Constrained Pre-Image for Kernel PCA. Aplication to Manifold Learning

Master Thesys
in Electrical Engineering

Pablo Arias Martínez

Advisors:
Gregory Randall, IIE, Facultad de Ingeniería, Universidad de la República
Guillermo Sapiro, ECE, University of Minnesota, USA

Academic Director:
Gregory Randall, IIE, Facultad de Ingeniería, Universidad de la República

Chair:
Marcelo Bertalmío, DTIC, Universitat Pompeu Fabra, Spain
Enrique Ferreira, IIE, Facultad de Ingeniería, Universidad de la República
Gonzalo Perera, IMERL, Facultad de Ingeniería, Universidad de la República

External Reviewer:
Vicent Caselles, DTIC, Universitat Pompeu Fabra, Spain

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República

Montevideo, Uruguay
December 7th, 2007

Contents

Chapter 1

Introduction

Manifold learning deals with the inference of geometrical properties from collections of data represented in vector form and embedded in an Euclidean space. In this context this type of datasets are often called *point clouds*.

Some of the aims of manifold learning are determining if the data is on (or close to) a manifold, estimating the dimensionality of the underlying manifold [46], inferring the underlying manifold [14, 35, 62], finding lower dimensional representations of data, intrinsic to the underlying manifold [14, 20, 23, 51, 63, 67, 73].

Much effort has been recently dedicated to these last aims: lower dimensional representations of data are very important because they allow computations that otherwise would be extremely costly, or even undoable. But these representations can also simplify operations beyond the gain in the computation cost. If the data lies in a curvy manifold for instance, the linear interpolation between two points on the manifold may lie outside it. With a representation of the data that "unfolds" the manifold one can perform linear operations between the low dimensional representatives and map them back to the high dimensional space.

The underlying manifold serves also as a model for the dataset, capturing its main structure. Projecting onto the manifold can be a way of obtaining the essence of a data point, in the same way that the projection over the principal axes found by *Principal Component Analysis* (PCA) can be used to capture the main structure of a *linear* manifold. The low dimensional coordinates of the data may also reflect the main modes of variation of the dataset.

Dimensionality reduction algorithms have been applied to data of many different fields, in particular to image and video processing, both, as a way to reduce computational complexity and to learn models that can be used as *prior* knowledge in applications of segmentation [26, 25] and tracking [65, 24, 28], among others.

Many dimensionality reduction algorithms are *transductive learning* algorithms. They compute a low dimensional representation for a given set (this is referred to as the *training* stage), but not a rule to generalize this representation to new points, as an *inductive learning* algorithm would do. In most learning algorithms the training stage is computationally costly and it is performed only once. It is when the algorithm "learns" from the data.

However the training set is just a finite sample from all possible events. Thus, a rule for extending the mapping to new points is highly desirable. The

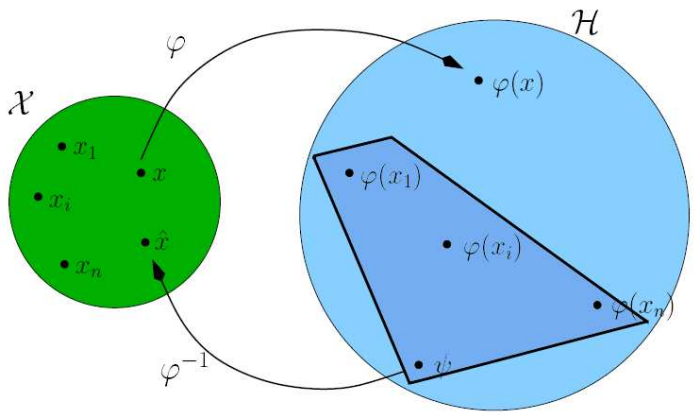

Figure 1.1: Representation of the out-of-sample and pre-image problems in kernel methods. The darker plane in the feature space represents the principal subspace.

problem of extending the low dimensional embedding to points absent in the training set is called the *out-of-sample extension problem*. Its counterpart, *i.e.* synthesizing a new high dimensional point from low dimensional coordinates is the *pre-image problem*.

In these thesis we study the pre-image problem taking advantage of the link between manifold learning and *kernel PCA* [10, 34], a technique born in the bosom of the *kernel methods* field. Kernel PCA is a *non-linear* generalization of PCA [55]. As all the kernel methods, it works by mapping the data into a space fitted with a dot product. PCA of the mapped data is then permformed. It has been noticed that some dimensionality reduction algorithms can be cast into these framework: the low dimensional coordinates are the principal components of the mapped data (*kernel principal components*).

As a kernel method, kernel PCA works without ever computing explicitly the mapping [53]. The only necessary information about the mapping are the dot products between mapped points. This information is encoded in a function k called the *kernel*.

In the kernel methods literature [53, 57], the space where the data lies is called the *input space* and we will denote it as $\mathcal{X}$. The so-called *feature space* is the destination space of the mapping, denoted by $\mathcal{H}$. Let $\varphi : \mathcal{X} \to \mathcal{H}$ be the mapping. The kernel function assigns to a pair of input space points $x, x' \in \mathcal{X}$ a real value: $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$.

This is a very general framework, but also an empty one. Its properties will depend on the choice of kernel, which is equivalent to say on the choice of the mapping. In fact, the kernel methods have successfully been used for *pattern classification* and *regression* by increasing dimensionality instead of reducing it.

An example of a kernel can be the Gaussian kernel:

$$k(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right) \tag{1.1}$$

where σ is a parameter we will refer to as *scale parameter* of the kernel. This is

an *analytic kernel* since it is given by an analytic expression that can be evaluated anywhere in the input space and is independent of the input space. Many kernel associated with dimensionality reduction algorithms are, in contrast, *data driven kernels*: The value of the kernel is known only between training points, and depends on the whole training set.

The benefit of the kernel PCA formalism is that it provides a natural solution to the *out-of-sample extension problem*, which combined with an appropriate kernel function can lead to an inductive dimensionality reduction algorithm.

We will focus on the pre-image problem for kernel PCA in manifold learning applications, a problem there is still no satisfactory solution for. Our aim is to compute the inverse of the kernel associated mapping φ^{-1}. We study this problem for analytic and data dependent kernels. The contributions of this work are the following.

- We experimentally show that existing pre-image algorithms fail to preserve the low dimensional coordinates. More formally the pre-image $\varphi^{-1}(\psi)$ of a feature space point $\psi \in \mathcal{H}$, when mapped back into the feature space $\varphi(\varphi^{-1}(\psi))$ will not have the same kernel principal components as ψ.

  This is something undesirable: the kernel principal components are the low dimensional representation of the data, they encode the coordinates of data inside the manifold. Failing to preserve them implies loosing all the information of ψ which is usefull to us.

- We propose a novel pre-image method that preserves the kernel principal components. With this methodology we design an algorithm for projecting points onto their underlying manifold. This can be usefull when observed points are noisy. We apply it to *manifold de-noising* obtaining good results in low dimensional data and encouraging results in higher dimensions, where we applied it to image data.

- Another contribution is a simple framework for extending data dependent kernels outside the training set, using classical *kernel regression* methods. Our whole kernel PCA plus pre-image framework is modular with the kernel: it can be changed easily, following the philosophy of kernel methods.

- Finally we also present approximate pre-images for the Gaussian and the polynomial kernel. For the Gaussian kernel we compare its performance with other existing pre-images.

This thesis is organized as follows. Chapter 2 starts with a brief overview on kernel methods with special attention paid to the kernel PCA algorithm, since it is the core of the manifold learning algorithms discussed here. This Chapter also presents some of the dimensionality reduction kernels that are going to be used later.

Chapter 3 discuss possible ways of extending data driven kernels, with more detail on our choice: the *Kernel Ridge Regression* algorithm [17]. It also discuss the issue of measuring the performance of the extension.

The core of this work is presented in Chapter 4. It begins by studying the pre-image problem for analytic kernels. Then the standard pre-image methodology is applied to data dependent kernels, where we show its inherent problems. The

chapter ends with the presentation of the proposed pre-image method and its application to manifold de-noising with synthetic and real datasets. Finally, future lines of research and concluding remarks are presented in Chapter 5.

Chapter 2

Kernel Methods

2.1 Introduction

In this Chapter we present an overview of some basic definitions and properties of kernel methods, focusing on kernel PCA. We will also study explicit representations of the kernel associated mapping. This escapes from the typical treatment of kernel methods where the mapping is only of theoretical importance. In particular the pre-image algorithms presented in Chapter 4 make use of representations of the empirical mapping which are presented here.

In the following section we will introduce kernel methods, providing a simple example, as in [53]. Kernel PCA is presented in §2.3, followed by the representations of the mapping in §2.4. We will introduce two types of mapping: the theoretical mapping and an approximation to it: the empirical mapping. The latter is only known at the training set, therefore we will need out-of-sample extension strategies for computing it at new points. These will be discussed in §2.5, together with some results showing the relation between the theoretical and the empirical mapping with its out-of-sample extension. Finally some dimensionality reduction kernels are presented in §2.6.

2.2 Definition and examples

A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that there exist a mapping $\varphi : \mathcal{X} \to \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space and the following inner-product relationship holds

$$k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} \qquad x, x' \in \mathcal{X} \tag{2.1}$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is the dot product in $\mathcal{H}$. The Hilbert space $\mathcal{H}$ is called the *feature* space and $\mathcal{X}$ the *input* space.

The kernel function can be considered as a generalization of the dot product, and therefore as a measure of similarity or correlation between input points. It induces a geometry in $\mathcal{X}$. This is useful mainly in two cases: to induce a geometry in an input space with no geometry, or to modify the existing one. An example of the first situation is the case of string data (chains of characters, words, texts). For this type of data it is possible to define a similarity measure, but not a dot product. By defining a kernel over pairs of strings, we can use the geometry of the feature space representatives of the strings.

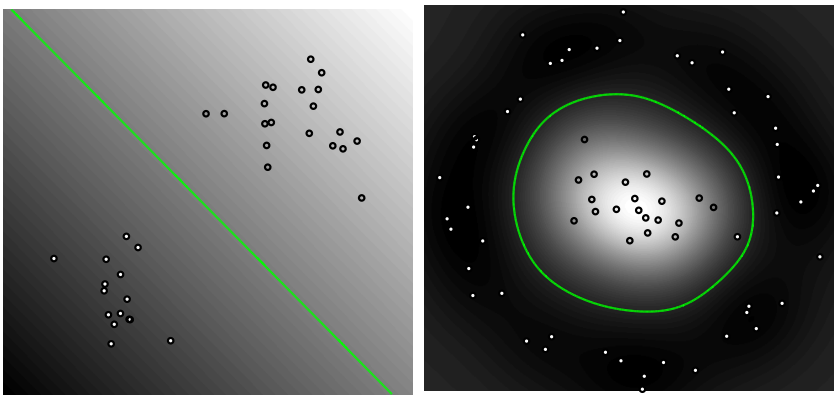

Figure 2.1: A Pattern classification problem. The data on the left can be separated with the line that bisects the segment joining the centers of each class. The dataset on the right does not allow such simple classifier; however, the Gaussian kernel (2.6) maps the set into a linearly separable one, and the same classifier, when *kernelized* yields good results.

In some other cases $\mathcal{X}$ has already its own geometry (for example inputs may be vectors in $\mathbb{R}^d$), but a given problem, complex with this geometry, could be simplified with an appropriate mapping (which is equivalent to say with an appropriate kernel).

This setting is useful in conjunction with algorithms that only take the dot products between the points as input. Suppose for example that we want to train a classifier for the dataset shown on the left of Figure 2.1. Let $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$ be the set of training points (shown as dots in the Figure). This is the data we have at the moment of training. Depending on the application the training set may have additional information. For instance labels in a classification problem or functional values in a regression problem.

In this case each point $x_i \in X$ has a label y_i that only takes two values, say $y_i = 1$ or $y_i = -1$. Therefore we can divide the training set X into two classes $x_{1,i}$ with $i = 1, \ldots, m_1$ and $x_{2,i}$ with $i = 1, \ldots, m_2$ and $m = m_1 + m_2$. The objective is to obtain a function over $\mathcal{X}$ (in the example $\mathcal{X}$ is the plane) that predicts the label of new points depending on their position on the plane. This is a *supervised learning* problem, because the label for each sample of the training set is given.

This example is very simple and a line seems a very reasonable solution, as shown in the Figure 2.1. The line can be determined by its normal direction $\boldsymbol{v}$ and the distance to the origin b. The following function,

$$f(x) = \langle x, \boldsymbol{v} \rangle - b \tag{2.2}$$

separates the input set $\mathcal{X}$ in two semiplanes: if $f(x) > 0$ then x is on one side of the line and if $f(x) < 0$ it is on the other. The line is the set of points for which $f(x) = 0$. Therefore, if we find a line that separates both sets, we could build a classifier using the sign of $f(x)$. In other words, $\operatorname{sign}(f)$ would be a rule induced by the learning algorithm.

Just to show the concept, we are going to pick the line that bisects the segment joining the class centers, c_1 and c_2. The normal vector $\boldsymbol{v}$ can be computed

as:

$$\boldsymbol{v} = c_1 - c_2 = \frac{1}{m_1}\sum_{i=1}^{m_1} x_{1,i} - \frac{1}{m_2}\sum_{i=1}^{m_2} x_{2,i} \tag{2.3}$$

We can determine the offset parameter b, imposing that the mean between c_1 and c_2 belongs to the line:

$$\begin{aligned} b &= \langle 1/2(c_1 + c_2), \boldsymbol{v}\rangle = \frac{1}{2}\left(\langle c_1, c_1\rangle - \langle c_2, c_2\rangle\right) \\ &= \frac{1}{2}\left(\frac{1}{m_1}\sum_{i,j=1}^{m_1}\langle x_{1,i}, x_{1,j}\rangle - \frac{1}{m_2}\sum_{i,j=1}^{m_2}\langle x_{2,i}, x_{2,j}\rangle\right) \end{aligned} \tag{2.4}$$

The label on an unseen point x will be determined by the sign of $f(x)$, given by:

$$f(x) = \langle x, \boldsymbol{v}\rangle - b = \frac{1}{m_1}\sum_{i=1}^{m_1}\langle x, x_{1,i}\rangle - \frac{1}{m_2}\sum_{i=1}^{m_2}\langle x, x_{2,i}\rangle - b \tag{2.5}$$

Note that the solution can be fully expressed in terms of the dot products between the points. If we did not know the precise location of the points, but knew their dot products $\langle x_i, x_j\rangle$ with $i, j = 1\ldots, m$ and $\langle x, x_i\rangle$ with $i = 1, \ldots, m$, we still would be able to build the classifier. The dot products between the training points can be encoded in a matrix $\boldsymbol{G}$, such that $\boldsymbol{G}_{ij} = \langle x_i, x_j\rangle$. This is called the *Gram* matrix and it is symmetric and positive semidefinite.

The problem on the right of Figure 2.1 is harder, in the sense that the solution it is not as simple as a line. If we use the geometry of the input set, we would have to develop a more complex algorithm to find the classifier. Another option is to define on the input space a new geometry, in which our simple classifier works well. In other words, concentrating the complexity on the geometry and not on the learning algorithm.

The success of the kernel methods is due to the fact that they allow to do that very easily. Substituting all the dot products in the Eqs. (2.4) and (2.5) by the kernel is equivalent to constructing the bisecting line (or more generally a hyperplane) in the feature space between the mapped points $\varphi(x_{1,i})$ and $\varphi(x_{2,i})$. This is called the *kernel trick*. Of course one question that rises immediately is that of how to choose a kernel such that its associated mapping allows to separate the classes with the bisecting hyperplane. It has been observed that many kernels do that [53]. Some of the most commonly used kernels are:

$$\textbf{Gaussian kernel: } k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right) \tag{2.6}$$

$$\textbf{Polynomial kernel: } k(x, x') = (\langle x, x'\rangle + c)^p \tag{2.7}$$

$$\textbf{Sigmoid kernel: } k(x, x') = \tanh(\kappa\langle x, x'\rangle + \theta) \tag{2.8}$$

Figure 2.1 shows on the right the classifier found using the Gaussian kernel.

The solution has been *kernelized*. Its new form is:

$$f(x) = \frac{1}{m_1}\sum_{i=1}^{m_1} k(x, x_{1,i}) - \frac{1}{m_2}\sum_{i=1}^{m_2} k(x, x_{2,i}) - \frac{1}{2}\left(\frac{1}{m_1}\sum_{i,j=1}^{m_1} k(x_{1,i}, x_{1,j}) - \frac{1}{m_2}\sum_{i,j=1}^{m_2} k(x_{2,i}, x_{2,j})\right) \quad (2.9)$$

Note that f can be expressed as a *kernel expansion*:

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x, x_i) - b \quad (2.10)$$

Thus we have built our classification function as a sum of kernel functions centered in the training set points: $k(\cdot, x_i)$. These kernel functions are a basis of the set of solutions. Any linear classifier in the feature space can be expressed as Eq. (2.10) and on the other hand, any kernel expansion can be interpreted as being a linear classifier in the feature space associated with the kernel.

In this case the coefficient α_i corresponding to a point x_i will be $1/m_1$ if it is of class 1 or $-1/m_2$ if it is of class 2. With the Gaussian kernel $k(\cdot, x_i)$ is a bump function centered in x_i. Its width will depend on the scale parameter σ. A higher σ yields a wider bump. The solution is made with the contributions of positive bumps centered on the class 1 points and negative bumps on the class 2 points. The constant coefficient b sets the zero level of the classifier.

This is what happens in the input space. Let us now analyze the situation in the feature space. Even if we do not know any explicit form of the mapping, we can draw some conclusions based on the dots products between them. In fact that is all we need to know. If the σ parameter is smaller than the gap between both classes, then $k(x_{1,i}, x_{2,j}) \approx 0$ for $i = 1, \ldots, m_1$ and $j = 1, \ldots, m_2$. This implies that both classes will lie in orthogonal subspaces of $\mathcal{H}$. Thus, they are linearly separable. In fact, any configuration of two classes can be mapped into a linearly separable set if σ is small enough: each mapped point will be orthogonal to the rest, therefore all the possible subsets are separable from their complement.

Many kernel algorithms have been developed by *kernelizing* existing ones. The most famous is probably the *Support Vector Machine* or SVM [13, 53, 56]. It is a linear classifier in the feature space which maximices the sum of the square distances between the hypeplane and the convex hull of each class. Classifiers found by SVM are sparse kernel expansions, *i.e.* $\alpha_i \neq 0$ in Eq. (2.10) only for a small fraction of the number of points.

There are also kernel methods for *regression* (inferring a real function from its samples) such as Kernel Ridge Regression [17], Support Vector Regression [60]; for density estimation [25] among other applications [53].

It was not until recently that kernel methods begun to be applied in the field of manifold learning and dimensionality reduction through the kernel PCA method, as we are going to see in §2.6. In the following section we are going to describe kernel PCA, and show its relation with manifold learning.

2.3 Kernel Principal Component Analysis

Kernel Principal Component Analysis, or kernel PCA, was introduced first by Schölkopf *et al.* in [54, 55] as a technique for doing non-linear component analysis (see [39] for more information on PCA). The main idea is to map the input patterns into the feature space and compute the principal axes there. The kernel PCA projections of the input patterns (referred to as kernel principal components) can then be used as features for classification [55]. Kernel PCA has also been applied to build generative models of a dataset in the input space [47], to image de-noising and as a shape prior in segmentation [26]. It has also been noted that many dimensionality reduction algorithms may be interpreted as kernel PCA [34].

These type of applications motivate the pre-image problem. For instance *kernel PCA de-noising* works by mapping a noisy input point x in the feature space, $\varphi(x) \in \mathcal{H}$, projecting it over the q principal axes in the subspace, $\mathbb{P}_q\varphi(x)$, and mapping the projection back into the input space, obtaining a new (hopefully noiseless) point: $x' = \varphi^{-1}(\mathbb{P}_q\varphi(x))$. This last step is the pre-image of the mapping. Recall Figure 1.1.

Although there is little understanding about the effects of this technique in the input set, it has been empirically shown that for non-linear datasets, where linear PCA would fail to capture the dataset structure, kernel PCA performs better [47, 43, 26]. In [47, 53] it has been argued that one of the reasons for kernel PCA to outperform linear PCA in de-noising applications is that the dimensionality of the feature space can be much greater than that of the input space. This may help at the time of separating the principal components encoding structural information from those encoding the noise.

The results depend heavily on the choice of the kernel, differently from what happens in pattern classification applications. See for example [38]. Among the kernels presented before, the Gaussian kernel is the most used one [26, 40, 43, 47].

In this section we will show how to compute the kernel PCA projections. This can be done in terms of kernel values among the input patters, and thus, does not involve the explicit computation of the mapping. However, as we will see in §2.5 the kernel PCA projections are itself a representation of the mapping.

Principal axes in the feature space

Suppose we have inputs $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$. Using a given kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ we map (implicitly) the data into the feature space $\Phi = \{\varphi(x_1), \ldots, \varphi(x_m)\} \subset \mathcal{H}$.

Usually principal axes are found by the diagonalization of the covariance matrix or its empirical estimate. If we could treat the feature space elements as vectors, we could express the estimate of the covariance matrix as:

$$\boldsymbol{C} = \frac{1}{m}\sum_{i=1}^{m} \varphi(x_i)\varphi(x_i)^T \tag{2.11}$$

The eigenvectors of $\boldsymbol{C}$ would be the principal axes, and their eigenvalues the variance along each axis. However we can not think of $\varphi(x_i)$ as a finite vector (we will see in §2.4.1 that $\mathcal{H}$ may be an infinite dimensional space), thus we need a more abstract formulation of the principal axes.

Proposition 2.1 (Optimality properties of PCA [39])**.** *The q first principal axes are an orthonormal basis of the q dimensional subspace which maximizes the variance of the projections of the data points onto it.*

A consequence of the previous proposition is that the principal axes can be found iteratively. The first one, as the direction that maximizes the variance of projections onto it:

$$p_1 = \arg\max_{p \in \mathcal{H}} \sum_{i=1}^{m} \langle \varphi(x_i), p \rangle_{\mathcal{H}}^2 \quad \text{subject to } \|p\|_{\mathcal{H}} = 1 \tag{2.12}$$

where we are assuming that the $\varphi(x_i)$ are centered. The second principal axis can be found by maximizing the same objective function, constraining it to be orthogonal to p_1:

$$p_2 = \arg\max_{p \in \mathcal{H}} \sum_{i=1}^{m} \langle \varphi(x_i), p \rangle_{\mathcal{H}}^2 \quad \text{subject to } \|p\|_{\mathcal{H}} = 1 \text{ and } \langle p, p_1 \rangle_{\mathcal{H}} = 0 \tag{2.13}$$

The third one can be found adding the constraint of being orthogonal to p_2, and so on.

However, we are still not able to use the kernel trick, since we can not compute $\langle p, \varphi(x_i) \rangle_{\mathcal{H}}$: we know the feature space dot product only between the mappings of the training points $\varphi(x_i)$. The following proposition will help us continue.

Proposition 2.2. *The principal axes with non-zero variance lie on the span of the dataset* $\Phi = \{\varphi(x_1), \ldots, \varphi(x_m)\}$.

Proof. To see why the proposition holds, suppose that the principal axis p_1 has components in the orthogonal complement of the span of Φ: $p_1 = p_1^{\Phi} + p_1^{\Phi^\perp}$. Only p_1^{Φ} contributes to the objective function of Eq. (2.12), since for each $\varphi(x_i)$

$$\langle p_1, \varphi(x_i) \rangle_{\mathcal{H}}^2 = \langle p_1^{\Phi}, \varphi(x_i) \rangle_{\mathcal{H}}^2 < \frac{1}{\|p_1^{\Phi}\|_{\mathcal{H}}^2} \langle p_1^{\Phi}, \varphi(x_i) \rangle_{\mathcal{H}}^2 \tag{2.14}$$

where the last inequality holds because $\|p_1^{\Phi}\|_{\mathcal{H}} < \|p_1\|_{\mathcal{H}} = 1$. Therefore, the unit norm vector $p_1^{\Phi} / \|p_1^{\Phi}\|_{\mathcal{H}}$ yields a greater projection variance than p_1, contradicting our assumption of p_1 being the principal axis. A similar argument can be used for other principal axes with non-zero eigenvalue. □

This means that p_1 can be expressed as $p_1 = \sum_{j=1}^{m} \alpha_i \varphi(x_i)$, and we can use the kernel trick to evaluate the objective function in Eq. (2.12):

$$\langle p_1, \varphi(x_i) \rangle_{\mathcal{H}} = \sum_{j=1}^{m} \alpha_j \langle \varphi(x_j), \varphi(x_i) \rangle_{\mathcal{H}} = \sum_{i=1}^{m} \alpha_j k(x_i, x_j) = [\boldsymbol{K}\boldsymbol{\alpha}]_i \tag{2.15}$$

where $\boldsymbol{K}_{ij} = k(x_i, x_j)$ is the kernel matrix, $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$ is the expansion coefficients vector of p_1 over Φ and $[\boldsymbol{K}\boldsymbol{\alpha}]_i$ denotes the ith component of vector $\boldsymbol{K}\boldsymbol{\alpha}$. We are going to assume that the kernel matrix is symmetric and positive

semidefinite. As we are going to see in §2.4, this is true from the definition of kernel.

Thus, we can express the objective function in (2.12) as $\|\boldsymbol{K}\boldsymbol{\alpha}\|^2$. Note that this is the Euclidean norm of a vector in $\mathbb{R}^m$: we translated the problem from a possible infinite dimensional space $\mathcal{H}$ to a vector problem. We can also express the constraint over the norm of p in vector form, since:

$$\|p\|_{\mathcal{H}}^2 = \left\langle \sum_{i=1}^{m} \alpha_i \varphi(x_i), \sum_{j=1}^{m} \alpha_j \varphi(x_j) \right\rangle_{\mathcal{H}} = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \quad (2.16)$$

Now we can rewrite the problem of Eq. (2.12) as a constrained optimization problem in terms of the expansion coefficients vector $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha}_1 = \underset{\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} = 1}{\arg\max} \|\boldsymbol{K}\boldsymbol{\alpha}\|^2 = \underset{\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} = 1}{\arg\max} \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{K} \boldsymbol{\alpha} \quad (2.17)$$

The *Lagrangian* of this problem is:

$$L(\boldsymbol{\alpha}, \lambda) = \boldsymbol{\alpha}^T \boldsymbol{K}^2 \boldsymbol{\alpha} - \lambda(\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} - 1) \quad (2.18)$$

Thus, making zero the derivative w.r.t. $\boldsymbol{\alpha}$ yields the following generalized eigenvalue problem:

$$\boldsymbol{K}^2 \boldsymbol{\alpha} = \lambda \boldsymbol{K} \boldsymbol{\alpha} \quad (2.19)$$

Note that for $\boldsymbol{\alpha}$ a solution of (2.19) with generalized eigenvalue λ the objective function of (2.17) takes λ as value. Therefore the eigenvector associated to the largest eigenvalue is the $\boldsymbol{\alpha}$ we are looking for.

The following proposition is very usefull because it allows us to solve a simpler problem. For a proof refer to §C.2 in [37].

Proposition 2.3. *For a symmetric matrix $\boldsymbol{K}$ the generalized eigenvalue problem (2.19) is equivalent to the eigenvalue problem:*

$$\boldsymbol{K}\boldsymbol{\alpha} = \lambda \boldsymbol{\alpha} \quad (2.20)$$

for the cases of interest w.r.t. the optimization problem (2.17).

Using Proposition 2.3, we can compute the coefficients vector of the first principal axis as the largest eigenvector of the kernel matrix $\boldsymbol{K}$ (*i.e.* the eigenvector with largest eigenvalue λ_1). Note that if $\boldsymbol{u}_1$ denotes the largest eigenvector with Euclidean unit norm, we will have to re-normalize it: the feature space norm of p_1 is given by $\boldsymbol{u}_1^T \boldsymbol{K} \boldsymbol{u}_1 = \lambda_1$. Thus

$$\boldsymbol{\alpha}_1 = \frac{1}{\sqrt{\lambda_1}} \boldsymbol{u}_1 \quad (2.21)$$

The optimization problem for the second principal axis p_2, (2.13), can also be expressed in vector form. Denote $\boldsymbol{\alpha}_1$ and $\boldsymbol{\alpha}_2$ the coefficients vectors of p_1 and p_2. The additional constraint on p_2 can be expressed as $\langle p_1, p_2 \rangle_{\mathcal{H}} = {\boldsymbol{\alpha}_1}^T \boldsymbol{K} \boldsymbol{\alpha}_2 = \lambda_1 {\boldsymbol{\alpha}_1}^T \boldsymbol{\alpha}_2 = 0$. The orthogonality between the principal axis translates into the orthogonality between their corresponding expansion coefficients vectors. It can be shown that the solution to the problem:

$$\begin{aligned} \boldsymbol{\alpha}_2 = \underset{\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} = 1}{\arg\max}\ & \boldsymbol{\alpha}^T \boldsymbol{K}^2 \boldsymbol{\alpha} \\ \text{subject to } & \boldsymbol{\alpha}_1^T \boldsymbol{\alpha} = 0 \end{aligned} \quad (2.22)$$

is given by the second largest eigenvector of matrix $\boldsymbol{K}$, properly normalized. Similarly, the coefficient vectors of the following principal axes correspond to the rest of the eigenvectors. For the ith principal axis we have that

$$\boldsymbol{\alpha}_i = \frac{1}{\sqrt{\lambda_i}}\boldsymbol{u}_i \tag{2.23}$$

We can express this results in matrix notation. Denote by $\mathbf{A}$ the $m \times m$ matrix with the coefficient vectors $\boldsymbol{\alpha}_i$ as columns and by $\boldsymbol{U}, \boldsymbol{\Lambda}$ the eigenvector and eigenvalue matrices of $\boldsymbol{K}$. Then:

$$\mathbf{A} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2} \tag{2.24}$$

It may be the case that the last $m - r$ eigenvalues are zero (or very small) *i.e.* the kernel matrix $\boldsymbol{K}$ has rank r. Or we might as well be interested in just the first r principal axes. In those cases, we can use the following expression:

$$\mathbf{A}_r = \boldsymbol{U}_{\cdot 1:r}\boldsymbol{\Lambda}_{1:r}^{-1/2} \tag{2.25}$$

where the subscript $(\cdot\ 1{:}r)$ denotes the submatrix built with the first r columns and the subscript $1{:}r$ (without the dot) indicates the upper left $r \times r$ submatrix. Note that in this case $\mathbf{A}_r$ is an $m \times r$ matrix, containing the coefficient vectors for the r principal components. In the following, we are going to use expression (2.24) indistinguishably in both cases unless the contrary is stated, because the overall treatment is basically the same.

Kernel PCA projections

Now that we have computed a representation of the principal axes, we can easily find the projection of the mapping of a test point x over the ith principal axis in the feature space:

$$\langle \varphi(x), p_i \rangle_{\mathcal{H}} = \langle \varphi(x), \sum_{j=1}^{m} \mathbf{A}_{ji}\varphi(x_j) \rangle_{\mathcal{H}} = \sum_{j=1}^{m} \mathbf{A}_{ji} k(x, x_j) = [\mathbf{A}^T \boldsymbol{k}_x]_i \tag{2.26}$$

where $\boldsymbol{k}_x = [k(x, x_1), \ldots, k(x, x_m)]^T$ is the kernel vector between x and the training set. Denoting by $\boldsymbol{y}_x$ the vector with all kernel PCA projections, we have that

$$\boldsymbol{y}_x = \mathbf{A}^T \boldsymbol{k}_x = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T \boldsymbol{k}_x \tag{2.27}$$

If we apply this to a point in the training set x_i, we get

$$\boldsymbol{y}_{x_i} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T \boldsymbol{k}_{x_i} = \boldsymbol{\Lambda}^{1/2}[\boldsymbol{U}_{i\cdot}]^T = [\sqrt{\lambda_1}\boldsymbol{U}_{i1}, \ldots, \sqrt{\lambda_m}\boldsymbol{U}_{im}]^T \tag{2.28}$$

Denote $\boldsymbol{Y}(X)$ the matrix which ith column is $\boldsymbol{y}_{x_i}$, then:

$$\boldsymbol{Y}(X) = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{K} = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T[\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T] = \boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T \tag{2.29}$$

The matrix $\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T$ encodes the projections of the m training points over the m principal axes in the kernel space. If $\boldsymbol{K}$ has rank r, the $\boldsymbol{Y}(X)$ would be $r \times m$ matrix with the projections over the first r principal axis.

Centering in the feature space

In the previous sections we have assumed that the kernel corresponded to a centered mapping. In general, the kernels will not be centered. Suppose for example that we are working with the Gaussian kernel. Since all the kernel values are greater than zero, the greatest angle between two points in the feature space will be smaller than $\pi/2$. Thus mapped points $\varphi(x_1), \ldots, \varphi(x_m)$ will lie inside a cone with angle smaller than $\pi/2$. Besides all of them will have unit norm, so they lie on the intersection with the unit sphere and the inside of the cone. The vertex of that cone is the origin, therefore they can not be centered.

