Weighted Distance Estimation for Kernel PCA

Nikolas Borrel-Jensen (s081782)
Department of Informatics and Mathematical Modelling
Technical University of Denmark
Copenhagen, Denmark
Email: mail@nikolasborrel.com

***Abstract*—We present a new method to extract signal dimensions from noise dimensions using a weighted distance metric in a Gaussian kernel for Kernel PCA, estimated by a non linear optimisation. 4 experiments estimating the weights in the Gaussian kernel using kernel PCA was performed on an artificial 22 dimension dataset consisting of 20 dimensions of Gaussian white noise: 3 experiments with different variance levels for the noise dimensions and 1 experiment concerning the choice of the scaling parameter. For small variances it was possible to detect the signal dimension perfectly, although the results for greater variances indicate that the optimisation could not detect the actual signal dimensions. It was also shown that the estimate of the scaling parameter is crucial for detecting signal dimensions.**

***Index Terms*—Gaussian Kernel, Kernel PCA, fMRI, Weighted Distance Estimation, Scale Parameter.**

I. Introduction

When doing a classification task in high dimensional data, it can be of great interest to locate in which dimensions noise exists and hence where the signal persists. This is of special interest in for example neuroimaging, where the task is to visualise the areas in the brain that is directly involved in a given task, ignoring for example the areas in the brain controlling the heart beat.

Using a weighted distance in a Gaussian kernel for Kernel PCA, we hope that after optimising the kernel w.r.t. to these weights, noisy dimensions can be detected and hence giving important information about the dimensions involved in a given classification task. Using an error function as the misclassification we optimise the weighted Gaussian kernel such that the error function is minimised. Because of the stochastic nature of the noise dimensions we hope that the variance in these dimensions is bigger than the signal dimension and for capturing these assumptions we use the NPAIRS[5] framework comparing the variability and mean for individual dimensions for each split.

The kernel PCA using a weightened Gaussian kernel has been implemented in Matlab, whereas the optimiser and the linear regression classifier used in this projects are Matlab implementions.

The report is organised as follows. In Section II the theory behind kernels and Kernel PCA is derived, in Section III the importance of the scaling parameter in the Gaussian Kernel is explained and an example illustrates the aspects discussed. The number of principal components to use is also discussed and tested on an artificial dataset. In Section IV the scheme for optimising the weights in the Gaussian kernel is presented and a solution is demonstrated. Finally, in Section V the framework for the experiments are explained together with important parameters for the optimiser and the weighted distance estimation for Kernel PCA method is applied on a dataset. A conclusion is made in Section VI.

II. Kernel PCA

In this Section we will formulate the defintion of a kernel and derive the Kernel PCA method. We will then illustrate the Kernel PCA method using a Gaussian Kernel on an artificial dataset refered to as the half moon data.

A. Gaussian Kernel

The main idea of a kernel is to map input data $\mathbf{x}$ into a high or even infinite space called the *feature space*, denoted by $\mathcal{F}$. A kernel is defined as a function $k : \mathbf{x} \times \mathbf{x} \mapsto \mathbb{R}$, such that there exists a mapping $\varphi : \mathbf{x} \mapsto \mathcal{F}$ where the following inner-product holds

$$k(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}') \tag{1}$$

We will focus on the Gaussian kernel

$$\mathbf{K}_{n,n'} = k(\mathbf{x}_n, \mathbf{x}_{n'}) = \exp - \left(\frac{d(\mathbf{x}_n, \mathbf{x}_{n'})}{\sqrt{c}} \right)^2 \tag{2}$$

with the usual Euclidian metric difference

$$d_{n,n'} = \|\mathbf{x}_n - \mathbf{x}_{n'}\| \tag{3}$$

B. Derivation of the Kernel PCA method

The *kernel PCA* is a nonlinear generalisation of the PCA, based on linear combinations of a *kernel function* obeying (1). For deriving the kernel PCA, we therefore consider a nonlinear transformation from $\mathbf{x}$ into $\varphi(\mathbf{x})$

$$\mathbf{x} \mapsto \varphi(\mathbf{x}) \in \mathcal{F} \tag{4}$$

Performing the standard PCA in the feature space implicitly gives a nonlinear transformation in the original data space.

Consider a data set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}^T$, where N is the number of observations in a dimensionality D. Performing the nonlinear transformation on these data results in the feature data $\mathbf{\Phi} = \{\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_N)\}^T$, where $\mathbf{\Phi}$ is a $N \times M$ matrix with a dimensionality M. Since the data in feature space can be of a high or even infinite dimension, we usually can not perform the standard PCA directly on the feature data. Instead, we will derive a method that only depends on the kernel [1][2].

Similar to the normal PCA, we usually need to center the data resisting in the feature space for kernel PCA. For now, let the centered data be denoted by $\tilde{\varphi}(\mathbf{x})$. We seek the normal direction of $\mathbf{v}_1$ which maximise the variance of the centered data in feature space $\tilde{\mathbf{\Phi}}$

$$\max_{\mathbf{v}_1} \sigma^2(\mathbf{v}_1\tilde{\mathbf{\Phi}}) \ s.t. \ \|\mathbf{v}_1\| = 1 \tag{5}$$

where $\|\mathbf{v}_1\| = \mathbf{v}_1^T\mathbf{v}_1$ is the Euclidian norm. The variance of the projection can be written as

$$\begin{aligned}\sigma^2(\mathbf{v}_1\tilde{\mathbf{\Phi}}) &= \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=1}^{M} v_i\tilde{\varphi}_i(\mathbf{x}_n)\right)^2 && (6)\\ &= \frac{1}{N}\sum_{n=1}^{N}\sum_{i,j=1}^{M} v_i\tilde{\varphi}_i(\mathbf{x}_n)v_j\tilde{\varphi}_j(\mathbf{x}_n)) && (7)\\ &= \sum_{i,j=1}^{M} v_iv_j\frac{1}{N}\sum_{n=1}^{N}\tilde{\varphi}_i(\mathbf{x}_n)\tilde{\varphi}_j(\mathbf{x}_n)) && (8)\end{aligned}$$

