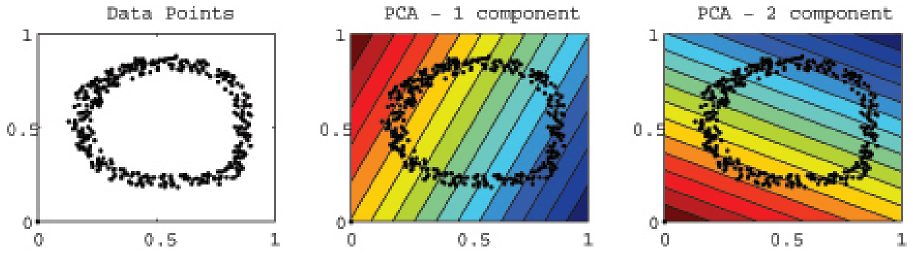
$$\begin{aligned} \|e\|^2 &= 1 \\ \|X\alpha\|^2 &= 1 \\ \alpha^T X^T X\alpha &= 1 \\ \alpha^T K\alpha &= 1 \\ \alpha^T \lambda\alpha &= 1 \\ \|\alpha\|^2 &= \frac{1}{\lambda} \end{aligned} \tag{6}$$

Finally the principal components can be written as where y is the projection of x on e

$$\begin{aligned} y &= e_T x \\ y &= \alpha^T X^T x \\ y &= \sum_i^N \alpha_i \langle x_i . x \rangle \end{aligned} \tag{7}$$

One thing to be noted is that we do not need e explicitly(we need only the dot product).

Two dimensional toy example illustrating PCA



PCA is linear

- It uses only second order statistics in the form of covariance matrix.
- The best it can do is to fit an ellipsoid around the data.
- Kernel Principal Component Analysis(KPCA) is an attractive method for extracting nonlinear features from a given set of multi variate data.
- While Principal Component Analysis(PCA) finds the best ellipsoidal fit for the data KPCA has the capability of extracting the nonlinear features which could be a more natural and compact representation of the data.

2. Kernel Principal Component Analysis(KPCA)

- The essential idea of KPCA is based on the hope that if we do some non linear mapping of the data points to a higher dimensional(possibly infinite) space we can get better non linear features which are a more natural and compact representation of the data.
- The computational complexity arising from the high dimensionality mapping is mitigated by using the *kernel trick*.
- KPCA belongs to a class of more general methodology which use the for algorithms which can be written only in terms of dot products and not on the variable themselves.

- Consider a nonlinear mapping $\phi(.) : \mathrm{R}^d \rightarrow \mathrm{R}^h$ from R^d the space of d dimensional data points to some higher dimensional space $\mathrm{R}^{h.}$.
- Once we have this mapping KPCA is nothing but Linear PCA done on the points in the higher dimensional space.
- As of now assume that the mapped data are zero centered(i.e $\sum_{i=1}^{N} \phi(x_i) = 0$). So now in our case the dot product matrix K becomes
$$[K]_{ij} = [\phi(x_i).\phi(x_j)] \tag{8}$$
K is called the Gram matrix.

- Solving the eigen value problem $K\alpha = \lambda\alpha$ gives the corresponding N eigen vectors. Note that the eigen vector needs to be normalized to satisfy $\|\alpha\|^2 = \frac{1}{\lambda}$ Finally the principal component of any data point x, which is the projection of $\phi(x)$ on e is given by

$$y = \sum_{i}^{N} \alpha_i \langle \phi(x_i).\phi(x) \rangle \tag{9}$$

  As mentioned earlier we do not need e explicitly(*we need only the dot product*).