Centering the data in the feature space can be translated into the kernel. Denote by $\overline{\varphi}$ the centered mapping:

$$\overline{\varphi}(x) = \varphi(x) - \frac{1}{m}\sum_{i=1}^{m}\varphi(x_i) \tag{2.30}$$

We can define a "centered" kernel w.r.t. the dataset X, denoted by $\overline{k}$, as

$$\begin{aligned}\overline{k}(x,x') =& \left\langle \varphi(x) - \frac{1}{m}\sum_{i=1}^{m}\varphi(x_i), \varphi(x') - \frac{1}{m}\sum_{i=1}^{m}\varphi(x_i) \right\rangle_{\mathcal{H}} \\ =& k(x,x') - \frac{1}{m}\sum_{i=1}^{m}k(x,x_i) - \frac{1}{m}\sum_{i=1}^{m}k(x',x_i) + \frac{1}{m^2}\sum_{i,j=1}^{m}k(x_i,x_j)\end{aligned} \tag{2.31}$$

Using the vector notation, we have that the centered kernel matrix for the training set is given by:

$$\begin{aligned}\overline{\boldsymbol{K}} &= \boldsymbol{K} - \mathbf{1}_{mm}\boldsymbol{K} - \boldsymbol{K}\mathbf{1}_{mm} + \mathbf{1}_{mm}\boldsymbol{K}\mathbf{1}_{mm} \\ &= (\boldsymbol{I} - \mathbf{1}_{mm})\boldsymbol{K}(\boldsymbol{I} - \mathbf{1}_{mm})\end{aligned} \tag{2.32}$$

where $\mathbf{1}_{qr}$ is a constant $q \times r$ matrix filled with ones. The centered kernel vector for test point x can be computed as:

$$\overline{\boldsymbol{k}}_x = \boldsymbol{k}_x - \mathbf{1}_{mm}\boldsymbol{k}_x - \boldsymbol{K}\mathbf{1}_{m1} + \mathbf{1}_{mm}\boldsymbol{K}\mathbf{1}_{mm} \tag{2.33}$$

Results with the Gaussian kernel

In this section we are going to show some results for kernel PCA projections with the Gaussian kernel (2.6). Figures 2.2 and 2.3 show the kernel PCA projections over the first four kernel principal axes for two distinct datasets, indicated by the cyan dots. The gray value in the images represents the magnitude of the kernel PCA projection over the corresponding principal axis.

The images were constructed as follows. A 50×50 rectangular mesh was defined covering the domain shown in each Figure. A pixel is associated to each point x in the mesh. The gray value for pixel x indicates the kernel PCA projection over the corresponding principal axis $\langle \varphi(x), p_i \rangle_{\mathcal{H}}$, with $i = 1, \ldots, 4$, computed according to Eq. (2.26) (after centering the kernel). The green curves are the level lines of the projection. The scale parameter σ was computed as the

average distance from each point towards its 10th nearest neighbor. Denoting by $x_{i,10}$ the 10th nearest neighbor of x_i:

$$\sigma = \frac{1}{m} \sum_{i=1}^{m} \|x_i - x_{i,10}\| \tag{2.34}$$

The behavior of the kernel PCA projections for the clusters dataset is very interesting. The value of the projections over p_1 and p_2 separate the three clusters. Points in the left cluster have a negative first principal component and a positive second one. The center cluster has zero first principal component and a negative second one. The right cluster has both principal components positive. Higher components differentiate between regions inside the clusters.

Note that for points x far away from the training set the kernel vector $\boldsymbol{k}_x$ vanishes, causing the value of the projection to go to zero.

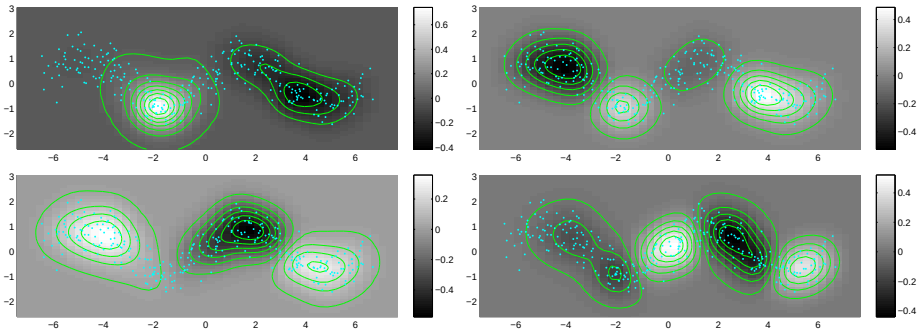

Figure 2.2: Kernel PCA projections over the first four principal axes, using the Gaussian kernel. Sinusoidal dataset. The gray scale of pixel x denotes the value of the projection $\langle p_i, \varphi(x) \rangle_{\mathcal{H}}$ with $i = 1, \ldots, 4$. Top: first and second components, bottom: third and fourth. The green curves denote the level curves of the projection.

2.4 Representations of the kernel mapping

There are many different mappings associated with a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, *i.e.* mappings that satisfy the dot product property (2.1). Next section deals with the ways in which the mapping can be represented. These instances of the mapping are of theoretical relevance because they are a constructive proof that (under appropriate hypothesis) a mapping exist. However they can not be computed explicitly.

In §2.4.2 we present the empirical mapping, and some vector representations of it which allow computational work, and will be used later in the derivation of pre-image algorithms.

Along the following sections we may use different notations to refer to the different representations of the mapping. However although they may different mathematical elements, they are equivalent from the kernel point of view, since they all have the same geometry. Therefore in the kernel literature such different notations are not common. We will use them whenever we are interested in referring the the mathematical object itself.

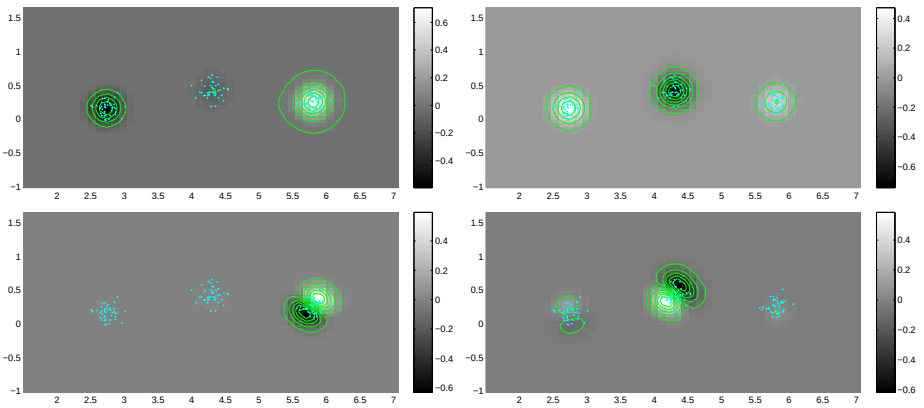

Figure 2.3: Kernel PCA projections over the first four principal axes, using the Gaussian kernel. Clusters dataset. The gray scale of pixel x denotes the value of the projection $\langle p_i, \varphi(x) \rangle_{\mathcal{H}}$ with $i = 1, \ldots, 4$. Top: first and second components, bottom: third and fourth. The green curves denote the level curves of the projection.

2.4.1 Theoretical mappings

Reproducing Kernel Hilbert Space

One possible way of constructing the mapping is to assign every point $x \in \mathcal{X}$ the function $\varphi_R(x) = k(x, \cdot)$. In this case, we are going to define $\mathcal{H}_R$ as a linear subspace of the set of real valued functions over the input space $\mathcal{X}$, denoted by $\mathcal{X}^{\mathbb{R}} = \{f : \mathcal{X} \to \mathbb{R}\}$[1]

$$\mathcal{H}_R = \left\{ f : \mathcal{X} \to \mathbb{R} \mid f = \sum_{i=1}^{l} \alpha_i k(x_i, \cdot)\,, \;\; \alpha_i \in \mathbb{R}, \;\; x_i \in \mathcal{X}, \;\; l \in \mathbb{N} \right\} \tag{2.35}$$

Note that even if $\mathcal{H}_R$ is built with finite linear combinations of elements $k(x, \cdot)$, its dimension can be infinite.

We still have to define a dot product in $\mathcal{H}_R$. We are going to define it among the elements of the generator first:

$$\langle k(x, \cdot), k(x', \cdot) \rangle_{\mathcal{H}_R} = k(x, x') \tag{2.36}$$

This definition can be linearly extended to the rest of the space. Consider $f = \sum_{i=1}^{l} \alpha_i k(x_i, \cdot)$ y $g = \sum_{j=1}^{l'} \alpha'_i k(x'_j, \cdot)$:

$$\langle f, g \rangle_{\mathcal{H}_R} = \sum_{i=1}^{l} \sum_{j=1}^{l'} \alpha_i \alpha'_j k(x_i, x'_j) \tag{2.37}$$

It can be shown that if the kernel satisfies the following definition, the defined function $\langle \cdot, \cdot \rangle_{\mathcal{H}_R}$ is a dot product [53].

[1] $\mathcal{H}_R$ as defined below is a *pre-Hilbert* space. To turn it into a Hilbert space it must be *completed* by adding the limits of all the Cauchy sequences. For now on we will refer with $\mathcal{H}_R$ to the completion of the space defined in (2.35).

Definition 2.1 (Positive definite kernel)**.** A kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel if for all finite sets $\{x_1, \dots, x_m\} \subset \mathcal{X}$ the kernel matrix $\boldsymbol{K}$ w.r.t. $\{x_1, \dots, x_m\}$, given by $\boldsymbol{K}_{ij} = k(x_i, x_j)$ is a positive definite matrix.

This Hilbert space is called *Reproducing Kernel Hilbert Space* for its *reproducing* property: $\langle k(x, \cdot), f \rangle_{\mathcal{H}_R} = f(x)$ for every $f \in \mathcal{H}_R$, and has been studied deeply in the field of functional analysis.

Mercer Map

Another possible kernel mapping is given by the Mercer theorem [53]. We are going to assume now that we have a density function $p(x)$ defined over the input space $\mathcal{X}$.

To construct this mapping we are going to associate a linear operator $T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$ to the kernel. $L_2(\mathcal{X})$ is the set of squared integrable (according to the density $p(x)$) real functions defined over $\mathcal{X}$:

$$\int_{\mathcal{X}} f^2(x) p(x) dx < \infty \tag{2.38}$$

The mapping will be obtained from the eigenfunctions of the operator. We are going to assume that the kernel function is bounded and continuous.

Theorem 2.1. *[Mercer] Suppose $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a symmetric, continuous and bounded function such that the integral operator $T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$ given by*

$$T_k f(x) = \int_{\mathcal{X}} k(x, x') f(x') p(x') dx' \tag{2.39}$$

is positive definite*, that is for all $f \in L_2(\mathcal{X})$*

$$\int_{\mathcal{X}} \int_{\mathcal{X}} k(x, x') f(x') f(x) p(x') p(x) dx' dx \geqslant 0 \tag{2.40}$$

Denote $\phi_j(x) \in L_2(\mathcal{X})$ the normalized orthogonal eigenfunctions of T_k associated with the eigenvalues $\gamma_j > 0$, sorted in non-increasing order:

$$\gamma_j \phi_j(x) = T_k \phi_j(x) = \int_{\mathcal{X}} k(x, x') \phi_j(x') p(x') dx' \tag{2.41}$$

Then

1. $\sum_{j=1}^{\infty} \gamma_j < \infty$
2. $k(x, x') = \sum_{j=1}^{\infty} \gamma_j \phi_j(x) \phi_j(x')$ *for all $(x, x') \in \mathcal{X} \times \mathcal{X}$. The series converges absolutely and uniformly in $\mathcal{X} \times \mathcal{X}$.*

A kernel fullfilling the hypothesis of the Theorem is called *Mercer* kernel. According to the second statement, if we define the mapping $\varphi_M : \mathcal{X} \to \ell_2$ as:

$$\varphi_M(x) = (\sqrt{\gamma_1}\phi_1(x), \sqrt{\gamma_2}\phi_2(x), \dots) \tag{2.42}$$

The mapping φ_M is an infinite sequence. With the usual dot product in ℓ_2 we have that

$$\langle \varphi_M(x), \varphi_M(x') \rangle_{\ell_2} = \sum_{j=1}^{\infty} \gamma_j \phi_j(x) \phi_j(x') = k(x, x') \tag{2.43}$$

for all $(x, x') \in \mathcal{X} \times \mathcal{X}$.

Note that in this case $\mathcal{H}_M = \ell_2$ has infinite dimensions as for the RKHS map.

To compute this mapping more structure is needed than for the RKHS: we need a density p defined in $\mathcal{X}$ that allows us to integrate. The following propositions elucidates the relationship between Mercer kernels and positive definite kernels.

Proposition 2.4. *Under the same conditions of Theorem 2.1 the kernel is also positive definite [53].*

Proposition 2.5 (Mercer map as kernel PCA)**.** *The Mercer map is aligned with the principal axes. In other words, the principal axes are given by the canonical coordinates of ℓ_2, and thus φ_M is already expressed in the principal components coordinate system.*

See for instance [15] for a proof. This means that Mercer kernels are also positive definite kernels and they allow a RKHS representation. The Mercer map can be viewed in this case, as the projections of the RKHS map over the principal axes in the feature space.

2.4.2 Empirical Kernel Map

In the previous section we saw two representations of the kernel map of theoretical relevance. Since these mappings are infinite dimensional they do not allow any computation except those done through the kernel trick. In this section we are going to present a finite dimensional approximation of the kernel map, more suited for computation. Before going on, let us anticipate briefly the path followed in the next sections.

The motivation for a having computable approximation of the kernel map is mainly due to the dimensionality reduction problem, as we are going to see in §2.6. However, these representations will also help us to understand and develop pre-image algorithms in Chapter 4.

Suppose we have a finite sampling of the input space $\mathcal{X}$, given by $X = \{x_1, \ldots, x_m\}$. If we perform only linear operations between the mapped points $\varphi(x_i)$ with $i = 1, \ldots, m$, we can restrict the feature space to the finite dimensional subspace generated by the set of m mappings $\varphi(x_i)$ with $i = 1, \ldots, m$:

$$\mathcal{H}^m = \left\{ \xi = \sum_{i=1}^{m} \alpha_i \varphi(x_i)\,, \;\; \alpha_i \in \mathbb{R}, \;\; x_i \in X \right\} \tag{2.44}$$

We will refer to $\mathcal{H}^m$ as the *empirical feature space*. In the following paragraphs we are going to discuss three ways to represent the elements of $\mathcal{H}^m$, that correspond to coordinates in different bases. Strictly speaking the presentation of the *empirical kernel map* that gives the name to this section, will be completed in §2.5. There we review how to compute the finite dimensional representations of $\mathcal{H}^m$ for a point $x \notin X$ (*i.e.* the out-of-sample extension). This will be done by approximating the theoretical mapping $\varphi(x)$ by its projection onto $\mathcal{H}^m$:

$$\varphi_m(x) \triangleq \mathbb{P}_{\mathcal{H}^m} \varphi(x) \tag{2.45}$$

where φ_m is the empirical kernel map.

Kernel vector and $\boldsymbol{\alpha}$-vector representations

The most straightforward way to represent an element $\xi \in \mathcal{H}^m$ is by expanding it in the generator given by $\Phi = \{\varphi(x_1), \ldots, \varphi(x_m)\}$. Denote by $\boldsymbol{\alpha}_\xi$ the coefficients vector of ξ:

$$\xi = \sum_{i=1}^{m} [\boldsymbol{\alpha}_\xi]_i \varphi(x_i) \tag{2.46}$$

The dot product between two elements ξ and ξ' of $\mathcal{H}^m$ with coefficient vectors $\boldsymbol{\alpha}_\xi$ and $\boldsymbol{\alpha}_{\xi'}$, can be expressed as:

$$\langle \xi, \xi' \rangle_{\mathcal{H}} = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_{\xi i} \alpha_{\xi' j} k(x_i, x'_j) = \boldsymbol{\alpha}_\xi{}^T \boldsymbol{K} \boldsymbol{\alpha}_{\xi'} \tag{2.47}$$

where $\boldsymbol{K}$ is the kernel matrix (*i.e.* $\boldsymbol{K}_{ij} = k(x_i, x_j)$ with $i, j = 1, \ldots m$), $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_m]^T$ and analogously for $\boldsymbol{\alpha}'$.

Eq. (2.47) shows that for computing the dot products among the elements of $\mathcal{H}^m$ we need their expansion coefficients vector $\boldsymbol{\alpha}$. Let $\xi \in \mathcal{H}^m$ for which we want to compute the $\boldsymbol{\alpha}_\xi$ coefficients. The coefficients can be obtained through the following optimization problem:

$$\boldsymbol{\alpha}_\xi = \arg\min F(\boldsymbol{\alpha}) = \arg\min \|\xi - \sum_{i=1}^{m} \alpha_i \varphi(x_i)\|^2_{\mathcal{H}} \tag{2.48}$$

The objective function can be rewritten in the following way:

$$\begin{aligned} F(\boldsymbol{\alpha}) = \|\xi - \sum_{i=1}^{m} \alpha_i \varphi(x_i)\|^2_{\mathcal{H}} &= \left\langle \xi - \sum_{i=1}^{m} \alpha_i \varphi(x_i), \xi - \sum_{j=1}^{m} \alpha_j \varphi(x_j) \right\rangle_{\mathcal{H}} \\ &= \langle \xi, \xi \rangle_{\mathcal{H}} - 2 \sum_{i=1}^{m} \alpha_i \langle \xi, \varphi(x_i) \rangle_{\mathcal{H}} + \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle \varphi(x_j), \varphi(x_i) \rangle_{\mathcal{H}} \end{aligned} \tag{2.49}$$

If we define the vector[2] $\boldsymbol{k}_\xi = [\langle \xi, \varphi(x_1) \rangle_{\mathcal{H}}, \ldots, \langle \xi, \varphi(x_m) \rangle_{\mathcal{H}}]^T$ we can express (2.49) in vector form:

$$F(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} - 2 \boldsymbol{\alpha}^T \boldsymbol{k}_\xi + \|\xi\|^2_{\mathcal{H}} \tag{2.50}$$

Making the derivative w.r.t. $\boldsymbol{\alpha}$ equal to zero yields:

$$\boldsymbol{K} \boldsymbol{\alpha}_\xi = \boldsymbol{k}_\xi \tag{2.51}$$

Under the assumption that $\xi \in \mathcal{H}^m$ (the minimum of F is zero), this vector equation has a solution. However, depending on the rank of the kernel matrix, the system can be underdetermined. A possible way to compute a unique $\boldsymbol{\alpha}_\xi$ is to choose the one with the smallest L_2-norm. This procedure is known as

[2]This is an abuse of notation: the dot product in the feature space $\langle \xi, \xi' \rangle_{\mathcal{H}}$ equals the kernel function value only if ξ and ξ' are in the image of the input space $\varphi(\mathcal{X})$, *i.e.* if ξ and ξ' have exact pre-images by the mapping. We will use also the notation $\boldsymbol{k}_x$ to refer to the kernel vector between a point $x \in \mathcal{X}$ and the training set X. Due to the dot product property, if $\xi = \varphi(x)$, then $\boldsymbol{k}_\xi = \boldsymbol{k}_x$.

Tikhonov regularization. This can be done using the *Penrose-Moore* pseudo-inverse $\boldsymbol{K}^+ = \boldsymbol{U}\boldsymbol{\Lambda}^+\boldsymbol{U}^T$, where $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ are eigenvectors and diagonal eigenvalue matrices, respectively and $\boldsymbol{\Lambda}^+$ denotes the Penrose-Moore pseudoinverse of $\boldsymbol{\Lambda}$, a diagonal matrix computed inverting all the non-zero eigenvalues of $\boldsymbol{K}$[3]. Then

$$\boldsymbol{\alpha}_\xi = \boldsymbol{K}^+\boldsymbol{k}_\xi \tag{2.52}$$

Therefore we can use two vector representations for ξ, $\boldsymbol{\alpha}_\xi$ and $\boldsymbol{k}_\xi$. If we use the kernel vector representation, the dot product between $\xi = \sum_{i=1}^m \alpha_{\xi i}\varphi(x_i)$ and $\xi' = \sum_{i=1}^m \alpha_{\xi' i}\varphi(x_i)$ can be expressed as:

$$\langle \xi, \xi' \rangle_{\mathcal{H}} = \boldsymbol{\alpha}_\xi^T \boldsymbol{K} \boldsymbol{\alpha}_{\xi'} \tag{2.53}$$

$$= \boldsymbol{k}_\xi^T {\boldsymbol{K}^+}^T \boldsymbol{K}\boldsymbol{K}^+ \boldsymbol{k}_{\xi'} = \boldsymbol{k}_\xi^T \boldsymbol{K}^+ \boldsymbol{k}_{\xi'} \tag{2.54}$$

In the last equality we have used that the pseudo-inverse of a symmetric matrix is also symmetric, and that $\boldsymbol{K}^+\boldsymbol{K}\boldsymbol{K}^+ = \boldsymbol{K}^+$.

These representations correspond to different basis of $\mathcal{H}^m$. The basis of the $\boldsymbol{\alpha}$ representation is $\Phi = \{\varphi(x_1), \ldots, \varphi(x_m)\}$. We are going to refer to this representation of the empirical map as the $\boldsymbol{\alpha}$ representation and to $\boldsymbol{k}_\xi$ as the kernel vector representation of $\xi \in \mathcal{H}^m$.

Kernel PCA representation

Another basis of $\mathcal{H}^m$ is given by the principal axes $\mathcal{P} = \{p_1, \ldots, p_m\}$. The coordinates on this basis are given by the orthogonal feature space projections over the basis elements. In §2.3 we computed those projections for the mappings of the training set $\varphi(x_i)$ with $i = 1, \ldots, m$ as well as for the mapping of a new point $\varphi(x)$. Generally, the latter will not belong to $\mathcal{H}^m$, and thus, its kernel PCA representation is infact that of its projection over $\mathcal{H}^m$. We will return to this point in §2.5.

In this section we will compute the kernel PCA projections of $\xi \in \mathcal{H}^m$. Note that ξ does not have to be the image of any x_i, and therefore this case was not covered in §2.3.

Suppose $\xi = \sum_{i=1}^m \alpha_{\xi i}\varphi(x_i)$. Let $\boldsymbol{y}_\xi = [y_{\xi 1}, \ldots, y_{\xi m}]^T$ be the vector with its principal components. The jth principal component is given by

$$[\boldsymbol{y}_\xi]_j = \langle \xi, p_j \rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^m \alpha_{\xi i}\varphi(x_i), \sum_{l=1}^m \mathbf{A}_{lj}\varphi(x_l) \right\rangle_{\mathcal{H}} = \boldsymbol{\alpha}_\xi^T \boldsymbol{K} \mathbf{A}_{\cdot j} \tag{2.55}$$

Therefore

$$\boldsymbol{y}_\xi = \mathbf{A}^T \boldsymbol{K} \boldsymbol{\alpha}_\xi = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T(\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T)\boldsymbol{\alpha}_\xi = \boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T\boldsymbol{\alpha}_\xi \tag{2.56}$$

Equivalently, we can also find the kernel PCA projections vector $\boldsymbol{y}_\xi$ from the kernel vector $\boldsymbol{k}_\xi$:

$$\boldsymbol{y}_\xi = \boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T\boldsymbol{K}^+\boldsymbol{k}_\xi = \boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{k}_\xi \tag{2.57}$$

[3] Equivalently if the rank of $\boldsymbol{K}$ is r:

$$\boldsymbol{K}^+ = \boldsymbol{U}_{\cdot 1:r}\boldsymbol{\Lambda}_{1:r}^{-1}\boldsymbol{U}_{\cdot 1:r}^T$$

Note that this is the same as in (2.28), the expression for the kernel PCA projection of input point x_i. Once again, being the rank of $\boldsymbol{K}$, $r < m$, $\boldsymbol{\Lambda}^{-1/2}$ must be substituted with $[\boldsymbol{\Lambda}^{1/2}]^+$.

The kernel PCA representation of the empirical map will be of particular relevance to us. As we are going to see in §2.6, the application of kernel methods to dimensionality reduction is precisely through the kernel PCA empirical map. The low dimensional representatives are the kernel PCA map. Dimensionality reduction is achieved by considering only the first principal components.

About the notation. In this section we have used the notations $\boldsymbol{\alpha}_\xi$, $\boldsymbol{k}_\xi$ and $\boldsymbol{y}_\xi$ to denote the different representations of ξ. If $\xi = \varphi(x_i)$ with $x_i \in X$ we will use $\boldsymbol{\alpha}_{x_i}$, $\boldsymbol{k}_{x_i}$ and $\boldsymbol{y}_{x_i}$ as a shortcut for $\boldsymbol{\alpha}_{\varphi(x_i)}$, $\boldsymbol{k}_{\varphi(x_i)}$ and $\boldsymbol{y}_{\varphi(x_i)}$.

The topic of the following section is the computation of $\boldsymbol{\alpha}_x$, $\boldsymbol{k}_x$ and $\boldsymbol{y}_x$ for a point x which is not in the training set X.

2.5 Out-of-sample extension

In §2.4.2 we presented three representations for the elements in the empirical kernel feature space $\mathcal{H}^m$. As stated before, these representations can be considered as kernel mappings, if the input space is restricted to $X = \{x_1, \dots, x_m\}$. In this section we are going to study the problem of extending these finite representations outside the training set, when X is just a finite sample of $\mathcal{X}$.

Generally the image of a new point will not be in $\mathcal{H}^m$. Consider for example the Gaussian kernel (2.6). With the RKHS map (recall $\varphi(x) = k(x, \cdot)$) it is easy to see that $\varphi(x) \notin \operatorname{span}\{\varphi(x_1), \dots, \varphi(x_m)\} = \mathcal{H}^m$. Infact adding x to the training set modifies all the representations. In particular, recomputing the kernel PCA projections would require recomputing the eigendecomposition of an expanded kernel matrix to find the new principal axes. Since the principal axes change, so do the kernel PCA projections of the original points. However, if the initial training data is large enough, we can expect $\varphi(x)$ to be close to $\mathcal{H}^m$ and to have a very small influence over the principal axes. We will base on this idea to present out-of-sample extensions of the empirical kernel map representations.

We are going to define the empirical kernel map $\varphi_m : \mathcal{X} \to \mathcal{H}^m$ as

$$\varphi_m(x) \triangleq \mathbb{P}_{\mathcal{H}^m} \varphi(x) \tag{2.58}$$

Projecting over $\mathcal{H}^m$ is equivalent to projecting over the span of the m principal axes in the feature space. Since $\varphi_m(x)$ lies in $\mathcal{H}^m$, we can compute $\boldsymbol{\alpha}_{\varphi_m(x)}$, $\boldsymbol{y}_{\varphi_m(x)}$ and $\boldsymbol{k}_{\varphi_m(x)}$ as we did in the last section.

We are going to start with the kernel PCA projections $\boldsymbol{y}_{\varphi_m(x)}$, since we already know them through Eq. (2.24):

$$\boldsymbol{y}_{\varphi_m(x)} = \boldsymbol{y}_x = [\Lambda^{1/2}]^+ \boldsymbol{U}^T \boldsymbol{k}_x \tag{2.59}$$

Recall that $\boldsymbol{k}_x = [k(x, x_1), \dots, k(x, x_m)]^T$. We will consider the case when $\boldsymbol{K}$ does not have full rank by using the pseudo-inverse of $\boldsymbol{\Lambda}$.

To compute the remaining representations of $\varphi_m(x)$ we are going to write it as an expansion in the Φ basis:

$$\begin{aligned}\varphi_m(x) &= \sum_{i=1}^{m}[\boldsymbol{y}_{\varphi_m(x)}]_i p_i = \sum_{i=1}^{m}[\boldsymbol{y}_{\varphi_m(x)}]_i \sum_{j=1}^{m}\mathbf{A}_{ji}\varphi(x_j) \\ &= \sum_{j=1}^{m}\left(\sum_{i=1}^{m}[\boldsymbol{y}_{\varphi_m(x)}]_i \mathbf{A}_{ji}\right)\varphi(x_j) = \sum_{j=1}^{m}\left[\mathbf{A}\boldsymbol{y}_{\varphi_m(x)}\right]_j \varphi(x_j) \end{aligned} \tag{2.60}$$

Therefore, we have the following relation between $\boldsymbol{\alpha}$-vector representation and the kernel PCA projection:

$$\boldsymbol{\alpha}_{\varphi_m(x)} = \mathbf{A}\boldsymbol{y}_{\varphi_m(x)} = \boldsymbol{U}[\boldsymbol{\Lambda}^{1/2}]^{+}\boldsymbol{y}_{\varphi_m(x)} \tag{2.61}$$

Substituting (2.59) we obtain $\boldsymbol{\alpha}_{\varphi_m(x)}$ in terms of $\boldsymbol{k}_x$:

$$\boldsymbol{\alpha}_{\varphi_m(x)} = \boldsymbol{K}^{+}\boldsymbol{k}_x \tag{2.62}$$

We can compute the kernel vector representation $\boldsymbol{k}_{\varphi_m(x)}$ substituting last equation in (2.52),

$$\boldsymbol{k}_{\varphi_m(x)} = \boldsymbol{K}\boldsymbol{K}^{+}\boldsymbol{k}_x \tag{2.63}$$

This expression may suggest that if $\boldsymbol{K}$ is not invertible, $\boldsymbol{k}_x$ may differ from $\boldsymbol{k}_{\varphi_m(x)}$. However this is not the case. One way of seeing it is by looking at the definitions of these kernel vectors. Their ith component of is given by $[\boldsymbol{k}_x]_i = k(x, x_i) = \langle\varphi(x), \varphi(x_i)\rangle_{\mathcal{H}}$ and $[\boldsymbol{k}_{\varphi_m(x)}]_i = \langle\varphi_m(x), \varphi(x_i)\rangle_{\mathcal{H}}$.

Since $\varphi_m(x)$ is the orthogonal projection of $\varphi(x)$ into $\mathcal{H}^m$, we have that the difference vector $\varphi(x) - \varphi_m(x)$ must be orthogonal to every $\varphi(x_i)$, with $i = 1, \ldots, m$. Thus:

$$\langle\varphi(x) - \varphi_m(x), \varphi(x_i)\rangle_{\mathcal{H}} = \langle\varphi(x), \varphi(x_i)\rangle_{\mathcal{H}} - \langle\varphi_m(x), \varphi(x_i)\rangle_{\mathcal{H}} = 0 \tag{2.64}$$

and therefore $\boldsymbol{k}_x = \boldsymbol{k}_{\varphi_m(x)}$. In other words $\boldsymbol{k}_x$ is the kernel representation of the empirical map for x.

Equivalently note that Eq. (2.63) can be rewritten as

$$\boldsymbol{k}_{\varphi_m(x)} = \left(\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T\right)\left(\boldsymbol{U}_{\cdot 1:r}\boldsymbol{\Lambda}_{1:r}^{-1}\boldsymbol{U}_{\cdot 1:r}^{T}\right)\boldsymbol{k}_x = \boldsymbol{U}_{\cdot 1:r}\boldsymbol{U}_{\cdot 1:r}^{T}\boldsymbol{k}_x \tag{2.65}$$

where r is the rank of $\boldsymbol{K}$. This equation is the Euclidean projection of vector $\boldsymbol{k}_x$ over the range of matrix $\boldsymbol{K}$. It can be shown (see Appendix A) that $\boldsymbol{k}_x$ is always on the range of the kernel matrix $\boldsymbol{K}$.

The fact the $\boldsymbol{k}_x$ is itself an empirical kernel map is not surprising. It can be seen as a finite version of the RKHS map. Besides it encodes all the information needed to project $\varphi(x)$ over $\mathcal{H}^m$.

This also explains the abuse of notation used by denoting $\boldsymbol{k}_x$, $\boldsymbol{y}_x$ and $\boldsymbol{\alpha}_x$ as shortcuts for $\boldsymbol{k}_{\varphi_m(x)}$, $\boldsymbol{y}_{\varphi_m(x)}$ and $\boldsymbol{\alpha}_{\varphi_m(x)}$.

2.5.1 Relation between φ_m and φ

In previous sections we presented the empirical mapping, and a way to extend this mapping to unseen points by projecting the true, unknown mapping $\varphi(x)$ onto the subspace spanned by the previous mappings $\mathcal{H}^m$. We have argued that

if the number of samples is large enough, $\varphi(x)$ should be close to its projection $\mathbb{P}_{\mathcal{H}_m}(\varphi(x)) = \varphi_m(x)$.

The purpose of this section is to give a more formal justification for the use of the empirical map φ_m as out-of-sample extension by reviewing some asymptotic results that show that φ_m converges to φ in the limit when $m \to \infty$. This section is based on Chapters 3 and 4 of M. L. Braun's PhD Thesis [15]. For a more detailed and comprehensive treatment refer there.

The kernel (or the kernel matrix for a finite sample) encodes the information about the mapping. All the approaches thus focus on the convergence of its spectral properties (eigenvalues and eigenvectors) to those of the kernel function associated operator (2.39). Observe that this implies the convergence of the kernel PCA representation of the empirical mapping to the Mercer map.

Asymptotic results

The asymptotic properties of the kernel PCA map are well known in the field of integral equations. Interestingly, there is a way of approximating the eigenvectors and eigenvalues of the kernel operator called the *Nyström method* [4], which coincides with the kernel PCA projection. The convergence properties of the Nyström method thus apply to the kernel PCA projection. In the following paragraphs we are going to present the Nyström method and show its relation with kernel PCA.