Recognising $\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N}\tilde{\varphi}(\mathbf{x}_n)\tilde{\varphi}(\mathbf{x}_n)^T$ as the covariance matrix, we can write the variance in matrix notation as

$$\sigma^2(\mathbf{v}_1\tilde{\mathbf{\Phi}}) = \mathbf{v}_1^T\mathbf{C}\mathbf{v}_1 \tag{9}$$

By formulating the constrained maximisation problem in (5) with Langrange multipliers, we get

$$\mathcal{L}(\mathbf{v}_1, \lambda_1) = \frac{1}{N}\mathbf{v}_1^T\mathbf{C}\mathbf{v}_1 + \lambda_1(\mathbf{v}_1^T\mathbf{v}_1 - 1) \tag{10}$$

By maximising this equation with respect to $\mathbf{v}_1$, we will find the principal component for which the variance is biggest

$$\begin{aligned}\frac{\partial\mathcal{L}(\mathbf{v}_1,\lambda_1)}{\partial\mathbf{v}_1} &= 2\mathbf{C}\mathbf{v}_1 - 2\lambda_1\mathbf{v}_1 = 0 && (11)\\ \frac{\partial\mathcal{L}(\mathbf{v}_1,\lambda_1)}{\partial\lambda_1} &= \mathbf{v}_1^T\mathbf{v}_1 - 1 = 0 && (12)\end{aligned}$$

Re-arranging the first equation gives us the eigenvalue problem

$$\mathbf{C}\mathbf{v}_1 = \lambda_1\mathbf{v}_1 \tag{13}$$

where $\mathbf{v}_1 \in \mathbb{R}^M\backslash\mathbf{0}$ and λ_1 is the eigenvector and eigenvalue of $\mathbf{C}$, respectively. By left multiplying with $\mathbf{v}_1^T$ and using $\|\mathbf{v}_1\| = 1$ we get $\lambda_1 = \mathbf{v}_1^T\mathbf{C}\mathbf{v}_1$, i.e. λ_1 is the variance of $\mathbf{C}$. Hence, the principal component where the variance is biggest corresponds to the eigenvector of $\mathbf{C}$.

Though, our goal is to solve this eigenvalue problem without working explicitly in the feature space. By combining (13) and the definition of $\mathbf{C}$, we get

$$\frac{1}{N}\sum_{n=1}^{N}\tilde{\varphi}(\mathbf{x}_n)\{\tilde{\varphi}(\mathbf{x}_n)^T\mathbf{v}_i\} = \lambda_i\mathbf{v}_i \tag{14}$$

Provided that $\lambda_i \neq 0$, it follows that the possibilities for $\mathbf{v}_1$ must lie in the span of the feature points, since $\mathbf{v}_i$ is given by a linear combination of the $\tilde{\varphi}(\mathbf{x}_n)$. Hence $\mathbf{v}_i$ can be written in the form

$$\mathbf{v}_i = \sum_{n=1}^{N}\alpha_{in}\tilde{\varphi}(\mathbf{x}_n) \tag{15}$$

where $\alpha_{i1}, \ldots, \alpha_{iN}$ are the coefficients of the expansion. Substituting (15) back into (13), we obtain

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\tilde{\varphi}(\mathbf{x}_n)\tilde{\varphi}(\mathbf{x}_n)^T\sum_{m=1}^{N}\alpha_{im}\tilde{\varphi}(\mathbf{x}_m) &= \lambda_i\sum_{n=1}^{N}\alpha_{in}\tilde{\varphi}(\mathbf{x}_n) \Rightarrow\\ \frac{1}{N}\sum_{n=1}^{N}\tilde{\varphi}(\mathbf{x}_n)\sum_{m=1}^{N}\alpha_{im}\tilde{\varphi}(\mathbf{x}_n)^T\tilde{\varphi}(\mathbf{x}_m) &= \lambda_i\sum_{n=1}^{N}\alpha_{in}\tilde{\varphi}(\mathbf{x}_n)\end{aligned}$$

The final step is now to express this in terms of the kernel function $k(\mathbf{x}_n, \mathbf{x}_m) = \tilde{\varphi}(\mathbf{x}_n)^T\tilde{\varphi}(\mathbf{x}_m)$. We can do that by left multiplying both sides by $\tilde{\varphi}(\mathbf{x}_l)^T$

$$\frac{1}{N}\sum_{n=1}^{N}k(\mathbf{x}_l,\mathbf{x}_n)\sum_{m=1}^{N}\alpha_{im}k(\mathbf{x}_n,\mathbf{x}_m) = \lambda_i\sum_{n=1}^{N}\alpha_{in}k(\mathbf{x}_l,\mathbf{x}_n) \tag{16}$$

which can be written in matrix notation as

$$\tilde{\mathbf{K}}^2\boldsymbol{\alpha}_i = \lambda_iN\tilde{\mathbf{K}}\boldsymbol{\alpha}_i \tag{17}$$

where $\boldsymbol{\alpha}_i$ is the column vector with the coefficients corresponding to the expansion in (15). We can find the solutions for $\boldsymbol{\alpha}_i$ by solving the eigenvalue problem

$$\tilde{\mathbf{K}}\boldsymbol{\alpha}_i = \lambda_iN\boldsymbol{\alpha}_i \tag{18}$$

Removing a factor $\tilde{\mathbf{K}}$ from both sides in (17) does not affect the principal component projection, since the solution of (17) and (18) differs only by eigenvectors of $\tilde{\mathbf{K}}$ having zero eigenvalue ([4] Appendix A).

