KERNEL METHODS FOR PRINCIPAL COMPONENT ANALYSIS (PCA)

A comparative study of classical and kernel pca.

Ezukwoke K.I[1], Zareian S.J[2]

[1, 2] Department of Computer Science
Machine Learning and Data Mining
{ifeanyi.ezukwoke, samaneh.zareian.jahromi}@etu.univ-st-etienne.fr
University Jean Monnet, Saint-Etienne, France

Abstract

Principal Component Analysis (PCA) is a statistical technique for linear dimensionality reduction. Its Kernel version *kernel-PCA* is a prominent non-linear extension of the classical dimensionality reduction technique. In this paper, we present an experimental comparison between the classical PCA and the kernel-PCA. We briefly introduce and compare the performances of PCA with it kernel versions using different kernels including (**rbf, laplace, sigmoid, polynomial**).

We also experiment the use of multi-kernels for PCA and show its comparison with classical PCA using circle, moons, classification, swiss roll and iris datasets.

Keywords
Dimensionality reduction, PCA, kernel-PCA

1 INTRODUCTION

Principal Component Analysis (PCA) is an unsupervised dimension reduction technique that depends on the orthogonal transformation of a higher dimensional data space to a lower dimensional subspace [1] . An idea originally proposed by Karl Pearson [2] as a statistical method to find lines or planes of best fit in the context of regression. It has been subsequently cited and reproduced in various context and especially in machine learning for dimension reduction and **denoising- in the context of image reconstruction**. [1]

This implicitly means it is used for feature extraction since some of the features in the original space may not be required for projecting the data in the reduced subspace.

2 DIMENSION REDUCTION

PCA aims to reduce the dimension D of a high dimensional data $\{x_i\} \in \mathbb{R}^D$ into lower dimension $\{x_j\} \in \mathbb{R}^k$ where $k \ll D$ and still preserves a huge percentage of the original space D in the final subspace k, where the new features have the largest variance [3].

We consider a dataset $x_i \in \mathbb{R}^D \ \forall i = 1, \ldots, n$. We aim to project the data unto k-dimensional subspace where $k \ll D$. We denote this projection with $\hat{x} = \mathbf{Ax}$ where $A = [\mathbf{u_i}, \ldots, \mathbf{u_k}]^T$ and $\mathbf{u_i}^T\mathbf{u_i} = 1 \ \forall i = 1, \ldots, k$. We want to maximize the variance of the subspace $\hat{\mathbf{x}}$ which is the trace of the covariance matrix $\{\sum\}$, We also prove that the new projection in the subspace is given by $\sum_{i=1}^{N}(\mathbf{dx_i^T u_k})$, where $\mathbf{u_k}$ is the eigenvector corresponsing to the eigenvalues of $\sum$.

[1] source code for project is available on github

2.1 Principal Component Analysis (PCA)

Classical PCA algorithm aims at finding a linear subspace of lower dimension than the original space. The PCA algorithm is described below in stepwise order. We use $\sum$ to denote the **covariance**, which is different from summation $\sum_{i=i}^{N}$ symbol.

We perform the classical PCA technique as follows. Given a dataset $x_i \in \mathbb{R}^D$. We proceed by centering the dataset around its mean value, we do this by taking the mean.

$$\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1}$$

$$\mathbf{dx_i} = x_i - \hat{x_i} \tag{2}$$

We then calculate the covariance matrix thus

$$\sum = \frac{1}{N}\sum_{i=1}^{N} dx_i dx_i^T \tag{3}$$

After which we compute the eigenvectors $\mathbf{u_k}$ corresponding the individual eigenvalues λ_k using

$$\sum \mathbf{u_k} = \lambda \mathbf{u_k} \tag{4}$$

We then sort the eigenvectors according to the corresponding eigenvalues in decreasing order and return the transformed data based on the numbers of components k. So that

$$\hat{x} = \mathbf{dx}^T \mathbf{u_k} \tag{5}$$

We summarize the PCA procedure succinctly

Algorithm 1: PCA procedure

Input : $x \in X$ where $x \in \mathbb{R}^D$
Output : $\hat{x} \in \mathbb{R}^k$ where $k \ll D$

begin
- $\hat{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$;
- $dx_i = x_i - \hat{x}$;
- $\sum = \frac{1}{N}\sum_{i=1}^{N} dx_i dx_i^T$;
- $\sum \mathbf{u}_k = \lambda_k \mathbf{u}_k$;
- $\mathbf{u}_k = \arg\operatorname{sort}(\mathbf{u}_k)$;
- $\hat{x} = \mathbf{u}_k \cdot dx_i$;

end

PCA is limited in use to finding a linear subspace, by taking advantage of the linear dependences between the feature vectors. This limitation makes the classical PCA unsuitable for non-linearly seperable data.