The *Nyström* approximation of the operator [4] was presented by Nyström in 1930 [49] is based on a Monte Carlo approximation of the integral

$$\gamma_j \phi_j(x) = \int_{\mathcal{X}} k(x, x')\phi_j(x')p(x')dx' \approx \frac{1}{m}\sum_{i=1}^{m} k(x, x_i)\phi_j(x_i) \tag{2.66}$$

Evaluating ϕ_j in $x_1, \ldots, x_m$ yields a matrix eigenvalue problem:

$$\hat{\gamma}_j \hat{\phi}_j(x_l) = \frac{1}{m}\sum_{i=1}^{m} k(x_l, x_i)\hat{\phi}_j(x_i) \qquad l = 1, \ldots, m \tag{2.67}$$

Denoting $\hat{\boldsymbol{\phi}}_j(X) = [\hat{\phi}_j(x_1), \ldots, \hat{\phi}_j(x_m)]^T$ we can express (2.67) in vector form:

$$\hat{\gamma}_j \hat{\boldsymbol{\phi}}_j(X) = \frac{1}{m}\boldsymbol{K}\hat{\boldsymbol{\phi}}_j(X) \tag{2.68}$$

Solving this matrix eigenvalue problem we can approximate the eigenvalues of the kernel and the value of its eigenfunctions at the sample points. Note that the $\hat{\boldsymbol{\phi}}_j(X) = \boldsymbol{U}_{\cdot j}$ is the jth eigenvector of $\boldsymbol{K}$, whereas $\hat{\gamma}_j = \lambda_j/m$. We can also extend the approximation outside the training set using

$$\hat{\phi}_j(x) = \frac{1}{m\hat{\gamma}_j}\sum_{i=1}^{m} k(x, x_i)\hat{\phi}_j(x_i) = \frac{1}{\lambda_j}\sum_{i=1}^{m} k(x, x_i)\boldsymbol{U}_{ij} \tag{2.69}$$

This is called the Nyström extension. Besides its applications in the numerical solution of integral equations, it has been applied to speed up the eigendecomposition of large kernel matrices [27, 72, 30], by solving a smaller problem taken from a submatrix of the original kernel matrix and extending the eigenvectors. The idea is that the eigenvalues and eigenvectors of a large kernel

matrix and its small submatrix are both approximations of the eigenvalues and eigenfunctions of the kernel operator, and therefore, they must be similar.

Note from Eq. (2.28) that the kernel projection on the jth principal axis of training set point x_i is given by

$$[\boldsymbol{y}_{x_i}]_j = \sqrt{\lambda_j}\boldsymbol{U}_{ij} = \sqrt{\lambda_j}\,\hat{\phi}_j(x_i) \tag{2.70}$$

We could use the Nyström extension to approximate the kernel PCA projection that would have been obtained if x had been included in the training set,

$$[\boldsymbol{y}_x]_j = \sqrt{\lambda_j}\hat{\phi}_j(x) = \frac{1}{\sqrt{\lambda_j}}\sum_{l=1}^{m} k(x, x_l)\hat{\phi}_j(x_l) = \frac{1}{\sqrt{\lambda_j}}[\boldsymbol{U}_{\cdot j}]^T\boldsymbol{k}_x \tag{2.71}$$

which is exactly the jth component of Eq. (2.27). This is the reason why sometimes the term Nyström extension is used for the kernel PCA extension (or projection). This has been noted by [72].

This link is very interesting for both fields. For the numeric field, it provides a geometric interpretation of the Nyström extension: the eigendecomposition of the kernel matrix can be seen as the principal axes of the data feature subspace $\mathcal{H}^m$. The Nyström extension neglects the change in the principal components by the inclusion of a new point (as our out-of-sample extension for the empirical kernel map).

For the kernel methods field, it allows to borrow the known results from the convergence of the Nyström extension and apply them to the kernel PCA projections. But it also sheds light about the relation of kernel PCA and the Mercer map. As we saw in §2.4.2 the empirical mappings can be represented by the kernel PCA projections $\boldsymbol{y}_x$. According to Eq. (2.71), we can see it as a finite sample estimate of the Mercer map. This is also coherent with the interpretation of the Mercer map shown in §2.4.1 where we saw that the Mercer map is aligned with the principal axes.

The results we are going to present basically show the convergence of the eigendecomposition of the kernel matrix to that of the kernel operator, and therefore the convergence of the kernel PCA mapping to that of the Mercer map.

Eigenvalues. The convergence of the eigenvalues is largely known in the field of numerical analysis of integral equations [4]. However, the hypothesis on those results are not suited for the context of machine learning. These results generally assume a compact domain with uniform density. In the machine learning setting the density is not uniform, and it is especially what we want to characterize. The following result by Koltchinskii and Giné [42] considers more appropriate hypothesis.

We will assume a sequence of random sets $(X_1, X_2, \ldots, X_m, \ldots)$ where each $X_m = \{x_1, \ldots, x_m\}$ is drawn i.i.d. from a data generating probability density function $p(x)$. For each set we compute a kernel matrix $\boldsymbol{K}_{X_m}$ with the values of the kernel function on the set. The following theorem is about the allmost sure convergence of the eigenvalues of this sequence of kernel matrices.

Theorem 2.2. *If k is a Mercer kernel, then*

$$\|\lambda(\boldsymbol{K}_{X_m}) - \gamma(T_k)\|_{\ell_2} \rightarrow_{\text{a.s.}} 0 \tag{2.72}$$

where $\boldsymbol{K}_{X_m}$ is the kernel matrix for a sample of size m, $\lambda(\boldsymbol{K}_{X_m}) \in \ell_2$ is an infinite sequence with the eigenvalues of $\boldsymbol{K}_{X_m}$ in decreasing order as its m first components and zeros in the rest, and $\gamma(T_k)$ is the sequence of eigenvalues of the operator T_k defined in Eq. (2.39).

In the above theorem, the eigenvalues are compared as infinite sequences using the ℓ_2 metric:

$$\|(a_n) - (b_n)\|_{\ell_2}^2 = \sum_{i=1}^{\infty}(a_i - b_i)^2 \tag{2.73}$$

Eigenvectors. The study of the convergence of the eigenvectors is more involved. Eigenvalues of the operator with multiplicity q greater than one, have a whole q dimensional eigenspace where each vector is an eigenvector. Furthermore, a perturbation of the operator would generate q different eigenvalues, slight modifications from the original one.

To circumvent this problem Koltchinskii [41], clusters the eigenvalues appropriately, so that each cluster corresponds to the perturbations of a single eigenvalue of the kernel operator. He studies the projections over the subspaces spanned by each cluster. The result states that the projection operator over the ith cluster of $\boldsymbol{K}_{X_m}$ converges to the ith eigenspace of T_k. Refer to [41, 15] for a detailed explanation.

Eigenfunctions. Bengio *et al.* [10] follow a different approach. They do not only study the convergence of the spectral properties of the kernel matrix, but also those of the Nyström extension. This is interesting to us, because it provides a formal justification that supports our intuitive choice of the kernel PCA projection (or the Nyström method) as an extension for the empirical mapping. The following propositions show firstly, that the Nyström approximation of the eigenfunctions of T_k, Eq. (2.69), are themselves the only eigenfunctions of the operator associated to the Monte Carlo approximation of the integral equation (2.66). Secondly, they give sufficiency conditions for the convergence of those eigenfunctions to the eigenfunctions of T_k.

Proposition 2.6 (Bengio *et al.* [10], Proposition 1)**.** *Denote by $\mathcal{L}(\mathcal{X}) \subset L_2(\mathcal{X})$ the set of square integrable, continuous real functions over $\mathcal{X}$. Let $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$ and $\boldsymbol{K}$ the kernel matrix for the set X. The operator $T_{k,X} : \mathcal{L}(\mathcal{X}) \to \mathcal{L}(\mathcal{X})$ given by*

$$T_{k,X} f(x) = \frac{1}{m}\sum_{i=1}^{m} k(x, x_i) f(x_i) \tag{2.74}$$

has $r \leqslant m$ non-zero eigenvalues given by $\hat{\gamma}_i$, the eigenvalues of the matrix $\frac{1}{m}\boldsymbol{K}$ (r is the rank of $\boldsymbol{K}$). The corresponding eigenfunctions are $\hat{\phi}_j$, where $\hat{\phi}_j$ is defined in Eq. (2.69).

The interesting part of this proposition is that $T_{k,X}$ has no other eigenfunctions associated with a non-zero eigenvalue.

Before addressing the convergence of the eigenfunctions, we are going to introduce some definitions, in order to consider a more general case which will

be usefull for us soon. Suppose that the kernel function k is unknown. Instead we compute empirical approximations to this function. These approximations will be kernels computed from the available data. We are going to refer to them as *data dependent kernels*. The functional form of a data dependent kernel will depend on the input set X. Since we are dealing with asymptotic properties when the number of points grows, we will use the notation X_m to explicit the amount of data, as we did in Theorem 2.2. As before we are also going to consider a sequence of random sets such that each set X_m is generated randomly from a i.i.d. process with probability density function $p(x)$.

Denote by k_{X_m} the data driven kernel, and $T_{k_{X_m},X_m}$ its associate operator as in Eq. (2.66). The subscript X_m appears twice, because the operator depends on X_m in two ways: firstly through the kernel and secondly through the discretization of the integral. Note that Theorem 2.2 deals with a sequence of matrices built from random datasets, while now we have a sequence of kernel functions built from random datasets.

Proposition 2.7 (Bengio *et al.* [10], Proposition 2)**.** *Suppose that the data dependent kernels k_{X_m} are uniformly bounded ($k_{X_m}(x, x') < c$ for all X_m, x, x') and that they converge uniformly in their arguments (x and x') and in probability to a kernel function k. Suppose that the eigenfunctions of $T_{k_{X_m},X_m}$ associated with non-zero eigenvalues also converge uniformly and in probability. Then their limit are the corresponding eigenfunctions of T_k.*

Note that the hypothesis of the Proposition are very strong and may be hard to check[4].

The case in which the k is unknown is common in the application of kernel methods to manifold learning and dimensionality reduction algorithms.

Finite size bounds

The convergence results are important because they show that the approximations used behave as the true mapping for a large enough number of points. However, in practice, we will have m data samples, and we would like to know how far are we from the actual mapping. In this case, finite sample size bounds are much more usefull. Most of the results for finite sample size were developed recently. The latest works are those of Shawe-Taylor *et al.* [58], Blanchard *et al.* [12] and Braun [15].

2.6 Dimensionality reduction kernels

As noted by [10, 34], some dimensionality reduction algorithms can be seen as kernel PCA. These algorithms are given a set of input points $X = \{x_1, \ldots, x_m\} \in \mathcal{X} \subset \mathbb{R}^d$. The aim of dimensionality reduction is to find a low dimensional representation of the data $Z = \{z_1, \ldots, z_m\} \subset \mathbb{R}^q$ with $q < d$, while preserving the geometrical properties of the original set. The methods differ mainly in the properties they aim to preserve.

[4] It should be noted that in [15] this result is considered

> "... commonplace in the numerical approximation of integral equations and are known at least since Nyström's initial paper [49]."

The algorithms we are going to consider obtain the mapping as the solution of an optimization problem, which can be computed as the eigendecomposition of a matrix, usually referred to as *transition* or *affinity* matrix. Generally the output dimension q is a parameter, which has to be given by the user. The low dimensional representative z_i is computed from the q largest or lowest eigenvectors (depending on the algorithm), in the same way as the kernel PCA mapping $\boldsymbol{y}_{x_i}$ of Eq. (2.25). Therefore it is almost straightforward to interpret the low dimensional representations as kernel PCA projections:

$$z_i = \left[[\boldsymbol{y}_{x_i}]_1, \ldots, [\boldsymbol{y}_{x_i}]_q\right]^T \tag{2.75}$$

where the kernel matrix is the affinity matrix or a simple transformation of it. These algorithms are often called *spectral dimensionality reduction* algorithms [52, 10].

In the following sections, we are going to give an overview of some of these dimensionality reduction algorithms: LLE [51], Diffusion Maps [20] and MVU [68]. Other methods such as IsoMap [63], Multidimensional Scaling (MDS) [23] and Laplacian Eigenmaps [8] are also spectral dimensionality algorithms, but we are not going to discuss them here.

Besides spectral dimensionality reduction, another new and related application of kernel methods is in the field of clustering. The *spectral clustering* [48, 71] and *normalized cuts* [59] methods are based on performing a simple clustering algorithm, such as *k-means*, after the data has been non-linearly mapped. The mapping is computed from the spectral decomposition of an affinity matrix, in the same way spectral dimensionality reduction does. The term *spectral embedding* is often used to refer to both spectral clustering and dimensionality reduction techniques.

2.6.1 Locally Linear Embedding

Locally Linear Embedding, or LLE was presented first by Roweis and Saul in [51]. The main idea is to find a low dimensional representation of the input data that preserves its local linear structure.

For each point x_i its n nearest neighbors are identified. Denote the $\eta(x_i)$ the set of indexes of the n nearest neighbors of x_i. The local linear structure of the manifold around x_i is defined by the weights that reconstruct x_i as a convex linear combination of its neighbors[5]. These weights are stored in the matrix $\boldsymbol{W}$, which is computed as

$$\boldsymbol{W}_{ij} = \arg\min_{\boldsymbol{W}} \|x_i - \sum_{i=1}^{m} \boldsymbol{W}_{ij} x_j\|^2 \tag{2.76}$$

$$\text{subject to } \sum_{j=1}^{m} \boldsymbol{W}_{ij} = 1 \text{ and } \boldsymbol{W}_{ij} = 0 \text{ if } j \notin \eta(x_i) \tag{2.77}$$

Note that each row of $\boldsymbol{W}$ can be computed independently from the others, reducing the above optimization to m least squares problems with n variables each.

[5] A convex linear combination must have coefficients which sum to 1.

The embedding is computed trying to preserve the expansion coefficients for each point. This can be formulated as the following optimization problem:

$$Z = \underset{z_1,\ldots,z_m}{\arg\min} \|z_i - \sum_{j=1}^{m} \boldsymbol{W}_{ij} z_j\|^2 \tag{2.78}$$

which is exactly the same function as (2.76) with the difference that in this case $\boldsymbol{W}_{ij}$ is given and the z_i's are the variables. This problem is ill posed because the objective function is invariant to affine transformations on Z. To remove this ambiguities, two constraints are added: the low dimensional representation must be centered and it must have unity covariance matrix.

The solution to the constrained optimization problem can be found from the $n+1$ bottom eigenvectors of the matrix: $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^T(\boldsymbol{I} - \boldsymbol{W})$. The lowest eigenvector is always the constant vector of value one: $\mathbf{1} = [1, 1, \ldots, 1]^T$ and its corresponding eigenvector is zero: the rows of $\boldsymbol{W}$ sum to unity and thus $(\boldsymbol{I} - \boldsymbol{W})\mathbf{1} = 0$.

To turn it into a kernel mapping, we define the matrix $\boldsymbol{K} = c\boldsymbol{I} - \boldsymbol{W}$. If $(\lambda_i, \boldsymbol{u}_i)$ is an eigenvalue-eigenvector pair of $\boldsymbol{M}$, $(c - \lambda_i, \boldsymbol{u}_i)$ is going to be one of $\boldsymbol{K}$, turning the smallest eigenvalues of $\boldsymbol{M}$ into the largest of $\boldsymbol{K}$. If $c \geqslant \lambda_{\max}$, then $\boldsymbol{K}$ will be positive semidefinite and symmetric, and thus it can be considered a kernel matrix.

The matrix $\boldsymbol{K}$ can be modified to eliminate the eigenvector $\mathbf{1}$:

$$(\boldsymbol{I} - \mathbf{1}\mathbf{1}^T)\boldsymbol{K}(\boldsymbol{I} - \mathbf{1}\mathbf{1}^T) \tag{2.79}$$

It is easy to show that this operation on the kernel matrix $\boldsymbol{K}$ is the centering operation of Eq. (2.31).

Performing kernel PCA with the centered version of $\boldsymbol{K}$ would yield basically the same mapping. There is only one difference: the mapping of LLE is computed using only the rows of the reduced eigenvector matrix without the normalization with the square root of the eigenvalues (recall for example Eq. (2.28)). This corresponds to a different scaling of the axes in the feature space, and thus can be ignored.

Actually the fact that the LLE mapping can be interpreted as kernel PCA is not surprising, after looking at the conditions used in the optimization problem that defines the mapping. Enforcing the convariance matrix of the mapped data to be the identity yields a mapping expressed in the coordinate system given by the principal axes, as kernel PCA does.

The algorithm has two parameters: n, the number of nearest neighbors and q the desired dimension of the embedding. The n parameter defines the local scale. This is a crucial parameter of the algorithm. Setting it high in cases when the manifold is poorly sampled can easily create wrong connections between different parts of the manifold, which may be close in the ambient space but far away w.r.t. the geodesic distance inside the manifold. This connections are called *shortcircuits*. On the other hand setting it too low, does not generate enough connections between the samples. The weight matrix will have less information for reconstructing the low dimensional dataset.

The dimension of the output space q, is also important. It is very hard to set in real situations, where the dimension of the manifold underlying the data is unknown. Recall from Eqs. (2.25) and (2.65) that the rank of the kernel is the

number of non-zero principal components in the feature space, and therefore the dimension of the kernel PCA mapping. The rank of the LLE kernel matrix will be generally higher than the dimension of the underlying manifold. Thus when imposing the dimension of the embedding to be q, even if it is the real dimension of the underlying manifold, there is a lot of information in the higher principal components that will be discarded. The LLE kernel matrix encodes much more information than is actually needed. Besides the inefficiency problem, this also makes the problem of choosing the embedding dimension q hard, since there seems to be no correlation between the rank of the kernel matrix and q.

This parameter however is present in almost every dimensionality reduction algorithm. There are works that approach the problem of estimating the dimensionality of the underlying manifold [22, 46].

Out-of-sample extension

The authors of LLE proposed to compute the out-of-sample extension for a new point x with the same idea used to build the mapping: find the $\boldsymbol{w}_x$ coefficients that approximate x as a convex combination of its n nearest neighbors in the training set, and define z_x as a linear combination of the z_i with the found coefficients. The same idea can be used to invert the mapping.

Contrary to the analytic kernels case, there is no simple kernel function k such that $\boldsymbol{K}_{ij} = k(x_i, x_j)$, needed for computing the kernel PCA projection of x. Bengio *et al.* [11] defined such a function based on the $\boldsymbol{w}_x$ coefficients of x. They reported good results and showed that when the constant c tends to infinity, their kernel PCA-based out-of-sample extension converges to the heuristic one.

2.6.2 Maximum Variance Unfolding

The Maximum Variance Unfolding algorithm was introduced by Weinberger and Saul in [68]. The low dimensional embedding computed by this algorithm preserves the local distances (*i.e.* the mapping is a local isometry), while maximizing the variance of the low dimensional representation. The intuition behind this is that maximizing the variance stretches the manifold eliminating the curvature.

As opposed to the rest of the spectral dimensionality reduction algorithms, MVU was conceived as a kernel method. Instead of looking for the coordinates of the mapping, they focus on the kernel matrix $\boldsymbol{K}$, translating the requirements on the mapping into requirements on the matrix. This can be done because all the conditions imposed on the mapping can be expressed in terms of dot products. The kernel matrix is found as the result of a *semidefinite programming* problem [66], where the variable is the whole $m \times m$ matrix itself.

The objective function is the variance of the low dimensional representation $Z = \{z_1, \ldots, z_m\}$. Assuming that Z is centered we can compute the variance in terms of the kernel matrix as:

$$\sum_{i=1}^{m} \|z_i\|_{\mathcal{H}}^2 = \sum_{i=1}^{m} \langle z_i, z_i \rangle_{\mathcal{H}} = \sum_{i=1}^{m} \boldsymbol{K}_{ii} = \operatorname{tr}(\boldsymbol{K}) \tag{2.80}$$

To guarantee that the found matrix is a valid kernel matrix, it has to be symmetric and positive semidefinite. The set of these matrices is a cone in the

space of real $m \times m$ matrices. This is a convex set. Semidefinite programming is an optimization technique designed for these kind of problems: linear functions, with linear restrictions over the symmetric, positive semidefinite matrices cone.

To enforce the centering of the mapping the following condition is needed:

$$0 = \sum_{i=1}^{m} z_i = \left\| \sum_{i=1}^{m} z_i \right\|_{\mathcal{H}}^2 = \sum_{i,j=1}^{m} \langle z_i, z_j \rangle_{\mathcal{H}} = \sum_{i,j=1}^{m} \boldsymbol{K}_{ij} \tag{2.81}$$

MVU is a local algorithm, and the local scale is defined by the number of nearest neighbors n. As we did for LLE, let us define for each input point x_i the set of nearest neighbors indices $\eta(x_i)$. The low dimensional representation of the whole neighborhood must be isometric. Thus for each x_j and $x_{j'}$ with $j, j' \in \eta(x_i)$ (each pair of common neighbors of x_i) we have the following constraints:

$$\|z_j - z_{j'}\|_{\mathcal{H}}^2 = \|x_j - x_{j'}\|^2 \tag{2.82}$$

which can be expressed in terms of the kernel as:

$$K_{jj} + K_{j'j'} - 2K_{jj'} = \|x_j - x_{j'}\|^2 \tag{2.83}$$

This defines a set of linear restrictions.

The matrix is then found as the solution to the semidefinite programming problem of maximizing (2.80) subject to (2.81) and (2.83). This is generally very costly, and its complexity increases with the number of nearest neighbors, since this increases the number of restrictions. However this technique has some benefits over LLE, regarding the choice of the embedding dimension. It has been observed, although there are no formal results about it, that maximizing the variance yields kernel matrices with low rank [61]. Thus, the rank of the matrix can be used as an important guide for determining the dimension of the embedding.

Weinberger *et al.* [70] propose to circumvent the problem of the prohibitive computational cost by expanding the columns of the matrix in a data dependent basis, and working with the coefficients vector. Modifications of the algorithm that allow some stretching in the local distances were introduced in [69]. This is a desirable feature when the samples over the manifold have noise.

Out-of-sample extension

The natural way of extending the mapping (and similarly for its inverse) for a new point x, is to preserve the distances towards its nearest neighbors. Although this is a kernel method, as with LLE, there is no simple way of finding a corresponding kernel function with an analytic expression.

2.6.3 Diffusion Maps

Diffusion Maps [20, 44] is a modification from the Laplacian Eigenmaps technique [8]. To understand how it works, first we are going to start by a justification of the Laplacian Eigenmaps algorithm given in [8].

We are going to consider the input dataset as a weighted graph whose vertexes are the input points. The weight between points will be stored in an affinity matrix $\boldsymbol{W}$, and can be establish by a nearest neighbors graph on the

dataset: $\boldsymbol{W}_{ij} = 1$ if $x_i \in \eta(x_j)$ or $x_j \in \eta(x_i)$, otherwise $\boldsymbol{W}_{ij} = 0$; or by a weighted graph such that the weight between points x_i and x_j is given by a Gaussian kernel:

$$\boldsymbol{W}_{ij} = \exp\left(-\|x_i - x_j\|^2/2\sigma^2\right). \tag{2.84}$$

The only thing we are going to ask our embedding is to keep as close as possible connected points. In order to have connected points lying close in the embedding, we will minimize the following cost function

$$\sum_{i,j=1}^{m} \|z_i - z_j\|^2_{\mathcal{H}} \boldsymbol{W}_{ij} \tag{2.85}$$

It can be shown that the function in the last expression can be written as

$$\sum_{i,j=1}^{m} \|z_i - z_j\|^2_{\mathcal{H}} \boldsymbol{W}_{ij} = 2\mathrm{tr}(\boldsymbol{Z}\boldsymbol{L}\boldsymbol{Z}^T) \tag{2.86}$$

where $\boldsymbol{Z} = [z_1, \ldots, z_m]$ is the $q \times m$ matrix whose columns are the embedding coordinates, and $\boldsymbol{L}$ is the $m \times m$ *Laplacian* matrix on the input graph: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$. $\boldsymbol{D}$ is a diagonal matrix such that $\boldsymbol{D}_{ii} = \sum_{j=1}^{m} \boldsymbol{W}_{ij}$, the degree of the ith node x_i.

The minimization of Eq. (2.85) on the matrix $\boldsymbol{Z}$ is not yet a well posed problem: setting $\boldsymbol{Z} = \boldsymbol{0}$ yields the optimum. To remove this degeneracy, the authors add the following constraints:

$$\boldsymbol{Z}\boldsymbol{D}\boldsymbol{Z}^T = \boldsymbol{I} \tag{2.87}$$

$$\boldsymbol{Z}\boldsymbol{D}\boldsymbol{1} = \boldsymbol{0} \tag{2.88}$$

the first one fix the scaling of the mapping whereas the second is analogous to a centering operation to remove translation ambiguities. The solution to this problem is given by the bottom eigenvectors of the following generalized eigenvalue problem

$$\boldsymbol{L}\boldsymbol{v} = \lambda \boldsymbol{D}\boldsymbol{v} \tag{2.89}$$

As in LLE, the lowest eigenvector is removed, since it is the constant vector of value 1, with eigenvalue 0.

To transform the generalized eigenvalue problem into an ordinary one, consider $\boldsymbol{w} = \boldsymbol{D}^{1/2}\boldsymbol{v}$. Then

$$\begin{aligned} &\boldsymbol{L}\boldsymbol{v} = \lambda \boldsymbol{D}\boldsymbol{v} \Rightarrow \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{v} = \lambda \boldsymbol{D}^{1/2}\boldsymbol{v} \Rightarrow \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}\boldsymbol{w} = \lambda \boldsymbol{w} \\ &\Rightarrow (\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2})\boldsymbol{w} = \lambda \boldsymbol{w} \Rightarrow \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}\boldsymbol{w} = (1-\lambda)\boldsymbol{w} \end{aligned} \tag{2.90}$$

Therefore, we can obtain the mapping (up to the scaling by the degree matrix) from the top eigenvectors of the matrix $\boldsymbol{K} = \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$. If the weights were computed with the Gaussian kernel, this matrix is symmetric and it can be shown that it is also positive semidefinite [19]. Note that in this case the entries of $\boldsymbol{K}$ are given by

$$\boldsymbol{K}_{ij} = \frac{\boldsymbol{W}_{ij}}{\sqrt{\sum_{l=1}^{m} \boldsymbol{W}_{lj}}\sqrt{\sum_{l=1}^{m} \boldsymbol{W}_{il}}} \tag{2.91}$$

It has been shown [8], that in the case that the input points are uniformly sampled from a manifold, the kernel converges to the heat diffusion operator on the manifold, the *Laplace-Beltrami operator* when the number of samples m tends to infinity and the scale parameter of the weight σ tends to zero. In the case of non-uniform sampling density, as shown by Coifman and Lafon [20], the limit of the kernel depends on the manifold density. In some cases this may be undesirable. They generalized Laplacian Eigenmaps with the following normalization in the weight matrix

$$\widetilde{\boldsymbol{W}}_{\alpha,ij} = \frac{\boldsymbol{W}_{ij}}{\left(\sum_{l=1}^{m} \boldsymbol{W}_{lj}\right)^{\alpha} \left(\sum_{l=1}^{m} \boldsymbol{W}_{il}\right)^{\alpha}} \tag{2.92}$$

$$\boldsymbol{K}_{\alpha,ij} = \frac{\widetilde{\boldsymbol{W}}_{\alpha,ij}}{\sqrt{\sum_{l=1}^{m} \widetilde{\boldsymbol{W}}_{\alpha,lj}} \sqrt{\sum_{l=1}^{m} \widetilde{\boldsymbol{W}}_{\alpha,il}}} \tag{2.93}$$

and have proved that setting $\alpha = 1$ yields a density invariant kernel (and thus a density invariant embedding). On the other hand $\alpha = 0$ recovers the Laplacian Eigenmaps embedding.

The kernel matrix can also be exponentiated $\boldsymbol{K}^t$. This has an interesting interpretation in terms of random walks [20]. The eigenvalues of this matrix are all between 0 and 1. Thus while t grows the variance of the higher principal components (lower eigenvalues) diminishes, and the energy of the mapping concentrates in the first principal components.

The basic parameters of this algorithm are the scale of the Gaussian weights σ and the embedding dimension q. As with LLE, there appears to be no clear correlation between the rank of the kernel matrix and the dimension of the underlying manifold. In fact, the smaller the scale, the slowest the decay of the eigenvalues. Recall that the eigenvalues of the kernel matrix represent the energy in each principal axis. Thus, a slow decay means that the energy in the feature space is distributed between several principal components, instead of concentrated in a few of them. This does not mean that the first principal components will not yield a good embedding. As in LLE, the kernel matrix has much more information than the one we actually need.

Out-of-sample extension

Coifman and Lafon use the *geometric harmonics* framework [21] to extend the eigenvalues. This is based in the Nyström extension for the eigenvectors of an auxiliary kernel. In §3.2.1 we are going to discuss their approach more in depth.

Recently Etyngier *et al.* [29] proposed an analytic expression for the Diffusion Maps kernel, which is an adaptation of an analytic kernel for Laplacian Eigenmaps proposed by [11]. These expressions can be used to extend the kernel PCA map using (2.27).

Chapter 3

Data Dependent Kernels Extension

3.1 Introduction

At the end of previous chapter, we presented a number of dimensionality reduction techniques that are kernel methods. However, the nature of these kernel algorithms is different from the usual pattern recognition setting. In the latter case the kernel matrix corresponds to the evaluation of a known kernel function over the training set.

In the dimensionality reduction applications of §2.6 the kernel is an unknown function. The kernel matrix is a result of the dimensionality reduction algorithm and generally, each value $\boldsymbol{K}_{ij}$ depends on the whole training set $X = \{x_1, \ldots, x_m\} \subset \mathcal{X}$, and not just on x_i and x_j. If any point is modified, the whole kernel matrix will change. This does not allow the out-of-sample extension of the mapping using the empirical kernel map extensions of §2.5.

We may circumvent this problem by assuming that there is an unknown data dependent kernel $k_X : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for which $\boldsymbol{K}_{ij} \approx k_X(x_i, x_j)$. We will refer to the problem of finding such a kernel the *kernel extension* problem. Ideally, we would like this kernel function to be an approximation of a limit kernel function k, as in Proposition 2.6.

Note that for the out-of-sample extensions of the empirical kernel map, we only need the kernel values between a new input point x and the training patterns, $k_X(x, x_i)$ with $i = 1, \ldots, m$. In fact, the kernel vector $\boldsymbol{k}_{X,x} = [k_X(x, x_1), \ldots, k_X(x, x_m)]^T$ is itself a representation of the empirical mapping, as we saw in §2.5.

In this chapter we are going to review a few methods to extend data driven kernel matrices to new points and select one. The objective is to compute the kernel vector on an unseen point x. The criteria for the choice of the extension method should take into account:

performance It should have a low prediction error, meaning that the extended kernel vector for x, $\boldsymbol{k}_{X,x}$, should be close to the kernel vector that would have been obtained if x was added to training set.

simplicity The purpose of the extension is to design pre-image algorithms for

this type of kernels. The extension method should be simple enough to allow the posterior analysis. For instance a simple closed form for $\boldsymbol{k}_{X,x}$ would be desirable.

low computational complexity The objective of out-of-sample methods is to leverage the computational cost of an algorithm. Training (*i.e.* finding the map for the training set) is costly since it involves at least the singular value decomposition (SVD) of the kernel matrix. Therefore, this should be done only once. Then each new point is mapped using the out-of-sample extension, thus avoiding retraining.

In §3.2 we briefly review different ways of extrapolating the kernel matrix. The selected method, Kernel Ridge Regression is described in §3.3. The Chapter ends with some results and a discussion about the selection of the parameters of the kernel framework.

3.2 Review of existing techniques

In this section we are going to study possible approaches to solve the kernel extension problem. Basically we are going to distinguish between the following types of approach:

- *Kernel matrix completion algorithms*: These methods impose that the extended new kernel matrix is a semipositive definite matrix. This problem is more involved however. They can be considered as interpolation methods, since the new matrix will keep the same value in the entries that were present in the old matrix. They just will interpolate new values on the new points. Another characteristic of these methods is that they do not provide an analytic extension function.

- *Generic regression approach*: The problem can be considered as m regression problems[1]. This approach considers each column of the matrix $\boldsymbol{K}$ as an independent function and using some regression technique learns a function for each column. Note that the ith column has the kernel values between x_i and the rest of the training set. Extending it will provide $k_X(x_i, \cdot)$.

The kernel matrix completion algorithms compute an approximation that fullfills the symmetry and positive semidefinitness properties of kernel matrices, however they have in general a greater computational complexity, and usually involve a heavy computation for each new point, reasons for which we discarded them. The interested reader may find a description of two algorithms of this kind in Appendix B.

3.2.1 Generic Regression Approaches

The generic regression algorithms reviewed below are themselves kernel methods. We will denote by h the auxiliary kernel h used by the regression algorithm.

[1] A regression problem is the problem of fitting a function to a set of points $\{(x_1, y_1), \ldots, (x_m, y_m)\}$. As opposed to the interpolation problem, the function does not have to take the value y_i in x_i.

To keep the notation simple, we are going to present the methods for the problem of learning a function $f : \mathbb{R}^d \to \mathbb{R}$, given m patterns $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$. Recall that the problem of extending the kernel can be reduced to m of these problems. In our setting the y_i would take the value of $\boldsymbol{K}_{ij}$ for a fixed j, and $f(x)$ would be $k_X(x, x_j)$.

The performance of a learned function f can be defined as the expected value of the *loss* function, which measures the error incurred when predicting $f(x)$ instead of y, the true outcome. Examples of loss function can be the squared error $(f(x) - y)^2$, the absolute value of the error $|f(x) - y|$, or the so-called ϵ-insensitive loss $|f(x) - y|_\epsilon = \min\{|f(x) - y|, \epsilon\}$, among others. The function f should minimize the expected loss:

$$R(f) = \mathbb{E}\{\text{loss}(f(x), y)\} = \int_{\mathcal{X}\times\mathbb{R}} \text{loss}(f(x), y)p(x, y)dxdy \tag{3.1}$$

The expected loss $R(f)$ is also referred to as the *risk*. The objective of a machine learning algorithm is therefore to minimize the risk. However, in practice the probability density function of the patterns $p(x, y)$ is unknown, and the *empirical risk* is minimized instead:

$$R(f) = \frac{1}{m}\sum_{i=1}^{m} \text{loss}(f(x_i), y_i) \tag{3.2}$$

This problem is badly posed if we do not add any constraint. There are an infinite number of different functions that would yield zero empirical risk. One trivial example is a function which is zero everywhere except on X, where $f(x_i) = y_i$. To exclude these solutions the search space must be reduced. Kernel regression methods restrict the search space to the RKHS $\mathcal{H}_h$[2], associated with a kernel h. Recall from §2.4.1 that this was a subset of real functions over $\mathcal{X}$.