The *kernel trick* basically makes use of this fact and replaces the dot product by a kernel function which is more easy to compute than the dot product.

$$k(x,y) = \langle \phi(x).\phi(y) \rangle \tag{10}$$

This allows us to compute the dot product without having to carry out the mapping. The most commonly used kernels are the polynomial, Gaussian and the tanh kernel.
For the more general case where the data is not centered in the higher dimensional space we will have to use the modified Gram Matrix(Derivation in the report)

- Given N data points in d dimensions let $X = [x_1 x_2 \ldots\ldots x_N]$ where each column represents one data point.
- Subtract the mean from all the data points.
- Choose an appropriate kernel $k(.,.)$.
- Form the $N \times N$ Gram matrix $[K]_{ij} = [k(x_i, x_j)]$.
- Form the modified Gram matrix

$$\tilde{K} = (I - \frac{1_{N\times N}}{N})^T K (I - \frac{1_{N\times N}}{N})$$

  where where $1_{N\times N}$ is an $N \times N$ matrix with all entries equal to 1

- Diagonalize $\tilde{K}$ to get its eigen values λ_n and eigen vectors α_n.
- Normalize $\alpha_n \Leftarrow \frac{\alpha_n}{\sqrt{\lambda_n}}$.
- Retain c eigen vectors corresponding to c largest eigen values such that $\frac{\sum_{j=1}^{c} \lambda_j}{\sum_{j=1}^{N} \lambda_j}$ equals the desired variance to be captured.
- Project the data points on the eigen vectors

$$y = \alpha^T (I - \frac{1_{N\times N}}{N}) (\begin{bmatrix} k(x_1, x) \\ . \\ . \\ . \\ k(x_N, x) \end{bmatrix} - K \frac{1_{N\times 1}}{N})$$

where $1_{N\times 1}$ is an $N \times 1$ matrix with all entries equal to 1. and use the projections instead of the data points.

3. Different Kernels

- The polynomial kernel of degree d is given by

$$k(x, y) = (x.y + c)^d \tag{11}$$

- The Gaussian or the radial basis function kernel is given by

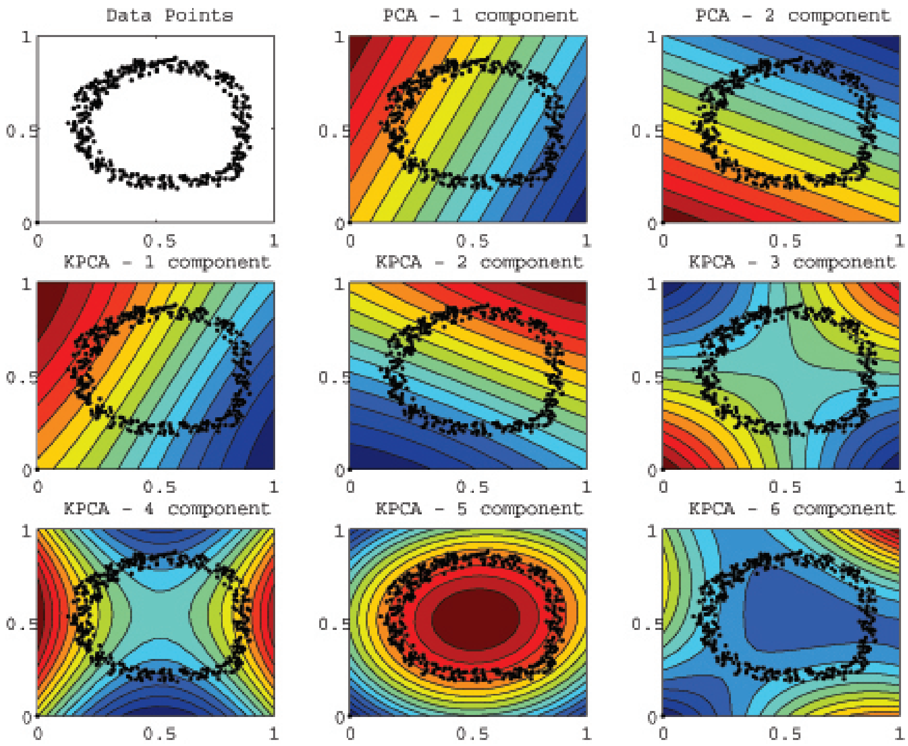
$$k(x, y) = exp(-\frac{\|x - y\|^2}{2\sigma^2}) \tag{12}$$

  where the parameter σ controls the support region of the kernel.