2.2 Kernel Principal Component Analysis (kernel-PCA)

The Kernel PCA seeks to address the limitation of the classical PCA algorithm by generalizing to non-linear dimensionality reduction [4]. By transforming the feature space x_i into a higher dimension space $\phi(x_i)$, the kernelized version is able to capture efficiently a subspace that reduces the dimension of the original feature space.

To construct the solution of the kernel-PCA we begin like the classical approach, except in this case we assume the new feature space have **zero mean**, such that

$$\frac{1}{N}\sum_{i=1}^{N} \phi(x_i) = 0 \tag{6}$$

We then compute the covariance matrix as

$$\sum = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T \tag{7}$$

The eigenvalue and corresponding eigenvectors is given by

$$\sum \mathbf{u_i} = \lambda_k \mathbf{u_i} \quad \forall i = 1, \ldots, k \tag{8}$$

We then substitute equation (7) into (8),

$$\frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\{\phi(x_i)^T \mathbf{u_k}\} = \lambda_k \mathbf{u_k} \tag{9}$$

Hence,

$$\mathbf{u_k} = \sum_{i=1}^{N} a_{ki}\phi(x_i) \tag{10}$$

where

$$a_{ki} = \frac{1}{\lambda_k N}\phi(x_i)^T \mathbf{u_k} \tag{11}$$

Following **Mercer's** theory for kernels, we know that kernel κ takes the form

$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = \phi(x_i)^T \phi(x_j) \tag{12}$$

Subsitute equation (10) into equation (9) and multiply both sides of equation by $\phi(x_l)$

$$\frac{1}{N}\sum_{i=1}^{N}\phi(x_l)\phi(x_i)\sum_{j=1}^{N}a_{kj}\phi(x_i)\phi(x_j) = \lambda_k a_{ki}\phi(x_l)\phi(x_i)$$

Which we can re-writen as

$$\frac{1}{N}\kappa(x_l,x_i)\sum_{i=1}^{N}a_{kj}\kappa(x_i,x_j) = \lambda_k\sum_{i=1}^{N}\kappa(x_l,x_i) \quad (13)$$

$$\mathbf{K^2 a_k} = \lambda_k N \mathbf{a_k} \quad (14)$$

where

$$\mathbf{K} = \kappa(x_i,x_j) \quad (15)$$

$\mathbf{a_k}$ is the eigen vector which is an N-dimensional column vectors of a_{ki}. $\mathbf{a_k}$ can be solved from

$$\mathbf{K a_k} = \lambda_k N \mathbf{a_k} \quad (16)$$

The resulting kernel principal components transformation is

$$\hat{x} = \phi(\mathbf{x})^T\mathbf{u_k} = \sum_{i=1}^{N}a_{ki}\kappa(\mathbf{x},\mathbf{x}_i) \quad (17)$$

Given an uncentered kernel matrix, we compute the zero mean of the kernel [4] thus

Let

$$\phi(x_i) = \phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j) \quad (18)$$

$$\hat{\mathbf{K}} = \left\|\phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j)\right\|_2 \quad (19)$$

$$= \left(\phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j)\right)^T \left(\phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j)\right)$$

After expansion we have that

$$= K_{ij} - \sum_{i=1}^{N}\phi(x_i)^T\phi(x_j)\frac{1}{N} - \frac{1}{N}\sum_{i=1}^{N}\phi(x_j)^T\phi(x_i) - \frac{1}{N}\sum_{i=1}^{N}\phi(x_j)\phi(x_j)$$

This can be rewritten in short as

$$\hat{\mathbf{K}} = \mathbf{K} - 1_{1/N}\mathbf{K} - \mathbf{K}1_{1/N} + 1_{1/N}\mathbf{K}1_{1/N} \quad (20)$$

Where $\mathbf{K} = \mathbf{K}_{ij}$. $\hat{\mathbf{K}}$ is called the Gram matrix (normalized kernel matrix).

