Paper Analysis
Kernel PCA and de-noising in feature spaces
by S. Mika et al.

Team 19
Federico Baldassarre, Zacharie Brodard, Alfredo Fanghella, Lucas Rodés i Guirao
Supervised by Hedvig Kjellström
DD2434 Machine Learning Advance Course Project
KTH Royal Institute of Technology
{fedbal, zacharie, ajfv, lucasrg} @kth.se

Abstract

Kernel PCA can be seen as a generalization of the linear PCA, where non-linear features can be extracted from data. S. Mika *et al* (1998) developed Kernel PCA theory and studied approximate pre-images using Gaussian kernels. Furthermore, they tested Kernel PCA and approximate pre-images on different experiments, which involved data reconstruction and de-noising. This paper reviews their results and implementation.

1 Introduction

Principal Component Analysis (PCA) theory has been studied for almost a century [1, 2] now. Given a data set of dimension N, PCA approach aims to find a low-dimensional linear subspace of $\mathbb{R}^N$, where the maximum variance of the data is retained. Thus, whenever the data does not lie in such a subspace, PCA will perform very poorly. However, the data might still lie in a low dimensional *nonlinear* subspace. To overcome this and still find a low-dimensional representation of the data, Kernel PCA first maps the data into a higher dimensional space and then performs regular PCA to find the linear manifold within the higher dimensional space where the mapped data lies on.

Overall, Kernel PCA finds a low-dimensional representation of the data by projecting the data in a higher dimensional space. As we will later see, we are actually only interested in the inner products of all data set pairs. Hence, the *kernel* trick is employed to avoid considering this higher dimensional space explicitly. In [3], only Gaussian Kernel [4] was considered. Pre-images of the test data are obtained for the purpose of de-noising, reconstruction and performance comparison with linear PCA.

This work is an analysis of [3] and aims at reproducing their results. We took the opportunity to formalize some algorithm steps, with special emphasis on the centering of the mapped data, using appropriate algebra tools, which was not carefully analyzed in the original paper.

1.1 Outline

Inspired by [3] and [5], Section 2 highlights the key mathematical background theory regarding Kernel PCA. Section 3 revisits the experiments done in [3]. Finally, in Section 4 we sum up our observations and we draw the conclusions of our experiments.

2 Kernel PCA

Consider the training set of observations $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell\}$, where $\boldsymbol{x}_i \in \mathbb{R}^N$ for $k = 1, \ldots, \ell$, being N and ℓ the number of features and the number of observations, respectively. Our goal is to use $\mathcal{X}$ to de-noise or reconstruct new observations from the test set $\mathcal{T} = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_L\}$, where $\boldsymbol{t}_m \in \mathbb{R}^N$ for $m = 1, \ldots, L$.

2.1 Train Data

Consider the function $\boldsymbol{\Phi} : \mathbb{R}^N \to \mathbf{F}$, which maps the data to a high dimension feature space $\mathbf{F}$ (meaning that $\dim(\mathbf{F}) > N$). Let us define the centered $\boldsymbol{\Phi}$-images of the train data $\boldsymbol{x}_i$ as $\boldsymbol{\Phi}_c(\boldsymbol{x}_i) = \boldsymbol{\Phi}(\boldsymbol{x}_i) - \frac{1}{\ell}\sum_{i=1}^{\ell}\boldsymbol{\Phi}(\boldsymbol{x}_i)$, whose covariance matrix is given by:

$$C = \frac{1}{\ell}\sum_{i=1}^{\ell}\boldsymbol{\Phi}_c(\boldsymbol{x}_i)\boldsymbol{\Phi}_c(\boldsymbol{x}_i)^T.$$

The eigenvalues $\lambda > 0$ and eigenvector $\mathbf{v} \in \mathbf{F}\backslash\{\mathbf{0}\}$ of $\bar{C}$ are obtained by solving

$$\lambda\mathbf{v} = C\mathbf{v} = \frac{1}{\ell}\sum_{i=1}^{\ell}\boldsymbol{\Phi}_c(\boldsymbol{x}_i)\boldsymbol{\Phi}_c(\boldsymbol{x}_i)^T \cdot \mathbf{v} = \frac{1}{\ell}\sum_{i=1}^{\ell}\big(\boldsymbol{\Phi}_c(\boldsymbol{x}_i)\cdot\mathbf{v}\big)\boldsymbol{\Phi}_c(\boldsymbol{x}_i), \tag{1}$$

where '$\cdot$' denotes the inner product operator. From (1) we observe that the eigenvector $\mathbf{v}$ lie in the span of the centered $\boldsymbol{\Phi}$-images of the train data and hence can be expressed as a linear combinations of them, using α_i with $i = 1, \ldots, \ell$ as coefficients:

$$\mathbf{v} = \sum_{i=1}^{\ell}\alpha_i\boldsymbol{\Phi}_c(\boldsymbol{x}_i). \tag{2}$$