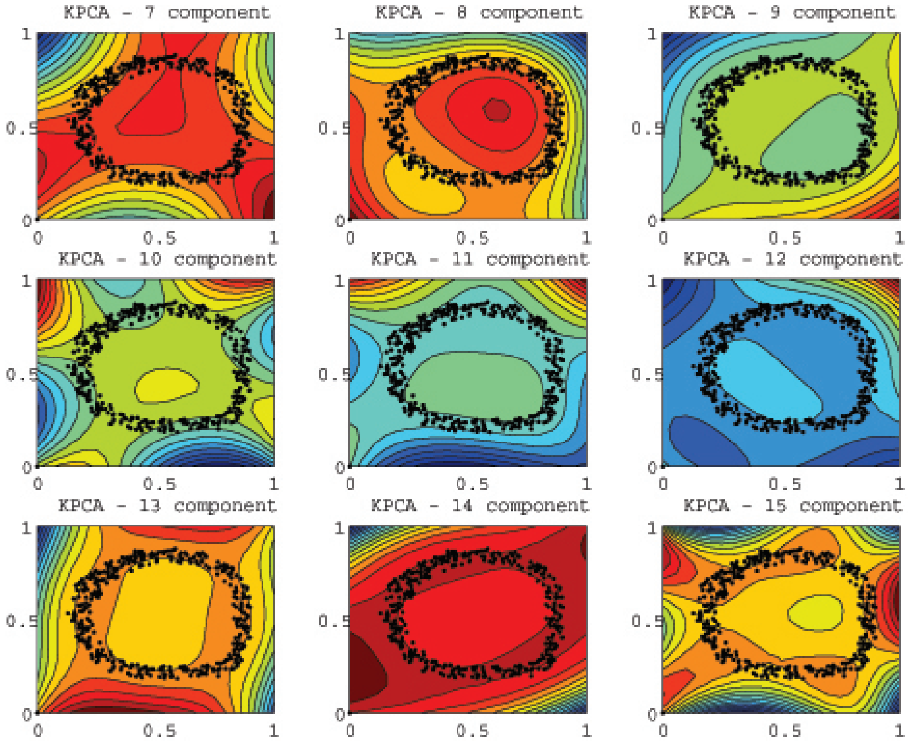
- The tanh kernel which is mostly used in Neural network type application is given by

$$k(x, y) = \tanh((x.y) + b) \tag{13}$$

Two dimensional toy example illustrating KPCA



Two dimensional toy example illustrating KPCA



4. Computational Complexity of KPCA

- The Gram matrix is an $N \times N$ matrix and finding the eigen values and eigen vectors is of $O(N^3)$ complexity.
- Also the memory storage is of $O(N^2)$
- However in most applications all the eigen vectors are not needed. The number of eigen vectors needed is usually $<< N$. In such cases we can use iterative methods to compute the eigen vectors and the eigen values and the complexity becomes $O(N^2)$.
- $O(N^2)$ complexity to compute the kernel principal components.

5. Iterative methods for Eigen value Problems

- Power Iteration
- Inverse iteration
- Rayleigh Quotient Iteration
- Deflation
- Subspace Iteration
- Krylov Subspace methods Reduce matrix to Hesenberg form using only matrix vector multiplication. These methods are based on the porjection methods onto Krylov subspaces. The three most important methods in this class are Arnoldi Iteration, Lanczos Iteration and the Jacobi Iteration.

6. Lanczos iteration

The Gram matrix in our case is real symmetric. The most commonly used method for Hermitian matrices is the Lanczos algorithm which is basically a version of Arnoldi's iteration for Hermitian matricies.