We summarize the precudure for kernel-PCA as follows

Algorithm 2: Kernel PCA Algorithm

Input : $\phi(x) \in \kappa(x,x_j)$ where $\phi(x) \in \mathbb{R}^D$. Let $\mathbf{K} = \kappa(x_i,x_j)$

Output : $\hat{x} \in \mathbb{R}^k$ where $k \ll D$

begin
- Select a kernel κ;
- Construct Gram matrix $\hat{\mathbf{K}} = \mathbf{K} - 1_{1/N}\mathbf{K} - \mathbf{K}1_{1/N} + 1_{1/N}\mathbf{K}1_{1/N}$;
- Solve eigen problem $\mathbf{K u_k} = \lambda \mathbf{u_k}$;
- Project data in new space $\hat{x} = \sum_{i=1}^{N}\mathbf{u_k}\kappa$;

end

The kernel PCA algorithm is expressed only in terms of dot products, this trick allows us to construct kernels only from training data $\{\mathbf{x_i}\}$ [5].

2.3 Kernels

We introduce the commonly used kernels and a brief overview of Multiple kernels used.

- **Linear kernel**

$$\kappa(x_i,x_j) = \mathbf{x}_i\mathbf{x}_j^T \quad (21)$$

- **Polynomial kernel**

$$\kappa(x_i,x_j) = (\mathbf{x}_i\mathbf{x}_j^T + c)^d \quad (22)$$

where $c \geq 0$ and d is the degree of the polynomial usually greated than 2.

- **RBF(Radial Basis Function) kernel**
  Sometimes referred to as the **Guassian kernel**.

$$\kappa(x_i, x_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2) \quad (23)$$

  where $\gamma = \frac{1}{2\sigma^2}$.

- **Sigmoid kernel**

$$\kappa(x_i, x_j) = \tanh(\gamma \mathbf{x}_i \mathbf{x}_j^T + c) \quad (24)$$

  where $c \geq 0$ and $\gamma = \frac{1}{2\sigma^2}$.

- **Cosine kernel**

$$\kappa(x_i, x_j) = \frac{\mathbf{x}_i \mathbf{x}_j^T}{||\mathbf{x}_i||||\mathbf{x}_j||} \quad (25)$$

2.4 Multi-kernel

The reason behind the use of multiple kernel is similar to the notion of multiclassification, where cross-validation is used to select the best performing classifier [6]. By using multiple kernel, we hope to learn a different similar in the kernel space not easily observed when using single kernel.

We can prove from **Mercer's Theorem** that a kernel is **Positive Semi-Definite (PSD)** if $u^T \kappa(x_i, x_j) u \geq 0$. Hence by performing arithmetic or any mathematical operation on two or more kernel matrix, we obtain a new kernel capable of exploiting different property or similarities of training data.

Given a kernel κ, we prove that κ is PD if

$$\langle u, \kappa u \rangle \geq 0 \quad (26)$$

Proposition: *A symmetric function* κ*:* $\chi \to \mathbb{R}$ *is positive semi-definite if and only if* $\langle u, \kappa u \rangle \geq 0$

Proof:
Suppose that κ is a kernel which is the inner product of the mapping functions $\langle \phi(x_i)\phi(x_j) \rangle$. κ is a kernel if its inner product are positive and the solution of $\kappa u = \lambda u$ gives non-negative eigenvalues.

So that,

$$\langle u, \kappa u \rangle = \sum_{i=1}^{N} u_i \cdot (\kappa u_i) \quad (27)$$

$$= \sum_{i=1}^{N} u_i \sum_{j=1}^{N} \langle \phi(x_i)\phi(x_j)_{\mathcal{H}} \rangle u_j \quad (28)$$

Where $\mathcal{H}$ represent the Hilbert space we project the kernel [7].

$$= \left\langle \sum_{i=1}^{N} \sum_{j=1}^{N} u_i \phi(x_i), u_j \phi(x_j) \right\rangle_{\mathcal{H}} \quad (29)$$

$$\langle u, \kappa u \rangle = \left\| \sum_{i=1}^{N} u_i \phi(x_i) \right\|_{\mathcal{H}}^2 \geq 0 \quad (30)$$

Therefore κ is positive definite.