However this may not be enough. There are kernels whose RKHS is so rich, that we could find an f with zero empirical risk for any finite set of labeled points (x_i, y_i). If the samples y_i have noise, we would therefore learn the noise and incorporate it in our function f. This is what is called *overfitting*.

Usually a regularization term is added, to prevent overfitting to the training set. Thus, generally, the kernel regression problem can be stated as:

$$f = \arg\min_{f\in\mathcal{H}_h} \sum_{i=1}^{m} \text{loss}(f(x_i), y_i) + \gamma\Omega(\|f\|_{\mathcal{H}_h}) \tag{3.3}$$

where the regularization term $\Omega(\|f\|_{\mathcal{H}_h})$ is a function of the feature space norm of f. The regularization coefficient γ determines the relative weight of the regularization term.

The *Representer Theorem* [53] proves that in this setting (adding some conditions on the loss function and Ω) the minimum of the regularized risk can be written as a kernel expansion using only the training set points:

$$f(x) = \sum_{i=1}^{m} \beta_i h(x, x_i) \tag{3.4}$$

[2] $\mathcal{H}_h$ refers actually to any feature space associated with h. We could be more precise in this case, since we are dealing with functions, and use the notation $\mathcal{H}_R^h$, for example. To keep the notation simple we are not going to do that.

This is equivalent to choosing the following kernelized linear function:

$$f(x) = \sum_{i=1}^{m} \beta_i h(x, x_i) = \left\langle \sum_{i=1}^{m} \beta_i \varphi_h(x_i), \varphi_h(x) \right\rangle_{\mathcal{H}_h} = \langle \omega, \varphi_h(x) \rangle_{\mathcal{H}_h} \tag{3.5}$$

where $\omega = \sum_{i=1}^{m} \beta_i \varphi_h(x_i) \in \mathcal{H}_h$ is a vector in the feature space of kernel h. With this definitions, we can rewrite Eq. (3.3) in a more suitable way for computational purposes:

$$f = \arg\min_{\beta \in \mathbb{R}^m} \sum_{i=1}^{m} \text{loss}(\langle \omega, \varphi_h(x_i) \rangle_{\mathcal{H}_h}, y_i) + \gamma \Omega(\|\omega\|_{\mathcal{H}_h}) \tag{3.6}$$

$$= \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^m} \sum_{i=1}^{m} \text{loss} \left(\sum_{j=1}^{m} \beta_j h(x_i, x_j), y_i \right) + \gamma \Omega(\|\omega\|_{\mathcal{H}_h}) \tag{3.7}$$

The methods we are going to see below are variants of this general kernel regression framework. They differ mostly in the loss function or the regularizer used.

Applied to our kernel matrix extension problem, these type of methods would yield m kernel expansions with coefficients vectors $\boldsymbol{\beta}_j$ for each of the kernel functions $k_X(x_j, \cdot)$:

$$k_X(x_j, x) = \sum_{i=1}^{m} [\boldsymbol{\beta}_j]_i h(x, x_i) \tag{3.8}$$

Through the use of an auxiliary analytic kernel h we are able to extrapolate the kernel matrix.

The approach we implemented to perform the out-of-sample extension was the Kernel Ridge Regression, mainly for its simplicity. However all these methods share the same type of solution: a closed form, analytic expression as in Eq. (3.8).

Support Vector Regression

Support vector type algorithms [53] search for a sparse solution. They obtain this with the ϵ-insensitive loss:

$$|y - f(x)|_\epsilon = \max\{0, |y - f(x)| - \epsilon\} \tag{3.9}$$

In words, the ϵ-insensitive loss function penalize errors only if they are bigger than ϵ. The regularizer used is $\|\omega\|^2_{\mathcal{H}_h}$.

The use of this loss function will cause many coefficients β_i to be zero. It can be proved that β_i takes a nonzero value, only if the prediction error in x_i is greater or equal than ϵ. These are called support vectors. All other vectors do not influence the solution. This causes the method to be robust to outliers. The smaller ϵ, the greater the number of support vectors, yielding a less sparser solution. The optimization problem in this case requires solving a quadratic program, although there are linear variants.

Generalized LASSO Regression

The (not kernelized) LASSO Regression [36] corresponds to the minimization of the following expression:

$$Q(\omega) = \|\omega\|_1 + \frac{\gamma}{m}\sum_{i=1}^{m}(y_i - \omega \cdot x_i)^2 \tag{3.10}$$

where ω denotes now a vector in the input space and the L_1 norm of ω as a regularization term is used. This generates a solution which tends to be sparse too. Its computation involves a linear optimization problem, which could be costly when m is large. The same idea can be extended to the kernel setting, substituting the $\|\omega\|_{\mathcal{H}_h}$ regularizer for $\|\boldsymbol{\beta}\|_1$, the L_1 norm of the coefficients vector, and using the squared error as the loss function. This was proposed in [50] with the name of Generalized LASSO Regression

Manifold Regularization

Belkin *et al.* [9] propose a framework for *transductive learning*, which generalizes the model of Eq. (3.6). Transductive learning is when the unlabeled data has to be present at the training stage together with the labeled data (for example semi-suppervised clustering). If the unlabeled data is not available during training we have *inductive learning*. As opposed to inductive learning, transductive learning does not generalize to new data points.

Their approach is based on Kernel Ridge Regression §3.3, the main difference being an additional regularization term which takes into account the geometry of the input samples, both labeled and unlabeled, ensuring that the solution is smooth with respect to the data distribution.

Suppose that from the m input samples, only $\ell \leqslant m$ are labeled. The aim is to find the function f such that:

$$f^* = \arg\min_{f\in\mathcal{H}_h} \frac{1}{\ell}\sum_{i=1}^{\ell}\|y_i - f(x_i)\|^2 + \gamma_A\|f\|^2_{\mathcal{H}_h} + \gamma_I\|f\|^2_I \tag{3.11}$$

where $\mathcal{H}_h$ is the RKHS associated with the (auxiliary) kernel h and $\|\cdot\|_{\mathcal{H}_h}$ is the norm of f in $\mathcal{H}_h$ (recall that the space of functions defined as kernel expansions is contained in the Reproducing Kernel Hilbert Space (RKHS)).

On the other hand, the term $\|\cdot\|_I$ is a measure of the complexity of f with respect to the intrinsic geometry of the whole set of input points, labeled and unlabeled. This is the novel manifold regularization term.

In the case when the data is sampled from a manifold $\mathcal{M}$, a natural choice of the term $\|\cdot\|_I$ would be the summation over the manifold of the norm of the gradient of f

$$\|f\|^2_I = \int_{\mathcal{M}} \langle \nabla f, \nabla f\rangle \tag{3.12}$$

Note that the norm of this gradient, is the euclidean norm in the input space. Even if we do not know the manifold, this can be numerically approximated using discrete approximations.

With some smoothness assumptions on the manifold term, the authors prove that the solution to the problem is a kernel expansion as Eq. (3.4).

Geometric Harmonics

Geometric harmonics [21] is a generalization of the Nyström extension to the problem of learning a general function over a finite set $X \subset \mathcal{X}$. The main idea is to write the function in terms of a basis given by the eigenvectors of an auxiliary kernel h, and then use the Nyström method to extend that mapping.

The first step is to compute the eigenvectors $\boldsymbol{u}_{h,i}$ and eigenvalues $\lambda_{h,i}$ of the auxiliary kernel matrix $\boldsymbol{H}$. The vector of function values $\boldsymbol{y} = [y_1, \ldots, y_m]^T$, can be expressed in the basis of eigenvectors

$$\boldsymbol{y} = \sum_{i=1}^{m} \langle \boldsymbol{y}, \boldsymbol{u}_{h,i} \rangle \boldsymbol{u}_{h,i} \tag{3.13}$$

Since the kernel h can be evaluated outside the training set, we can use the Nyström method to extend the eigenvectors

$$\boldsymbol{u}_{h,j}(x) = \frac{1}{\lambda_{h,j}} \sum_{i=1}^{m} h(x_i, x)[\boldsymbol{u}_{h,j}]_i \tag{3.14}$$

There is a problem with this expansion, and is that of dividing between λ_j. Therefore we must exclude the smaller eigenvalues from expression (3.13). This is equivalent to projecting $\boldsymbol{y}$ onto the space spanned by the principal eigenvectors. The number of eigenvectors considered is given by a parameter δ, that measures the maximum condition number of the extension operator. The condition number of the extension is given by the smallest λ_j, therefore fixing δ is equivalent to fixing, for a given kernel, the number of eigenvalues over which to project.

However for some functions, by projecting over the largest eigenvalues we may incur in a big projection error, extending a function which is a oversimplified version of $\boldsymbol{y}$. This can be fixed by changing the kernel. In particular, if h is the Gaussian kernel, it is enough to use a smaller σ_h parameter (with a smaller σ_h the eigenvalues decay slower, and it is possible to consider more eigenvalues while having a smaller condition number). This however diminishes the distance to which the function can be extended.

Therefore, for complex functions, a smaller σ_h will be necessary and the extension will be shorter ranged. However, if the function is smooth, we can use a larger σ_h with a wider extension range.

The authors propose an iterative algorithm to set the value for σ_h, starting with a large value and making it smaller if it is necessary. This is a costly procedure, because for each σ_h the eigendecomposion of the kernel matrix H needs to be computed.

In our case, for extending a data driven kernel k, we can use some heuristic method based on what we know about the kernel. For instance for the diffusion maps kernel, we could use the same σ_h parameter. We will discuss further this issue in §3.3.2.

Lafon *et al.* [45] use the framework to compute an out-of-sample extension for the Diffusion Maps embedding, treating each coordinate of the embedding as a function defined over the training set.

3.3 Kernel Ridge Regression

In the linear case (plain *Ridge Regression*) the objective is to find a linear function that minimizes the total L_2 error between the predicted value $(x_i, f(x_i))$ and the corresponding pattern (x_i, y_i). The function is fully parametrized by a vector $\omega \in \mathbb{R}^d$.

$$f(x) = \omega \cdot x \tag{3.15}$$

The parameter vector ω is computed by minimizing the following regularized empirical risk:

$$Q(\omega) = \frac{1}{2}\|\omega\|^2 + \frac{\gamma}{m}\sum_{i=1}^{m}(y_i - \omega \cdot x_i)^2 \tag{3.16}$$

In the linear case, the regularization term penalizes solutions with high coefficients. This is the Tikhonov regularization. It will be more usefull in the kernelized version of this algorithm.

The method can be extended to handle nonlinear functions using the kernel trick. The resulting algorithms is known as *Kernel Ridge Regression* (KRR) or *Regularized Least Squares* [17]. Suppose we have a kernel $h : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and its mapping $\varphi_h : \mathbb{R}^d \to \mathcal{H}_h$. This kernel should not be confused with the data driven kernel we want to extend. This is just an auxiliary kernel function (with a known analytic expression), used to define a richer function family. For example we can use the Gaussian kernel (2.6).

We are going to solve the linear regression problem in the feature space. The corresponding cost function is:

$$Q(\omega) = \frac{1}{2}\|\omega\|^2_{\mathcal{H}_h} + \frac{\gamma}{m}\sum_{i=1}^{m}(y_i - \langle\omega, \varphi_h(x_i)\rangle_{\mathcal{H}_h})^2 \tag{3.17}$$

Note that this expression is of the form of Eq. (3.6), with the squared error as a loss function and the square feature space norm as a regularizer. This can be stated as a constraint optimization problem:

$$L(\omega, \boldsymbol{\xi}, \boldsymbol{\beta}) = \frac{1}{2}\|\omega\|^2_{\mathcal{H}_h} + \frac{\gamma}{m}\sum_{i=1}^{m}\xi_i^2 + \sum_{i=1}^{m}\beta_i(y_i - \langle\omega, \varphi_h(x_i)\rangle_{\mathcal{H}_h} - \xi_i)^2 \tag{3.18}$$

$$\text{subject to: } \xi_i = y_i - \langle\omega, \varphi_h(x_i)\rangle_{\mathcal{H}_h} \tag{3.19}$$

The saddle point conditions for L are:

$$\frac{\partial L}{\partial \omega} = 0 \Rightarrow \omega = \sum_{i=1}^{m}\beta_i\varphi_h(x_i) \tag{3.20}$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow \beta_i = \frac{2\gamma}{m}\xi_i \tag{3.21}$$

$$\frac{\partial L}{\partial \beta_i} = 0 \Rightarrow \xi_i = y_i - \langle\omega, \varphi_h(x_i)\rangle_{\mathcal{H}_h} \tag{3.22}$$

Substituting these relationships into (3.18) and (3.19), and solving for β yields the following expression:

$$\boldsymbol{\beta} = \left(\boldsymbol{H} + \frac{m}{2\gamma}\boldsymbol{I}\right)^{-1}\boldsymbol{y} \tag{3.23}$$

where $\boldsymbol{H}$ is the kernel matrix of h and $\boldsymbol{y} = [y_1, \ldots, y_m]^T$. According to (3.20) the function f is given by:

$$f(x) = \langle \omega, \varphi_h(x) \rangle_{\mathcal{H}_h} = \sum_{i=1}^{m} \beta_i \langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{H}_h} = \sum_{i=1}^{m} \beta_i h(x_i, x) \tag{3.24}$$

This solution can be extended to a more general case, adding a constant b to the kernel expansion:

$$f(x) = \sum_{i=1}^{m} \beta_i h(x_i, x) + b \tag{3.25}$$

The following modification on Eq. (3.23) allows to find $\boldsymbol{\beta}$ and b:

$$\begin{bmatrix} \boldsymbol{H} + \frac{m}{2\gamma}\boldsymbol{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix} \tag{3.26}$$

One of the problems of this approach is that it involves the inversion of a potentially large matrix. Recall that this kernel matrix is not the kernel matrix we want to extend, thus we would need to perform the SVD of the kernel matrix (needed for kernel PCA) and then invert the matrix in Eq. (3.23). However there is a way to alleviate this problem using a low rank approximation of $\boldsymbol{H}$. The idea is to find a subset of $\{\varphi_h(x_1), \ldots, \varphi_h(x_m)\}$ which could serve as a basis in the feature space.

It is possible that such a subset does not exist. The Gaussian kernel for example is a full rank kernel. Therefore the set $\{\varphi_h(x_1), \ldots, \varphi_h(x_m)\}$ is linearly independent. However one still can keep only a subset of n vectors $\varphi_h(x_i)$ that spans the subspace of principal variation. Finding this subspace is the aim of kernel PCA, and this involves computing a SVD of $\boldsymbol{H}$. In [7] an alternative greedy algorithm is used to find an appropriate basis. Following this approach, one can reformulate the whole problem using only $n < m$ points, and thus inverting an $n \times n$ matrix. The result is a sparse coefficient vector $\boldsymbol{\beta}$, with at most n non-zero coefficients. The resulting function will be a sum of n kernels.

3.3.1 Application to the kernel extension problem

Suppose we have computed a kernel matrix $\boldsymbol{K}$ for the set $X = \{x_1, \ldots, x_m\}$. $\boldsymbol{K}$ could be for instance, the kernel matrix of the Diffusion Maps method.

As was said before, we are going to "learn" m functions to extend each of the m columns of $\boldsymbol{K}$, using Kernel Ridge Regression. The jth function $k_X(x_j, x)$ will be expressed as a kernel expansion (3.25) with coefficients vector $\boldsymbol{\beta}_j$ and a b_j. We are going to consider the same kernel h for all the m functions. Then:

$$k_X(x_j, x) = \sum_{i=1}^{m} [\boldsymbol{\beta}_j]_i h(x_i, x) + b_j \tag{3.27}$$

We will always use for our experiments the Gaussian kernel with scale parameter σ_h. The scale parameter is also constant for all m functions.

The coefficients vector $\boldsymbol{\beta}_j$ and the constant b_j are computed according Eq. (3.26), where the functional values $\boldsymbol{y}$ for the training set are given by $\boldsymbol{K}_{\cdot j}$, the jth column of $\boldsymbol{K}$.

Let us define the $m \times m$ matrix $\boldsymbol{B}$ having in its columns the coefficients vectors $\boldsymbol{\beta}_j$, $\boldsymbol{B} = [\boldsymbol{\beta}_1 \ \cdots \ \boldsymbol{\beta}_m]$ and the vector $\boldsymbol{b} = [b_1, \ldots, b_m]^T$. Using Eq. (3.26) we can find $\boldsymbol{B}$ and $\boldsymbol{b}$ as:

$$\begin{bmatrix} \boldsymbol{B} \\ \boldsymbol{b}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{H} + \frac{m}{2\gamma}\boldsymbol{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{K} \\ \mathbf{0}^T \end{bmatrix} \tag{3.28}$$

where $\mathbf{0}$ is a m dimensional vector of zeros.

We can also extend Eq. 3.27 to find the kernel vector $\boldsymbol{k}_{X,x}$ with this matrix notation

$$\boldsymbol{k}_{X,x} = \boldsymbol{B}^T \boldsymbol{h}_x + \boldsymbol{b} \tag{3.29}$$

where $\boldsymbol{h}_x$ is the auxiliary kernel vector $\boldsymbol{h}_x = [h(x_1, x), \ldots, h(x_m, x)]^T$.

Results

We have used the kernel vector $\boldsymbol{k}_{X,x}$ to extend the kernel PCA projections according to Eq. (2.26), for the Diffusion Maps kernel. Recall that this kernel has several parameters: the scale σ_k, the parameter α (see Eq. (2.93)) and the exponent t. For these experiments we set $\alpha = 1$ and $t = 1$. The former makes the embedding density invariant. The scale parameter σ_k was set as the average distance to the 10th nearest neighbor as in (2.34).

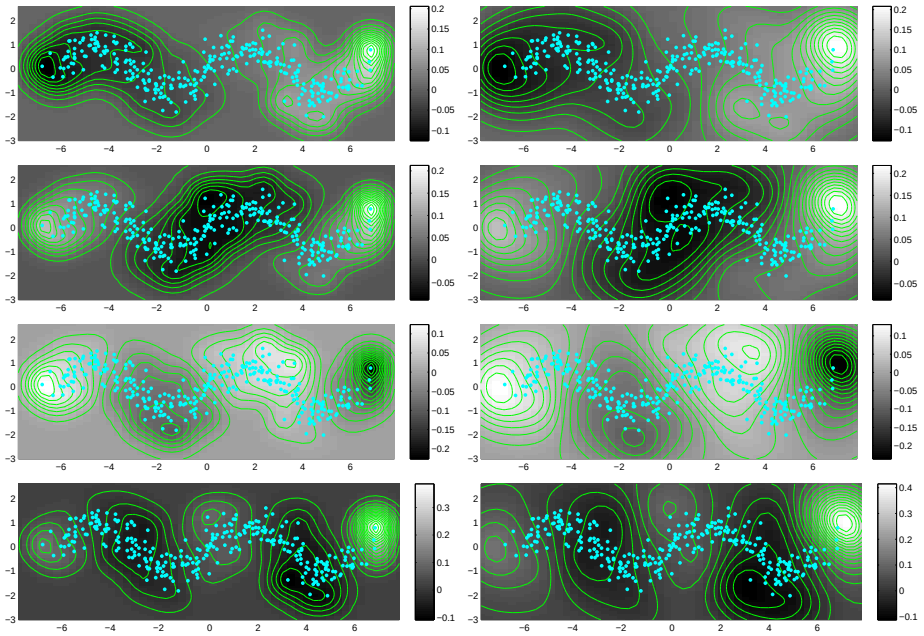

Figure 3.1: Kernel PCA projections over the first four principal axes of the Diffusion Maps kernel, extended with the KRR with two different σ_h. The gray value of pixel x depicts the value of the projection $\langle p_i, \varphi(x) \rangle_{\mathcal{H}}$. The ith row shows the projection over p_i with $i = 1, \ldots, 4$. On the left, small σ_h and on the right high σ_h.

Figure 3.1 shows a noisy sinusoidal dataset $X = \{x_1, \ldots, x_m\}$ together with the kernel PCA projection over the first four principal axes for the Diffusion

Maps kernel. For the extension we used the Kernel Ridge Regression with the Gaussian kernel, using two different values of σ_h. The images were generated as those in Figures 2.2 and 2.3, with the difference that those were for the Gaussian kernel and these are for the Diffusion Maps kernel. We used Eq. (3.29) to compute the kernel vector needed for the kernel PCA projection (2.26).

The images in the left column of the Figure were computed using $\sigma_h = \sigma_k \approx 0.52$. The σ_h on the right column was set to the average distance to the 30th nearest neighbor, which yielded the value $\sigma_h \approx 0.99$. Note that the smaller σ_h the projections go to zero faster because the kernel vector $\boldsymbol{k}_{X,x}$ vanishes. Recall that this kernel vector is a linear combination of Gaussian kernels. A higher σ_h allows to extend the kernel farther away from the dataset. We will refer to σ_h as the *extension scale parameter*.

The top images depict the projection over the first principal component. For points x far away from the training set, Points on the left part of the dataset have a negative projection that gradually increases while traversing the manifold towards the right. The projections over other principal components also vary along the manifold, showing more oscillations.

Recall that the kernel PCA projections are the coordinates of the kernel PCA representation of the mapping. The first coordinate of the mapping effectively varies following the underlying sinusoidal manifold. Note that the level curves, *i.e.* the set of points with the same first coordinate, intersect the manifold "orthogonally" to this manifold.

3.3.2 Choice of the parameters

Kernel Ridge Regression has two main parameters: the regularization parameter γ and the kernel function h. If we use the Gaussian kernel we have to determine its extension scale σ_h.

A common practice to choose the parameters of a learning algorithm, is to separate the training test, into two sets. Train the algorithm with one set, and test it over the other measuring the empirical risk. If this is done with several parameters, we can at the end pick the set of parameters with a smaller empirical risk. This is called *cross-validation*. A variation of the cross-validation is the n-folded cross-validation. The idea is to divide the training set randomly into n equal sets. For each set of parameters train the algorithm n times, each time leaving one of the n subsets out of the training, and computing the empirical error on the part that was left out. These errors are then averaged and the parameters with the smallest average errors are chosen.

Leave One Out Kernel Ridge Regression

If $n = m$ then we have *Leave One Out* (LOO) validation: the algorithm is trained m times, each time removing a different point from the training set. The error measured is the prediction of the functional value on the removed point. Denote $f^{(i)}$ the function obtained removing x_i from the training set. The LOO cross-validation error, known also as the *Predicted Residual Sum-of-Squares* (PRESS) [1], can be defined as:

$$P = \sum_{i=1}^{m} (y_i - f^{(i)}(x_i))^2 \tag{3.30}$$

review Notice that the computation of the PRESS involves solving m learning problems with a training set of $m-1$ points. However, for the Kernel Ridge Regression it can be computed in closed form, in terms of the auxiliary kernel matrix [18]:

$$P = \sum_{i=1}^{m} (y_i - f^{(i)}(x_i))^2 = \sum_{i=1}^{m} \left(\frac{\beta_i}{\boldsymbol{C}_{ii}^{-1}} \right)^2 \tag{3.31}$$

where

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{H} + \gamma \boldsymbol{I} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \tag{3.32}$$

Refer to [18] or the Appendix §C.2 for a proof.

The minimization of expression (3.31) allows to find the optimal parameters of the Kernel Ridge Regression w.r.t. the PRESS criterion. This optimization however is costly. If implemented with a steepest descent (see Appendix §C.2.2), each iteration involves a matrix inversion. Generally, the PRESS will be a nonlinear function of the regression parameters, and will suffer from local minima.

Figure 3.2 shows two local minima found by a steepest descent algorithm, for a simple regression problem. The data was generated by sampling a sinusoid and adding Gaussian noise to the sampled value. The initial parameters $\gamma_0, \sigma_{h,0}$ of both iterations where very close. The result on the right is better. However it has a larger PRESS. This shows that the PRESS does not guaranties a good generalization. The left result is a typical case of overfitting. The function has "learned" the noise. The value of σ_h, higher for the right Figure, is crucial to prevent this to happen.

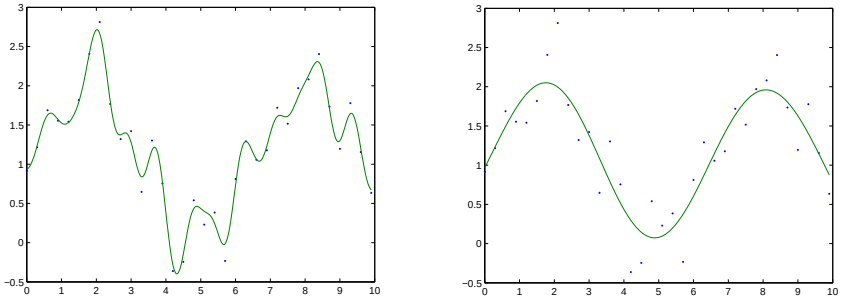

Figure 3.2: Results of the optimization. The PRESS value is 5.08 for the result shown on left and 5.62 for the one on the right.

Application to data driven kernels

Let us analyze the PRESS for our problem of extending data driven kernels. Suppose we have a set of extension parameters γ and σ_h for which we want to compute the PRESS in order to evaluate if they are appropriate.

We will denote by $\boldsymbol{K}_X$ the kernel matrix computed with the training set X. Consider the set $X^{(i)}$ obtained by removing the x_i vector from X. Let $\boldsymbol{K}^X_{X^{(i)}}$ be the $(m-1)\times(m-1)$ matrix obtained by removing the ith column and ith row from matrix $\boldsymbol{K}_X$. The notation $\boldsymbol{K}^X_{X^{(i)}}$ indicates that this matrix is the restriction to the reduced set $X^{(i)}$ of a kernel matrix computed with the dataset X.

We can compute the kernel vector at x_i, as

$$\boldsymbol{k}^{(i)}_{X,x_i} = [\boldsymbol{B}^X_{X^{(i)}}]^T \boldsymbol{h}^{(i)}_{x_i} + \boldsymbol{b}^X_{X^{(i)}} \tag{3.33}$$

where $\boldsymbol{h}^{(i)}_{x_i} = [h(x_1,x_i),\ldots,h(x_{i-1},x_i),h(x_{i+1},x_i),\ldots,h(x_m,x_i)]^T$, and the matrix $\boldsymbol{B}^X_{X^{(i)}}$ and vector $\boldsymbol{b}^X_{X^{(i)}}$ are computed according to Eq. (3.28) using $\boldsymbol{K}^X_{X^{(i)}}$ instead of $\boldsymbol{K}$.

This kernel vector is compared against the "true" kernel vector for x_i, obtained from the ith column of matrix $\boldsymbol{K}_X$. Not that the kernel vector $\boldsymbol{k}^{(i)}_{X,x_i}$ does not have the kernel value between x_i and itself, therefore we must exclude the ith component from the column $[\boldsymbol{K}_X]_{\cdot i}$. Let us denote the true kernel vector by

$$\boldsymbol{k}_{X,x_i} = \left[[\boldsymbol{K}_X]_{1i},\ldots,[\boldsymbol{K}_X]_{(i-1)i},[\boldsymbol{K}_X]_{(i+1)i},\ldots,[\boldsymbol{K}_X]_{mi}\right]^T \tag{3.34}$$

The PRESS error term corresponding to the removal of x_i is given by:

$$P^{(i)} = \frac{1}{m-1}\sum_{j=1}^{m-1}\left([\boldsymbol{k}_{X,x_i}]_j - [\boldsymbol{k}^{(i)}_{X,x_i}]_j\right)^2 \tag{3.35}$$

This expression corresponds to the mean square error between the components of both kernel vectors. The PRESS is obtained by averaging $P^{(i)}$ with $i=1,\ldots,m$:

$$\text{PRESS} = \frac{1}{m}\sum_{i=1}^{m} P^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{m-1}\sum_{j=1}^{m-1}\left([\boldsymbol{k}_{X,x_i}]_j - [\boldsymbol{k}^{(i)}_{X,x_i}]_j\right)^2\right] \tag{3.36}$$

Note however that the case of extending empirical kernels is not under the assumptions of the PRESS, because the values of the functions we want to extend depend on the whole training set.

When removing x_i from that training set, the whole kernel matrix will change. This is not considered in the PRESS error estimate: the reduced kernel matrix $\boldsymbol{K}^X_{X^{(i)}}$ is the restriction of the matrix $\boldsymbol{K}_X$ to the reduced set, assuming that the rest of the entries remain constant.

A more appropriate estimate of the extension error, can be obtained by recomputing the kernel matrix $\boldsymbol{K}_{X^{(i)}}$for the reduced set $X^{(i)}$. By plugging the recomputed kernel matrix into Eqs. (3.28) and (3.29) we obtain another kernel vector extension $\boldsymbol{k}_{X^{(i)},x_i}$. This vector, and not $\boldsymbol{k}^{(i)}_{X,x_i}$ is the one that we had computed if x_i was not part of the training set.

We will refer to this modification of the PRESS, as the *Recomputed Kernel* PRESS or RK-PRESS.

$$\text{RK-PRESS} = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{1}{m-1}\sum_{j=1}^{m-1}\left([\boldsymbol{k}_{X,x_i}]_j - [\boldsymbol{k}_{X^{(i)},x_i}]_j\right)^2\right] \tag{3.37}$$

Although the RK-PRESS is a more appropriate criterion for our problem, its computation is much heavier than the PRESS, since it does not have a closed form expression. Therefore we are interested in studying the differences between both approaches. We performed some experiments comparing both measurements which are presented in the Appendix §C.3.

However in practice we selected the value of the extension parameters according to other more intuitive considerations, explained in the following section.

Interpretation of the scale

The scale parameter σ_h in the KRR is related with the complexity of the function we want to extend, and will also determine the range of the extension. We use the term complexity in an informal way, referring to the variability of the function. For instance the left function in Figure 3.2 is more complex than the function in the right. A complex function will require a smaller σ_h and therefore a shorter extension domain. This makes sense: the more complex the function, the more uncertainty we will have about its value far from the training set.

The problem of manifold learning has naturally two scales. Dimensionality reduction algorithms usually gather local information. The low dimensional representation is found by "coordinating" the local information. This is evident for LLE and MVU. Another example of that, although not a spectral dimensionality reduction algorithm is the *Locally Linear Coordination* method [62]. Brand [14] presents similar idea and an interesting discussion for the choice of the size of the local neighborhoods. Diffusion Maps also works in this way. Its scale parameter σ_k defines the notion of locality. Note that a manifold is defined as a set that *locally* looks like an Euclidean space.

The σ_h parameter is related with the larger scale structure of the manifold. If the manifold has high curvature, σ_h should be low. On the other hand for a simpler manifold, a higher σ_h can be used, thus enlarging the range of the extension.

The columns of the Diffusion Maps matrix are complex functions to extend, even if the manifold is simple. Criteria such as the PRESS or the RK-PRESS would suggest to use a small extension scale σ_h ignoring the geometry of the manifold underlying the data.

The reason for this apparent divorce between the complexity of the kernel matrix and the geometry of the manifold is that some kernels encode much more information than what is needed to describe the manifold. An example of this is shown in Figure 3.3 comparing the decay of eigenvalues of two kernel matrices, MVU and Diffusion Maps, computed for the same dataset shown in Figure 3.1 (the training points are the cyan dots in the Figure). MVU has encoded most of the information in the first principal component. On the other hand, the slow decay of the spectrum of Diffusion Maps indicates a complex kernel matrix.

Figure 3.1 compares the kernel PCA projections of the Diffusion Maps kernels using two different extension scales. The value of σ_h in the right column was computed as the average distance from each point towards its 30th nearest neighbor. Such a large value for σ_h will have a high error on extending the kernel values according to the PRESS or RK-PRESS criteria. However this kernel PCA projections reflect geometric properties of the manifold.

These results show that altough according to the PRESS or the RK-PRESS criteria the measured error is high we are still able to compute a meaningful

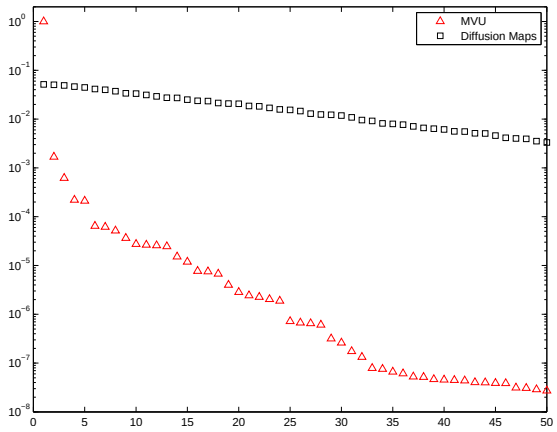

Figure 3.3: Comparison of the spectrum of two kernels for the same dataset. The eigenvalues shown were normalized dividing by the sum, showing the relative energy encoded in each principal component. Note that the first eigenvalue for MVU is allmost 1.

out-of-sample extension of the first kernel PCA projections. Projections over higher principal axes are more complex, therefore the error in the extension will increase for the higher principal axes.

This suggest that the Diffusion Maps kernel matrix encodes information that may not be needed. If this intuition is true a lower rank approximation of this matrix, discarding high eigenvalues may encode the necessary information. We expect the PRESS and RK-PRESS values for this lower rank matrix to better reflect the structure of the manifold. We did not verify this hypothesis, although we did use this intuitive idea without any problems arriving to the results presented in the next Chapter.

Chapter 4

The Pre-Image Problem in Kernel PCA

4.1 Introduction

In this Chapter we are going to discuss the pre-image problem. As said before, the main motivation for the pre-image is to estimate estimate the input point that corresponds to a new feature space point. A pre-image is needed in applications in which the output is a new data element. For instance if the kernel PCA map "unfolds" the manifold, we might perform an interpolation inside the manifold by linearly interpolating the feature space representatives and mapping back the result.

In particular we will apply the pre-image algorithm for projecting points into the underlying manifold. This can be used for *de-noising* noisy samples from the manifold. Besides its applications, the pre-image problem is also interesting as a way to study the geometry induced by the kernel in the input space.