Normalisation can be normalised by [1]

$$\boldsymbol{\alpha}_i \leftarrow \frac{\boldsymbol{\alpha}_i}{\sqrt{\lambda_iN}} \tag{19}$$

For the illustration and experiments in this project, normalisation has not been used since we are not concernt about the scaling of the projections.

The resulting principal component projection can now be performed by only considering the kernel and the eigenvectors. Using (15), the projection of a point $\mathbf{x}$ onto the ith eigenvector $\mathbf{v}_i$ is given by

$$\begin{aligned}y_i(\mathbf{x}) &= \tilde{\varphi}(\mathbf{x})^T\mathbf{v}_i = \sum_{n=1}^{N}\alpha_{in}\tilde{\varphi}(\mathbf{x})^T\tilde{\varphi}(\mathbf{x}_n) && (20)\\ &= \sum_{n=1}^{N}\alpha_{in}k(\mathbf{x},\mathbf{x}_n) && (21)\end{aligned}$$

C. *Test errors*

To find the best suited kernel and the optimal parameters for this kernel, we need a test set to evaluate the classification error, so that overfitting can be avoided. A slightly modified version of (21) can be obtained: Let the training set be $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}^T$ and the test set be $\mathbf{X}' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\}^T$, where N and M are the number of observations for the training and test set, respectively. Using (15), the projection of a test point $\mathbf{x}'$ onto the ith eigenvector is given by

$$\begin{aligned}y_i(\mathbf{x}') = \tilde{\varphi}(\mathbf{x}')^T\mathbf{v}_i &= \sum_{n=1}^{N}\alpha_{in}\tilde{\varphi}(\mathbf{x}_n)^T\tilde{\varphi}(\mathbf{x}') && (22)\\ &= \sum_{n=1}^{N}\alpha_{in}k(\mathbf{x}_n,\mathbf{x}') && (23)\end{aligned}$$

Using this formula, it is possible to evaluate the test error compared to the chosen kernel and the parameters used.

D. Centering data in feature space

Similar to the standard PCA, the input (feature) data to the kernel PCA should have zero mean, $\sum_{n=1}^{N}\tilde{\varphi}(\mathbf{x}_n)=0$, which can be computed by

$$\tilde{\varphi}(\mathbf{x}_n)=\varphi(\mathbf{x}_n)-\frac{1}{N}\sum_{l=1}^{N}\varphi(\mathbf{x}_l) \tag{24}$$

The elements of the kernel of the projected data points after centralising are given by

$$\begin{aligned}\tilde{K}_{nm} &= \tilde{\varphi}(\mathbf{x}_n)^T\tilde{\varphi}(\mathbf{x}_m)\\ &= k(\mathbf{x}_n,\mathbf{x}_m)-\frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_l,\mathbf{x}_m)\\ &\quad -\frac{1}{N}\sum_{l=1}^{N}k(\mathbf{x}_n,\mathbf{x}_l)+\frac{1}{N^2}\sum_{j=1}^{N}\sum_{l=1}^{N}k(\mathbf{x}_j,\mathbf{x}_l)\end{aligned}$$

In matrix notation, this can be written as

$$\tilde{\mathbf{K}}=\mathbf{K}-\mathbf{1}_N\mathbf{K}-\mathbf{K}\mathbf{1}_N+\mathbf{1}_N\mathbf{K}\mathbf{1}_N \tag{25}$$

where $\mathbf{1}_N$ denotes the $N\times N$ matrix where every element takes the value $\frac{1}{N}$.

III. KERNEL PCA USING A GAUSSIAN KERNEL

A. The scaling parameter and the impact on the projection

We will use the derived Kernel PCA together with a Gaussian Kernel using the Euclidian distance to illustrate the strength of these methods by showing a problem consisting of two separated classes, class 1 and class 2, as depicted in Figure 1. The task is to classify the data, such that it can be determined whether a new test point is belonging to either class 1 or class 2. Since the two data sets are etangled, a linear classifier can not solve the task and the idea is therefore to transform the data by using kernel PCA making the classification task solvable by a linear classifier. When performing kernel PCA using a Gaussian Kernel (2) with the Euclidian metric, two parameters has to be estimated, namely (1) the scale of the kernel c and (2) the number of principal components. The number of principal components to use for projecting the data concerns the kernel dimensions and since we are working in the kernel space, it is not obvious how many dimensions to use.

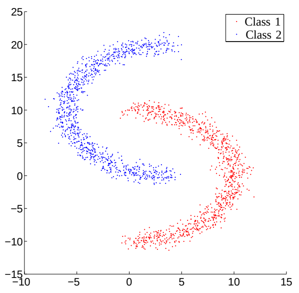


Fig. 1. Artificial data set consisting of 2000 data points with a variance of $\sigma^2=0.7^2$.

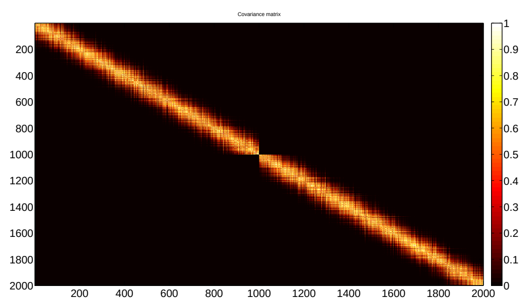
We would like neighbour point to have a high correlation and points far from each other to have a very low correlation, hence giving a kernel making the kernel PCA capable of separating the two classes. Concerning the scale of the kernel, choosing a small $c, k(\mathbf{x}_n,\mathbf{x}_{n'})\to 0$ for $n\neq n'$ which clearly is not a good choice since it corresponds to all points in the feature space being orthogonal to each other due to the way the general kernel in (1) is defined, dispite the class label. Choosing $c\to\infty$, the kernel will only consists of values near 1 and this corresponds intuitively to the linear PCA.