No information is lost if (1) is we multiply the equation for $\boldsymbol{\Phi}_c(\boldsymbol{x}_i)$.

$$\lambda(\boldsymbol{\Phi}_c(\boldsymbol{x}_i)\cdot\mathbf{v}) = (\boldsymbol{\Phi}_c(\boldsymbol{x}_i)\cdot C\mathbf{v}) \quad \text{for all } i = 1, 2, \ldots, \ell. \tag{3}$$

Combining the equations obtained from (3) with (2) and rewriting everything in matrix form, we obtain

$$\ell\lambda\boldsymbol{\alpha} = K_c\boldsymbol{\alpha}, \tag{4}$$

where we have defined the $\ell \times \ell$ matrix K_c with coefficients $(K_c)_{ij} := k_c(\boldsymbol{x}_i, \boldsymbol{x}_j) = \big(\boldsymbol{\Phi}_c(\boldsymbol{x}_i) \cdot \boldsymbol{\Phi}_c(\boldsymbol{x}_j)\big)$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_\ell)^T$. Here, the Kernel trick is used, which allows us to operate in the feature space $\mathbf{F}$ without explicitly computing the new coordinates of the data in this space.

However, it is defined using the non-centered mapped data, i.e. $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \big(\boldsymbol{\Phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\Phi}(\boldsymbol{x}_j)\big)$. In this regard, we can operate as in [6]: define the matrix K with coefficients $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, such that we express K_c in terms of K as

$$K_c = K - \mathbf{1}_{\boldsymbol{\ell}}K - K\mathbf{1}_{\boldsymbol{\ell}} + \mathbf{1}_{\boldsymbol{\ell}}K\mathbf{1}_{\boldsymbol{\ell}}, \tag{5}$$

where $\mathbf{1}_{\boldsymbol{\ell}}$ is an $\ell \times \ell$ matrix with all coefficients equal to $1/\ell$. Next, let us consider the eigenvalues of K_c (solutions $\ell\lambda$ in (4)), denoted by $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_\ell$, with their respective eigenvector $\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \ldots, \boldsymbol{\alpha}^\ell$, which are normalized such that $\lambda_k(\boldsymbol{\alpha}^k \cdot \boldsymbol{\alpha}^k) = 1$ since, $(\mathbf{v}^k \cdot \mathbf{v}^k) = 1$ for all $k = 1, \ldots, \ell$.

2.2 Test Data

Given a test point $\boldsymbol{t}_m$, let us define β_m^k as the projection of its centered $\boldsymbol{\Phi}$-image onto the k:th eigenvector, i.e. $\beta_m^k := (\mathbf{v}^k \cdot \boldsymbol{\Phi}_c(\boldsymbol{t}_m)) = \sum_{i=1}^{\ell} \alpha_i^k k_c(\boldsymbol{t}_m, \boldsymbol{x}_i)$, where α_i^k is the i:th coefficient of $\boldsymbol{\alpha}^k$. Next, we can construct the centered $\boldsymbol{\Phi}$-image of the test point $\boldsymbol{t}_m$ using its projections β_m^k onto the first n principal components, i.e.

$$\mathrm{P_n}\boldsymbol{\Phi}_c(\boldsymbol{t}_m) = \sum_{k=1}^{n} \beta_m^k \mathbf{v}^k, \tag{6}$$

where $\mathrm{P_n}$ is the corresponding projection operator onto n principal components. Although perfect reconstruction might not be achieved, Kernel PCA ensures that (i) the error $\sum_i ||\boldsymbol{\Phi}_c(\boldsymbol{t}_m) - \mathrm{P_n}\boldsymbol{\Phi}_c(\boldsymbol{t}_m)||^2$ is minimal (interesting for de-noising applications) and that (ii) the preserved variance is maximal among all combinations of n orthogonal vectors in $\mathbf{F}$. In this regard, let us find a vector $\boldsymbol{z}_m \in \mathbb{R}^N$ such that it verifies $\boldsymbol{\Phi}_c(\boldsymbol{z}_m) = \mathrm{P_n}\boldsymbol{\Phi}_c(\boldsymbol{t}_m)$. In general, we seek to find a value of $\boldsymbol{z}_m$ such that $\rho(\boldsymbol{z}_m) = ||\ \boldsymbol{\Phi}_c(\boldsymbol{z}_m) - \mathrm{P_n}\boldsymbol{\Phi}_c(\boldsymbol{t}_m)||^2$ is minimized.