The pre-image of $\psi \in \mathcal{H}$ (feature space point) is a point $x \in \mathcal{X} \subset \mathbb{R}^d$ such that $\varphi(x) = \psi$. However such a point x might not exist. The set of points that have an exact pre-image is given by

$$\varphi(\mathcal{X}) = \{\varphi(x) \ : \ x \in \mathcal{X}\} \tag{4.1}$$

Consider for example a two dimensional input space $\mathcal{X} \subset \mathbb{R}^2$, and a Gaussian kernel. In this case, since φ is a smooth mapping (a direct consequence of the smoothness of the kernel function) and $\mathcal{X}$ is a two dimensional set, $\varphi(\mathcal{X})$ is a two dimensional manifold, contained in the infinite dimensional space $\mathcal{H}$. That manifold is non-linear, as it is easy to see with the RKHS representation of the mapping. The set $\varphi(\mathcal{X})$ contains all the functions $k(\cdot, x)$ with $x \in \mathcal{X}$. Any linear combination of these elements will not be in $\varphi(\mathcal{X})$.

A way to circumvent this problem is to look for an approximate pre-image, *i.e.* a point $x \in \mathbb{R}^d$ such that $\varphi(x)$ is "as close as possible" to ψ. Different

optimality criteria could be used, such as

$$\text{Distance: } x = \arg\min_{x \in \mathcal{X}} \|\varphi(x) - \psi\|_{\mathcal{H}}^2 \tag{4.2}$$

$$\text{Collinearity: } x = \arg\max_{x \in \mathcal{X}} \left\langle \frac{\varphi(x)}{\|\varphi(x)\|_{\mathcal{H}}}, \frac{\psi}{\|\psi\|_{\mathcal{H}}} \right\rangle_{\mathcal{H}} \tag{4.3}$$

These are unconstrained optimizations. Such pre-images are going to be discussed in the next section and we will see that they present some problems. Section §4.3 describes a way to circumvent these problems by adding constraints to the optimization. Along this Chapter we will focus on the Gaussian kernel as an analytic kernel and the Diffusion Maps kernel as a data dependent kernel for which we will use the KRR extension framework. Among the kernels we tested these where the two representatives of both kernel classes that performed the best. Some derivations performed for the polynomial kernel are presented in Appendix D.

4.2 Unconstrained pre-image

Most of previous work in the pre-image problem in kernel methods was focused in solving problems (4.2) or (4.3) for analytic kernels. In particular the Gaussian kernel is the one that received most of the attention.

In this section we are going to review some of the approaches presented previously in the literature, as well as some novel work developed during this thesis.

4.2.1 Gaussian kernel

The Gaussian kernel belongs to the family of *Radial Basis Function* kernels given by

$$k(x, x') = \kappa(\|x - x'\|^2) \tag{4.4}$$

where $\kappa : \mathbb{R} \to \mathbb{R}$ is a function. For these kernels the collinearity and the distance criterion coincide:

$$\|\varphi(x) - \psi\|_{\mathcal{H}}^2 = \langle \varphi(x), \varphi(x) \rangle_{\mathcal{H}} + \langle \psi, \psi \rangle_{\mathcal{H}} - 2\langle \varphi(x), \psi \rangle_{\mathcal{H}} \tag{4.5}$$

$$= k(x, x) + \|\psi\|_{\mathcal{H}}^2 - 2\langle \varphi(x), \psi \rangle_{\mathcal{H}} \tag{4.6}$$

For Radial Basis Function kernels, $k(x, x) = \|\varphi(x)\|_{\mathcal{H}}$ is constant, thus minimizing the distance criterion is the equivalent to maximizing the collinearity.

Collinearity and distance criterion

Iterative methods. Mika *et al.* [47], present an analytic solution for the Gaussian kernel. First, they assume that $\psi \in \mathcal{H}^m$, and therefore is a linear combination of the mappings of the training set. Denoting by $\boldsymbol{\alpha}_\psi$ the linear combination coefficients vector, we have that

$$\psi = \sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \varphi(x_i) \tag{4.7}$$

As we saw before the cost function in (4.3) is equivalent to

$$\langle \varphi(x), \psi \rangle_{\mathcal{H}} = \left\langle \varphi(x), \sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \varphi(x_i) \right\rangle_{\mathcal{H}} = \sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i k(x, x_i) \tag{4.8}$$

where we used the dot product property (2.1). The maximum can be found by taking the gradient, leading to the following expression for the optimal x:

$$x = \frac{\sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i k(x, x_i) x_i}{\sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i k(x, x_i)} = \frac{\sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \exp(-\|x - x_i\|^2 / 2\sigma^2) x_i}{\sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \exp(-\|x - x_i\|^2 / 2\sigma^2)} \tag{4.9}$$

This implicit equation can be solved by a fixed point iteration, but suffers from local minima and instabilities [26, 47].

Closed form approximations. In [26] an approximation to avoid the iteration is proposed. The distance between the mapped points can be computed in terms of the dot products, and therefore in terms of the kernel,

$$\|\varphi(x) - \varphi(x')\|^2_{\mathcal{H}} = k(x, x) + k(x', x') - 2k(x, x') \tag{4.10}$$

Since the Gaussian kernel is normalized, we obtain:

$$\|\varphi(x) - \varphi(x')\|^2_{\mathcal{H}} = 2(1 - k(x, x')) \tag{4.11}$$

This equation only holds for points in $\mathcal{H}$ that have an exact pre-image, *i.e.* for the points that belong to the image of the mapping $\varphi(\mathcal{X})$. However, in [26] the authors make the assumption that ψ lies close to the manifold $\varphi(\mathcal{X})$, and therefore it exists one x such that $\psi \approx \varphi(x)$. This is then used to estimate the kernel values between x and the training points:

$$\hat{k}(x, x_i) = \tfrac{1}{2}(2 - \|\psi - \varphi(x_i)\|^2_{\mathcal{H}}) \qquad i = 1, \ldots, m \tag{4.12}$$

Substituting this approximation in the iterative equation (4.9) leads to a direct formula.

In [3, 2] we proposed a modification of the direct formula approximation of [26] using the kernel estimate given by [43]. In [43], the authors use a similar approach to estimate the kernel vector $\boldsymbol{k}_x$ on the pre-image x. However they do not assume that $\|\psi\|_{\mathcal{H}} = 1$. This norm can be computed in terms of the kernel, assuming that $\psi = \sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \varphi(x_i)$:

$$\|\psi\|^2_{\mathcal{H}} = \left\langle \sum_{i=1}^{m} [\boldsymbol{\alpha}_\psi]_i \varphi(x_i), \sum_{j=1}^{m} [\boldsymbol{\alpha}_\psi]_j \varphi(x_j) \right\rangle_{\mathcal{H}} = \boldsymbol{\alpha}_\psi^T \boldsymbol{K} \boldsymbol{\alpha}_\psi \tag{4.13}$$

Therefore their kernel estimate between the pre-image x and x_i is given by:

$$\hat{k}(x, x_i) = \tfrac{1}{2}(1 + \|\psi\|^2_{\mathcal{H}} - \|\psi - \varphi(x_i)\|^2_{\mathcal{H}}) = \langle \psi, \varphi(x_i) \rangle_{\mathcal{H}} \tag{4.14}$$

Note that this is the kernel vector representation of ψ: $\boldsymbol{k}_\psi$ (recall §2.4.2). In other words, it is ψ itself expressed on a different basis of the $\mathcal{H}^m$. The relation between this estimate for $\boldsymbol{k}_x$ and the one in Eq. (4.12) is the following.

$$\begin{aligned}\hat{\boldsymbol{k}}_x(i) &= \tfrac{1}{2}(2 - \|\varphi(x_i) - \psi\|^2_{\mathcal{H}}) \\ &= \tfrac{1}{2}(2 - 1 - \langle\psi,\psi\rangle_{\mathcal{H}} + 2\langle\varphi(x_i),\psi\rangle_{\mathcal{H}}) \\ &= \tfrac{1}{2}(1 - \langle\psi,\psi\rangle_{\mathcal{H}}) + \|\psi\|_{\mathcal{H}}\boldsymbol{k}_\psi(i) \end{aligned} \tag{4.15}$$

In the case that $\varphi(x) \approx \psi$, then $\|\psi\|_{\mathcal{H}} \approx 1$, and the difference between both approximations will be small.

Discussion: Kernel vector estimates

Both for the Gaussian and the polynomial kernel (see Appendix D) we used the estimate $\boldsymbol{k}_x \approx \boldsymbol{k}_\psi$ to derive closed form approximations to the solution of the iterative pre-images. We have already noticed that $\boldsymbol{k}_\psi$ is the kernel vector representative of ψ. In [3] we provided an alternative interpretation. In this section we are going to discuss both interpretations and determine the conditions under which the estimate is valid.

We are going to start by reviewing the justification we gave in [3]. Approximating $\varphi(x)$ by the empirical kernel map $\varphi_m(x)$ yields:

$$\|\varphi(x) - \psi\|^2_{\mathcal{H}} \approx \|\varphi_m(x) - \psi\|^2_{\mathcal{H}} = \|\boldsymbol{y}_x - \boldsymbol{y}_\psi\|^2 \tag{4.16}$$

In the last equality we have just substituted $\varphi_m(x)$ and ψ by their kernel PCA representations. We can express $\boldsymbol{y}_x$ in terms of the kernel vector $\boldsymbol{k}_x$ using Eq. (2.27), obtaining

$$\|\varphi_m(x) - \psi\|^2_{\mathcal{H}} = \|\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{k}_x - \boldsymbol{y}_\psi\|^2 \tag{4.17}$$

If we minimize this expression w.r.t. the vector $\boldsymbol{k}_x$ we get the following optimal kernel vector:

$$\begin{aligned}\boldsymbol{k}^*_x &= \boldsymbol{U}\boldsymbol{\Lambda}^{1/2}\boldsymbol{y}_\psi = \boldsymbol{Y}(X)^T\boldsymbol{y}_\psi \\ &= [\langle\varphi(x_1),\psi\rangle_{\mathcal{H}},\ldots,\langle\varphi(x_m),\psi\rangle_{\mathcal{H}}]^T = \boldsymbol{k}_\psi \end{aligned} \tag{4.18}$$

where $\boldsymbol{Y}(X)$ is the matrix whose columns are the kernel PCA mappings of the training set points $\boldsymbol{y}_{x_i}$, with $i = 1,\ldots,m$ (see Eq. (2.29)). Recall that with this representation, the feature space dot product is the Euclidean dot product: $\langle\varphi(x_i),\psi\rangle_{\mathcal{H}} = \boldsymbol{y}_{x_i}{}^T\boldsymbol{y}_\psi$. Therefore $\boldsymbol{k}_\psi$ is the kernel vector that minimizes expression (4.17).

It should be noted that this fact can not be interpreted easily. Suppose that instead of minimizing (4.17) w.r.t. the kernel vector we minimize it w.r.t. x and then compute the kernel vector for the optimal x^*, $\boldsymbol{k}_{x^*}$. This solution will be different to $\boldsymbol{k}_\psi$.

A simpler justification for the estimate $\boldsymbol{k}_x \approx \boldsymbol{k}_\psi$ is that it comes from a change of basis: $\boldsymbol{k}_\psi$ and $\boldsymbol{y}_\psi$ are just representations of ψ, in two different basis. We now prefer this explanation over the one we gave in [3].

Using the latter, we can easily provide a geometric interpretation of the assumption $\boldsymbol{k}_x \approx \boldsymbol{k}_\psi$. It is equivalent to assuming:

$$\varphi_m(x) = \mathbb{P}_{\mathcal{H}^m}(\varphi(x)) \approx \psi \tag{4.19}$$

The kernel vector estimate will be valid when $\varphi_m(x)$ is close to ψ. However, as we saw in §2.5, when m is large enough, $\varphi_m(x)$ is an approximation of $\varphi(x)$. Thus assuming the validity of the estimate is equivalent to assuming that $\psi \approx \varphi(x)$, or equivalently that ψ has an exact pre-image ($\psi \in \varphi(\mathcal{X})$).

Other criteria

Kwok and Tsang [43] use the kernel vector estimate to obtain the distance in the input space between the searched pre-image x and the n nearest neighbors of ψ in the Feature Space. They do this by inverting the Gaussian kernel expression (2.6):

$$[\boldsymbol{k}_\psi]_i \approx [\boldsymbol{k}_x]_i = \exp\left(-\frac{1}{2\sigma^2}\|x - x_i\|^2\right) \qquad i = 1, \ldots, m \tag{4.20}$$

then

$$\|x - x_i\|^2 \approx 2\sigma^2 \log([\boldsymbol{k}_\psi]_i) = 2\sigma^2 \log(\langle \psi, \varphi(x_i)\rangle_{\mathcal{H}}) \qquad i = 1, \ldots, m \tag{4.21}$$

Finding x now reduces to a localization problem solved by standard MDS [32, 23]. For locating x they use only the distances the input points corresponding to n nearest neighbors of ψ in the Feature Space. The number of nearest neighbors is a parameter of the algorithm. Note that this approach is not based in any of the two optimality criteria mentioned above.

This idea can also be applied for the polynomial and sigmoid kernels. In those cases, instead of locating x using the distances, they use the dot products between x and the points in the training set.

Bakır *et al.* [5] use Kernel Ridge Regression to learn a mapping from the kernel PCA representation of $\mathcal{H}^m$ to the input space $\mathcal{X}$. Their pre-image can be applied to any kernel, including data dependent kernels. This approach follows the inverse path that most of the approaches follow.

In fact, leaning the kernel with Kernel Ridge Regression, as we do for data dependent kernels, is a way of learning the mapping: Recall from §2.4.2 that the kernel vector $\boldsymbol{k}_x$ (or its projection over the range of the kernel matrix) serves as a representation of the empirical mapping. Thus, while we learn the mapping Bakır *et al.* learn directly its inverse. The main advantages of leaning the mapping will become apparent in §4.3, since it will allow us to controll the kernel principal components of the pre-image. On the other hand the advantages of learning the inverse directly, is that they can apply it to non-Euclidean input spaces, as proposed in [6].

In [74] the authors propose an LLE-type pre-image. Basically they express ψ as a convex linear combination of its neighbors, and used the same coefficients to interpolate the pre-image in the input space. In §4.2.2 we are going to discuss a similar approach. The same authors propose in [75] a variation of the feature space distance optimization in a *weakly supervised setting* by adding information about the input patterns that the pre-image should be close to and far away from.

Results

In this section we are going to show some results for the pre-images of the Gaussian kernel. We are going to use synthetic datasets of two different na-

ture: Samples from a smooth manifold, and from a clustered dataset, both in a (relatively) high dimension ($d = 10$) and a low dimension input space ($d = 2$).

Manifold dataset. In the high dimensional setting, $m = 800$ points are sampled from a uniform distribution in the $d-1$ dimensional cube of side 2π. The 10th dimension is defined as

$$[x_i]_{10} = \sin\left(\sum_{j=1}^{d-1}[x_i]_j\right) \tag{4.22}$$

Gaussian noise with variance $\sigma_n = 0.2$ is added in all the components. The same idea for the low dimensional dataset, with $d = 2$: $m = 300$ points $[x_i]_1$ are sampled from a uniform distribution in $[0, 2\pi]$, the second component is defined as $[x_i]_2 = \sin([x_i]_1)$, noise with variance $\sigma_n = 0.4$ is added.

Clustered Dataset. The dataset consist of ten clusters with Gaussian distributions of fixed variance and random centers. The number of points per cluster is constant, 40 and 80 in the low and high dimensional setting respectively.

The scale parameter of the Gaussian kernel will be set as the mean squared distance from a point to its 10th nearest neighbor. Denoting by $x_{i,10}$ the 10th nearest neighbor of x_i,

$$\sigma = \frac{1}{m}\sum_{i=1}^{m}\|x_i - x_{i,10}\|^2 \tag{4.23}$$

To compare the pre-image algorithms we are going to compute, for each point in the dataset its kernel PCA projection over the q principal components in the feature space: $\mathbb{P}_q\varphi(x_i)$, with $q = 1$ and $q = 10$. The pre-images of these projections is computed. This is an arbitrary choice: All we need is to have new points in the feature space to compute their pre-image. However, computing a pre-image of the kernel PCA projection is one of the most common applications. Let us denote by

$$k\mathbb{P}_q(x) = \varphi^{-1}(\mathbb{P}_q\varphi(x)) \tag{4.24}$$

where φ^{-1} will depend on the pre-image algorithm. We are going to compare five methods:

A. Pre-image presented in [3]. Modification of the direct formula approximation used in [26] with the kernel vector estimate proposed in [43].

B. Pre-image by distance based localization in the input space [43]. The number of nearest neighbors was set to $n = 10$.

C. Direct formula approximation of [26].

D. Fixed point iterative pre-image [47].

E. Steepest descent of distance criterion (see 4.2.2).

We are going to perform a numerical and a visual (in the low dimensional setting) comparison. For the numerical comparison we will use the distance criterion (4.2):

$$e_d(x) = \|\varphi(k\mathbb{P}_q(x)) - \mathbb{P}_q\varphi(x)\|^2_{\mathcal{H}} \tag{4.25}$$

computed over the whole training set: $e_d(x_i)$ with $i = 1, \ldots, m$. Thus for a given dataset, we will have m error measures for each method. Instead of comparing the methods using a single number (as could be the average distance error) we are going to compare the distributions of the errors $e_d(x_i)$, by means of the error histograms. The iterative approaches D and E are initialized with the corresponding x_i.

Low Dimensional Manifold Dataset. Figure 4.1 shows the error histograms for the five methods with $q = 1$ and $q = 10$ principal componets (left and right respectively). Each Figure shows the supperposition of the five histograms, identified by the color legend. The numbers in the horizontal axis indicate the centers of the bins[1]. For example, in the left figure, the first bin is centered at $e_d = 0.54$. This bin counts the number of times that the feature space distance falls between $e_d = 0.51$ and $e_d = 0.57$. Each histogram has 10 bins.

For each bin, five bars are displayed, each one corresponding to one of the five methods compared, according to the legend. For instance, methods B, D and E in the left Figure had a e_d error between $e_d = 0.51$ and $e_d = 0.57$ around 40 times from a total of $m = 300$ trials. Pre-images A and C did not have any distance error in that range.

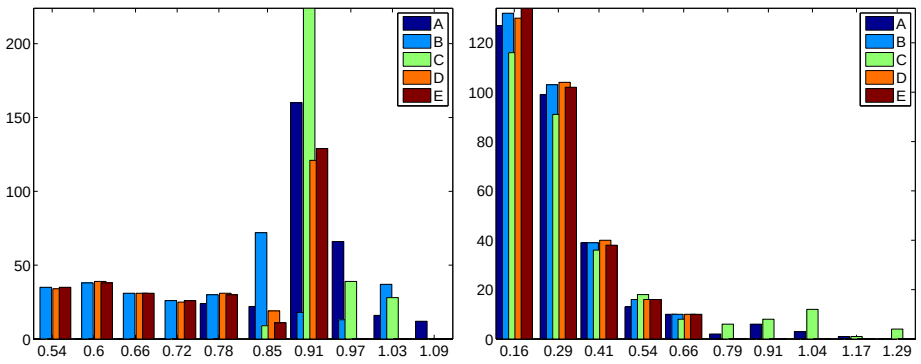

Figure 4.1: Comparison of the five pre-image algorithms for the Gaussian kernel applied to kernel PCA projections with manifold data in low dimensions. Histogram of the feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

As we can see, the behavior of the algorithms depends heavily on the number of components q. With $q = 1$ all the methods show higher e_d values. Besides, the performance with $q = 10$ is much more homogeneous between different methods:

[1] The bin centers are equidistant. Sometime the distance between adjacent centers do not coincide: For instance in the left of Figure 4.1, the distance between the 6th and the 5th bins is 0.7, whereas for the others it is 0.6. The reason for this is that the bin centers are rounded to fit them in the plot.

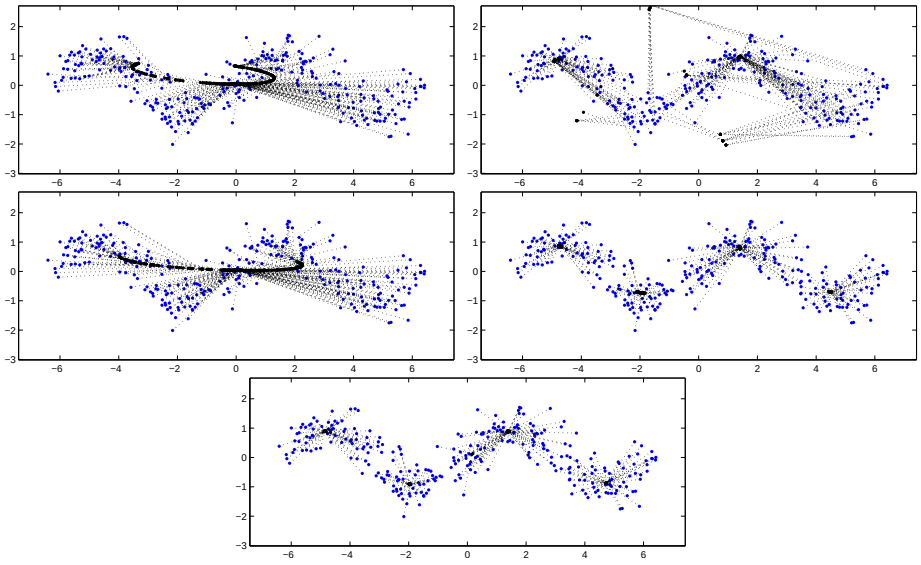

Figure 4.2: Pre-image results for the 2 dimensional manifold data with $q = 1$. Top: A and B, medium C and D and bottom E. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$

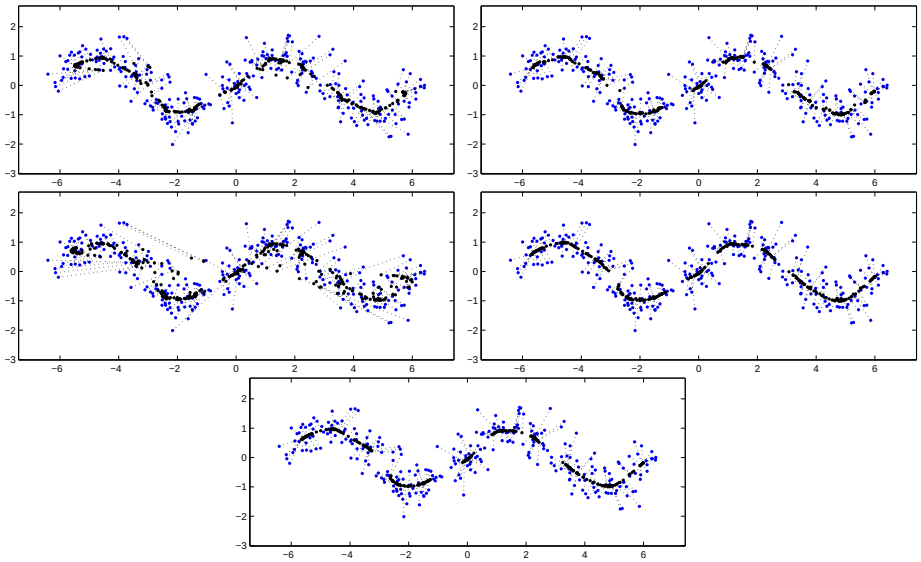

Figure 4.3: Pre-image results for the 2 dimensional manifold data with $q = 10$. Top: A and B, medium C and D and bottom E. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$

With $q = 1$ algorithms A and B yield much worse results. Both phenomena have the same cause.

The feature space projection $\mathbb{P}_q\varphi(x_i)$ will generally lie outside $\varphi(\mathcal{X})$. This means that $\mathbb{P}_q\varphi(x_i)$ will not have an exact pre-image. The best we can do (in terms of the distance criterion), is to find the point x such that $\varphi(x)$ is as close as possible to $\mathbb{P}_q\varphi(x_i)$. Iterative approaches, as well as their direct formula approximations, aim to minimize this distance. If we do not get trapped in any local minima, $\varphi(x)$ will be the point on $\varphi(\mathcal{X})$ closest to $\mathbb{P}_q\varphi(x_i)$.

Naturally, $\mathbb{P}_{10}\varphi(x_i)$ will lie closer to the manifold than $\mathbb{P}_1\varphi(x_i)$. This explains why does the error distribution shifts when the number of principal components increases. The kernel PCA projections are closer to $\varphi(\mathcal{X})$. Regarding the iterative methods D and E, it should not be concluded that they give worse results when $q = 1$. In both cases they yield the best possible results, since by design they minimize the error criterion.

Different is the situation for the closed form approximations (algorithms A and C). These algorithms assume that $\mathbb{P}_q\varphi(x)$ lies close to $\varphi(\mathcal{X})$. This is not true for $q = 1$, and that is the reason for which they are outperformed by the algorithms D and E. When $q = 10$, the approximation $\mathbb{P}_q\varphi(x) \approx \varphi(x)$ is valid and their performance is equivalent to the iterative approaches' one. Pre-image B is based on the same assumptions, yet it yields very good results even with $q = 1$.

Figures 4.2 and 4.3 show the pre-images found for $q = 1$ and $q = 10$. We can confirm that pre-images B, D and E yield similar results both with $q = 1$ and $q = 10$. Pre-images A and C are now closer to the iterative pre-images. Pre-image C shows some errors.

High Dimensional Manifold Dataset. Figure 4.4 show the error histograms for the 10 dimensional sinusoidal manifold.

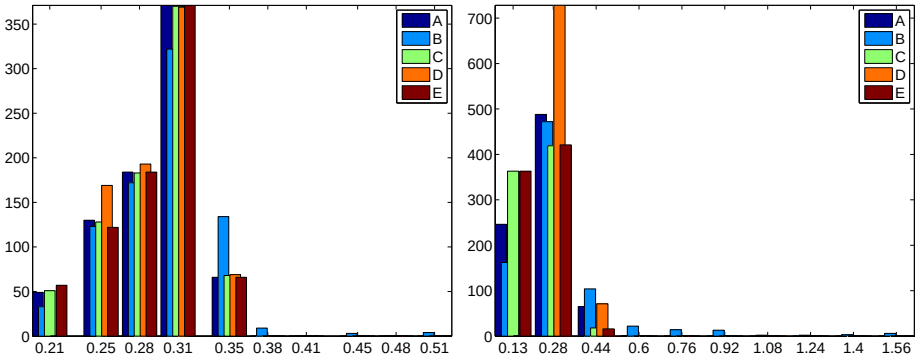

Figure 4.4: Comparison of the five pre-image algorithms for the Gaussian kernel applied to kernel PCA projections with manifold data in high dimensions. Histogram of the feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

In this case, the behavior of the error distributions with $q = 1$ and $q = 10$ principal components is very similar. The decrease in the errors for $q = 10$ is much less important than in the low dimensional setting. A possible reason for this because of the way the σ is elected, Eq. (4.23). The bigger the scale parameter, the simpler the mapping: A smaller σ will define more localized

kernel functions. Thus points relatively close in the input space will be almost orthogonal in the feature space, increasing the dimensionality of the spaced spanned by the data ($\varphi(\mathcal{X})$) as well as its curvature.

When the dimension of the input set increases, the points are farther appart one from each other, and the distance to the 10th nearest neighbor is larger related to the size of the whole set. Thus the mapping becomes simpler. Intuitively, the simpler the mapping, the closer we are from the hypothesis of the direct formula approximations: $\mathbb{P}_q\varphi(x_i)$ will lie closer to $\varphi(\mathcal{X})$. Another consequence of this is that pre-images A and C perform similarly to the iterative pre-images.

Notice that the pre-image B has long tails that indicate some larger errors. Recall that this algorithm determines the pre-image from its distance to n points, chosen as the nearest neighbors of $\mathbb{P}_q\varphi(x)$. When the dimension increases, more distance constraints are needed.

The fixed point iteration pre-image never gets to the bin of smaller errors. This may be corrected with a better stopping condition.

Low Dimensional Clustered Dataset. Figure 4.5 shows the histograms of the error for the pre-image methods in the clustered dataset. Figures 4.6 and 4.7 show the computed pre-images for $q = 1$ and $q = 10$ principal components.

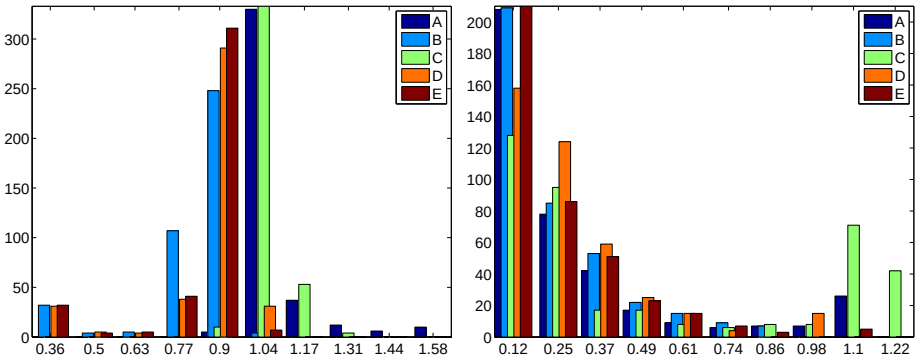

Figure 4.5: Comparison of the five pre-image algorithms for the Gaussian kernel applied to kernel PCA projections with cluster data in low dimensions. Histogram of the feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

The histograms of the error behave similarly than in the case of low dimensional manifold data, except that in this case, method B outperforms the iterative methods with $q = 1$. It doubles the number of errors in the bin centered at 0.77 and has less errors in higher value bins. However, the results seem worse in Figure 4.6, in the center of the top row. The reason for this contradiction is that the distance criterion has several local minima centered in each cluster, where the iterative approaches get trapped, and prevent the pre-image of the kernel PCA projection of a point in one cluster to be in a different one. Pre-image B instead is closer to the global minimum.

With 10 principal components the feature space distances are overall lower, and the direct approximations A and C behave similarly to the iterative pre-

images, although with some errors that generate the high tails (approximately 20 for A and 100 for C). These can also be seen on the left and right plots at the top row of Figure 4.7.

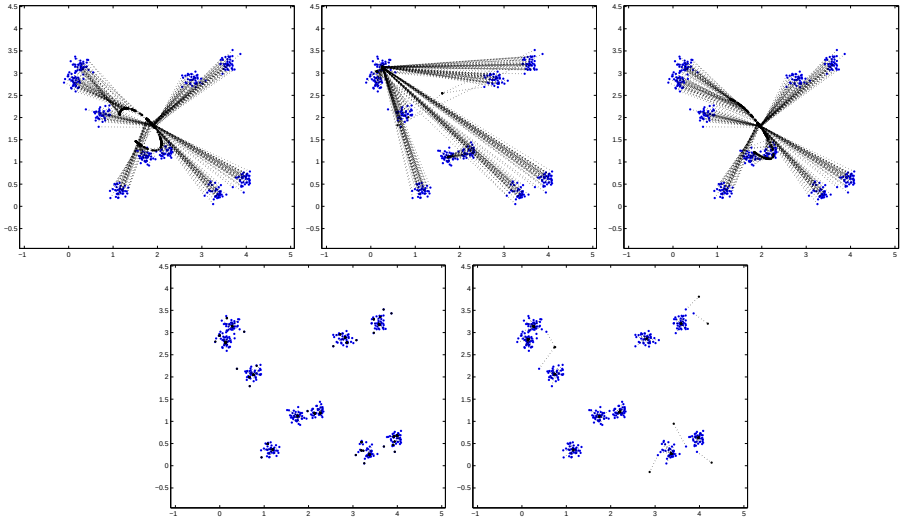

Figure 4.6: Pre-image results for the 2 dimensional cluster data with $q = 1$. Top: A and B, medium C and D and bottom E. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$

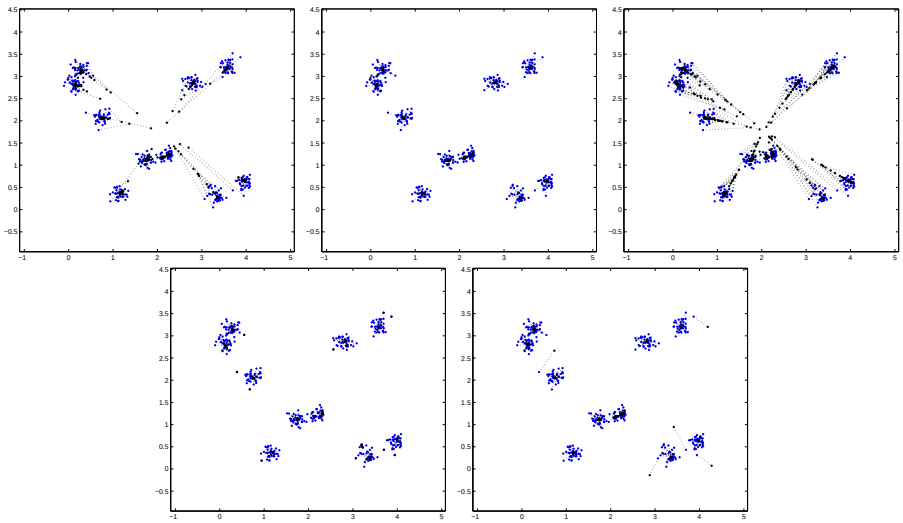

Figure 4.7: Pre-image results for the 2 dimensional cluster data with $q = 10$. Top: A and B, medium C and D and bottom E. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$

Notice also that pre-image E drags the points to the center of the cluster,

when $q = 1$. This happens also with $q = 10$. It also appears a small curve in the two clusters located at the center of the bottom right plot of Figure 4.7.

