Review of “Kernel PCA and De-Noising in Feature Spaces”

Group 7
Henrik Karlsson henrik10@kth.se
Alexios Kotsakis alexiosk@kth.se
Markus Videll mvidell@kth.se
Wenyi Zhao wenyizh@kth.se

April 29, 2017

1 Background

In this report our aim is to re-implement some of the methods used in "Kernel PCA and De-Noising in Feature Space" [MSS+99]. The paper presents a method to de-noise datapoints by using PCA with a Gaussian kernel instead of a linear PCA.

Principal component analysis (PCA) is a statistical procedure which aims to convert a set of possibly correlated variables into a set of linearly uncorrelated variables. The PCA cannot detect nonlinear structures in a given dataset, however, the Kernel PCA is well suited to extract nonlinear structures within the data.

The paper tries to use kernel PCA to de-noise data that have been exposed to different kinds of noise. This is done by mapping the data onto a high dimensional space $\mathbf{F}$ using a non-linear function $\Phi : \mathbf{R}^N \to \mathbf{F}$.

To de-noise a data point x it is first mapped onto the high dimension space $\mathbf{F}$ using using the function Φ. It is then projected using the high variance eigenvectors in the image of the training data. We then want to find the pre-image z so $\Phi(z)$ is equal to the projected $\Phi(x)$. Since the eigenvectors with small variance has been removed z should be noise free and only the core structure should be kept.

This method can be easily computed because of the kernel trick. $\Phi(x)$ is never computed because only the inner product $\Phi(x_n)\Phi(x_m)$ is needed.

2 Kernel PCA

The Kernel PCA can be seen as a generalization of the PCA that maps the data into some feature space $\mathbf{F}$ via a (usually nonlinear) function

$$\Phi : R^N \to \mathbf{F} \tag{1}$$

and then performs a PCA on the mapped data.

To perform a PCA in feature space, we need to find eigenvalues $\lambda > 0$ and eigenvectors $\mathbf{V} \in \mathbf{F} \setminus \{\mathbf{0}\}$

$$\lambda \mathbf{V} = C\mathbf{V} \tag{2}$$

where C is the covariance matrix in feature space given by

$$C = \frac{1}{N} \sum_{n=1}^{N} \Phi(\mathbf{x}_n)\Phi(\mathbf{x}_n)^T \tag{3}$$

assuming the projected data has a zero mean in feature space.

As $C\mathbf{V} = \frac{1}{N}\sum_{n=1}^{N}(\Phi(\mathbf{x}_n)\cdot\mathbf{V})\Phi(\mathbf{x}_n)$, all solutions $\mathbf{V}$ must lie in the span of $\Phi(x_1), \ldots, \Phi(x_N)$, hence (3) is equivalent to

$$\lambda(\Phi(\mathbf{x}_k) \cdot \mathbf{V}) = (\Phi(\mathbf{x}_k) \cdot C\mathbf{V}) \tag{4}$$

and there exists coefficients α_n $(n = 1, \ldots, N)$ such that

$$\mathbf{V} = \sum_{n=1}^{N} \alpha_n \Phi(\mathbf{x}_n) \tag{5}$$

Combining (4) and (5), we get

$$\Phi(\mathbf{x}_k) \cdot C\mathbf{V} = \lambda \Phi(\mathbf{x}_k) \cdot \mathbf{V} \implies$$

$$[\Phi(\mathbf{x}_k) \cdot \frac{1}{N} \sum_{n=1}^{N} \Phi(\mathbf{x}_n)][\Phi(\mathbf{x}_n) \cdot \sum_{m=1}^{N} \alpha_m \Phi(\mathbf{x}_m)] = \lambda \Phi(\mathbf{x}_k) \cdot \sum_{n=1}^{N} \alpha_n \Phi(\mathbf{x}_n) \implies$$

$$\frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_m [\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_n)][\Phi(\mathbf{x}_n) \cdot \Phi(\mathbf{x}_m)] = \lambda \sum_{n=1}^{N} \alpha_n [\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}_n)]$$

Now we define the matrix K by $K_{nm} = [\Phi(\mathbf{x}_n) \cdot \Phi(\mathbf{x}_m)]$ to arrive at,

$$K^2 \alpha = N\lambda K\alpha \implies K\alpha = \lambda' \alpha \quad \text{with} \quad \lambda' = N\lambda \tag{6}$$

where α is the column vector with elements $\{\alpha_1, \dots, \alpha_N\}$. We select a subset of the m largest eigenvalues $\alpha_1, \dots, \alpha_m$.