For the dataset depicted in Figure 1, what we need is instead $\sqrt{c}\ll d_1$ and $\sqrt{c}\gg d_2$, where d_1 is the (average) distance between two neighbour points and d_2 is the smallest distance between two points in each of the two classes. This choice will make the correlation between neighbour point big, whereas the correlation between two points far away from each other, as for example points in each of the two classes, will be negligible, corresponding to a covarance in the kernel matrix near 1 and 0, respectively. If the computation time allows us to do so, the best way is to find c is by minimising an error function w.r.t. c. Otherwise a simple method for approximating the scale parameter is to compute the standard deviation of the data and use this as the scaling factor.

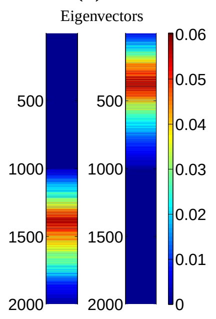
In Figure 2, the Gaussian kernel for the data in Figure 1 is depicted for $c=4$ together with the two principal components corresponding to the eigenvalues with the highest values. The data points are not shuffled and are enumerated from point 1 to 1000 in class 1 and 1001 to 2000 in class 2, making the kernel and eigenvectors directly comparable with the dataset. From the figure we see, that neighbour points has a high value whereas points far from each other has a value near 0. The important area is near point 1000. Here we see that the covariance between the last points in class 1 and the points in class 2 is near 0 and visa versa. This is a very important property for separating the two classes succesfully. In Figure 3, the projection of the data onto the first two principal components in Figure 2 is shown. We see that the two classes have been separated to an extend that a linear classifier can be used with succes. Be aware that we have not used a testset for the reason to make the figures more clear. A logistic regression classifier implemented in the Matlab function `glmfit` has been used, since this method is stable to outliers in contrary to for example a linear discriminant and it proved to be very important because of the distribution of the points in the two classes after projection.

B. The number of principal components and the impact on the projection using training and test set

In Figure 5 the test error is shown for a different number of principal components corresponding to various values of c for the dataset in Figure 4 consisting of 500 data points where 250 data points are used for training and 250 data points are used for testing. For 10 principal components (and hence doing the linear regression classification in 10 dimensions), we get the



(a)



(b)

Fig. 2. (a) Visualisation of the Gaussian kernel for the data in Figure 1 and $c = 4$. (b) The two eigenvectors corresponding to the most significant eigenvalues.

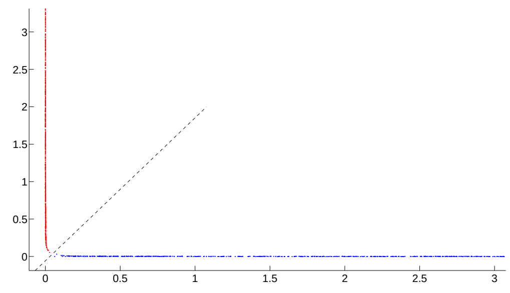


Fig. 3. Projected data using Kernel PCA with a Gaussian kernel.

best result with a misclassification error on the test set at 0.4 %, corresponding to 1 data point being misclassified. We have centered the feature data as explained in Section II-D and in Figure 6 the centered suboptimal solution for the projection on the first 2 principal components is depicted.

IV. Kernel optimisation

Let the important signal be contained in the i'th dimension. Using the Euclidian metric difference and denoting the activation signal $\mathbf{x}_{act}$ and the base signal $\mathbf{x}_{base}$, we get

$$\begin{aligned} d^2_{act,base} &= \|\mathbf{x}_{act} - \mathbf{x}_{base}\|^2 \\ &= \sum_{j=1}^{J} (x_{act,j} - x_{base,j})^2 \\ &= \sum_{j=1\setminus i}^{J} (x_{act,j} - x_{base,j})^2 \\ &\quad + (x_{act,i} - x_{base,i})^2 \end{aligned} \tag{26}$$

Assume having the same significant variance in all the dimensions, the distance measure $J \cdot 2\sigma^2 + \mathbf{s}^2_{dist}$ will overrule the signal distance $\mathbf{s}_{dist}$ due to the term $J \cdot 2\sigma^2$. In data where noisy dimensions exists, it can therefore be of great interest to extract the signal dimensions.

We will investigate how to optimise the Gaussian kernel given by (2) using the weighted distance metric

$$d^2_{n,n'} = \sum_{j=1}^{J} w_j (x_{nj} - x_{n'j})^2 \tag{27}$$

By minimising an error function E w.r.t. the weights $\{w_i\}$ in (27) we hope that the coefficients of the weights will indicate which dimensions contains the signal. We have the following knowledge about the error function:

- The matrix $\mathbf{K}$ consisting of the elements given by the function $k(\mathbf{x}_n, \mathbf{x}_{n'})$ has to be positive semi-definite for being a valid kernel and therefore we need the weights $\{w_j\}$ to be greater than zero [2].

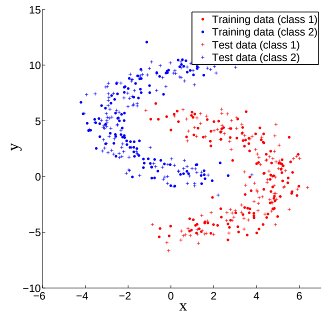


Fig. 4. The data consisting of 500 data points, where 250 points are used for training and 250 points are used for testing.

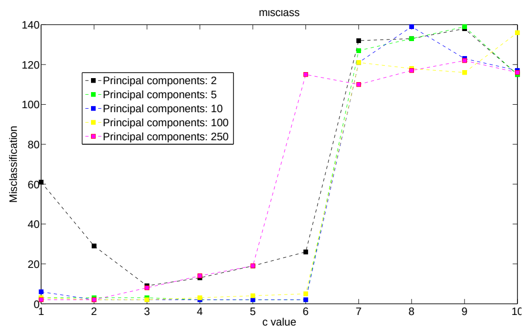


Fig. 5. The number of misclassifications for the projection on 2, 5, 10, 100 and 250 principal components using a linear regression classifier.