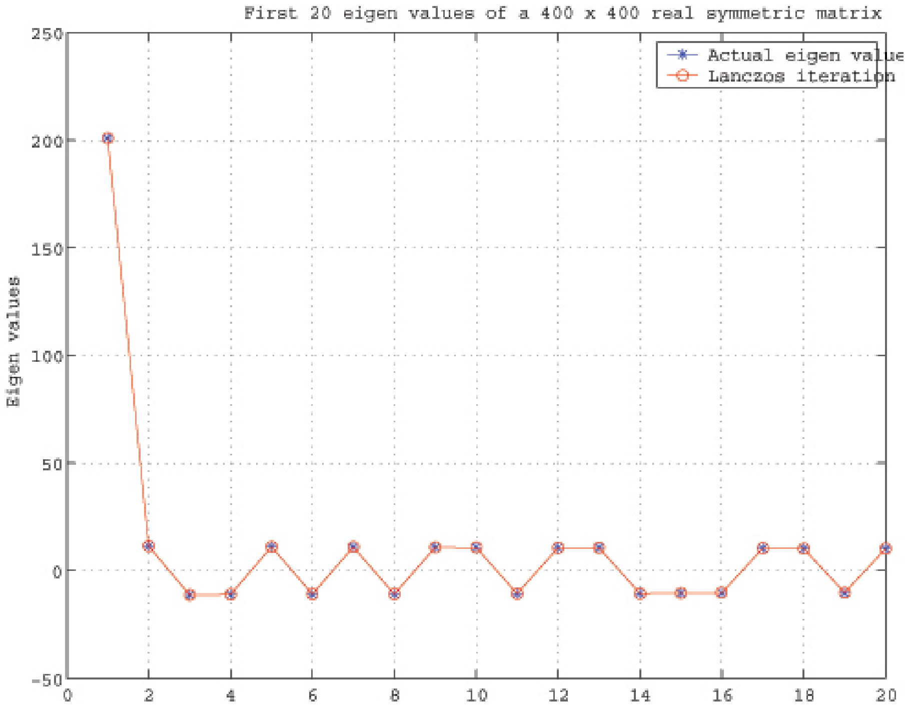
Start Choose an initial vector v_1 of norm unity. Set $\beta_1 = 0$, $v_0 = 0$

Iterate for $j = 1, 2, ...k$ do

$$\begin{aligned} w_j &= Av_j - \beta_j v_{j-1} \\ \alpha_j &= (w_j, v_j) \\ w_j &= w_j - \alpha_j v_j \\ \beta_{j+1} &= w_{j2} \\ v_{j+1} &= w_j / \beta_{j+1} \end{aligned} \tag{14}$$

- α_j and β_j are the diagonal and the subdiagonal entries of symmetric tridiagonal matrix T_k.
- Lanczos method does not produce the eigen values directly but produces the tridiagonal matrix T_k whos eigen values and eigen vectors are to the determined by some other method.
- In general if only a few eigen values are needed T_k is a very small matrix as compared to the original matrix and can be diagonalized directly.
- The above algorithm gurantees that the vectors v_i are orthogonal. However in reality exact orthogonality of these vectors is observed only at the beginning of the process.
- The problem can be overcome by reorthogonalizing the vectors as needed or do some partial or selective orthogonalization.

Lanczos Iteration



- Lanczos iteration provides a good approximation for the extreme eigen values and they require only one matrix-vector multiplication per step.
- Also most of the commercially available software have of the option of providing the function which does the matrix-vector product.
- So we can write a fast efficient matrix-vector multiplication code which can effectively fasten up the iterative procedure.

7. Polynomial kernel

$v = Ku$ where $[K]_{ij} = [k(x_i, x_j)]$ for the polynomial kernel of order p $k(x, y) = (x.y + c)^p$ i.e we want to evaluate v_i for $i = 1.....N$

$$v_i = \sum_{j=1}^{N} k(x_i, x_j)u_j$$

$$v_i = \sum_{j=1}^{N} (x_i.x_j + c)^p u_j$$

$$v_i = \sum_{j=1}^{N} u_j \sum_{k=0}^{p} \left(\begin{array}{c} p \\ k \end{array} \right) (x_i.x_j)^k c^{p-k}$$

$$v_i = \sum_{k=0}^{p} \left(\begin{array}{c} p \\ k \end{array} \right) c^{p-k} \sum_{j=1}^{N} u_j (x_i.x_j)^k$$