Using this property of the kernel κ we are able to create serveral other kernels including

- **LinearRBF**
  Here we combine two kernels, precisely **Linear and RBF kernel** using their inner product.

$$\hat{\mathbf{K}}_{\mathbf{linrbf}} = \kappa(x_i, x_j)^T \kappa(x_i, x_l) \quad (31)$$

- **RBFPoly**
  Here we combine **RBF and Polynomial kernels** using their inner product.

$$\hat{\mathbf{K}}_{\mathbf{rbfpoly}} = \kappa(x_i, x_j)^T \kappa(x_i, x_l) \quad (32)$$

- **RBFCosine**
  RBF and Cosine kernels using their inner product.

$$\hat{\mathbf{K}}_{\mathbf{rbfcosine}} = \kappa(x_i, x_j)^T \kappa(x_i, x_l) \quad (33)$$

- **EtaKernel**
  The **EtaKernel** is a composite combination of **LinearRBF, RBFPoly and RBFCosine** and it is given by

$$\hat{\mathbf{K}}_{\mathbf{etarbf}} = \hat{\mathbf{K}}_{\mathbf{linrbf}}^T \hat{\mathbf{K}}_{\mathbf{rbfpoly}} + \hat{\mathbf{K}}_{\mathbf{rbfpoly}}^T \hat{\mathbf{K}}_{\mathbf{rbfcosine}}$$

3 EXPERIMENT

3.1 Dataset

We perform a comparison between PCA and its kernel version using different datasets including circle, moon, classification, swiss roll and iris datasets. Given N, the total number of samples. Circle ($N = 1000$), Moon ($N = 1000$), Classification ($N = 1000$), Swiss roll ($N = 1000$), Iris ($N = 150$).

3.1.1 Data Description

All datasets except Iris have equal numbers of samples ($N = 1000$). Iris contains only $N = 150$ datapoints. Iris dataset contains $four$ features, circle dataset contains two features (two circles with different class), Moon dataset contains two features, Classification dataset contains $twenty$ features and Swiss roll contains $three$ features.

The original data is plotted in 2D as seen on the first column of Figure 1.

3.2 PCA and Kernel PCA Results

We begin experimentation with 2 principal components and a small gamma. We take gamma as $1/N$ where ($\gamma = 0.001$) for all datasets except iris ($\gamma = 0.007$) and polynomial degree ($d = 3$). Figure 1 shows the result of performing kernel and classical PCA on all datasets. Linear PCA follows the column after the **original** column.

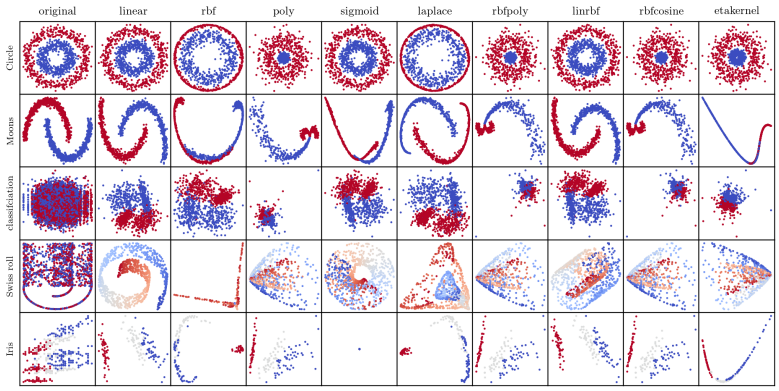

Figure 1: PCA and Kernel PCA with $\gamma = 0.001$ $d = 3$

We observe the result of PCA and its kernel version on **Circle** data using the above configurations. We notice little change in the linear PCA result. However, **rbf and laplace** kernels show considerable amount grouping for the *red* class, hence the tight shrinkage, as well as expansion for the *blue* class. **rbfpoly and rbfcosine** show similar properties for their results. We see infact that using multiple kernel PCA reduces the dimension but does not help seperability.

Similar properties can be seen for **moon and classification** datasets. Both PCA and kernel PCA with 2 components are unable to create a subspace where the components are linearly seperable. **Laplace** kernel however shows it is capable of creating a lower dimensional subspace of the original space where we can see a tangible difference in the class seperation.

For Iris dataset we observe **linear pca**, **poly kernel**, **rbfpoly kernel**, **linrbf kernel** and **rbfcosine** have the capacity to reduce the dimension and project it unto a lower subspace where the classes are almost linearly seperable.

We further our experiment by keeping the initial value of principal components $k = 2$, but changing the value of gamma ($\gamma = 0.01$) and $d = 4$ for all datasets. Figure 2 shows the result of this change. We observe rbf kernel result for **circle and moon** datasets

takes the form of **linear pca**. Consequently, rbfpoly and rbfcosine produces similar result for circle data (conical shape with one class at the vertex of the cone), however, we see **Swiss roll** has taken a donut shape for **rbf kernel** and **iris** also takes a linear form with this configuration on **rbf**. The result of **sigmoid kernel** is invertedly shifted for circle dataset.