We can compute the nonlinear principal components for the given input $\mathbf{x}$ by projecting on eigenvectors $\mathbf{V}^k (k = 1, ..., m)$ in feature space F. Yielding

$$\beta_k = \mathbf{V}^k \cdot \Phi(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i^k (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})) \tag{7}$$

$$= \sum_{i=1}^{N} \alpha_i^k \mathbf{k}(\mathbf{x}_i, \mathbf{x}) \tag{8}$$

Where $\beta_k (k = 1, ..., m)$ is the k'th nonlinear component of input $\mathbf{x}$. The projection $P_m \Phi(x)$ of $\Phi(x)$ onto the subspace spanned by the m largest eigenvectors is then

$$P_m \Phi(x) = \sum_{i=1}^{m} \beta_i \cdot \mathbf{V}^i \tag{9}$$

As $P_m \Phi(x)$ is in the feature space, we have to find its pre-image z in order to recover the de-noised pattern. However the exact pre-image may not even exist, and so we wish to recover a z that can satisfy $\Phi(z) \simeq P_m \Phi(x)$. [MSS+99] addressed this problem by minimizing the squared distance between $\Phi(z)$ and $P_m \Phi(x)$:

$$\rho(z) = \|\Phi(z) - P_m \Phi(x)\|^2 = \|\Phi(z)\| - 2(\Phi(z) P_m \Phi(x)) + \Omega \tag{10}$$

where Ω includes terms independent of z. To solve this nonlinear optimization problem, we substitute (4) and (8) and obtain the form we need to maximize.

$$\rho(z) = \Phi(z) P_m \Phi(x) + \Omega' = \sum_{i=1}^{N} \gamma_i k(z, \mathbf{x}_i) + \Omega' \tag{11}$$

Where $\gamma_i = \sum_{k=1}^{m} \alpha_i^k \beta_k$

For particular choices of kernels, such as Gaussian kernels of the form $k(x, y) = exp(-\|x-y\|^2/c)$, by setting the gradient with respect to z equals zero, we get:

$$z = \frac{\sum_{i=1}^{N} \gamma_i exp(-\|z - \mathbf{x}_i\|^2/c)\mathbf{x}_i}{\sum_{i=1}^{N} \gamma_i exp(-\|z - \mathbf{x}_i\|^2/c)} \tag{12}$$

Assuming $P_m\Phi(x) \neq 0$ and we can conclude the numerator will always be positive, and since the Gaussian Kernel is smooth, we can solve the problem by a fixed-point iteration method. We can arrive at a iteration form of z as

$$z_{t+1} = \frac{\sum_{i=1}^{N} \gamma_i exp(-\|z_t - \mathbf{x}_i\|^2/c)\mathbf{x}_i}{\sum_{i=1}^{N} \gamma_i exp(-\|z_t - \mathbf{x}_i\|^2/c)} \tag{13}$$

With this method the pre-image we obtained will be in the span of $\mathbf{x}_i$. However, this method is numerically unstable[MSS+99], which can be solved by choosing different starting value of z.

3 Experiments

The authors compares the kernel PCA and the linear PCA by running them on several toy examples and one real world example.

3.1 Toy Example 1

In the first toy example, the aim was to de-noise points in $\mathbf{R}^{10}$. Eleven "centers" were defined in $\mathbf{R}^{10}$, uniformly chosen in $[-1, 1]^{10}$. For each of these, we generated 100 training points and 33 test points from a Gaussian distribution with mean from the generated centers and variance σ^2 in each direction. De-noising was performed on all the test points. This was done for several values of σ^2, and for various number of components n, using both linear and kernel PCA.

To compare how well the two methods performed, we calculated the mean square distance from all the de-noised test points to their corresponding center, and studied the ratio of the them for each σ and number of components n.

σ	$n = 1$	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9
0.05	919.23	538.17	341.90	226.85	132.62	108.28	70.69	59.65	36.25
0.1	240.76	136.57	72.76	51.38	31.60	27.90	20.39	19.83	14.20
0.2	47.32	29.88	20.12	14.62	9.60	8.57	6.75	6.07	4.71
0.4	8.49	5.72	4.26	3.02	2.22	2.05	1.49	1.72	2.02
0.8	1.10	0.87	0.78	0.69	0.64	0.68	0.69	0.73	0.76

Table 1: De-noising results. Each cell is the ratio between the mean squared distances from the centers for linear PCA and kernel PCA.

The ratio between these values are shown in table 1, where ratios larger than 1 indicate how much better kernel PCA performed compared to linear PCA. We can see that kernel

PCA performs better than linear PCA, especially in cases with low values of σ and few components where the difference can be easily noticed.

In the original paper, there is an even bigger disparity between the two methods for $\sigma = 0.05$. The reason why linear PCA is significantly worse for lower values of σ, is that most training and test points will lie very close to their corresponding centers. But then linear PCA has to capture eleven separate directions - one to each center - with fewer than ten features. As σ increases, there might be more overlap between the different clusters, so the linear PCA does not lose as much information by limiting the number of features.