2.3 Pre images for Gaussian Kernels

Let us now consider only the case where $k(\boldsymbol{x}, \boldsymbol{y}) = k(||\boldsymbol{x} - \boldsymbol{y}||^2)$ (holds for Gaussian Kernel), which implies that $k(\boldsymbol{x}, \boldsymbol{x}) \equiv$ const for all $\boldsymbol{x}$. This yields to the equivalent maximization problem of

$$\rho(\boldsymbol{z}_m) = \sum_{i=1}^{\ell} \gamma_{i,m} k(\boldsymbol{z}_m, \boldsymbol{x}_i) + \Omega', \tag{7}$$

where $\gamma_{i,m} = \sum_{k=1}^{n} \beta_m^k \alpha_i^k$ and Ω' stands for all terms independent of $\boldsymbol{z}_m$. To minimize (7), we find $\boldsymbol{z}_m$ such that $\nabla_{\boldsymbol{z}_m}\rho(\boldsymbol{z}_m) = 0$. Assuming Gaussian Kernel, the following iterative scheme is obtained for the estimation of $\boldsymbol{z}_m$

$$\boldsymbol{z}_m^{new} = \frac{\sum_{i=1}^{\ell} \gamma_{i,m} \exp\big(-||\boldsymbol{z}_m^{old} - \boldsymbol{x}_i||^2/c\big)\, \boldsymbol{x}_i}{\sum_{i=1}^{\ell} \gamma_{i,m} \exp\big(-||\boldsymbol{z}_m^{old} - \boldsymbol{x}_i||^2/c\big)}, \qquad \text{for some } c > 0. \tag{8}$$

We observe that (8) is a normalized weighted sum of training data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$. The superscripts '^{old}' and '^{new}' are used to mark a value at the beginning and at the end of each iteration, respectively.

3 Implementation

To ease the implementation we expressed the algorithm to obtain the weighting factors $\gamma_{i,m}$ in matrix form. First, we defined the $\ell \times n$ matrix A containing the first n Eigenvalues of K_c in its columns, such that $(A)_{ik} = \alpha_i^k$. Furthermore, we defined the $L \times \ell$ Kernel matrix of the test data K^{test}, with coefficients $(K^{test})_{mi} = k(\boldsymbol{t}_m, \boldsymbol{x}_i)$. Similarly to (5) we centered it using

$$K_c^{test} = K^{test} - \mathbf{1_\ell}'K - K^{test}\mathbf{1_\ell} + \mathbf{1_\ell}'K\mathbf{1_\ell},$$

where $\mathbf{1_\ell}'$ is an $L \times \ell$ matrix will all coefficients equal to $1/\ell$. We used this matrices to obtain the $L \times n$ matrix $B = K_c^{test}A$, whose coefficients are given by $(B)_{ik} = \beta_i^k$. Finally, we defined the $L \times \ell$ matrix Γ, which contains the weighting factors as $(\Gamma)_{mi} = \gamma_{i,m}$ and is obtained as

$$\Gamma = BA^T = K_c^{test}AA^T.$$

3.1 Experiments

11 gaussians

The first experiment we reproduced was a de-noising experiment using a gaussian Kernel. The goal here was to realize a comparison between Kernel PCA and linear PCA. The data set was constructed as follows :

- Consider 11 spherical gaussian distributions in $\mathbb{R}^{10}$ with centers picked uniformly at random in $[-1, 1]^{10}$ and a constant variance σ^2.
- Construct a training data set by sampling 100 points from each Gaussian.
- Construct a testing data set by sampling again 33 points from each Gaussian.

We then trained our Kernel PCA implementation on the training data set, getting the α matrix. For each of the test data points we then deduced the γ factors and applied the iterative scheme described in equation (8) with the considered data point as starting point. Using the data point as starting point is common in de-noising algorithms. It allows to find a local minimum for $\rho(z_m)$ that is closer to our data point than the one we may get with a random starting point.