Using the compress operator

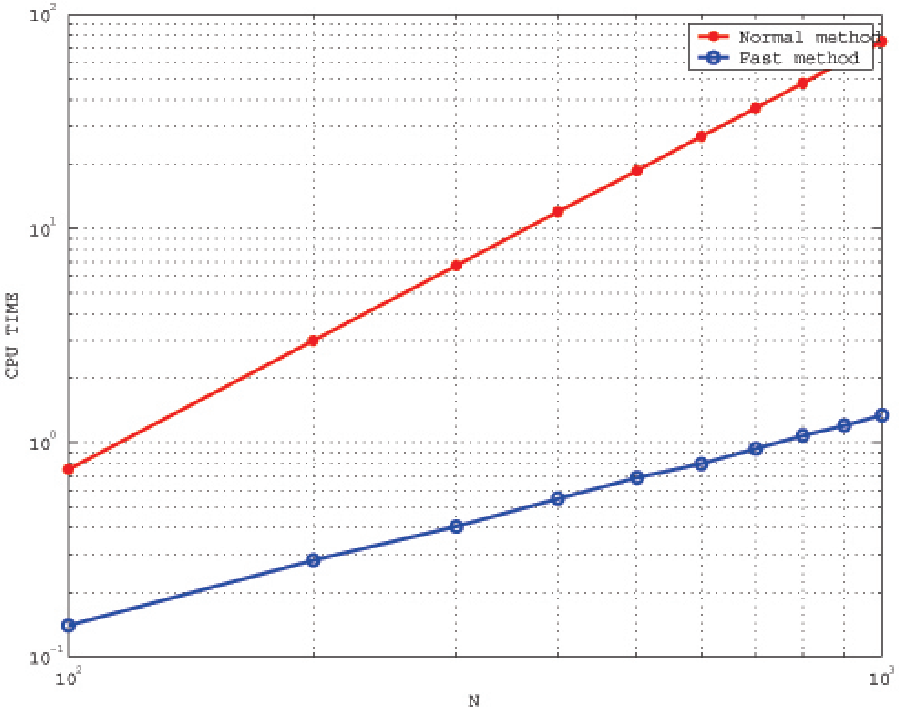
$$v_i = \sum_{k=0}^{p} \left(\begin{array}{c} p \\ k \end{array} \right) c^{p-k} \sum_{j=1}^{N} u_j comp(x_i^k).comp(x_j^k)$$

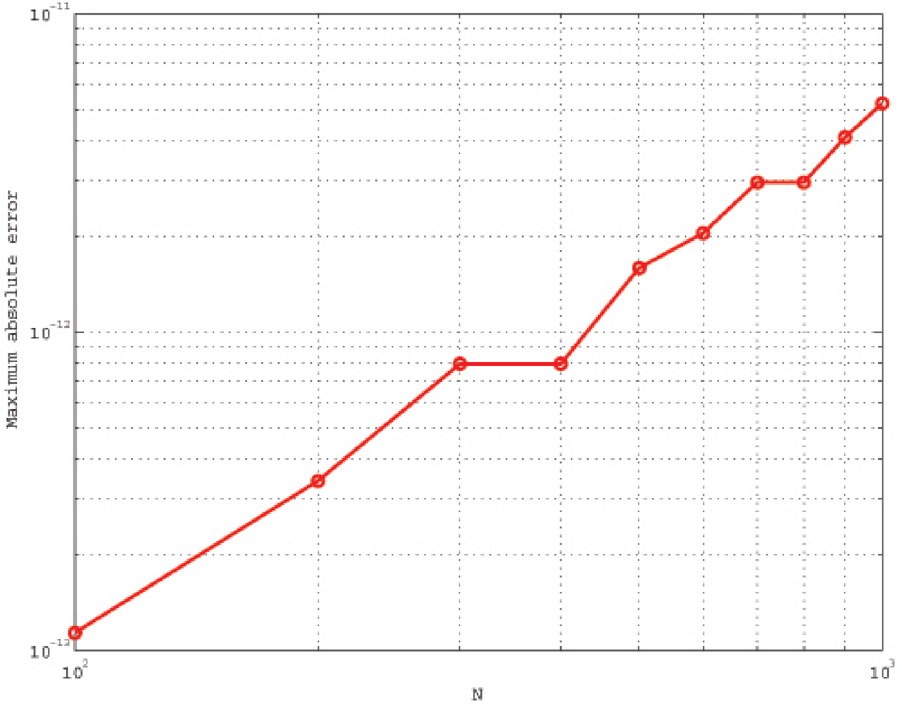
$$v_i = \sum_{k=0}^{p} comp(x_i^k).[\sum_{j=1}^{N} u_j \left(\begin{array}{c} p \\ k \end{array} \right) c^{p-k} comp(x_j^k)]$$