We have not gotten as good results as the original paper. We have several cases where linear PCA performs better than kernel PCA, which the original authors never did.

3.2 Toy Example 2

In the second toy example we are interested in how well the kernel PCA can de-noise two figures, and it is compared to other methods. We choose to compare it to linear PCA, while the authors compared it to a nonlinear autoencoder and Principal Curves as well. This example visualizes in a more clear way the differences between linear and kernel PCA, as it works in 2 dimensions. Figure 1 shows the results.

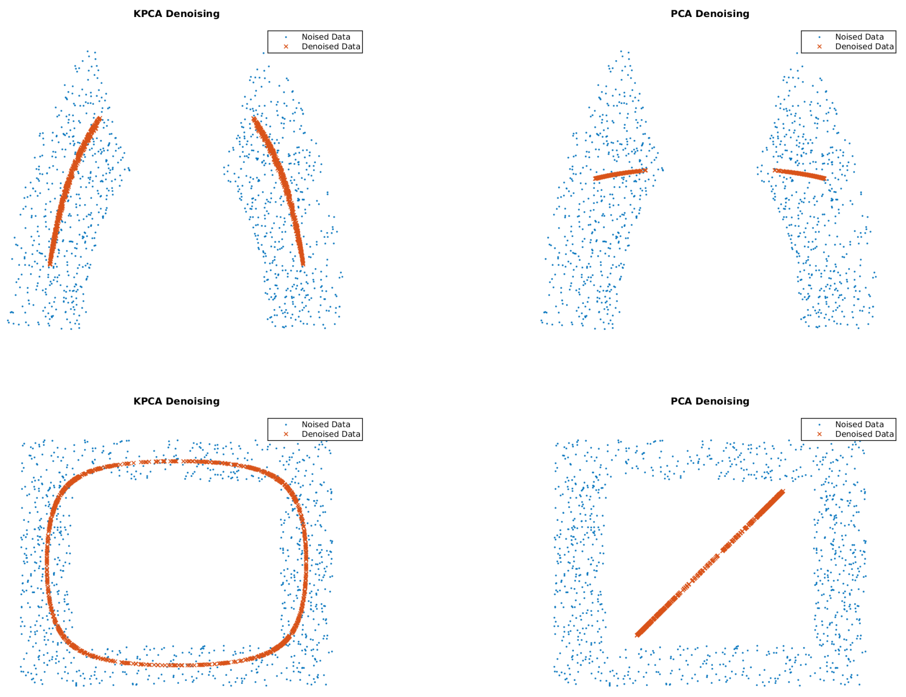

Figure 1: Comparison between KPCA and PCA on de-noising points from a circle and a box. Kernel PCA does a much better job than linear PCA in both test cases.

In the circle case the training has been done on a circle with a discontinuity. The noisy data is generated to yield a uniform distribution similar to the datasets in the original paper. As in the original paper, the model parameters were set as to produce the best result possible.

The kernel PCA uses four features and the linear PCA uses only one. As shown the kernel PCA is able to de-noise the samples with good results, while the linear PCA does not. These results are very similar to the original paper. Our kernel PCA performs a bit worse in de-noising the lower points of the circle, and our linear PCA gives much shorter line segments. The reason for this may be the dataset. The authors do not go into details on how their dataset was generated, but it may be a bit different than ours.

In the box case the training was done on a square shape. Our kernel PCA does learn a circular structure as in the original paper. Likewise the linear PCA is only able to create a straight line.

3.3 USPS Example

The real world example was the USPS database of 16x16-dimensional handwritten digits. We downloaded the database and left the values unaltered, a value of 1 indicates full black and a values of -1 indicates full white. Just as the authors, we randomly chose for each digit 300 samples from training set and 50 samples from the test set. We used the Gaussian kernel $c = 0.93$ which is twice the mean variance of our data set, the orignal paper had a $c = 0.5$ which was twice the mean variance of their dataset. We then generated noisy data sets using the test set (see figure 2), one set with Gaussian noise with zero mean and a standard deviation $\sigma = 1$ and another set with what the authors calls 'speckle' noise (salt-and-pepper noise) with $p = 0.4$.

There are some significant differences in our figure 2 and figure 4 of the original paper. We did not use the same data set that was used in the original paper, the most notable difference is that the original paper have thicker digits. A Gaussian noise of $\sigma = 0.5$ produced much less noise in the data set we used, therefore a Gaussian noise of $\sigma = 1$ was used as it generated data with Gaussian noise. The linear PCA with 256 components did not reproduce every bit of noise as the original paper, however, it retains significantly more noise than the kernel PCA. In other aspects the original table and our table are very similar.