We also conducted de-noising on the same data-set using a linear PCA algorithm.

To compare performances of the different methods, we computed the mean square distance of all de-noised test points to the center of their Gaussian. We then computed the ratio of these values for Kernel PCA and linear PCA. Results are shown in Table 1.

σ	$n = 1$	2	3	4	5	6	7	8	9
0.05	496.24	1614.73	1816.32	1598.64	972.89	623.70	364.37	153.05	90.50
0.1	178.33	752.52	803.60	586.85	396.99	259.77	152.16	104.50	89.89
0.2	37.65	63.62	18.41	5.16	3.35	2.82	2.73	2.30	3.43
0.4	6.03	4.06	1.77	1.78	1.87	2.08	5.49	9.24	9.54
0.8	0.62	0.98	1.25	1.54	1.83	1.80	1.75	1.57	1.53

(a) Our results

σ	$n = 1$	2	3	4	5	6	7	8	9
0.05	2058.42	1238.36	846.14	565.41	309.64	170.36	125.97	104.40	92.23
0.1	10.22	31.32	21.51	29.24	27.66	23.53	29.64	40.07	63.41
0.2	0.99	1.12	1.18	1.50	2.11	2.73	3.72	5.09	6.32
0.4	1.07	1.26	1.44	1.64	1.91	2.08	2.22	2.34	2.47
0.8	1.23	1.39	1.54	1.70	1.80	1.96	2.10	2.25	2.39

(b) Paper's results

Table 1: 11 gaussians experiment results

Our results, although they are not exactly similar to the ones presented in the paper, allow to make the same observations:

- Kernel PCA performs better than regular PCA for almost all σ and n.
- Extreme superiority of Kernel PCA for small values of σ: linear PCA cannot cover the 11 centers with less than 10 dimensions. For Kernel PCA it is easier, especially since we can use the test data point (which is located very close when to the center when σ is small) as a starting point for our iterations. This is supported by the fact that using a random starting point in $[-1, 1]^{10}$ gives far worse results.

Several reasons can be put forward to account for the differences between our results and the ones of the paper. Although all parameters are given, there are some parts that are still open for interpretation (such as choosing a new starting point in case of instabilities). It is also possible that we miss-interpreted some of the parameters, yielding to different results. Other explanations can be an error in our implementation, although we reviewed it several time and applied successfully the same code to the two other de-noising examples.

De-noising shapes

To reproduce this example we first generated synthetic data. Starting from a half circle and a square, Gaussian noise was added (in the radial direction and the direction orthogonal to the each side, respectively). We then de-noised the data using Kernel PCA, standard PCA, the principal curves algorithm and, for the square, a variation of the principal curves algorithm in which the starting curve is a circle. Due to lack of an available implementation, the nonlinear auto-encoder was omitted. Results are shown in figure 1.