$$v_i = \sum_{k=0}^{p} comp(x_i^k).A_k$$

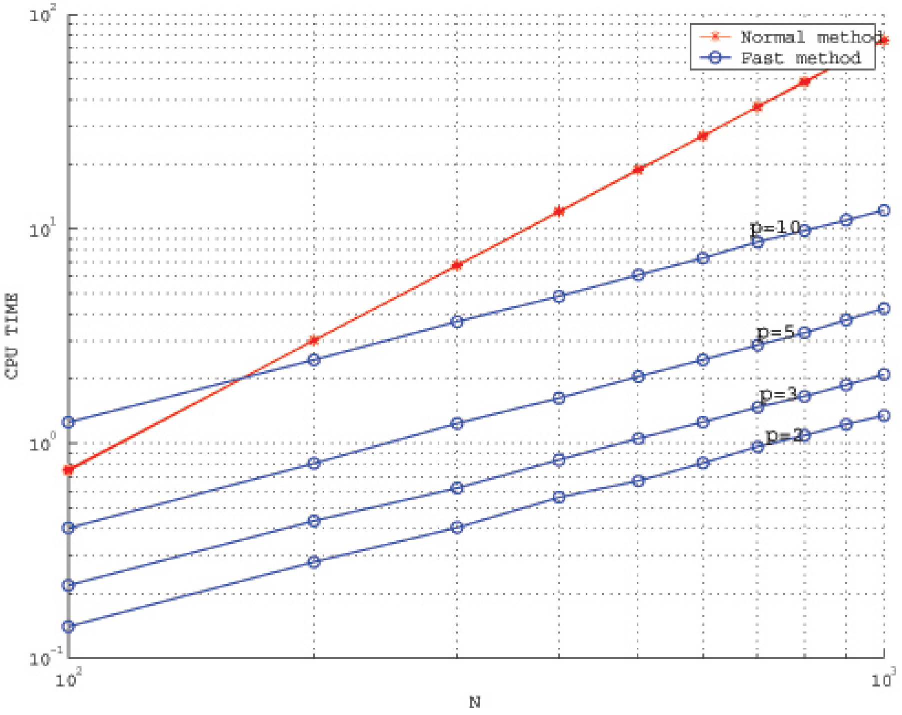
where

$$A_k = \sum_{j=1}^{N} u_j \left(\begin{array}{c} p \\ k \end{array} \right) c^{p-k} comp(x_j^k)$$


Normal method
Fast method
CPU TIME
N
10^{-1}
10^{0}
10^{1}
10^{2}
10^{2}
10^{3}



10^-11
10^-12
10^-13
10^2
10^3
Maximum absolute error
N



Normal method
Fast method
p=10
p=5
p=3
p=2
CPU TIME
N
10^{-1}
10^{0}
10^{1}
10^{2}
10^{2}
10^{3}

8. Gaussian kernel

$$K\left(x_i, x_j\right) = \exp(\tfrac{-|x_i - x_j|^2}{2\sigma^2}) \qquad i = 1 \cdots N$$

$$= \exp(\tfrac{-|x_i - x_* + x_* - x_j|^2}{2\sigma^2})$$

$$= \exp(\tfrac{-|(x_i - x_*) - (x_j - x_*)|^2}{2\sigma^2})$$

$$= \exp\left(\tfrac{-|x_i - x_*|^2}{2\sigma^2}\right) \exp\left(\tfrac{-|x_j - x_*|^2}{2\sigma^2}\right) \exp\left(\tfrac{1}{\sigma^2}\left(x_i - x_*\right) . \left(x_j - x_*\right)\right)$$

$$= \exp\left(\tfrac{-|x_i - x_*|^2}{2\sigma^2}\right) \exp\left(\tfrac{-|x_j - x_*|^2}{2\sigma^2}\right) \textstyle\sum_{m=0}^{\infty} \tfrac{1}{\sigma^{2m} m!} \left(x_i - x_*\right)^m . \left(x_j - x_*\right)^m$$