High Dimensional Clustered Dataset. The results for the pre-image of kernel PCA for a 10 clusters dataset in $\mathbb{R}^{10}$ can be seen in Figure 4.8 for $q = 1$ and $q = 10$ principal components. In this case there is not a substantial change in the error ranges w.r.t. the low dimensional setting, as opposed to what happens between the low and high dimensional versions of the manifold dataset. For this dataset we argued that the differences between the different dimensions were due to the way in which the scale parameter σ is chosen. In that case, the scale in the high dimensional setting was larger compared with the size of the dataset, and therefore the mapping was simpler. This is not the case for the clustered dataset.

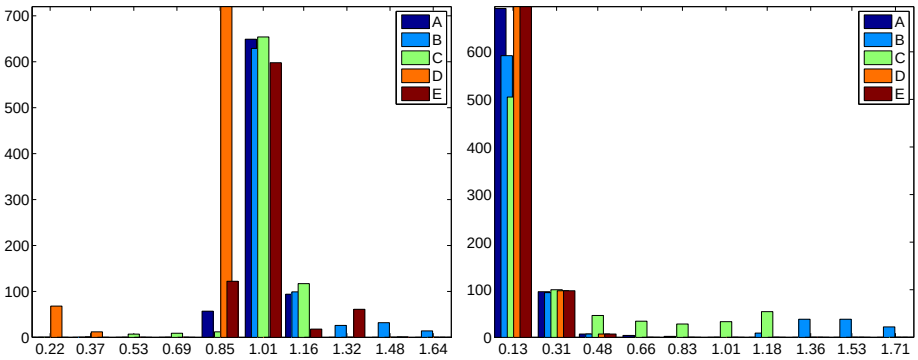

Figure 4.8: Comparison of the five pre-image algorithms for the Gaussian kernel applied to kernel PCA projections with cluster data in high dimensions. Histogram of the feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

4.2.2 Data dependent kernels

Data dependent kernels do not have an analytic expression and their values are known only at the training set X, through the kernel matrix $\boldsymbol{K}$. A discussed in Chapter 3, we are going to use a regression method to extend the empirical kernel matrix. In particular kernel regression methods define an extension of the kernel matrix as a linear combination of an auxiliary kernel h. Thus for extending the ith column of $\boldsymbol{K}$, we have

$$k_X(x, x_i) = \sum_{j=1}^{m} [\boldsymbol{\beta}_i]_j h(x, x_j) + b_i \tag{4.26}$$

where $\boldsymbol{\beta}_i$ and b_i are the coefficients of the kernel expansion model. All kernel regression methods share this type of solution. In particular we used Kernel Ridge Regression to compute the coefficients $\boldsymbol{\beta}_i$ and b_i, but any other kernel method for regression could be used, just as Kernel Lasso Regression or Support Vector Regression.

Recall from §3.3 that we could express the data dependent kernel vector $\boldsymbol{k}_{X,x}$ as

$$\boldsymbol{k}_{X,x} = \boldsymbol{B}^T \boldsymbol{h}_x + \boldsymbol{b} \tag{4.27}$$

where $\boldsymbol{B}_{ij} = [\boldsymbol{\beta}_i]_j$, $\boldsymbol{b} = [b_1, \ldots, b_m]^T$ and $\boldsymbol{h}_x$ is the auxiliary kernel vector given. The ith column of $\boldsymbol{B}$ is the coefficients vector of the kernel expansion for extending $k_X(\cdot, x_i)$.

For the auxiliary kernel h we are going to consider a Gaussian kernel $h(x, x') = \exp(\|x - x'\|^2/2\sigma_h^2)$. The value of σ_h, the extension scale parameter, will be set manually following the criterion mentioned at the end of §3.3.

Feature space distance minimization

We will assume, as before, that $\psi \in \mathcal{H}^m$. For computing a pre-image for the mapping of a data dependent kernel, we are going to minimize the following expression:

$$D_m(x) = \|\varphi_m(x) - \psi\|^2_{\mathcal{H}} \tag{4.28}$$

This is an approximation to Eq. (4.2). When working with analytic known kernels we could compute the true distance by means of the kernel trick as:

$$\begin{aligned} D(x) &= \|\varphi(x) - \psi\|^2_{\mathcal{H}} = \langle \varphi(x) - \psi, \varphi(x) - \psi \rangle_{\mathcal{H}} \\ &= k(x, x) + \|\psi\|^2_{\psi} - 2\sum_{i=1}^{m} \alpha_i k(x, x_i) \end{aligned} \tag{4.29}$$

However with data dependent kernels, using the kernel expansion framework we can estimate $k_X(x, x_i)$ with $i = 1, \ldots, m$ but we do not know $k_X(x, x)$. We will restrict our analysis to the empirical kernel feature space $\mathcal{H}^m$, since we do not need $k_X(x, x)$ to work in $\mathcal{H}^m$.

We have three possible representations of the empirical mapping that can used to evaluate $D_m(x)$. Using the kernel vector representation yields:

$$D_m(x) = \|\varphi_m(x) - \psi\|^2_{\mathcal{H}} = (\boldsymbol{k}_{X,x} - \boldsymbol{k}_{\psi})^T \boldsymbol{K}^+ (\boldsymbol{k}_{X,x} - \boldsymbol{k}_{\psi}) \tag{4.30}$$

The gradient of this expression is given by

$$\nabla D_m(x) = \frac{2}{\sigma_h^2} (\boldsymbol{k}_{X,x} - \boldsymbol{k}_{\psi})^T \boldsymbol{K}^+ \boldsymbol{B}^T \mathrm{Diag}(\boldsymbol{h}_x)(\boldsymbol{X} - x\boldsymbol{1}_{1m})^T \tag{4.31}$$

where $\mathrm{Diag}(h_x)$ is a diagonal matrix whose diagonal is the vector h_x, $\boldsymbol{1}_{1m}$ is a $1 \times m$ matrix full of ones and $\boldsymbol{X}$ is a matrix which ith column is x_i. The matrix $(\boldsymbol{X} - x\boldsymbol{1}_{1m})^T$ has in its ith row $(x_i - x)^T$.

Using this gradient in the steepest descent algorithm for the feature distance function we can find pre-images of the data dependent kernels.

Note that this pre-image can also be applied to the Gaussian kernel. In fact, the Gaussian kernel is a particular case of this framework with $\boldsymbol{B} = \boldsymbol{I}$, $\boldsymbol{b} = \boldsymbol{0}$ and $\sigma_h = \sigma$. This is in fact the pre-image E (see §4.2.1).

Linear combination pre-image

We can also apply the LLE pre-image to any data dependent kernel. The idea of this method is to express ψ as a convex linear combination of its n nearest

neighbors in the feature space:

$$\psi \approx \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \tag{4.32}$$

where $\eta_\psi \subset \{1, \ldots, m\}$ is the set of the indices of the nearest neighbors of ψ. Assuming that $\psi = \sum_{i=1}^{m} \alpha_i \varphi(x_i)$ we have to solve the following problem:

$$\begin{aligned} \boldsymbol{\beta} &= \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^n} \left\| \psi - \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \right\|_{\mathcal{H}}^2 \\ &= \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^n} \left\| \sum_{j=1}^{m} \alpha_i \varphi(x_i) - \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \right\|_{\mathcal{H}}^2 \qquad (4.33) \\ &\text{subject to } \mathbf{1}_{n1}^T \boldsymbol{\beta} = 1 \qquad (4.34) \end{aligned}$$

The constraint imposes that $\boldsymbol{\beta}$ sums to one. This problem has a closed form solution, see Appendix E for details.

Results

We are going to focus on the Diffusion Maps kernel, for this is a data dependent kernel which has no formal approach to its pre-image problem. As with the Gaussian kernel, we are going to test both pre-image algorithms for manifold and clustered data, in low and high dimensions also varying the number of kernel principal components used.

The scale parameter of the Diffusion Maps kernel σ_k will be set as the average distance from a point to its 5th nearest neighbor. The α parameter and the kernel matrix exponent t were both set to 1.

The KRR parameters were set to $\sigma_h = \sum_{i=1}^{m} \|x_i - x_{i,30}\|$ and $\gamma = 0.01$. Note that the value of σ_h is larger than σ_k, following the ideas presented in §3.3.2.

We are going to refer to the pre-images with the following code:

SD. Steepest descent algorithm for the minimization of the distance criterion (4.28).

LC. Convex linear combination pre-image of problem (4.33) and (4.34).

As we did in the last section, we are going to compare the pre-images numerically and visually. The experiments are basically the same as before: we project each training point x_i onto the space spanned by the q kernel principal axes, $\mathbb{P}_q \varphi(x_i)$, and compute the pre-image of this projection. The initial point for the *SD* pre-image is x_i itself. The numerical comparison is done through the inspection of the histograms of the empirical feature space distance errors D_m.

Manifold Dataset. The feature space distance histograms are shown in Figure 4.9. We can see that with $q = 1$ the *SD* pre-image has a better performance, which can also be appreciated in Figure 4.10. The reason for this is very interesting. Neighbors of $\mathbb{P}_1 \varphi(x_i)$ in the feature space may lie in distant parts of the input space dataset. The linear combination of those neighbors will lie close to

$\mathbb{P}_1\varphi(x_i)$ in the feature space. However, the mapping is highly non-linear and the same linear combination in the input space, when mapped back again to the feature space may be far away from $\mathbb{P}_1\varphi(x_i)$. This is related with the high non-linearities of the mapping, something we already discussed in §3.3.2. With $q = 10$ the results of both pre-images are very similar. As for the Gaussian kernel (recall Figure 4.3), it takes a high number of principal components to discover the sinusoidal manifold underlying the samples.

The histograms for the high dimensional data show that contrary to the low dimensional case, with $q = 1$ principal components the *LC* pre-image outperforms the iterative approach. The reason for this will be explained later is §4.3. They main idea is that since relative size of the σ_k is larger the mapping is simpler (recall the discussion for results of the Gaussian kernel with manifold data). This can be seen noting that the range of the distance errors is overall much smaller with $d = 10$: the projection $\mathbb{P}_q(x)$ is closer to the manifold $\varphi(\mathcal{X})$, similarly to what happened with the Gaussian kernel. In the low dimensional case this method failed because the neighbors of the projection $\mathbb{P}_1(x)$ may be in different locations of the manifold. This does not happen with a simpler mapping.

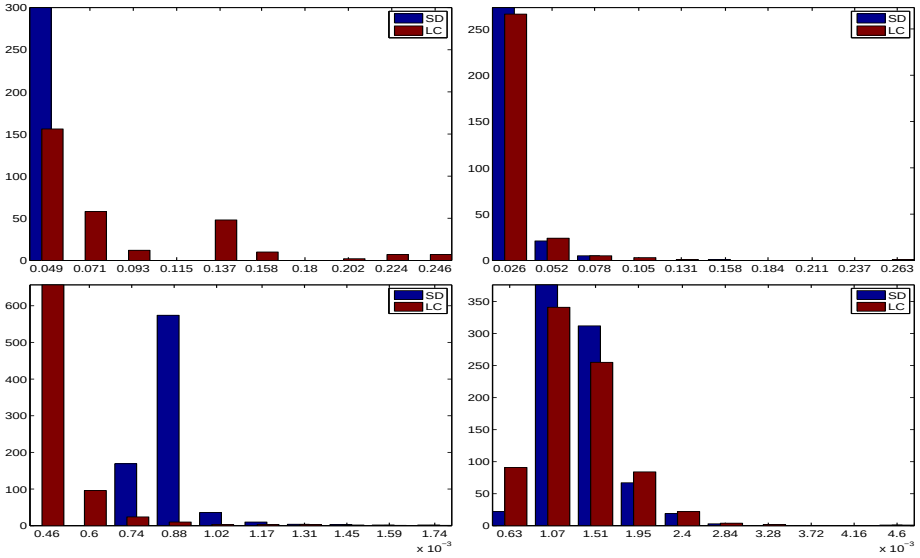

Figure 4.9: Comparison of the two pre-image algorithms for the Diffusion Maps kernel applied to kernel PCA projections with manifold data in low (top) and high (bottom) dimensions. Histogram of the empirical feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

Clustered Dataset. The feature space distance histograms are shown in Figure 4.11. As with the manifold data, the results in the low dimensional setting show that with only one principal component the *LC* pre-image performs badly. Again, the reason is that the neighbors of $\mathbb{P}_1\varphi(x_i)$ may lie in different clusters. This can also be seen in the top right plot of Figure 4.12. In high dimensions,

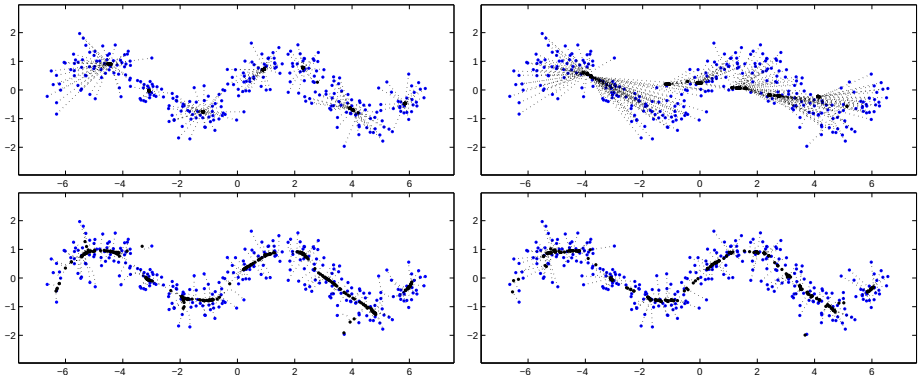

Figure 4.10: Pre-image results for the 2 dimensional manifold data. Left: SD and right LC. Top row: Results with $q = 1$, bottom with $q = 10$. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$

with $q = 1$, the LC method outperforms notoriously the SD pre-image.

When $q = 10$ both methods perform almost the same in terms of the feature space distance measure, although there are some differences looking at the bottom row in Figure 4.11. The most significant difference is with the twin clusters in the upper right corner of the figure. The LC maps the points in both cluster to the center of the right cluster whereas the SD pre-image differentiate both clusters.

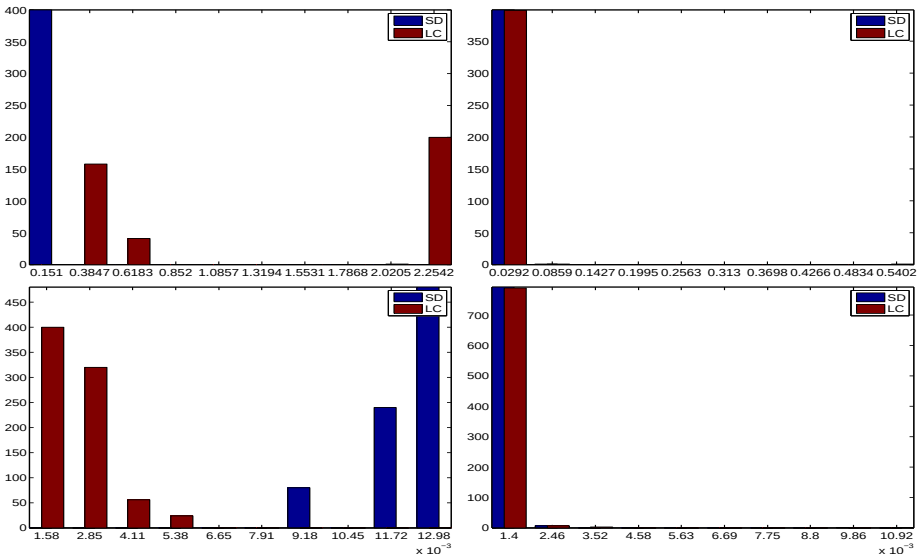

Figure 4.11: Comparison of the two pre-image algorithms for the Diffusion Maps kernel applied to kernel PCA projections with clustered data in low (top) and high (bottom) dimensions. Histogram of the empirical feature space distance. Left: Kernel PCA projection with $q = 1$ principal components. Right with $q = 10$. See text for details.

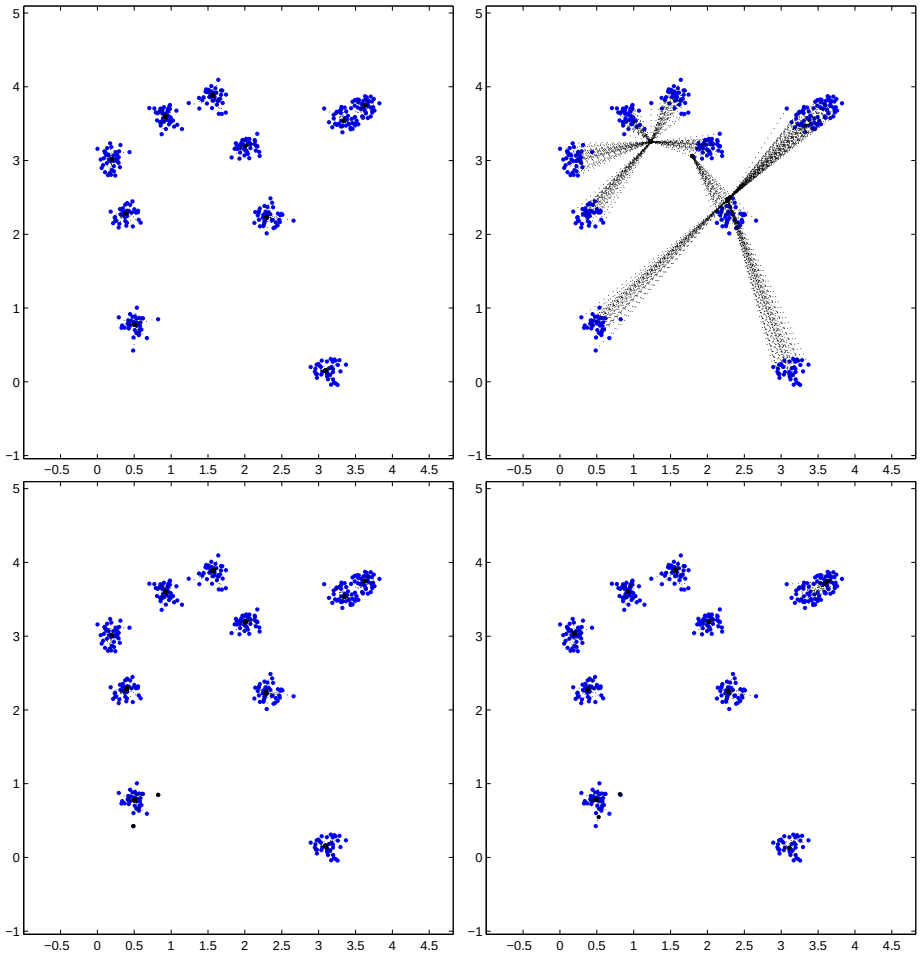

Figure 4.12: Pre-image results for the 2 dimensional cluster data. Left: SD and right LC. Top row: Results with $q = 1$, bottom with $q = 10$. The blue points are the original training set. The black dots are the computed pre-images. The dotted lines connect x_i with $k\mathbb{P}_1(x_i)$.

General remarks

As a general conclusion, it can be said that the LC pre-image works well if ψ is close to the manifold $\varphi(\mathcal{H})$, as with the non-iterative pre-images for the Gaussian kernel. Otherwise the neighbors may lie in distant parts of the input space.

In this section we compared different pre-images approaches using the feature space distance error as a performance measure. However we did not evaluate if the obtained pre-images were meaningfull. For instance we can apply the kernel PCA projection with $q = 10$ principal components to project noisy samples of the sinusoidal manifold dataset onto the underlying manifold. From this point of view, the results using $q = 1$ principal component were bad.

It is also noticeable that there seems to be no substantial changes between

the Diffusion Maps and the Gaussian kernel. With manifold data, both need a high number of principal components to remove the noise, taking into account that the data comes from a noisy one-dimensional manifold. One could expect this from the Gaussian kernel, but Diffusion Maps is a dimensionality reduction kernel, and the projection over the kernel principal components is the low dimensional representation. For instance for the manifold dataset, we have already seen in Figure 3.1, that the first principal component for Diffusion Maps varies encodes the position along the underlying sinusoidal manifold.

In the next section we are going to see a different pre-image that allows a more efficient use of the information of the principal components.

4.3 Constrained pre-image

In this section we are going to propose an alternative pre-image criterion, to circumvent the problems shown in the last section. Before that, we are going to find out what are the reasons for the observed behavior.

We are going to use the sinusoidal manifold dataset and the Diffusion Maps kernel to illustrate the ideas, since we are interested in applying the kernel PCA projection to learn the manifold underlying the data.

In section §4.3.1 we are going to study the geometry induced by the kernel in the input space. Then we will infer how things should look like in the high dimensional feature space §4.3.2. In §4.3.3 the new pre-image with some initial results will be presented.

4.3.1 Input space geometry induced by the kernel

Since the kernel PCA projections are the coordinates of the mapping, looking at their values in the input space, allow us to infer some properties of the mapping. Figure 4.13 shows the projection over the first principal components for the Diffusion Maps kernel. The images were created as those in Figure 3.1.

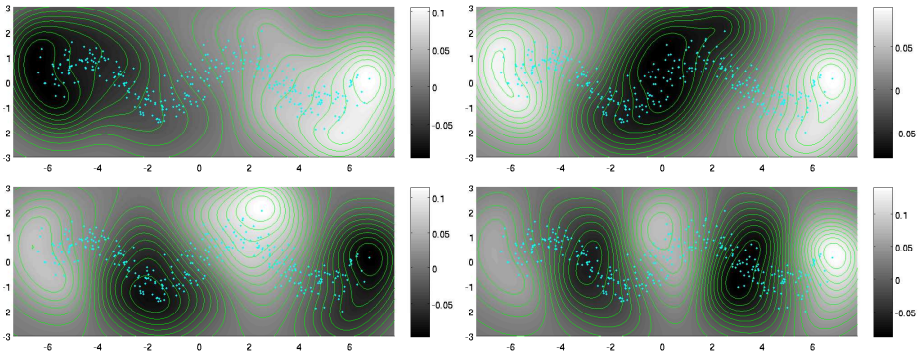

Figure 4.13: Kernel PCA projections over the first four principal components. The gray scale of pixel x denotes the value of the projection $\langle p_i, \varphi_m(x)\rangle_{\mathcal{H}}$ with $i = 1, \ldots, 4$. Top: First and second components, bottom: Third and fourth. The green curves denote the level curves of the projection.

As we can see, the first principal components vary along the manifold. The

level curves of the projections represent all the points that will have the same coordinates in the kernel PCA representation of the mapping. These curves traverse the dataset "orthogonally" to the underlying manifold. Thus, the mapping did discover the geometry of the underlying manifold, and the projection over the first principal component could serve as a lower dimensional (one dimension) representation of the dataset.

Figure 4.14 shows the empirical feature space pre-image $D_m(x) = \|\varphi_m(x) - \psi\|^2_{\mathcal{H}}$, where in this case $\psi = \mathbb{P}_1\varphi_m(x_i)$. The white circle shows the position of x_i. The iterations of the pre-image algorithm are shown as white dots. As we can see, they converge to one of the local minima of the feature space distance. The problem is that this minimum is not where we would like it to be for projecting x_i on the manifold. Furthermore the location of the local minima will be allmost the same independently of x_i. This is the reason why in the top right plot of Figure 4.10 the pre-images are attracted to a few points in the input space. These points are the local minima of the empirical feature space distance towards the projection $\mathbb{P}_1\varphi_m(x_i)$.

We can also observe that the value of the projection over the first principal component is not conserved, meaning that if x denotes the resulting pre-image $\mathbb{P}_1\varphi_m(x) \neq \mathbb{P}_1\varphi_m(x_i)$. The initial iterations of the pre-image algorithm follow approximately the level curve, but then they keep on going along the center of the manifold.

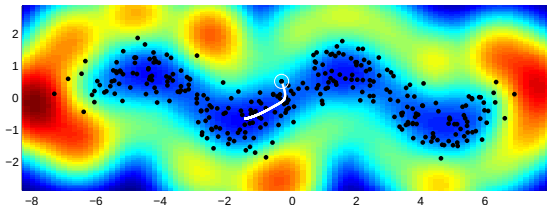

Figure 4.14: Kernel PCA projection of a point. Iterations of the pre-image of the projection onto the first principal component. The colored image represents the feature space distance to the projected point.

In order to find out the reason of this behavior, we decompose $D_m(x)$ in the sum of the norms of two orthogonal vectors:

$$\begin{aligned} D_m(x) &= \|\varphi_m(x) - \mathbb{P}_p\varphi_m(x_i)\|^2_{\mathcal{H}} \\ &= \|\varphi_m(x) - \mathbb{P}_p\varphi_m(x)\|^2_{\mathcal{H}} + \|\mathbb{P}_p\varphi_m(x) - \mathbb{P}_p\varphi_m(x_i)\|^2_{\mathcal{H}} \end{aligned} \tag{4.35}$$

The first term measures the distance from $\varphi_m(x)$ to the subspace spanned by the first principal components, while the second term measures the distance between the projections of $\varphi_m(x)$ and $\varphi_m(x_i)$.

Since the first one does not depend on x_i we can think of it as a regularization term: The pre-image x should be such that $\varphi_m(x)$ is close to the principal subspace. The information about x_i is encoded in the second term, the data fitting term. Figure 4.15 depicts both terms, as seen in the input space.

The complete distance map from Figure 4.14 corresponds to the addition of both terms. It is very similar with the subspace distance map in the right of Figure 4.15. The reason for this becomes apparent from the color scale bars in

Figure 4.15: The magnitude of the data fitting term is much smaller than that of the regularization term. When adding both terms, the data fitting term is negligible.

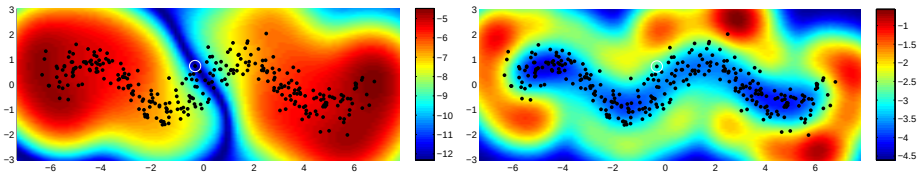

Figure 4.15: Components of the feature space distance (the scale is logarithmic with base e). Left: Distance between the projections. Right: Distance to the subspace.

So far we have seen that the first principal components encodes the one dimensional underlying manifold in the dataset, and that the distance to the principal component, as can be observed in Figure 4.15 captures the distance to the underlying manifold. In the next section we are going to explain why the data fitting term in the distance has such a small contribution.

4.3.2 Diffusion Maps' feature space

Figure 4.16 shows a representation of what could be happening in the feature space. The mapping $\varphi_m : \mathcal{X} \to \mathcal{H}$ is continuous and differentiable, since it is built from linear combinations of Gaussians. As in this case $\mathcal{X} \subset \mathbb{R}^2$, the image $\varphi_m(\mathcal{X})$ is a 2D manifold in the feature space, which is the smooth curvy manifold shown in the drawing. This manifold is the set of feature space points that have exact pre-image.

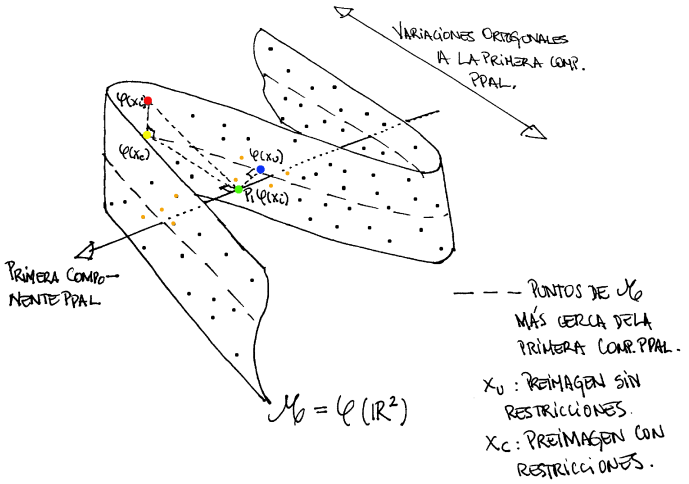

Figure 4.16: Schematic representation of the feature space. See text.

The principal axis is drawn as the arrow that traverses $\varphi_m(\mathcal{X})$. The dashed line going through $\varphi_m(\mathcal{X})$ is the mapping of the unknown sinusoidal curve underlying the noisy samples in the input space. The input space level curves of the kernel PCA projection are seen in the feature space as the intersection of $\varphi_m(\mathcal{X})$ with planes (or hyperplanes) orthogonal to the principal axis. For example the red dot $\varphi_m(x_i)$ and the yellow dot $\varphi_m(x_c)$ in Figure 4.16 have the same projection $\mathbb{P}_1\varphi_m(x_i)$, which is depicted as the green dot. The yellow dot is closer to the principal axis. In fact it is the point in $\varphi_m(\mathcal{X})$ which minimizes the distance to the principal axis, while projecting on $\mathbb{P}_1\varphi_m(x_i)$.

As we saw in Figure 4.15, the distance to the principal axis varies more than the distance between the projections. This is represented in the drawing by the strong oscillations of $\varphi_m(\mathcal{X})$ across the principal axis. A displacement of a point on $\varphi_m(\mathcal{X})$ will generate a large change in the distance distance towards the principal axis (regularization term), and a small change in the value of the $\mathbb{P}_1$ projection (data fitting term).

Denote by x_u the pre-image of $\psi = \mathbb{P}_1\varphi_m(x_i)$ computed by minimizing the empirical feature space distance. Its image $\varphi_m(x_u)$ (the blue dot) is the closest point in $\varphi_m(\mathcal{X})$ to ψ. Due to the strong variations of the distance to the principal axis, getting close to it becomes more important than keeping the projection constant.

This also explains the errors of the LC pre-image. The orange dots represent the nearest neighbors of ψ. Recall that ψ was approximated by a linear combination of its neighbors. Since the variation $\varphi(\mathcal{X})$ along the principal axis is smaller than the variation across it, the neighbors may lie in different parts of $\varphi(\mathcal{X})$ which are close in the feature space but can be distant in the input space. This is a consequence of the high curvature of $\varphi(\mathcal{X})$. Since the mapping is not linear, the same linear combination in the input space can be mapped far away from ψ.

Although the mapping aligns the underlying sinusoidal manifold with the principal axis in the feature space, it curves the manifold in other orthogonal directions. The magnitude of the variations of $\varphi(x)$ in the orthogonal complement of principal axis is much higher than the the variations along the principal axis itself. This seems contradictory since the principal axis maximices the variance of the projections onto it.

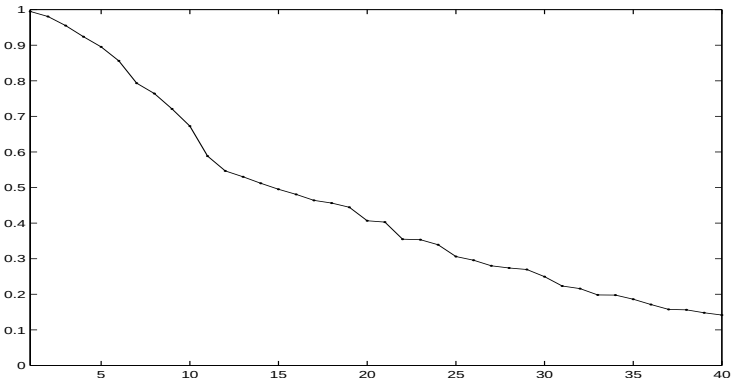

Figure 4.17: First 40 eigenvalues of the Diffusion Maps kernel applied to the sinus dataset.

Figure 4.17 shows the eigenvalues of the kernel matrix in descending order. As can be seen in the Figure, the decay is very small. The energy of the first one is only a 5% of the total energy. This explains the very small relative weight of the second term in equation (4.35). Individually, the energy of the first principal component is higher, but it is just a small fraction of the total energy.

4.3.3 Constrained pre-image

The first principal component of the Diffusion Maps kernel captures the position of the points along the underlying principal manifold in the feature space. The optimization of the distance criterion fails to preserve this information. Generally speaking if we perform dimensionality reduction using the q principal components of the Diffusion Maps kernel, we are assuming that these principal component somehow parametrized the position on the underlying manifold.

When computing the pre-image of a kernel PCA projection the pre-image algorithm should keep the value of the kernel PCA projection constant. This can be done by defining the pre-image as the solution of a constrained optimization problem [2]:

$$x_\psi = \arg\min_{x \in \mathcal{X}} \|\varphi(x) - \psi\|^2_{\mathcal{H}} \tag{4.36}$$

$$\text{subject to } \mathbb{P}_q \varphi(x) = \mathbb{P}_q \psi \tag{4.37}$$

We will refer as $D(x)$ to the feature space distance $\|\varphi(x) - \psi\|^2_{\mathcal{H}}$. This can also be formulated in terms of φ_m and D_m if the actual mapping is unknown.

The condition in Eq. (4.37) imposes q constraints:

$$\langle \varphi(x), p_i \rangle_{\mathcal{H}} = \langle \psi, p_i \rangle_{\mathcal{H}} \qquad i = 1, \ldots, q \tag{4.38}$$

In the case discussed in the previous section when $q = 1$, Eq. (4.37) restricts the search domain to the level curve of the kernel PCA projection. Adding more constraints in this case would restrict the search domain to a few points given by the intersection of the level curves of the projections.

However, more constraints may be necessary if the dimension of the input space is higher. Consider the case in which the dataset consist of samples from a q dimensional manifold in $\mathcal{X} \subset \mathbb{R}^d$. Assume that the q principal components in the feature space capture the intrinsic coordinates of the manifold. The resulting search domain will be a $d - q$ dimensional manifold, coinciding with the dimension of the normal space at any point of the manifold (the so-called *co-dimension* of the manifold).

For example if the data was sampled from a curve in $\mathbb{R}^3$ and we set the number of principal components in $q = 1$, the search domain will be a two dimensional manifold. If the manifold is two dimensional and we set $q = 2$ the search domain will be a curve. Figure 4.18 illustrates these situations.