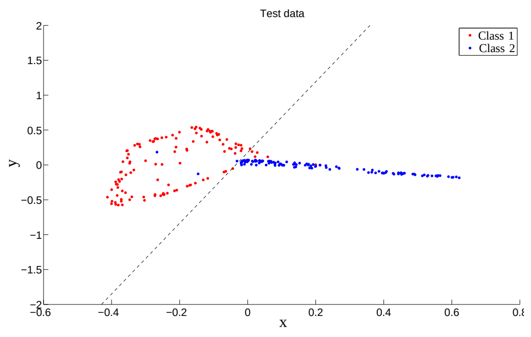


Fig. 6. The projection on the two first principal components with the centered feature data done implicitly by using the kernel.

- Since the kernel PCA uses eigenvalues to perform the projection, the optimisation problem is highly non-linear.

We can formulate the *Nonlinear Programming Problem* (NPL) as [3]

$$\min_{\mathbf{w}} E(\mathbf{w}), \text{ subject to } \mathbf{w} > \mathbf{0} \tag{28}$$

This is a constrained optimisation problem and can be solved by using a varity of optimation schemes. The Matlab function `fmincon` for constrained optimisations can be used with either Interior Point, Sequential Quadratic Programming or Active-Set algorithms. By empirical tests the Interior Point algorithm seemed to both converges faster and found more optimal solutions than the other optimation schemes. It uses an approximated re-formulation consisting only of equalities, making the problem easier to solve. For $\mu > 0$, the approximated problem is

$$\min_{\mathbf{w},\mathbf{s}} E_\mu(\mathbf{w},\mathbf{s}) = \min_{\mathbf{w},\mathbf{s}} E(\mathbf{w}) - \mu \sum_i \ln(s_i)$$

subject to (29)

$$-\mathbf{w} + \mathbf{s} = \mathbf{0}$$

`fmincon` for Interior Point uses either a *direct step* in $(\mathbf{w}, \mathbf{s})$ solving a Karush-Kuhn-Tucker equation or a conjugate gradient step using a trust region. If the resulting Hessian from the direct step is not positive definite, the algorithm uses a conjugate gradient step. The number of slack variables s_i corresponds to the number of inequality constraints.

V. Experiments

In this section we will show the results from experiments on the half moon data with the weightened distance estimation for Kernel PCA. The experiments should clarify the following questions:

- Under which conditions is it possible to extract signal dimensions from the weight coefficients? Does the variance in the noise dimensions has an impact on the ability to localise the signal?
- What should the initial weights be, i.e. which value should the scaling factor c in the kernel have?
- How many principal components should be used to get the best results, i.e. what should the dimensionality of the classification be?

A. Experiment overview

For the experiments we use a dataset of 22 dimensions, where the first two dimensions contains the half moon data and the remaining 20 dimensions contains Gaussian white noise. 500 points are used for training as can be seen in Figure 8 and the NPAIRS [5] framework with 50 splits ($Q = 50$) is used (i.e. 250 points in each subset in the splits), and the estimated weights are applied on a testset consisting of 500 points. The scaling factor c used with this testset is estimated using another training- and testset each consisting of 500 data points, ensuring that the c value is not being overfitted for the data.

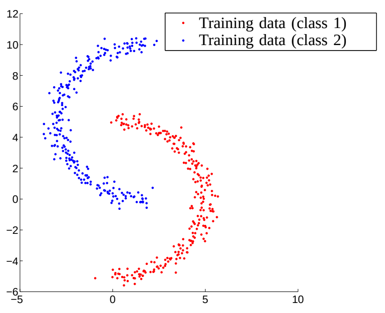


Fig. 8. The dataset consisting of the trainingset for the NPAIRS framework.

B. Experiment setup

The NPAIRS framework is performed by doing $Q = 50$ splits on the desribed trainingset, where each split results in two subsets consisting of 250 data points. For each of the subsets the optimal scaling factor c is computed with the weights for the noise dimensions set to 0 and the others set to 0. This means that the c value is computed only with respect to the half moon data, as defined below:

$$d^2_{n,n'} = \sum_{j=1}^{J} \frac{1}{c} w_j (x_{nj} - x_{n'j})^2$$

where (30)

$$w_j = \begin{cases} 1 & \text{if } j = \{1,2\} \\ 0 & \text{if } j = \{3,\ldots,22\} \end{cases}$$

Choosing the error function as the misclassification using a logistic regression classifier with the optimal c in (30) for the initial weights $w_j = 1$ and $j = \{1, \ldots, 22\}$, we optimise the error function (28) using the Matlab function `fmincon`.

There are many parameters to choose for the `fmincon` function and some are crucial to adjust for the problem to be solveable:

Weights between 0 and 1.

Since we have normalised the weights by c in (30)