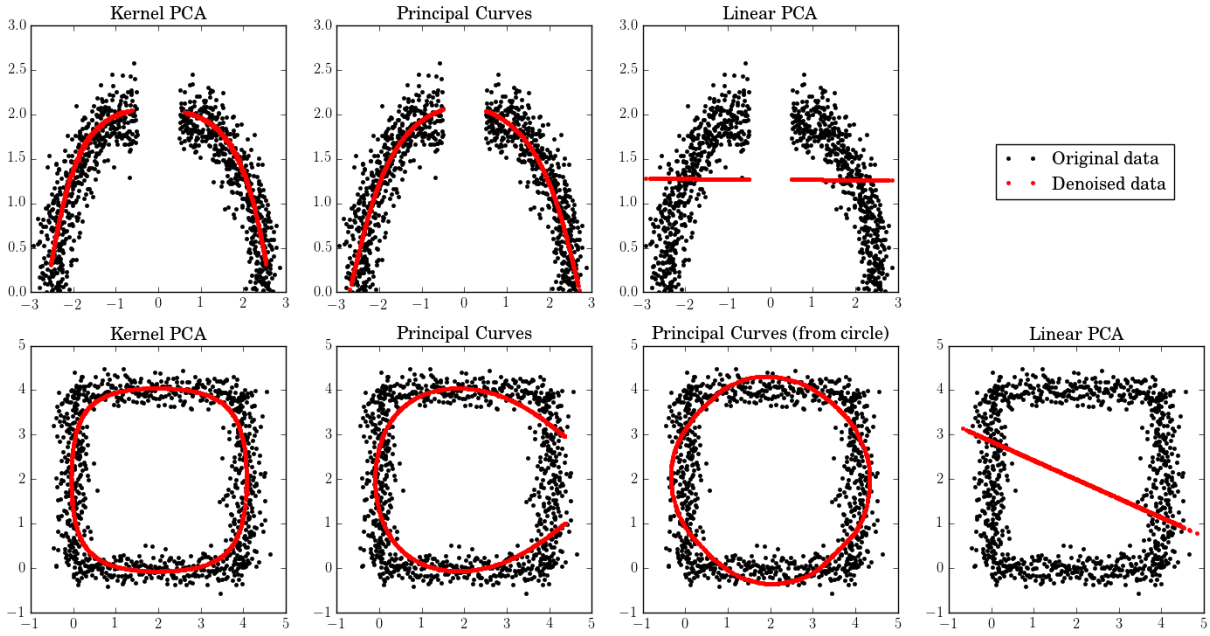


Figure 1: Noisy half-circle and square, and de-noised data.

For the half-circle, Kernel PCA underestimates the length of the real curves, but is vastly superior to linear PCA. The principal-curves algorithm performs subjectively better in this case. However, on occasions principal curves will overestimate the curves' length slightly for data generated with the same technique. In the original paper similar results are obtained for both types of PCA, but the principal curves algorithm does not perform as well. This may be caused by differences in how the data was obtained, which was not specified in [3].

For the square, Kernel PCA yields the best results. The standard principal curves algorithm gives an open curve, and its variation a curve that looks more like a circle than the Kernel PCA one. Again, linear PCA is the worst algorithm. The result for the principal curves algorithm is not as good as in [3], but the overall results are similar.

USPS data set

In the spirit of the original paper, we considered the USPS data set. It contains 16x16 labeled images of handwritten digits scanned by the U.S. Postal Service, for a total of 9298 samples. Each sample is represented by a 256-dimensional real vector and the corresponding integer label. For each class we have picked 300 data points for training and 50 for testing. The test data was rendered noisy by Gaussian noise and Speckle noise. The former was characterized by zero mean and $\sigma = 0.5$, the latter by a probability $p = 0.4$ of changing a bit to white or to black.

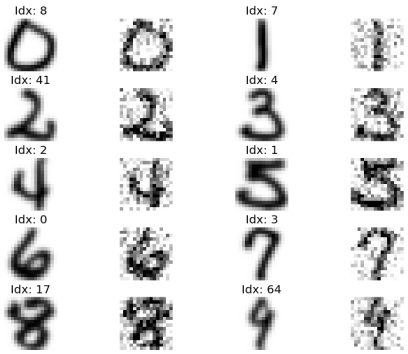
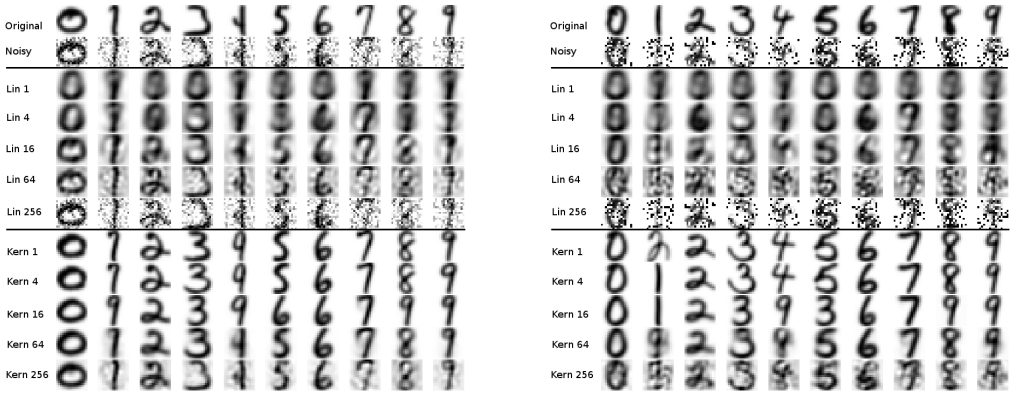


Figure 2: The first appearance of each class, original and noisy versions

Fig. 2 illustrates the first appearance of each digit in the data set, first without noise and then with Gaussian noise.

We then trained a linear PCA and a Kernel PCA algorithm to perform de-noising of all test data using a variable number of features. In Fig. 3 we illustrate how we first generate a noisy (Gaussian in Fig. 3a, Speckle in Fig. 3b) instance out of a number in the test data set and then de-noise it using both linear PCA and Kernel PCA for different number of features.