Pre-image algorithm for kernel PCA projections

The constraint pre-image problem as stated in Eqs. (4.36) and (4.37) is a non-linear optimization problem with non-linear constraints. In this section we

[2] During the writing of this thesis a related pre-image method, applied to the Diffusion Maps mapping was presented in [29]. It is also constrained optimization problem, with the same constraints and a different objective function.

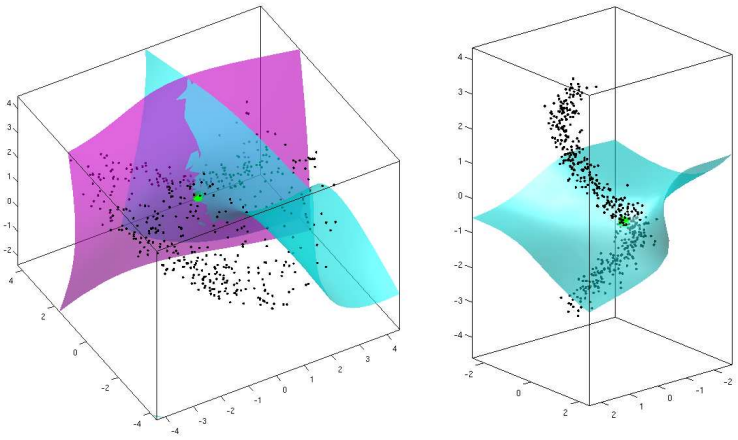

Figure 4.18: Search spaces resulting from keeping the kernel PCA projections constant. Left: Data sampled from a two dimensional manifold; the cyan surface is the set of points that have the same projection over p_1 than the green dot, and the magenta surface corresponds to the projection over p_2. The intersection is a curve. Right: Data sample from a one dimensional manifold; the cyan surface corresponds to the projection over p_1.

propose a solution to that problem for a simpler case in which we already know a point that meet the restrictions. This is precisely the case for the projection of a point x_0 onto the manifold.

$$k\mathbb{P}_q(x_0) = \arg\min_{x \in \mathcal{X}} \|\varphi(x) - \mathbb{P}_q\varphi(x_0)\|^2_{\mathcal{H}} \tag{4.39}$$

$$\text{subject to } \mathbb{P}_q\varphi(x) = \mathbb{P}_q\varphi(x_0) \tag{4.40}$$

The initial point of the iteration will be x_0 itself. To keep the kernel PCA projections constant, the direction of the displacement should be tangent to the search manifold[3]. This can be achieved by projecting the gradient of the objective function ∇D onto the tangent space to the kernel PCA projection level manifold that passes by x_0. Denote by $\mathcal{P}_{x_0,q}$ the search space:

$$\mathcal{P}_{x_0,q} = \{x \in \mathcal{X} \mid \mathbb{P}_q\varphi(x) = \mathbb{P}_q\varphi(x_0)\} \tag{4.41}$$

The tangent space to the manifold $\mathcal{P}_{x_0,q}$ at point x is orthogonal to the space spanned by the gradients of the functions $\langle\varphi(x), p_i\rangle_{\mathcal{H}}$, with $i = 1, \ldots, q$. Defining the vector functions

$$c_i(x) = \langle\varphi(x), p_i\rangle_{\mathcal{H}} \quad , \quad i = 1, \ldots, q \tag{4.42}$$

we must have that the modified gradient $\nabla^{\perp} D$ must be orthogonal to

$$\mathcal{P}^{\perp}_{x_0,q}(x) = \text{span}\{\nabla c_1(x), \ldots, \nabla c_q(x)\} \tag{4.43}$$

Let $\{v_1, \ldots, v_q\}$ be an orthonormal basis of $\mathcal{P}^{\perp}_{x_0,q}(x)$. This can be obtained, from example, with the *Gram-Schmidt* orthogonalization algorithm applied to

[3]We are assuming that the kernel PCA projections are differentiable functions. This is true for the Gaussian kernel, and for the KRR extension of data dependent kernels.

$\nabla c_1(x), \ldots, \nabla c_q(x)$. Then we can compute the modified gradient by removing from ∇D the components in the directions given by v_i with $i = 1, \ldots, q$:

$$\nabla^{\perp} D(x) = \nabla D(x) - \sum_{i=1}^{m} \langle v_i, \nabla D(x) \rangle v_i \tag{4.44}$$

This yields the following algorithm:

Pre-Image for kernel PCA projection

Input arguments: x_0, μ, τ, i_M, q.

Algorithm:

1. $\|\nabla^{\perp} D(x)\| = \tau + 1$; $i = 0$; $x = x_0$;
2. `while` $\|\nabla^{\perp} D(x)\| > \tau$ `and` $i < i_M$ `do:`
   (a) `compute` $\nabla D(x)$ `and` $\nabla c_i(x), \quad i = 1, \ldots, q$
   (b) `compute` $\{v_1, \ldots, v_q\}$ `by orthogonalizing` $\{c_1(x), \ldots, c_q(x)\}$
   (c) $\nabla^{\perp} D(x) = \nabla D(x) - \sum_{i=1}^{m} \langle v_i, \nabla D(x) \rangle v_i$
   (d) $x = x + \mu \nabla^{\perp} D(x)$
3. `return` x

Recall that the kernel PCA projection functions $c_i(x) = \langle \varphi(x), p_i \rangle_{\mathcal{H}}$ with $i = 1, \ldots, q$ are just the first q components of the kernel PCA representation of the empirical mapping of x, which we denoted as $\boldsymbol{y}_x$ in §2.5:

$$c_i(x) = [\boldsymbol{y}_x]_i = \left[\boldsymbol{\Lambda}^{-1/2} \boldsymbol{U}^T \boldsymbol{k}_x \right]_i \tag{4.45}$$

The gradient of this function can be easily computed. In the Appendix F can be found the calculus of the gradient for data dependent kernels, using the kernel regression framework to extend $\boldsymbol{k}_{X,x}$.

Results on synthetic data

The results are shown in Figure 4.19 for the Diffusion Maps kernel. The same figure displays the level curves of the kernel PCA projection over the principal axis. As can be seen in the figure, the de-noised version of each point keeps the projection value, with a small drift due to the discretization of the flow. We can also see that in some parts the level curves never intersect the sinusoidal manifold, and the flow is kept trapped (upper left corner).

There are some important aspects to consider regarding the choice of the parameters. Basically there are two main parameters: The scale parameter of the Diffusion Maps kernel σ_k and the scale parameter of the auxiliary kernel σ_h. According to [20] the scale paremeter of the Diffusion Maps algorithm should be small. However, we found good results with relatively high values of σ_k. In the Figure 4.19 σ_k was computed as the average distance to the 20th nearest neighbor, as in Eq. 4.23. The numerical value was $\sigma_k \approx 0.8$.

The extension scale σ_h, can be larger and still obtain good results. In fact, setting it to a high value, prevents overfitting to the noise. In Figure 4.19 it can

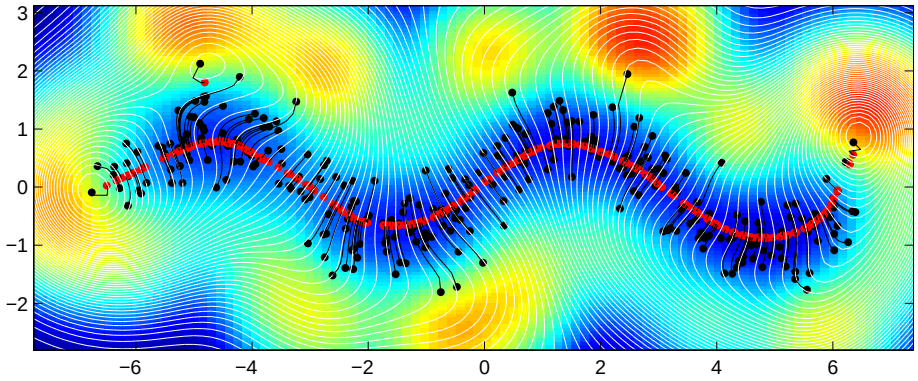

Figure 4.19: De-noising of the sinusoidal manifold using the proposed pre-image algorithm. The level curves of the projection are depicted in white. The black curves show the evolution of each point with the iterations.

be seen that level curves are affected by the outliers. An example of this are the closed level curves in the upper left corner of the figure. For this experiment σ_h was set to twice the average distance to the 30th nearest neighbor, yielding a value of $\sigma_h \approx 1$.

This value can be set higher, resulting in a broader extension domain. The domain of the extension can be seen in Figures 4.13. When the distance to the dataset if greater than σ_h, the extension of the kernel PCA projection vanishes. This causes the feature space distance function to suddenly drop (see Figure 4.14), and therefore points far away from the dataset can not be projected onto the manifold.

On the other hand, σ_h acts also as a regularizer. If σ_h is too high, we may oversimplify the learned manifold. In the sinusoidal dataset, this would result in a flattened sinusoidal.

Figures 4.20 and 4.21 show results on manifold de-noising in $\mathbb{R}^3$ for curves and a two dimensional manifold. The de-noising was performed by assigning each point x_i its kernel projection over the manifold $k\mathbb{P}_1(x_i)$ computed according Eq. (4.36). For the curves 4.20 one constraint was used ($q = 1$). Notice that the same framework can also be applied for the closed curve on the right.

Application to the Gaussian kernel

The Gaussian kernel can be seen as a particular case of the auxiliary kernel expansion framework. The expansion coefficients are $\boldsymbol{B} = \boldsymbol{I}$, the identity matrix, and $\boldsymbol{b} = \boldsymbol{0}$. Thus, all the formulas derived so far apply to this kernel.

Figure 4.22 shows the result of the constrained pre-image algorithm used with the Gaussian kernel. As we can see, the constraint prevents the points to accumulate in the local minima of the feature distance function. In this case however, the kernel principal component do not encode the position along the sinusoidal manifold. The feature space distance function also does not reflect the distance to the manifold.

Observe that some level curves become parallel to the gradient of the function. In these regions the modified gradient $\nabla^{\perp} D_m$ becomes zero and prevents

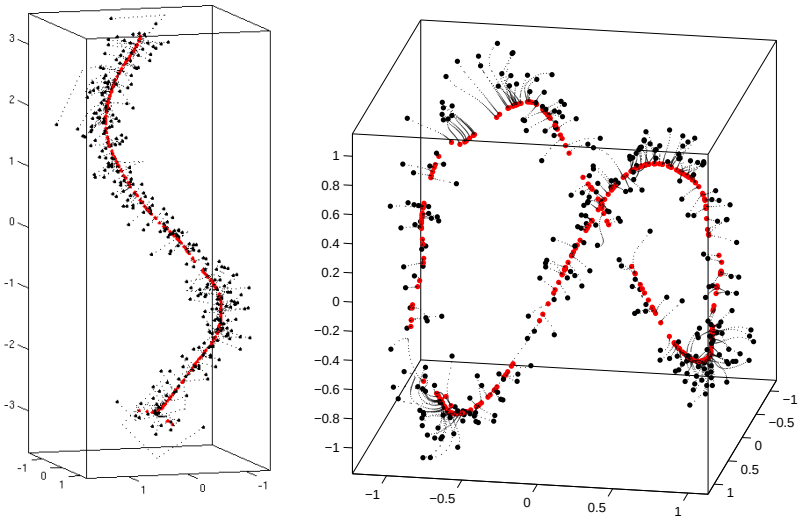

Figure 4.20: De-noising of curves in $\mathbb{R}^3$ with kernel PCA projection over the principal axis; only one constraint was imposed in the pre-image algorithms. On the left: Open curve. On the right: Closed curve.

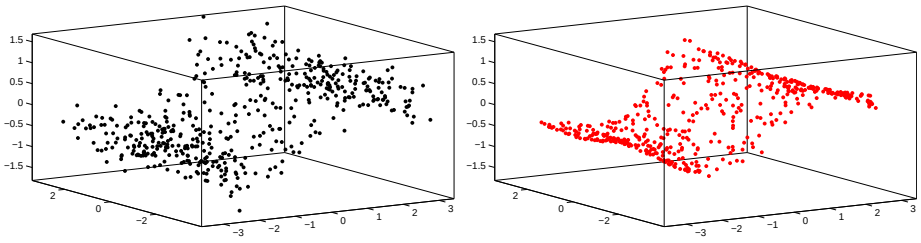

Figure 4.21: De-noising of a two dimensional dataset in $\mathbb{R}^3$. Kernel PCA projection. The first two principal components were kept constant in the pre-image algorithm. Left: Noisy data, right: Result of the de-noising.

some points from reaching the manifold.

4.4 Results on images

In this section we present some preliminary advances in the application of the proposed framework to image de-noising, as an example of a high dimensional input space. The images we will consider are part of a larger dataset of gray scale images of size $s = [s_x, s_y]$. The images are represented as vectors $x_i \in \mathbb{R}^{s_x s_y}$ by concatenating their columns. Denote by $X = \{x_1, \ldots, x_m\}$ the set of images. We will assume that the images lie on a lower dimensional manifold and will de-noise image x_i by computing the pre-image $k\mathbb{P}_q(x_i)$ of the kernel PCA projection according.

The image datasets considered are shown in Figure 4.23, belonging to the Teapots dataset. The dataset consists of $m = 400$ pictures from a teapot ac-

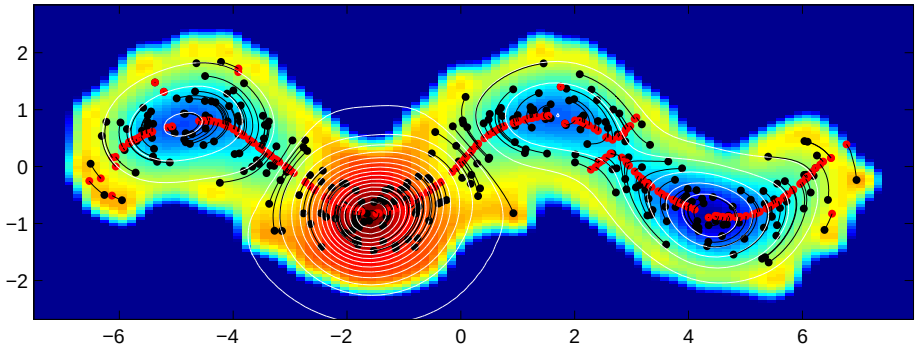

Figure 4.22: De-noising of the sinusoidal manifold using the proposed pre-image algorithm with the Gaussian kernel. The level curves of the projection are depicted in white. The black curves show the evolution of each point with the iterations.

quired while rotating the camera 360 degrees around the teapot. The rotation angle between each pair of consecutive pictures is approximately the same. We considered two variations of the dataset: The original size of $s = [76, 101]$, and a subsampled version by a factor of three, $s = [26, 34]$. The latter yields an input space of $d = 884$ dimensions, whereas the former gives $d = 7676$. The subsample images were previously filtered to prevent aliasing.

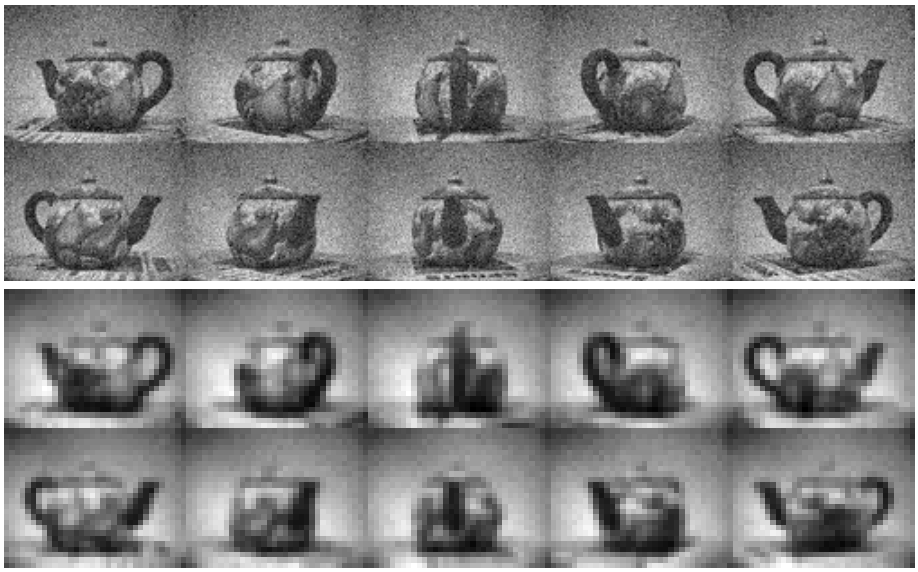

Figure 4.23: Teapots image dataset. Pictures taken from a teapot while rotating the camera around it. Gray scale images obtained by averaging RGB channels. White Gaussian noise added. The images of the bottom where obtained by low pass filtering and down-sampling by a factor of 3.

Gaussian noise was added before computing the kernel matrix [4]. The standard deviation of the noise was $\sigma_n = 0.02$ for the subsampled images and $\sigma_n = 0.02$ for the original ones. The gray value of the images is normalized to $[0, 1]$. The average energy of an image is approximately 0.2.

Since the images were obtained by rotating a camera smoothly, they should lie on a one dimensional manifold. The embedding computed by the projections over the two kernel principal components is shown in Figure 4.24. The embedding is a closed curve, representing the fact that the camera rotated 360 degrees. This supports the choice of $q = 1$.

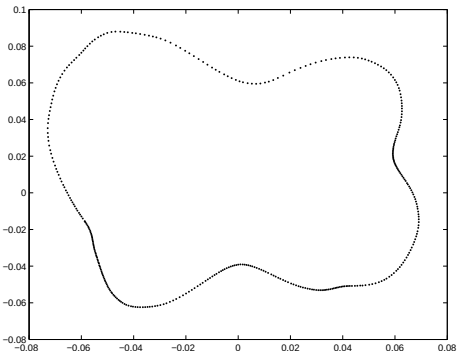

Figure 4.24: Low dimensional representation of the teapots dataset by the projections over the two kernel principal axes of the Diffusion Maps kernel. Subsampled images with noise.

Figure 4.25 shows results obtained with the Diffusion Maps kernel for the subsampled images. The parameters of the kernel and its extension where: $\sigma_k = 0.5427$, computed as the average distance to the the 3rd nearest neighbor; and $\sigma_h = 0.6621$, the average distance to the the 5th nearest neighbor. For the constrained pre-image algorithm we used $q = 1$ and $q = 2$ constraints (left and center images in the bottom row of Figure 4.25) and a step of $\mu = 0.1$. The tolerance on the norm of the gradient for the stopping condition was set to $\tau = 10^{-5}$.

The top row in the Figure shows the original noiseless image on the left and the noisy image on the right. The bottom right image was obtained using the unconstrained optimization pre-image of the projection over the two first principal axes $\mathbb{P}_2\varphi(x_i)$. The same step and stopping tolerance were used. As can be seen, the result of the unconstrained pre-image is very noisy, and the handle of the teapot is almost erased. The constrained pre-image has removed the noise but also the details on the body of the teapot. However the main structural elements are kept, like the light reflexes and shadows.

Observing carefully it can be seen that teapot in the pre-image obtained with one constraint has rotated (the handle seems to be closer to the camera). The pre-image obtained with $q = 2$ constraints has not (at least not noticeable). However it has some blurring on the handle. It has also preserved more the details of the body of the teapot. According to our experiments on synthetic data, for a q dimensional manifold, constraining the q principal components

[4] The subsampling was done before the addition of the noise, and after a low pass filtered to prevent aliasing.

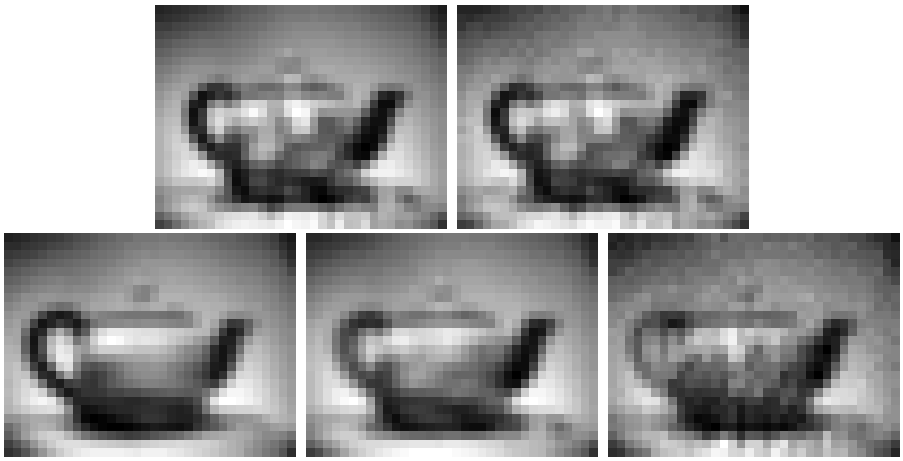

Figure 4.25: De-noising by the pre-image of the kernel PCA projection, with the subsampled images. Top row: Original image and with noise added. Bottom, from left to right: Constrained pre-image with $q = 1$, constrained pre-image with $q = 2$, unconstrained pre-image of the kernel PCA projection over two principal axes.

of the pre-image should be enough. However in this case using two principal components seems to work better.

The algorithm took around 2000 iterations to converge. The step was intentionally given a small value to minimize the drift on the constrained kernel projection values. With a step of $\mu = 1$ (ten times higher) very similar results can be obtained, as can be seen in Figures 4.26 and 4.27, with a number of iterations between 80 and 300 depending on the image.

Figures 4.26 and 4.27 show a comparison between $q = 2$ (second row in each Figure) and $q = 2$ (third row) constraints, for 14 images at equally spaced camera rotation angles. It is notorious that $q = 2$ yields better results. For example, the last three teapots in the third row of Figure 4.26 are in the same position. Although in both cases the results for the front and rear views of the teapot are worse than the side views, with $q = 1$ the results for these views are particularly bad: The handle vanishes completely and there is a lot of noise. Also in the 3rd and 5th positions in Figure 4.26 (before and after the rear view) the angle is preserved with $q = 2$ constraints, but not with $q = 1$.

The fourth row of Figures 4.26 and 4.27 shows the results obtained with the *LC* pre-image using $n = 10$ nearest neighbors. The reconstruction for many different images is the same, similarly to what happened in our previous experiments with synthetic data. As happened before, the feature space distance can be missleading due to the high curvature of the embedding. Recall the decomposition of the feature space distance of Eq. 4.35: The data fitting term is negligible compared with the regularization term. Therefore the neighbors found are close to the principal subspace, but may have a completely different position in the input space.

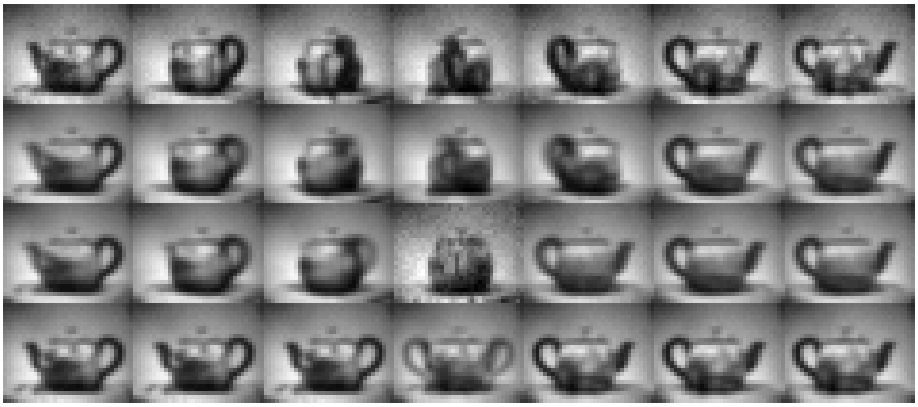

Figure 4.26: Kernel PCA de-noising for the first 7 of 14 images. Top: Original noisy images; second: Constrained pre-image using $q = 2$; third row: Constrained pre-image using $q = 1$; fourth row: LC pre-image with 10 nearest neighbors.

Figure 4.27: Kernel PCA de-noising for the last 7 of 14 images.

Results without subsampling

To test the method in an even higher dimensional space, we performed kernel PCA de-noising without subsampling the images. The results are shown in Figure 4.28. Increasing the dimension of the input space with the same training set makes this problem harder. For instance the de-noised image may not be in the span of the training images. The right image in the Figure 4.28 shows the projection of the clean image on the span of the training set. This is the best de-noising result that can be achieved in the span of the training set (in terms of L_2 distance).

Figure 4.29 shows some results with the proposed constrained pre-image varying the number of principal components q. From left to right $q = 2$, $q = 30$ and $q = 100$. Again this results contradict our intuition: Fixing the first two principal components (nor the first one) does not assure to capture the rotation angle of the teapot. When adding more constraints the results get better until

Figure 4.28: Result without subsampling. Left: Original image, center: Noise with standard deviation $\sigma_n = 0.08$ has been added. Right: Projection of the clean image over the span of the training set.

the noise begins to be captured, as with regular PCA. However when considering $q = 30$ principal components better and faster results can be obtained with the LC pre-image.

Figure 4.29: First tree images: Constrained pre-image varying the number of principal components q. From left to right $q = 2$, $q = 30$ and $q = 100$. The kernel parameter σ_k was set as a fifth of the average distance to the nearest neighbor. The right most image correspond to the LC pre-image with 10 nearest neighbors when projecting over $q = 30$ principal components.

The datasets we worked with in this section are of a different nature than the synthetic datasets used in the previous section. In this case, both for the original images and the subsampled ones the dimension of the input space exceeds the number of training set points m. Therefore the span of the training set will not cover the whole input space. This is important: The pre-image is a linear combination of the input samples, thus, they will be constrained to the span of the training set.

On the other hand the situation in the empirical feature space $\mathcal{H}^m$ is the opposite: Its dimension is the number of inputs $m = 400$. The image of the input space by the actual mapping $\varphi(\mathcal{X})$ will be a $d = 884$ dimensional manifold for the subsampled images. Its image by the empirical kernel map $\varphi_m(\mathcal{X})$ will be the projection of $\varphi(\mathcal{X})$ over the empirical feature space $\mathcal{H}^m$.

The constrained pre-image algorithm was developed with the intuition gained from the analysis of a problem that may have a different nature from this one. For the examples we analyzed in §4.3, the image of the input space by the empirical mapping $\varphi_m(\mathcal{X})$ was a manifold of dimension two, therefore a submanifold of $\mathcal{H}^m$. This may be the cause of some of the unexpected behaviors observed.

Results with the Gaussian kernel

The Gaussian kernel has been already used to perform kernel PCA de-noising [47]. In this section we are going to apply it to the downsampled teapot images. The results obtained are shown in Figure 4.30. The first column shows the original image without noise and with noise added, and the second corresponds to the de-noising results with the Diffusion Maps kernel, with $q = 2$ both for the constrained and the unconstrained pre-image. x

The results with the Gaussian kernel are shown in the third column, with the constrained (top) and the unconstrained (bottom) pre-image. The advantage of the constrained pre-image is not as significant as with the Diffusion Maps kernel. The unconstrained pre-image looks a bit noisier and the teapot is rotated. Comparing the constrained pre-images of both kernels Diffusion Maps yields better results. The teapot's handle is sharper, and the details on the body of the teapot are more similar to the original.

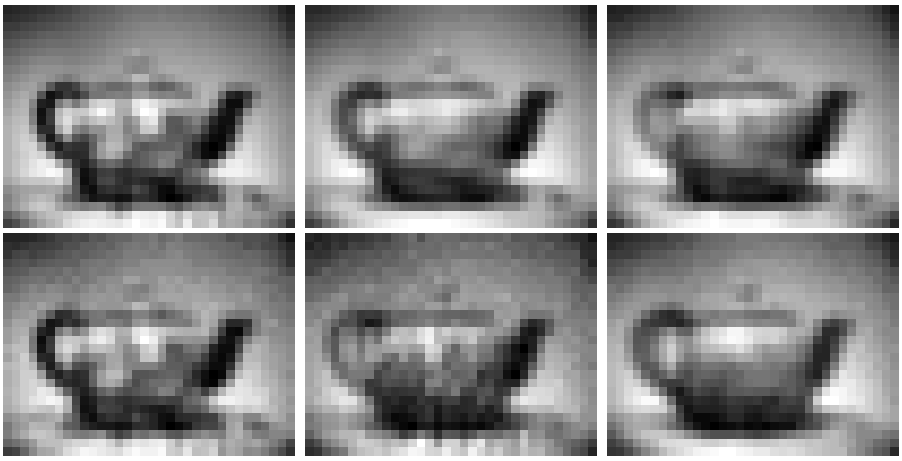

Figure 4.30: Results with the Gaussian kernel. First column original clean and noisy images. Center column: Results with the Diffusion Maps kernel using $q = 2$ principal components, top constrained pre-image, bottom unconstrained; right: Results with the Gaussian kernel with $q = 2$ principal components, top constrained pre-image, bottom unconstrained. The scale parameter of the kernel was computed as the average distance to the third nearest neighbor. The optimization step is $\mu = 0.1$.

Chapter 5

Conclusions

5.1 Concluding remarks

The kernel PCA interpretation sheds light over spectral dimensionality reduction techniques. The theory of kernel methods is well advanced. Its asymptotic results and finite size bounds may provide a better understanding of spectral dimensionality reduction. However the study of the induced geometry in the input space seems to be harder than the geometry of feature space, known though kernel function. Understanding the pre-image problem is crucial for controlling the input space geometric aspects of kernel methods. The work presented contributes to this goal.

We showed experimentally that the unconstrained optimization of the feature space distance has problems with some kernels. We proposed a solution to these problems by forcing the kernel principal components to remain constant in the pre-image criterion.

It can be argued however, that the source of the problem is the kernel, instead of the pre-image. The problems of the pre-image arise from the high variations of the kernel mapping. Thus, another interesting line of research is that of designing a kernel whose mapping does not have those variations.

To the time of writing we do not have enough experimental evidence nor theoretical results to affirm that the proposed constrained pre-image provides an effective solution or to tell the limits of this approach: which are the necessary conditions needed for a good performance. We would like to address this issues in the future.

The preliminary results for low dimensional datasets are good and encouraging: the developed algorithm project points onto the manifold, transforming the kernel PCA projection in an input space projection according to the kernel-induced geometry. However in very high dimensional settings, the results contradict the intuition we have, even for simple datasets, such as the teapot images dataset.

The proposed approach can be applied to both analytic and data dependent kernels. To achieved this we extended data dependent kernels using Kernel Ridge Regression, a standard and simple non-linear regression technique, which is itself another kernel method. Any other kernel regression technique can be used, such as Support Vector Regression of Kernel Lasso Regression, as long as

the output is a kernel expansion function.

It would be interesting to see if the choice of the regression method for extending the kernel has some influence in the results. Perhaps for some applications a sparse kernel expansion (such as those obtained with Support Vector Regression) yields a better performance.

For the manifold learning kernels, the framework adds basically two parameters the scale of the extension kernel, σ_h and the number of kernel PCA projections p. The σ_h parameter is related with the larger scale structure of the manifold. If the manifold has high curvature, σ_h should be low. On the other hand the lower the σ_h the smaller the domain of the extension.

A crucial parameter is the number of principal components p. According to our experience with synthetic datasets, p should be the dimension of the underlying manifold. For real high dimensional data we found better results using more constraints. We are currently trying to understanding the reason for this behavior.

5.2 Future work

In this section we enumerate some possible lines for future work:

- More experimental results are needed. Variations in the kernel function and framework parameters need to be further tested.
- Study of the theoretical properties of the proposed framework. The asymptotic results of the kernel PCA could be used to study the convergence of the learned manifold when the number of training samples tends to infinity.
- So far, we have only implemented the proposed pre-image method for projecting a point onto the manifold, because we have an initial condition which belongs to the feasible set. However the same idea can be applied to the computation of the pre-image of a generic feature space point. This is usefull for applications such as interpolating inside the manifold.
- There are other ways of extending the kernel that would be interesting to try. In particular it may be usefull to have a local extension length, dependent on the local complexity of the manifold: in high curvature regions a small σ_h should be used, but high curvature regions allow a larger one.
- Inclusion of other terms in the optimization problem. For high dimensional settings the solution is limited to the span of the training set. An interesting way of lifting this restriction is by adding other type of priors to the objective functions. An example of this could be a *total variation* regularization term.

Appendix A

The range of the kernel matrix

In the following we are going to prove the following proposition.

Proposition A.1. *The kernel vector* $\boldsymbol{k}_x = [k(x, x_1), \ldots, k(x, x_m)]^T$ *is in the range of the kernel matrix for the set* $X = \{x_1, \ldots, x_m\}$*, denoted by* $\boldsymbol{K}$*.*

By the range of a matrix we refer to the span of its columns.

Appendix B

Kernel Matrix Completion

B.0.1 Euclidean distances: Semidefinite Programming

Graepel [33] presents an approach for completing a kernel matrix with missing entries. The completion is forced to be a symmetric positive semidefinite matrix. Suppose $\hat{\boldsymbol{K}}$ is the incomplete kernel matrix. Let $\boldsymbol{M}$ be a mask matrix, that is one on the missing entries and zero otherwise. Then we can formulate the problem as:

$$\boldsymbol{K} = \underset{\boldsymbol{K} \in S_m}{\arg\min} \|\boldsymbol{M} \circ (\boldsymbol{K} - \hat{\boldsymbol{K}})\|^2 \tag{B.1}$$

$$\text{subject to } \boldsymbol{\mathcal{A}K} = \boldsymbol{b} \tag{B.2}$$

where the $\circ$ operator denotes the Hadamard product (*i.e.* point-wise multiplication of matrices), and the norm is the *Frobenious* norm. The constraint $\boldsymbol{\mathcal{A}K} = \boldsymbol{b}$ can be used to add an equality constraint over the matrix. The operator $\boldsymbol{\mathcal{A}} : S_m \to \mathbb{R}^p$ is a linear operator defined over the space S_m of symmetric positive semidefinite $m \times m$ matrices.