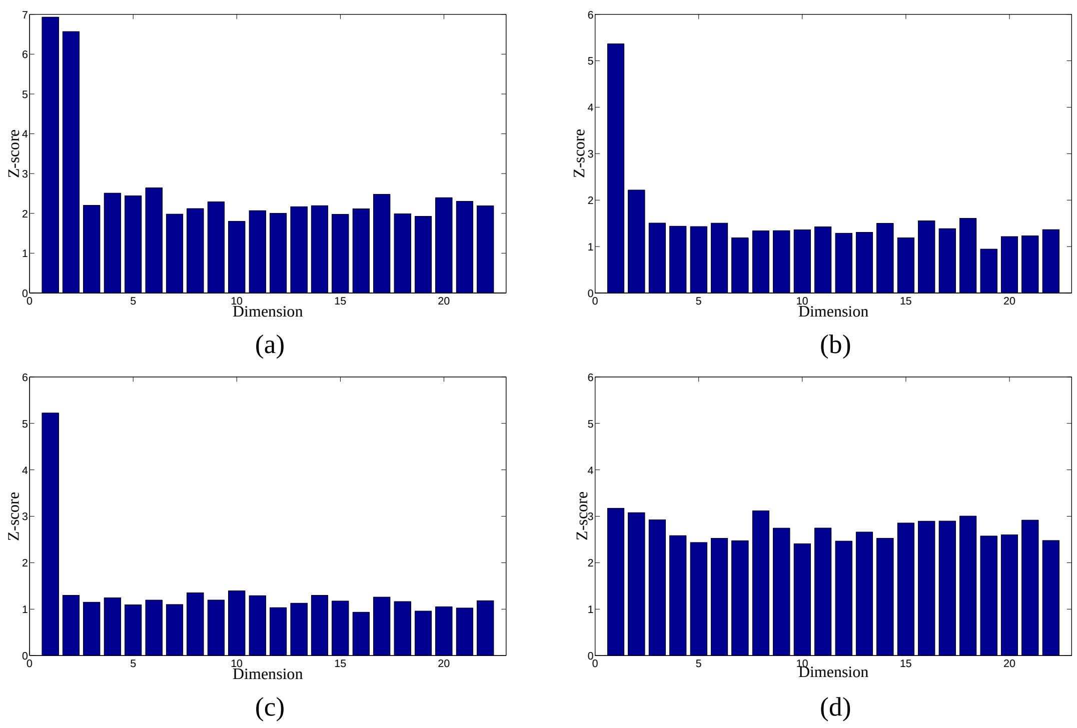


Fig. 7. Z-scores for each of the 22 dimensions after NPAIRS with $Q = 50$ using optimal c for each subset. (a) $\sigma_{data} = 0.3$, $\sigma_{noise} = 0.2$. (b) $\sigma_{data} = 0.3$, $\sigma_{noise} = 0.5$. (c) $\sigma_{data} = 0.3$, $\sigma_{noise} = 0.8$.

to take optimal values between 0 and 1, we restrict the weights to be in that range.

Change in variables for finite-difference gradients. `DiffMinChange` and `DiffMaxChange` controls the perturbation levels for w_j used in the calculation of finite-difference gradients and it turned out that the parameter `DiffMinChange` is very important to adjust correctly. I found that `DiffMinChange` $= 1/3$ and `DiffMaxChange` $= 1$ performed the best.

Finite differences method. `FinDiffType` concerns the finite differences and is used to estimate gradients. The parameter can be either “forward” (default), or “central” which takes twice as many function evaluations but should be more accurate. I found that the extra computational time used with the “central” option was indeed profitable.

For each of the subsets in the q’th split, denote the optimal weights found by optimising the error function by w_{qj} and w'_{qj}, respectively. Let the dimensions corresponding to the weights be given by $j = \{1, \ldots, 22\}$. Comparing the weights for each dimension j using Z-scores is done by

$$\sigma_j = \sqrt{\frac{1}{Q}\sum_{q=1}^{Q}(w_{qj} - w'_{qj})^2} \tag{31}$$

$$\mu_j = \frac{1}{Q}\sum_{q=1}^{Q}\left(\frac{w_{qj} + w'_{qj}}{2}\right) \tag{32}$$

$$z_j = \frac{\mu_j}{\sigma_j} \tag{33}$$

Comparing the z values in (33) for each dimensions, we hope to be able to extract the signal dimensions.

For the first two dimensions we expect the variance to be low and the means to be high (≈ 1), since we have found the optimal c in (30) corresponding to $w_j = 1$ for $j = \{1, 2\}$. In Figure 9 the misclassification w.r.t. c is depicted, and we see that there exists many values for c that gives good misclassifications. Since we want to use the optimal c for the specific dataset, we are not concerned about overfitting. Choosing the highest value of c that gives the best misclassification error, we hope that the optimiser will not diminish the weights w_1 and w_2 since this would correspond to augmenting the value of c. Mathematically, this can easily be seen by the following. Let c_0 be the optimal scaling parameter with the noise dimensions weighted by 0, $w_0 = 1$ be the optimal weight for dimension 1 and 2 and $w \in [0, 1]$ be the weight to be optimised. From the following equation

$$w \cdot \frac{1}{c_0} = w_0 \cdot \frac{1}{c} \tag{34}$$

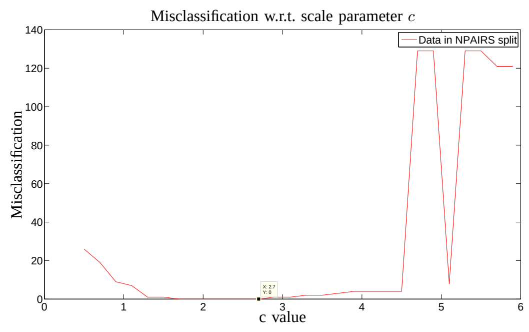


Fig. 9. The misclassification error for a chosen dataset in one of the Q splits in the NPAIRS framework using two principal components. We use the training error, since we want to use the optimal c for the specific dataset, hence not concerning ourself about overfitting.

we get

$$\lim_{w \to 0} c = \infty \tag{35}$$

and hence we will choose the biggest value of c giving the best misclassification, hoping for the weights for the signal dimensions to be more or less fixed at the initial optimal value $w = 1$.

In the remaining 20 noise dimensions, we expect the variances to be high because of the stochastic nature of the Gaussian noise, and the means to be lower that for the signal dimensions (≈ 0.5), since the values of the weights are resticted to be between 0 and 1 and should be distributed equally in that range due to the randomness of the noise.