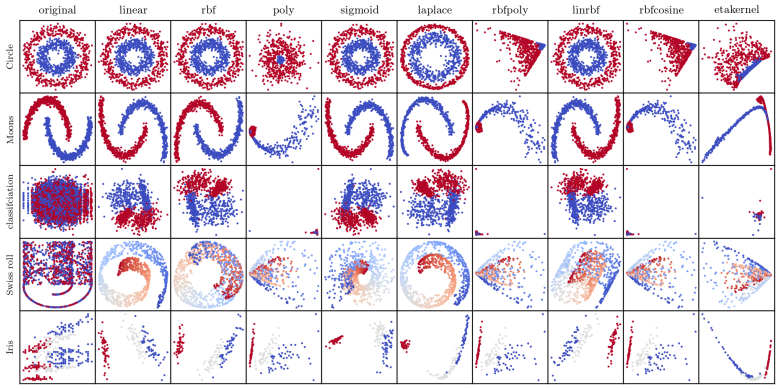

Figure 2: PCA and Kernel PCA with $\gamma = 0.01$ $d = 4$

The sigmoid kernel produces similar result with the linear pca for moon dataset, dotted donut shape for Swiss roll and almost a linearly seperable form for iris data; **Laplace kernel** shows no big difference as circle, moon and classification takes previous form. It however, takes a spiral shape for swiss data. **rbfpoly** stays the same, **linrbf, rbfcosine and etakernel** all show an inversion from previous results. We observe that increasing the polynomial degree impacts the resulting kernel result for moon and classification datasets. No linearly seperable result was achieved by increasing the polynomial degree we observe. We observe the **etakernel** returns an aligned structural subspace, where it tries to join or align the classes into unitary space by overlapping them.

Again we change the configuration such that **gamma** ($\gamma = 1$) and $d = 5$. The figure 3 shows the result of this configuration.

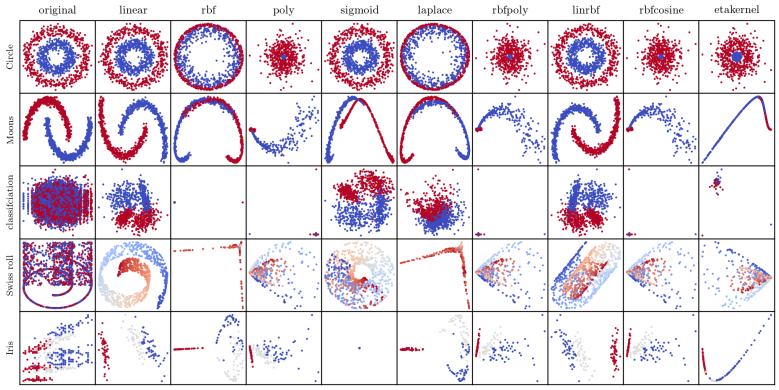

Figure 3: PCA and Kernel PCA with $\gamma = 1$ $d = 5$

We observe similar result for **rbf** and **laplace** kernels on circle, moon, swiss rol and iris datasets. Polynomial shows almost similar result to when $\gamma = 0.01$ for all datasets. The only new information here is the result from laplace and rbf as they try to linearly seperate the classification, swiss roll and iris datasets.

Finally we adjust the **gamma** ($\gamma = 10$) and $d = 6$ and observe a unique representation of our data in the new subspace. Figure 4 shows the variety of representation using a high γ and polynomial degree d.

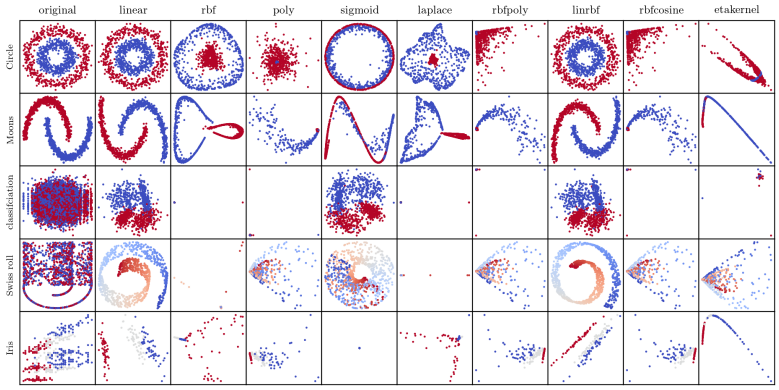

Figure 4: PCA and Kernel PCA with $\gamma = 10$ $d = 6$

Rbf shows a surprising behavior on the circle dataset by trying to collapse one class inside and expanding the other class. A behavior not observed for small gammas. On the moon dataset it evidently separates the classes into **half-butterfly** wings. It however fails to project the iris dataset in a linearly seperable subspace. We can conclude that linear PCA is best for iris data compression.