The original paper claims that the Kernel PCA was better than the PCA by a factor of 1.6 for the Gaussian noise and a factor of 1.2 for the speckle noise. Our data (figure 3) confirms that the Kernel PCA is better by a factor of 1.6 for the Gaussian noise but that it is better by a factor of 1.4 for the speckle noise.

Our results showed that the kernel PCA was better at reconstruction by a factor of 1.06 to 1.2 when using 1 to 20 components, this is unlike the original paper which showed that the methods were almost equivalent for reconstruction.

4 Discussion

Linear PCA allows the extraction of n principal components (n equals the amount of features in the dataset). The general idea is that the last components of PCA holds the the greatest bit of noise. Kernel PCA on the other hand, allows the extraction of N principal components (N equals the amount of observations). Therefore, kernel PCA allows you to extract a much larger set of components.

This paper shows that we were able to re-implement most of the method in our source paper [MSS+99]. Firstly, our implementation showed that kernel PCA performs well, just

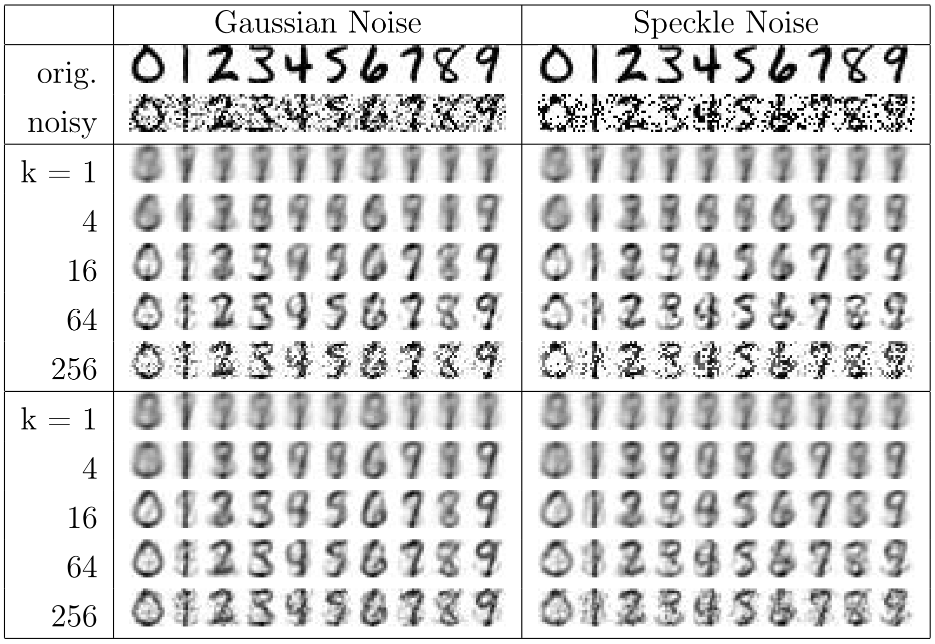

Figure 2: De-Noising of USPS dataset. Top two rows: original data followed by noisy versions of that data (see text). Following five rows: the reconstruction by a linear PCA . The last five rows: the reconstruction by a kernel PCA with a Gaussian kernel $c = 0.93$. k indicates the number of components used.

k =	1	2	4	8	16	32	64	128	256
Gaussian noise	1.04	1.05	1.04	1.02	1.00	0.98	1.01	1.15	1.59
Speckle noise	1.06	1.08	1.08	1.08	1.08	1.07	1.06	1.14	1.43

Figure 3: De-noising of USPS data. Fraction of mean square distance towards the original sample for $k = 2^{0,1,\ldots,8}$ first components.

not as well the original paper suggested. From our perspective, this occur mainly because of two reasons. First of all, it is an iterative scheme. It may thus get stuck in local optima. To overcome this, we may need to make repeated guesses on where to initialize the algorithm for the pre-image. Secondly, it appears that the proposed method is really sensitive to the values of the constant c. The computational experience showed that changing these, significantly affected the results.

Finally, a general comment, the paper was not written clearly, leaving many pieces of information to our intuition for solving. For example, we never got any advice on how to initialize the the iterative scheme, how exactly the data in the second toy example was generated or how the eigenvector vizualizations were generated.

New methods for computing the pre-image has been published after their paper was written. [KT04] writes about a different method of computing the pre-image z. Their method is non-iterative and instead uses distance constraints and shows significant improvements.

References

[KT04] JT-Y Kwok and IW-H Tsang. The pre-image problem in kernel methods. *IEEE transactions on neural networks*, 15(6):1517–1525, 2004.

[MSS$^+$99] Sebastian Mika, Bernhard Schölkopf, Alex Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel PCA and de-noising in feature spaces. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 11*, pages 536–542. MIT Press, 1999.