C. Results

We will start by investigating how the misclassification depends on the weights for the half moon data from Figure 8 without taking the additional noise dimensions into account. A 3D surface plot of the weights w_1 and w_2 and the corresponding misclassification z is shown in Figure 10. In (a) 2 principal components are used and we see that the global minimum is very near $w_1 = w_2 = 1$ selecting c as the biggest value for the best misclassification. In (b) we have used a smaller value of c still giving the best misclassification and doing that, many global minima for $w_1 \in [0.2, 1], w_2 \in [0.2, 1]$ is present. In (c) we have used 10 principal components with $c = 100$, but because of the low noise on the dataset ($\sigma_{data} = 0.3$) it seems like the scale parameter does not have any impact on the optimal misclassification ($= 0$) and hence we suffer from the same problem as in (b).

In Figure 7 the Z-scores (33) for each of the 22 dimensions is shown for standard deviations of $0.2, 0.5$ and 0.8 in (a), (b) and (c), respectively. In (d) we have chosen the smallest value of the scaling parameter c that gives the best misclassification for a standard deviation of 0.2. The first two dimensions contains the half moon data and the classification is done using 2 principal components.

For $\sigma = 0.2$ in (a), `fmincon` clearly detects the signal dimensions but as the noise level raises the detection of dimension 2 becomes imposible. Dimension 1 is clearly separated in all cases (a), (b) and (c) but the reason for that should be found in the way we have constructed our half moon dataset: because the variance is smaller in dimension 1 than in dimension 2, the Euclidian distance is small for more points than in the other dimension giving a higher correlation in the kernel since $k(\mathbf{n}_i, \mathbf{n}'_i) \rightarrow 1$ for $\|\mathbf{n}_i, \mathbf{n}'_i\| \rightarrow 0$, where i is the dimension. Therefore the second dimension has a greater influence on the projection than the first dimension. This can also be seen in Figure 10 (a), (b) and (c), and in Figure 11, where $E(1,0) < E(0,1)$.

In Figure 11 the 2 signal dimensions, w_1 and w_2 together with the corresponding misclassification rate, z, are plotted fixing the 20 weights for the noise dimensions found by `fmincon`. For (a) the noise level is $\sigma_{noise} = 0.2$ and the optimal weights $\mathbf{w}_{opt} = [1, 1, 0, \ldots]$ gives a misclassification of 2, which is the same error as with the optimised weights $\mathbf{w}_{est} = [0.97, 0.79, 0.60, 0.76, \ldots]$. We also see from the plot, that more than one global minimum is giving the misclassification of 2. The error in $(w_1, w_2) = (1, 1)$ has a misclassification of 3 giving a worse error due to the impact from the fixed noise dimensions. In (b) the noise level is $\sigma_{noise} = 0.8$ and we see a problem with a nonsmooth curve giving rise to local minima. This can arise for all variance levels in the noise dimensions, but it seems that it occurs more frequently for high variance levels.

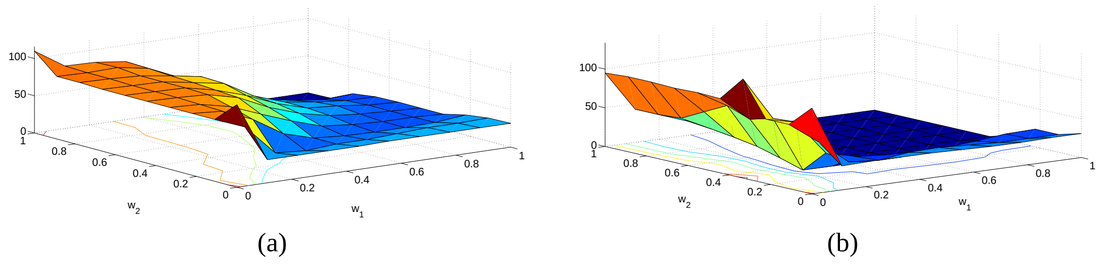
For (d) in Figure 7 none of the first two dimensions can be detected. In Figure 12 the mean and standard deviation are compared for the experiments for the smallest and highest value of c, respectively, for a standard deviation of 0.2.

We first observe that the means in (a) for dimension one has dropped from approximate 0.9 to 0.7 and dimension two from 0.85 to 0.65 when choosing the smallest c. This can be explained by the chosen c, giving a solution for a smaller weight that is as good as $w = 1$ giving the optimal solution. In (b) we see that the standard deviation is a little smaller when choosing the smallest c in the two first dimensions, which contribute to a better Z-score, though the last 20 dimensions has dropped dramatically. This is hard to explain other than the `fmincon` optimiser somehow does not find solutions in the outer range of the weights (near 0 and 1). Because the means have become smaller for the two first dimensions and the variance has become smaller in the noise dimensions, it is not possible to destinguish the signal dimensions from the Z-scores.

In Table I the misclassification is shown for the optimal weight as defined in (31), equal weights where $w_j = 1$ for $j = \{1, \ldots, 22\}$ and the optimised weights where $w_j = 0$ if the Z-scores are less than a given threshold. We see that choosing the thresholds 3,2 and 2 for (a), (b) and (c) in Figure 7, we get perfect classifications for (a) and (b) but 13.2 % misclassification for (c).

VI. Conclusion

We have shown a novel method for separating signal dimensions from noise dimensions. The best results are obtained with low noise in the noise dimensions and for a standard deviation of $\sigma_{noise} = 0.2$, the signal dimensions are clearly separated. When the noise are increasing ($\sigma_{noise} = 0.5 - 0.8$),



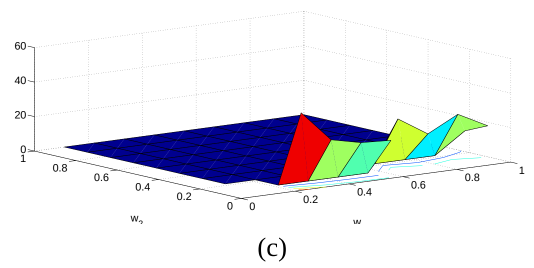


Fig. 10. Surface for 2 weight dimensions for the data and the resulting misclassification without additional noise. (a) Using the biggest c value that gives the best misclassification, only one global minimum very near $w_1 = w_2 = 1$ is present and since the curve is smooth, we can expect the optimser to find near-optimal solutions. (b) Using a small c value still giving the best misclassification, many global minima is present, giving rise to a big variance in the weights. (c) Using 10 principal components, the value c seems not to have great impact on the classification because of the low noise for the data, hence the same problem as in (b) arise.