Polynomial kernel continues to shrink one class of moon and circle data as the degree of polynomial continues to increase, while swiss data result is unchanged from previous. **Sigmoid kernel** create a circular representaion for circle data with one class acting almost as a membrane for the other. An overlapping $S - shape$ representation is observed for moon data, $donut - shape$ for swiss roll and a single dot representation for iris data.

Laplace kernel almost returns similar result as rbf for circle and classification and swiss roll datasets. **Rbfpoly and rbfcosine** return similar representation for all dataset and **etakernel** returns a curved-line representation for moon and iris dataset.

3.3 Performance analysis

We compare the performances of linear and its different kernel versions.

Explained Variance (%) for $k = 2$, $\gamma = 10$ and $d = 6$									
kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	rbf cosine	eta kernel
Circle	81/18	9/8	91/7	112/26	3/2	**99/0.6**	83/17	99/0.1	107/0
Moon	50/50	8/8	28/26	**71/69**	2/2	34/30	50/49	34/30	905/29
Classification	9/8	0.1/0.1	4/3	30/27	0.1/0.1	21/12	8/7	22/12	**34/17**
Swiss roll	39/32	0.3/0.3	34/26	**100/96**	0.2/0.1	57/26	38/32	57/26	94/4
Iris	92/5	5.4/4.4	97	99/0.8	1.6/1.4	**100/0**	144/9	**100/0**	**100/2**

Running Time (*secs*)									
kernels	linear	rbf	poly	sigmoid	laplace	rbfpoly	linrbf	rbf cosine	eta kernel
Circle	**0.61**	1.29	0.83	0.87	0.69	1.22	0.82	1.21	2.25
Moon	0.63	1.37	0.80	**0.46**	0.70	1.16	0.91	1.15	2.27
Classification	0.65	0.94	0.84	**0.63**	0.91	1.10	1.44	1.10	2.92
Swiss roll	0.65	0.79	0.81	0.52	**0.47**	1.18	1.29	1.19	2.58
Iris	**0.010**	0.013	0.016	0.01	0.01	0.02	0.17	0.02	0.04

From the table above, we observe the best performing algorithm with high **explained variance** are the kernel pcas. **rbfpoly** is the best performing for circle dataset. **sigmoid** for moon dataset, **etakernel** for classification dataset, **sigmoid** for Swiss roll and again **rbfpoly** for Iris dataset.

We also observe that linear pca is the fastest in terms of running time when compared to its kernel versions.

4 CONCLUSION

In this paper we present a qualitative and experimental detailed difference between PCA and its kernel version. We introduced the concept of multi-kernel and apply it for different datasets including circle, moon, classification, swiss roll and iris datasets.

We compared the performances of classical PCA and its kernel version and conclude that although kernel PCA serves better to reduce dimension of non-linearly sepearble data, its only drawback is with high computation time when compared to classical PCA.

We conclude that kernel PCAs are better performing than classical PCA for high dimensional data reduction.

References

[1] Diamantaras K.I, Kung S.Y. Principal component neural networks: theory and applications. John Wiley & Sons, Inc., ISBN:0-471-05436-4, New York, USA, 1996.

[2] Pearson K. On lineas and Planes of closest fit to systems of points in space. *Philos Mag A*, 6:559–572, 1901.

[3] Bishop, Christopher M. Pattern Recognition and Machine Leaning. Springer, Cambridge, U.K, 2006.

[4] Bernhard Schölkopf, Alexander Smola, KlausRobert Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5): 1299-1319, 1998.

[5] Weinberger, Kilan Q., Sha, Fei et al. Learning a kernel matrix for nonlin- ear

dimensionality reduction. *Proceedings of the 21st International Conference on Machine Learning (ICML-04)*, pp. 839–846. ACM Press, 2004.

[6] Mehmet Gönen, Ethem Alpaydin. Multiple Kernel Learning Algorithms. *Journal of Machine Learning Research*, 12:2211-2268, 2011.

[7] John Shawt-Taylor, Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press, ISBN:9780511809682, pg47-83, 2011.

[8] Ben-Hur A., Noble W.S, Kernel methods for predicting protein-protein interations. *Bioimformatics*. 21(Supp 1):i38-46, 2005.