truncating upto p terms,

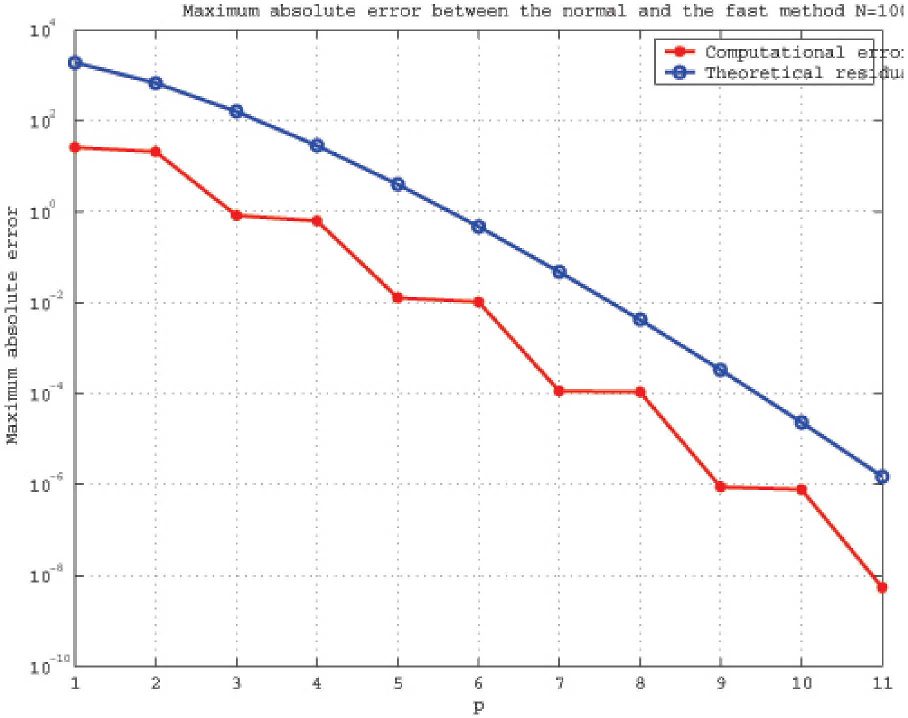
$$= \exp\left(\tfrac{-|x_i - x_*|^2}{2\sigma^2}\right) \exp\left(\tfrac{-|x_j - x_*|^2}{2\sigma^2}\right) [\textstyle\sum_{m=0}^{p-1} \tfrac{1}{\sigma^{2m} m!} \left(x_i - x_*\right)^m . \left(x_j - x_*\right)^m + error_p]$$

$$error_p| \leq \frac{\left(\frac{\sqrt{d}}{2}\right)^p \left(\frac{\sqrt{d}}{2}\right)^p}{\sigma^{2p} p!} \exp\left(\frac{1}{\sigma^2}\left(\frac{\sqrt{d}}{2}\right)\left(\frac{\sqrt{d}}{2}\right)\right)$$

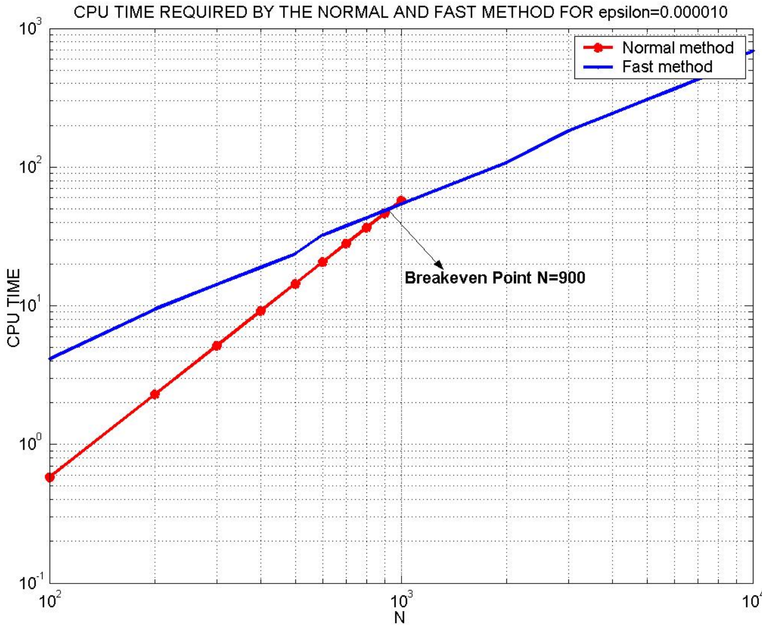
$$v_i = \sum_{m=0}^{p-1} \exp\left(\frac{-|x_i - x_*|^2}{2\sigma^2}\right) Compress((x_i - x_*)^m) \cdot$$
$$C_m + \sum_{j=1}^{N} \exp\left(\frac{-|x_i - x_*|^2}{2\sigma^2}\right) \exp\left(\frac{-|x_j - x_*|^2}{2\sigma^2}\right) error_p \, u_j$$