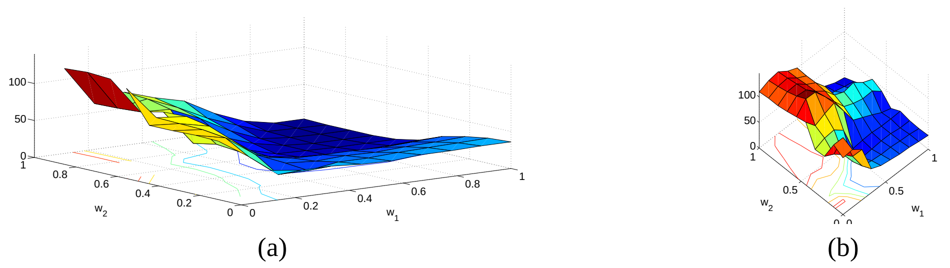
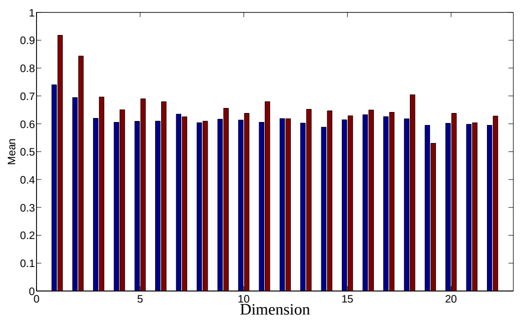


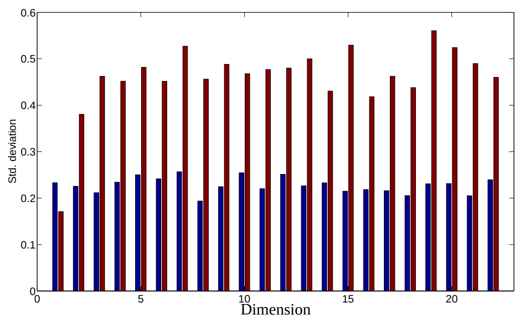
Fig. 11. Surface for the 2 weight dimensions for the data and the resulting misclassification w.r.t. all 22 dimensions. The optimised weights for the 20 noise dimensions found by the termination of `fmincon` are fixed. (a) The optimal weights $\mathbf{w}_{opt} = [1, 1, 0, \ldots]$ gives a misclassification of 2, which is the same error as for the optimised weights $\mathbf{w}_{est}$. We also see from the plot, that more than one global minimum is giving a misclassification of 2. The error in $(w_1, w_2) = (1, 1)$ has a misclassification of 3, giving a worse error due to the impact from the noise dimensions. (b) Because of the more nonsmooth surface, the optimisation is harder.

Data from Fig. 7	Class. error optimal $\mathbf{w}$	Class. error equal $\mathbf{w}$	Class. error optimised $\mathbf{w}$
(a)	0.000	0.000	0.000
(b)	0.000	0.152	0.000 (0.1320*)
(c)	0.000	0.230	0.132

TABLE I
TEST ERRORS FOR THE OPTIMAL WEIGHTS $[1, 1, 0, \ldots, 0]$, EQUAL WEIGHTS $[1,1,\ldots,1]$ AND THE ESTIMATED WEIGHTS. TRESHOLDS FOR THE Z-VALUES ARE SET TO 3, 2 AND 2 FOR (A), (B) AND (C), RESPECTIVELY, I.E. WEIGHTS BELOW THAT VALUE ARE SET TO 0. c VALUE IS ESTIMATED INDIVIDUALLY FOR THE DIFFERENT WEIGHT VECTORS (AND CORRESPONDING NOISE LEVEL) BY USING 500 TRAINING POINTS AND 500 TEST POINTS. (*THRESHOLD OF 3 USED).



(a)



(b)

Fig. 12. Comparison for two experiments, where the first observation in each group is for choosing the smallest value of c and the second observation in each group is for choosing the highest value of c, giving the smallest misclassification in both cases. (a) Comparison of the mean. (b) Comparison of the standard deviation.

the optimiser has problems detecting dimension 2. The reasons for that could be because of the non-smoothness of the error function resulting in `fmincon` to find local minima. We have also shown that the value of the scaling paramter c is crucial when the dimensions is compared using Z-scores, possibly making the estimation of c in a real-world a serious issue. At last, many splits in the NPAIRS framework has to be done making the computation time relatively big. 50 spilts are used, but more splits would probably give even more reliable results to draw conclusions.

To sum up: for a low noise level in the noise dimensions the signal dimensions can be clearly separated, but when the noise is augmented the optimiser fails to detect dimension 2 for the signal.

References

[1] Abrahamsen, T. Kernel methods for de-noising with neuroimaging application. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, 2009.

[2] Bishop, C. M. *Pattern Recognition and Machine Learning.* Springer Science, 2006.

[3] Matlab. *fmincon Interior Point Algorithm*, r2010a ed., February 2010.

[4] Schölkopf, B., Smola, A., and Müller, K.-R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation 10, 1299-1319* (1998).

[5] Strother, S. C. e. a. *The quantitative evaluation of functional neuroimaging experiments: The NPAIRS data analysis framework.* Neuroimage 15:747-771, 2002.