(a) Gaussian noise added on test sample

(b) Speckle noise added on test sample

Figure 3: Row by row: original images, noisy images, de-noised images using linear PCA on 1, 4, 16, 64, 256 features and de-noised images using Kernel PCA on 1, 4, 16, 64, 256 features

As noted in the original study, we are not surprised to see that a number of features of 256 yielded very different results for the linear and Kernel PCAs. In fact, linear PCA can identify a maximum number of Eigenvectors that is equal to the dimension of the data. De-noising through linear PCA using 256 features means using all of the projected features and thus fully reconstructing the input image. The Kernel PCA works completely different, in particular the algorithm is able to extract up to 3000 Eigenvectors, i.e. the number of training samples. So, in this case 256 is still far from using the whole set of Eigenvectors and the reconstructed image still results de-noised.

As a second experiment on the USPS data set, the authors of [3] tested the reconstruction properties of Kernel PCA. Reconstruction is different from de-noising in the sense that the goal is not to eliminate noise from a noisy data point, but rather to approximate a data point using as few dimensions as possible.

It is noticeable that with a low number of dimensions Kernel PCA is able to produce an image that is closer to the original than the Linear PCA one. Increasing the number of dimensions results in better results for the linear algorithm, while the image produced by Kernel PCA tends to resemble "a more prototypical three"[3].

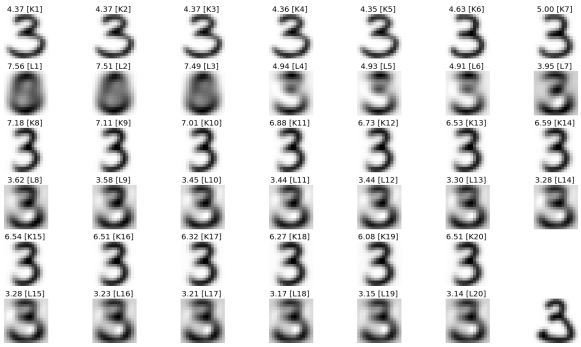


Figure 4: A Linear PCA and a Kernel PCA trained on the same data set as before, reconstruction the image of a '3' using a number of features in the range $[1, 20]$. The value above represents the euclidean distance from the original (bottom right)

4 Discussion

It is our opinion that the comparison between kernel and linear PCA conducted in the first experiment is not as relevant as it could be. In [3], the two algorithms are compared on the same number of components (n). However, this does not allow to effectively compare the algorithms as de-noising methods. In the case of de-noising, the number of components is merely a parameter of the algorithm: in a real use-case, confronted with the choice of using one or the other method, we could try each algorithm, tune its parameters (including the number of components n) and then make the decision based on how each algorithm performed. How the algorithms perform for a given n is not so relevant for comparing them, especially since the components are in spaces of different dimension.

Even more, in the Gaussians example of [3] the authors illustrate a numerical comparison using the ratios between the MSE for each algorithm. This we think might be misleading, in fact in the USPS example, Linear PCA achieves smaller error values, but subjectively, one can argue that the results from Kernel PCA look more like the real number.

Of course, comparing using the same number of dimensions is important if the algorithms are going to be used for dimensionality reduction, but it may be misleading in other cases.

Overall, the results obtained from [3] were reproduced quite faithfully. The divergences found with our results can be addressed to the initial randomization and small differences in the definition of the data sets.

Through these experiments we have tested how Kernel PCA can be applied to identify non-linear manifolds in the data space. For this reason it has a wider area of application compared to linear PCA. However, the increased power comes at the cost that all the training data must be stored to obtain a projection for new data. This has also the effect of increasing the computational cost of the algorithm.

We consider Kernel PCA to be suitable for datasets that contain a small number of points represented in a high number of dimensions. For this case in fact the cost of computing K and the space needed to store the data are more acceptable.

References

[1] K Person. On lines and planes of closest fit to system of points in space. philiosophical magazine, 2, 559-572, 1901.

[2] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[3] Sebastian Mika, Bernhard Schölkopf, Alexander J Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. In *NIPS*, volume 11, pages 536–542, 1998.

[4] Paul F Evangelista, Mark J Embrechts, and Boleslaw K Szymanski. Some properties of the gaussian kernel for one class learning. In *International Conference on Artificial Neural Networks*, pages 269–278. Springer, 2007.

[5] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[6] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.