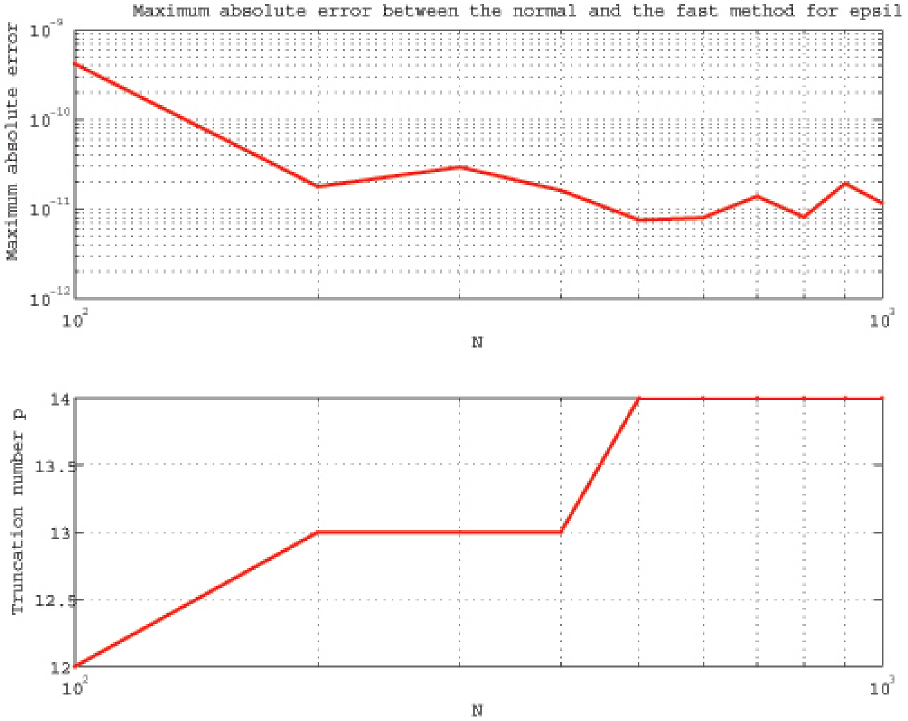
where

$$C_m = \frac{1}{\sigma^{2m} m!} \sum_{j=1}^{N} \ u_j \exp\left(\frac{-|x_j - x_*|^2}{2\sigma^2}\right) Compress(\left(x_j - x_*\right)^m)$$



Maximum absolute error between the normal and the fast method N=10
Computational erro
Theoretical residu
Maximum absolute error
p
10^{4}
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
1
2
3
4
5
6
7
8
9
10
11


CPU TIME REQUIRED BY THE NORMAL AND FAST METHOD FOR epsilon=0.000010
Normal method
Fast method
Breakeven Point N=900
CPU TIME
N
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{2}
10^{3}
10^{4}



Maximum absolute error between the normal and the fast method for epsil
Maximum absolute error
10^{-9}
10^{-10}
10^{-11}
10^{-12}
10^{2}
10^{3}
N
Truncation number p
14
13.5
13
12.5
12
10^{2}
10^{3}
N

9. Other Applications

Same idea can be extended to other machine learning algorithms like Kernel Linear Discriminant Analysis(KLDA), Kernel Biased Discriminant Analysis(KBDA), Kernel Independent Component Analysis(KICA) etc.

10. Conclusions

A fast method for diagonalizing the Gram matrix using iterative techniques.