It turns out that this problem can be formulated as a standard quadratic programming algorithm, the size being the number of missing entries. The method, as stated by the authors, is very computationally costly.

B.0.2 Kullback Leibler Divergence [64]

The approach presented in [64] uses non-Euclidean Geometry in a space of positive semidefinite (PSD) matrices to complete the missing entries while preserving positive semidefinitness. The main idea is to associate each PSD matrix with a probability density function (PDF), treating the matrix as the covariance matrix of a Gaussian variable with zero mean. The *Kullback-Leibler divergence* between this PDFs can be used as a dissimilarity measure between the covariance matrices. The main advantage of this approach is that it allows the use of the *information geometry* framework, and in particular the *em* algorithm (this is not the same as the Expectation-Maximization algorithm, although in some cases they coincide).

We briefly review some basics notions of information geometry needed to understand this approach. Information Geometry is a field of statistics that study information and probability by means of differential geometry. This is

based in the fact that a space of probabilities can be considered as a differential manifold endowed with a Riemannian metric. This provides interesting geometric interpretations of some well known statistical algorithms. For instance, the maximum likelihood estimate corresponds to a projection over a certain manifold. The EM algorithm can be considered as a special case of the *em* algorithm.

Let us consider two manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$. The *em* basically finds one point $M_1 \in \mathcal{M}_1$ and $M_2 \in \mathcal{M}_2$ minimizing the KL divergence between M_1 and M_2. The points M_1 and M_2 correspond to PDFs, and therefore the KL divergence can be used to compare them. The *em* finds M_1 and M_2 by iteratively alternating projections onto the manifolds $\mathcal{M}_1$ and $\mathcal{M}_2$. Since the KL divergence is not symmetric, there are two different type of projections over a manifold $\mathcal{M}$:

- e-projection: $\pi^e_{\mathcal{M}}(B) = \arg\min_{A\in\mathcal{M}} \mathrm{KL}(A, B)$
- m-projection: $\pi^m_{\mathcal{M}}(B) = \arg\min_{A\in\mathcal{M}} \mathrm{KL}(B, A)$

The *em* algorithm has the following steps, given a initial matrix, $M^0 = M^0_1 = M^0_2$.

- e-step: $M^{k+1}_1 = \pi^e_{\mathcal{M}_1}(M^k_2)$
- m-step: $M^{k+1}_2 = \pi^m_{\mathcal{M}_2}(M^{k+1}_1)$

Under certain hypothesis regarding the manifolds $\mathcal{M}_1$ and $\mathcal{M}_1$, the uniqueness of each of the projections in each step can be proved. However the iterative algorithm can converge to a local minima, and its sensible to the initialization.

The approach followed in [64] uses the *em* to complete the kernel matrix. Suppose we have a matrix D with missing rows and columns:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}^{nn} & (\boldsymbol{K}^{ln})^T \\ \boldsymbol{K}^{ln} & \boldsymbol{K}^{ll} \end{bmatrix} \tag{B.3}$$

where $\boldsymbol{K}^{nn}$ is the block formed by the known entries, and the rest is unknown. Let us consider the manifold $\mathcal{M}_1$ as the manifold of symmetric positive semidefinite matrices that coincide with K in the first $n \times n$ entries:

$$\mathcal{M}_1 = \left\{ \boldsymbol{K} \in \mathbb{R}^{m\times m} : \boldsymbol{K}^{ln} \in \mathbb{R}^{l\times n}, \boldsymbol{K}^{ll} \in \mathbb{R}^{l\times l}, (\boldsymbol{K}^{ll})^T = \boldsymbol{K}^{ll}, \boldsymbol{K} \succeq 0 \right\} \tag{B.4}$$

This is clearly the manifold on which the desired solution is. This is called the data manifold. But still we need another manifold, $\mathcal{M}_2$, which we are going to refer to as the model manifold. This manifold encodes the *a priori* information we have.

In [64] the authors assume that this information is given in the form of another kernel matrix, $\boldsymbol{H} \in \mathbb{R}^{m\times m}$. The manifold $\mathcal{M}_2$ is defined as the spectral variants of the kernel $\boldsymbol{H}$. These are defined as the set of matrices which have the same eigenvectors, but different set of (non-negative) eigenvalues.

The *em* algorithm is then applied between this two manifolds. The limit point in the $\mathcal{M}_1$ manifold is the completed kernel matrix. The e-step and the m-step can be computed in closed form, however the final completion must be found iteratively. The cost of each iteration depends mainly on the size missing data.

Appendix C

Selection of the extension scale

In this Chapter we present some of the work performed for estimating optimal parameters for the data dependent kernel extension framework which finally was not used.

C.1 Matrix Calculus

In this section we are going to briefly review some properties of matrix calculus, those needed for the derivatives of the PRESS. We based on [16].

Definition C.1. In the following we are going to handle functions $\mathcal{F} : \mathbb{R}^{p\times q} \to \mathbb{R}^{n\times m}$. The output, as well as the argument of $\mathcal{F}$ are matrices. For computing its differential, we are going to treat those both matrices as vectors by concatenating the columns. Consider $Y = \mathcal{F}(X)$. Then, $d\mathcal{F}/dX = dY/dX \in \mathbb{R}^{nm\times pq}$ such that:

$$\left[\frac{dY}{dX}\right]_{ij} = \frac{\partial Y_{:i}}{\partial X_{:j}} \tag{C.1}$$

where A: refers to the vectorized version of matrix A.

Basic Properties. Before reviewing some basic properties of the calculus with matrices, we need a few definitions. The Kronecker product $A\otimes B$ between two matrices $A \in \mathbb{R}^{p\times q}$ and $B \in \mathbb{R}^{n\times m}$ is a $np \times mq$ block matrix given by

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1m}B \\ A_{21}B & A_{22}B & \cdots & A_{2m}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1}B & A_{n2}B & \cdots & A_{nm}B \end{bmatrix} \tag{C.2}$$

The Hadamard product, or component-wise product $A \circ B$ between $A, B \in \mathbb{R}^{nm}$ is a $n \times m$ matrix such that $[A \circ B]_{ij} = A_{ij}B_{ij}$.

Chain rule If $Z = Z(Y)$ and $Y = Y(X)$, then $dZ/dX = dZ/dY\,dY/dX$.

Linearity $d[\alpha Y(X) + \beta Z(X)]/dX = \alpha dY/dX + \beta dZdX$ where $\alpha, \beta \in \mathbb{R}$.

Product $d[Y(X)Z(X)]/dX = (I \otimes Y)dZ/dX + (Z^T \otimes I)dY/dX$

Inverse $dX^{-1}/dX = -X^{-1} \otimes X^{-1}$

Product with a scalar function Let $f : \mathbb{R}^{n\times m} \to \mathbb{R}$. Then

$$d[Y(X)z(X)]dX = zdYdX + Y\colon dz/dX$$

The following results are going to be used later.

- $d[\mathbf{c}^T X^T X\mathbf{c}]/dX = 2[X\mathbf{c}\mathbf{c}^T]$:
- $d[X^T CX]dX = I \otimes X^T C + X^T C \otimes I$

C.2 Leave-One-Out Cross-Correlation for Kernel Ridge Regression

In this section we are going to show that the LOO error can be computed efficiently in a closed form that can be minimized to find the optimal parameters (w.r.t. the LOO performance measure). This derivation is based in [18], presented here with more detail.

C.2.1 Closed for expression for the PRESS

Let $X = \{x_1, \ldots, x_m\} \subset \mathcal{X} \subset \mathbb{R}^d$, be the set of input points. For given parameters γ and σ_h, the coefficients $\boldsymbol{\beta}$ and b can be found by solving the following equation:

$$\begin{bmatrix} \boldsymbol{M} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \beta \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix} \tag{C.3}$$

where $\boldsymbol{M} = \boldsymbol{H} + \gamma \boldsymbol{I}$, and $\mathbf{1}$ is a column vector of dimension m whose entries are all one. Let us define the following matrices as:

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{M} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} = \begin{bmatrix} c_{11} & \boldsymbol{c}_1^T \\ \boldsymbol{c}_1 & \boldsymbol{C}_1 \end{bmatrix} \tag{C.4}$$

where $c_{11} \in \mathbb{R}$, $\boldsymbol{c}_1 \in \mathbb{R}^m$ and $\boldsymbol{C}_1 \in \mathbb{R}^{m\times m}$ Consider $\boldsymbol{\beta}^{(i)}$, $b^{(i)}$ parameters learned when removing x_i from the training set. For $i = 1$ we have that:

$$\begin{bmatrix} \boldsymbol{\beta}^{(1)} \\ b^{(1)} \end{bmatrix} = \boldsymbol{C}_1^{-1} \begin{bmatrix} y_2 \\ \vdots \\ y_m \\ 0 \end{bmatrix} \tag{C.5}$$

Therefore, the learned model predicts the following value for x_1:

$$y_1^{(1)} = \boldsymbol{c}_1^T \begin{bmatrix} \boldsymbol{\beta}^{(1)} \\ b^{(1)} \end{bmatrix} = \boldsymbol{c}_1^T \boldsymbol{C}_1^{-1} \begin{bmatrix} y_2 \\ \vdots \\ y_m \\ 0 \end{bmatrix} \tag{C.6}$$

On the other hand, the last m equations of the system (C.3) can be rewritten as $[\boldsymbol{c}_1 \boldsymbol{C}_1][\boldsymbol{\beta}^T, b]^T = [y_2, \ldots, y_m, 0]^T$, thus we have that:

$$y_1^{(1)} = \boldsymbol{c}_1^T \boldsymbol{C}_1^{-1} [\boldsymbol{c}_1 \ \boldsymbol{C}_1] \begin{bmatrix} \boldsymbol{\beta} \\ b \end{bmatrix} = \boldsymbol{c}_1^T \boldsymbol{C}_1^{-1} \boldsymbol{c}_1 \beta_1 + \boldsymbol{c}_1 \begin{bmatrix} \beta_2 \\ \vdots \\ \beta_m \\ b \end{bmatrix} \tag{C.7}$$

We now consider the first equation on the system (C.3), which states that $y_1 = c_1 1 \beta_1 + \boldsymbol{c}_1^T [\beta_2, \ldots, \beta_m, b]^T$. As a result we obtain that

$$r_1 = y_1 - y_1^{(1)} = \beta_1 (c_{11} - \boldsymbol{c}_1^T \boldsymbol{C}_1^{-1} \boldsymbol{c}_1) = \frac{\beta_1}{\boldsymbol{C}_{11}^{-1}} \tag{C.8}$$

The last equality is a consequence of the block matrix inversion lemma [31] applied to the matrix $\boldsymbol{C}$. This equation can be generalized to any point in the training set, by appropriately permuting the equations and the unknowns in the linear system (C.3):

$$r_i = y_i - y_i^{(i)} = \frac{\beta_i}{\boldsymbol{C}_{ii}^{-1}} \tag{C.9}$$

C.2.2 Gradient of the PRESS

For minimizing the PRESS, we need express Eq. (3.31) in a more convenient way. From Eq. (C.3) it can be seen that

$$b = \frac{\mathbf{1}^T M^{-1} \boldsymbol{y}}{\mathbf{1}^T M^{-1} \mathbf{1}} \quad \text{and} \quad \beta = \left(M^{-1} - M^{-1} \mathbf{1} \frac{\mathbf{1}^T M^{-1}}{\mathbf{1}^T M^{-1} \mathbf{1}} \right) \boldsymbol{y} = D \boldsymbol{y} \tag{C.10}$$

Using again the block matrix inversion lemma, it can be easily seen that D equals the first $m \times m$ block of C^{-1}. Therefore the PRESS can be expressed in matrix form in the following way:

$$P = \boldsymbol{y}^T D^T \text{Diag}(D)^{-1} \text{Diag}(D)^{-1} D \boldsymbol{y} \tag{C.11}$$

Recall from Eq. (C.11) that the PRESS can be written as

$$P(\sigma, \gamma) = \mathbf{y}^T D \text{Diag}(D)^{-1} \text{Diag}(D)^{-1} D \mathbf{y} = \mathbf{y}^T T^T T \mathbf{y} \tag{C.12}$$

where $T = \text{Diag}(D)^{-1} D$. Using the chain-rule, where need to compute the following terms for computing the derivative:

$$\frac{\partial P}{\partial \sigma} = \frac{dP}{dT} \frac{dT}{dD} \frac{dD}{dM^{-1}} \frac{dM^{-1}}{dM} \frac{dM}{d\sigma} \tag{C.13}$$

and similarly for γ. The first term is the quadratic form from Eq. REF. The derivative is an m^2 row vector given by:

$$\frac{dP}{dT} = 2[T \mathbf{y} \mathbf{y}^T]_{:}^T \tag{C.14}$$

The second term needs the calculation of $d\text{Diag}(D)/dD$ which can be easily computed using the definition. The result is a m^2 diagonal matrix, its diagonal being $I_{:}$. Using some basic properties it can be shown that

$$\frac{dT}{dD} = I \otimes \text{Diag}(D)^{-1} + (D \otimes I) \frac{d\text{Diag}(D)}{dD} \tag{C.15}$$

For computing the term dD/dM^{-1} recall that

$$D = M^{-1} - M^{-1}\mathbf{1}\mathbf{1}^T M^{-1}\frac{1}{\mathbf{1}^T M^{-1}\mathbf{1}} \tag{C.16}$$

Thus,

$$\frac{dD}{dM^{-1}} = I_{m^2} - \frac{I \otimes M^{-1}\mathbf{1}\mathbf{1}^T + M^{-1}\mathbf{1}\mathbf{1}^T \otimes I}{\mathbf{1}^T M^{-1}\mathbf{1}} + \frac{[M^{-1}\mathbf{1}\mathbf{1}^T M^{-1}]\colon [\mathbf{1}\mathbf{1}^T]\colon^T}{(\mathbf{1}^T M^{-1}\mathbf{1})^2} \tag{C.17}$$

The term dM^{-1}/dM can be computed directly from Eq REF. Finally, we only need the terms $dM/d\sigma$ and $dM/d\gamma$. Recall that $M = H + \gamma I$, and $H_{ij} = \exp(-\|x_i - x_j\|^2/2\sigma^2)$. Then:

$$\frac{dM}{d\sigma} = \sigma^{-3}E^2 \circ H \quad \text{and} \quad \frac{dM}{d\gamma} = I\colon \tag{C.18}$$

C.3 Comparison between the PRESS and the RK-PRESS

In this section we show some results comparing the PRESS and the RK-PRESS measurements. For performing the comparison we used synthetic datasets over which we computed the Diffusion Maps kernel.

Two Gaussian clusters in $\mathbb{R}^2$. The centers of the clusters are $[-1, 0]$ and $[1, 0]$, and the variances are 0.5 and 0.1. Each cluster has 100 points. The Diffusion Maps kernel is computed using the average distance to the nearest neighbor, which yielded $\sigma_k = 0.0658$.

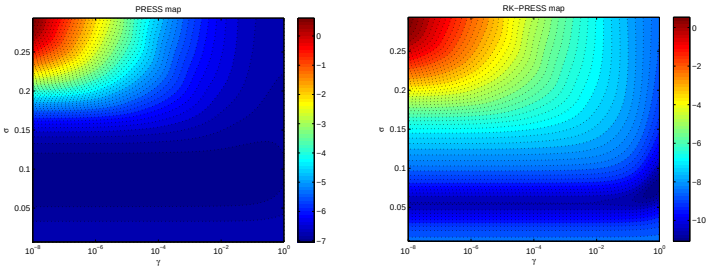

Figure C.1: PRESS and RK-PRESS map for the two Gaussian cluster dataset. The logarithm is taken in both cases for visualization.

The PRESS and the RK-PRESS were computed for a 60×60 grid of parameters γ and σ_h. The Figures C.1 and C.2 show the results. Figure C.1 shows that the overall behavior is similar. A big value of σ_h and a small γ is in both cases a bad combination. For σ_h close to σ_k, there is a wide range of values of γ for which the results are almost constant. The optimal parameters are in this region.

Figure C.2 shows a cut with a fixed value of γ comparing the PRESS and the RK-PRESS. This shows that the major differences are in the region where both

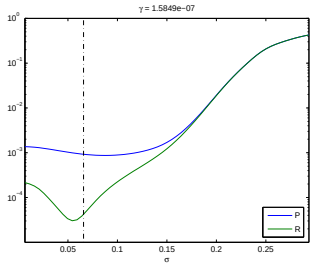

Figure C.2: PRESS and RK-PRESS for the two Gaussian cluster dataset. The logarithm is taken in both cases. The Figure shows the variation with σ_h, for $\gamma = 1.58\dot{1}0^{-7}$. The vertical black line shows the σ_k.

attain their best performance. In this case, the PRESS is a upper bound for the RK-PRESS. This can be helpfull: by minimizing the PRESS we are bounding the RK-PRESS. However, according to the Figure, using directly $\sigma_h = \sigma_k$ would yield a worse PRESS, but a better RK-PRESS than the minimizer of the PRESS.

Two Gaussian Clusters in $\mathbb{R}^{10}$. The centers of the clusters are $[-1, 0, \ldots, 0]$ and $[1, 0, \ldots, 0]$, and the variances are 0.5 and 0.1. Each cluster has 100 points. The Diffusion Maps kernel is computed using the average distance to the nearest neighbor, which yielded $\sigma_k = 0.7285$.

In this case, expecting the same regularity as before, we computed the PRESS and the RK-PRESS only for a coarser grid 30×24 grid of parameters σ_h and γ. The Figures C.3 and C.4 show the results. The behavior this time has some differences. The PRESS map is almost flat, as can be seen also in Figure C.4. The RK-PRESS has more variation (always bellow the PRESS), and has a valley almost surrounding the σ_k.

Figure C.1 shows that the overall behavior is similar. A big value of σ_h and a small γ is in both cases a bad combination. For σ_h close to σ_k, there is a wide range of values of γ for which the results are almost constant. The optimal parameters are in this region.

Figure C.4 confirms these observations, showing that there is almost a two orders of magnitude difference between the PRESS and the RK-PRESS. Again, using $\sigma_h = \sigma_k$ yields much better results (according to the RK-PRESS) than the minimizer of the PRESS. As in the two-dimensional case, the profile shown in Figure C.4 holds for a wide range of γ: at least from $\gamma = 10^{-5}$ to $\gamma = 0.1$.

Thickened square in $\mathbb{R}^2$. This dataset is the one depicted in Figure C.5. The number of points is $m = 200$.

The behavior for this dataset is similar to the one of the two clusters in $\mathbb{R}^2$. Again there is a region of small PRESS and RK-PRESS around σ_k which is almost invariant to γ. For the RK-PRESS slightly better results are obtained with γ close to 1 .

In a second experiment with this dataset, we increased the σ_k, computing it as the average distance to the eighth nearest neighbor. The results are shown in Figure C.8. For the RK-PRESS, there is a minimum around $\sigma \approx \sigma_k = 0.12$

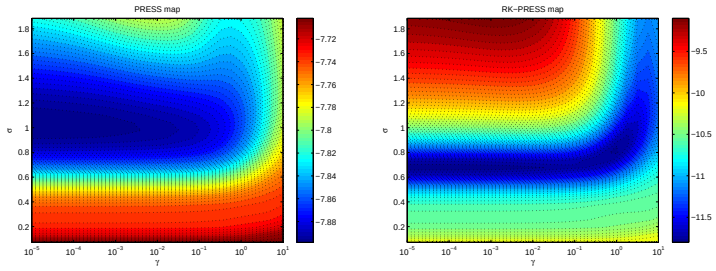

Figure C.3: PRESS and RK-PRESS map for the two Gaussian cluster dataset. The logarithm is taken in both cases.

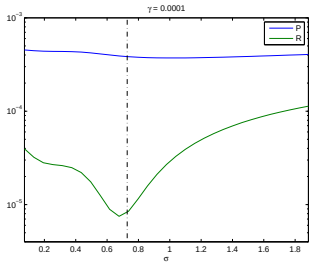

Figure C.4: PRESS and RK-PRESS for the two Gaussian cluster dataset. The logarithm is taken in both cases. The Figure shows the variation with σ_h, for $\gamma = 1\dot{1}0^{-3}$. The vertical black line shows the σ_k.

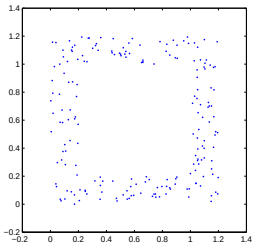

Figure C.5: Thickened square dataset.

and $\gamma \approx 0.4$. However, the PRESS shows a different overall shape, as can be seen left in Figure C.8.

The experiments so far show that instead of minimizing the PRESS in σ and γ, using directly $\sigma = \sigma_k$ would yield better results, according to the RK-PRESS. Using this σ the regularization parameter is not relevant, at least in the first experiments. In all the experiments, except the last one, choosing λ between 10^{-6} and 10^{-2} gives similar results.

The last case shows a stronger dependency with λ: the PRESS is grows with

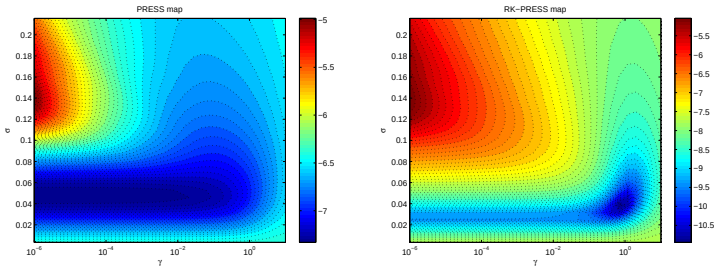

Figure C.6: PRESS and RK-PRESS map for the thickened square cluster dataset. The logarithm is taken in both cases.

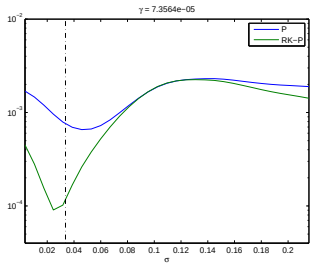

Figure C.7: PRESS and RK-PRESS for the thick square dataset. The logarithm is taken in both cases. The Figure shows the variation with σ_h, for $\gamma = 7.35\dot{1}0^{-5}$. The vertical black line shows the σ_k.

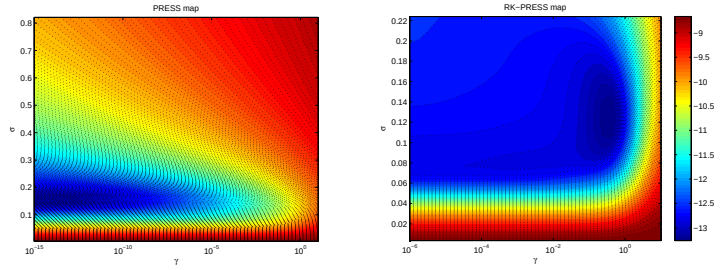

Figure C.8: PRESS and RK-PRESS map for the thickened square cluster dataset, with a larger σ_k. The logarithm is taken in both cases. Note that in the left the ranges of σ and γ are wider.

λ for any fixed σ. However the RK-PRESS behaves as in the previous cases: there is a minimum for λ between 0.1 and 1, and it is almost constant between 10^{-6} and 10^{-2}. This evidence shows that choosing $\sigma_h = \sigma_k$ and $\lambda \in [10^{-6}, 10^{-2}]$ would be a good strategy, even better that minimizing the PRESS.

Appendix D

Pre-images of the polynomial kernel

In this section we present the pre-images developed for the polynomial kernel. This kernel did not yield good results on the kernel PCA de-noising applications, and therefore we did not include these computations in the main body of this work.

The (inhomogeneous) polynomial kernel is defined as

$$k(x, x') = (\langle x, x' \rangle^d + c) \tag{D.1}$$

Note that the dot product $\langle x, x' \rangle$ is the input space dot product. The polynomial kernel (as well as the sigmoid kernel) is one of the following class of kernels:

$$k(x, x') = g(\langle x, x' \rangle) \tag{D.2}$$

where in the case of the polynomial kernel, the function g is given by

$$g(s) = (s + c)^d \tag{D.3}$$

D.1 Collinearity criterion

Iterative methods. We are going to consider our pre-image $x \in \mathcal{X}$ as the a maximizer of the collinearity criterion,

$$x = \arg\max_{x \in \mathcal{X}} \frac{\langle \psi, \varphi(x) \rangle_{\mathcal{H}}}{\|\varphi(x)\|_{\mathcal{H}} \|\psi\|_{\mathcal{H}}} \tag{D.4}$$

As before we are going to assume that ψ is a linear combination of the images of the training set. Substituting in the previous equation, expanding the dot product and using the kernel trick yields,

$$\frac{\langle \psi, \varphi(x) \rangle_{\mathcal{H}}}{\|\varphi(x)\|_{\mathcal{H}} \|\psi\|_{\mathcal{H}}} = \frac{\sum_{i=1}^m \alpha_i k(x, x_i)}{\sqrt{k(x, x)} \|\psi\|_{\mathcal{H}}} = \frac{\sum_{i=1}^m \alpha_i g(\langle x, x_i \rangle)}{\sqrt{g(\langle x, x \rangle)} \|\psi\|_{\mathcal{H}}} \tag{D.5}$$

Computing the gradient of the this expression, and making it equal to zero yields also an implicit approach

$$x = \frac{\sum_{i=1}^{m} \alpha_i \frac{g'(\langle x, x_i \rangle)}{g'(\langle x, x \rangle)} x_i}{\sum_{i=1}^{m} \alpha_i \frac{g(\langle x, x_i \rangle)}{g(\langle x, x \rangle)}} \tag{D.6}$$

For the this kernel, as well as for the Gaussian, the iterative formula can be expressed in terms of k, since

$$g'(\langle x, x' \rangle) = d\, g(\langle x, x' \rangle)^{1-1/d} = d\, k(x, x')^{1-1/d} \tag{D.7}$$

Therefore we obtain the following implicit equation:

$$x = \frac{\sum_{i=1}^{m} \alpha_i \frac{k(x, x_i)^{1-1/d}}{k(x, x)^{1-1/d}} x_i}{\sum_{i=1}^{m} \alpha_i \frac{k(x, x_i)}{k(x, x)}} \tag{D.8}$$

Closed form approximation. Following the same idea of [26, 3] we compute a direct formula approximation of the iterative approach, estimating the pre-image's kernel vector $\boldsymbol{k}_x$. Note that for the polynomial kernel we need also the value of $k(x, x)$. As [43] we are going to set $k(x, x) = \|\psi\|^2_{\mathcal{H}}$.

We can approximate Eq. (D.6) using $k(x, x_i) \approx \langle \psi, \varphi(x_i) \rangle_{\mathcal{H}}$ and $k(x, x) \approx \|\psi\|^2_{\mathcal{H}}$. Note that that using this approximations, the denominator of the implicit formula for the collinearity pre-image reduces to one:

$$\frac{\sum_{i=1}^{m} \alpha_i k(x, x_i)}{k(x, x)} \approx \frac{\sum_{i=1}^{m} \alpha_i \langle \psi, \varphi(x_i) \rangle_{\mathcal{H}}}{\|\psi\|^2_{\mathcal{H}}} = \frac{\langle \psi, \sum_{i=1}^{m} \alpha_i \varphi(x_i) \rangle_{\mathcal{H}}}{\|\psi\|^2_{\mathcal{H}}} = \frac{\langle \psi, \psi \rangle_{\mathcal{H}}}{\|\psi\|^2_{\mathcal{H}}} = 1 \tag{D.9}$$

Then the following closed form approximation yields:

$$x \approx \sum_{i=1}^{m} \alpha_i \frac{\langle \psi, \varphi(x_i) \rangle^{1-1/d}_{\mathcal{H}}}{\|\psi\|^{2-2/d}_{\mathcal{H}}} x_i \tag{D.10}$$

Note that the kernel estimates were deduced minimizing the distance criterion between the Nyström extension and ψ.

D.2 Distance criterion

We choose our pre-image $x \in \mathcal{X}$ to be a minimizer of the distance criterion:

$$x = \arg\min_{x \in \mathcal{X}} \|\psi - \varphi(x)\|^2_{\mathcal{H}} = \arg\min_{x \in \mathcal{X}} \langle \psi - \varphi(x), \psi - \varphi(x) \rangle_{\mathcal{H}} \tag{D.11}$$

If we suppose that ψ lies in the span of the images of the training set $X = \{x_1, \ldots, x_m\}$, we arrive to the following expression for the distance criterion:

$$\begin{aligned}\|\psi - \varphi(x)\|^2_{\mathcal{H}} &= k(x,x) - 2\sum_{i=1}^{m} \alpha_i k(x, x_i) + \|\psi\|^2_{\mathcal{H}} \\ &= g(\langle x, x\rangle_{\mathcal{H}}) - 2\sum_{i=1}^{m} \alpha_i g(\langle x, x_i\rangle) + \|\psi\|^2_{\mathcal{H}} \end{aligned} \tag{D.12}$$

Taking the gradient with respect to the variable x yields

$$\nabla\|\psi - \varphi(x)\|^2_{\mathcal{H}} = 2g'(\langle x, x\rangle)x - 2\sum_{i=1}^{m} \alpha_i g'(\langle x, x_i\rangle)x_i \tag{D.13}$$

If we make the gradient equal to zero, we arrive to the following implicit relation that the pre-image must satisfy:

$$x = \frac{\sum_{i=1}^{m} \alpha_i g'(\langle x, x_i\rangle)x_i}{g'(\langle x, x\rangle)} \tag{D.14}$$

$$x = \sum_{i=1}^{m} \alpha_i \frac{k(x, x_i)^{1-1/d}}{k(x,x)^{1-1/d}} x_i \tag{D.15}$$

The implicit equation for the collinearity criterion is very similar as the one for the distance criterion, the only difference being the factor:

$$\frac{\sum_{i=1}^{m} \alpha_i k(x, x_i)}{k(x,x)} = \frac{\langle \psi, \varphi(x)\rangle_{\mathcal{H}}}{\|\varphi(x)\|^2_{\mathcal{H}}} \tag{D.16}$$

Closed form approximation. Since the factor in which Eqs. (D.8) and (D.15) differs cancels when assuming that $\varphi(x) \approx \psi$, the closed form approximations for both criteria coincide.

Appendix E

Convex Linear Combination Pre-Image

In this section we are going to find a closed form expression for the expansion coefficients of $\psi = \sum_{i=1}^{m} \alpha_i \varphi(x_i)$ in terms of its n nearest neighbors in the empirical feature space $\mathcal{H}^m$:

$$\psi \approx \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \tag{E.1}$$

where $\eta_\psi \subset \{1, \ldots, m\}$ is the set of the indices of the nearest neighbors of ψ.

First not that to compute the distances in the feature space one can use the kernel trick:

$$\|\psi - \varphi(x_i)\|_{\mathcal{H}}^2 = (\boldsymbol{\alpha}_\psi - \boldsymbol{e}_i)^T \boldsymbol{K} (\boldsymbol{\alpha}_\psi - \boldsymbol{e}_i) \tag{E.2}$$

where $\boldsymbol{e}_i$ is a vector with 1 in its ith component and 0s in the rest.

Since we are looking for a convex linear combination, the problem to solve is the following,

$$\begin{aligned} \boldsymbol{\beta} &= \underset{\boldsymbol{\beta} \in \mathbb{R}^n}{\arg\min} \left\| \psi - \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \right\|_{\mathcal{H}}^2 \\ &= \underset{\boldsymbol{\beta} \in \mathbb{R}^n}{\arg\min} \left\| \sum_{j=1}^{m} \alpha_i \varphi(x_i) - \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \right\|_{\mathcal{H}}^2 \qquad \text{(E.3)} \\ &\text{subject to } \mathbf{1}_{n1}^T \boldsymbol{\beta} = 1 \qquad \text{(E.4)} \end{aligned}$$

Expressing the norm as a dot product, expanding it and using the kernel trick again, the objective function can be expressed as:

$$\left\| \sum_{j=1}^{m} \alpha_i \varphi(x_i) - \sum_{i=1}^{n} \beta_i \varphi(x_{\eta_\psi(i)}) \right\|_{\mathcal{H}}^2 = \boldsymbol{\beta}^T \boldsymbol{K}_{\eta\eta} \boldsymbol{\beta} - 2\boldsymbol{\beta}^T \boldsymbol{K}_{\eta\cdot} \boldsymbol{\alpha} + \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \tag{E.5}$$

where $\boldsymbol{K}_{\eta\eta}$ is the $n \times n$ submatrix of $\boldsymbol{K}$ obtained by removing all rows and columns except those corresponding to the nearest neighbors of ψ; and $\boldsymbol{K}_{\eta\cdot}$

is the $n \times m$ submatrix of obtained removing the rows corresponding to the non-nearest neighbors.

To find the optimum we introduce the Lagrange multiplier μ:

$$L(\boldsymbol{\beta}, \mu) = \boldsymbol{\beta}^T \boldsymbol{K}_{\eta\eta} \boldsymbol{\beta} - 2\boldsymbol{\beta}^T \boldsymbol{K}_{\eta\cdot} \boldsymbol{\alpha} + \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} + 2\mu(\mathbf{1}_{n1}^T \boldsymbol{\beta} - 1) \tag{E.6}$$

Solving for $\boldsymbol{\beta}$ and μ yields:

$$\mu = \frac{1 - \mathbf{1}_{n1}^T \boldsymbol{K}_{\eta\eta}^+ \boldsymbol{K}_{\eta\cdot} \boldsymbol{\beta}}{\mathbf{1}_{n1}^T \boldsymbol{K}_{\eta\eta}^+ \mathbf{1}_{n1}} \tag{E.7}$$

$$\boldsymbol{\beta} = \boldsymbol{K}_{\eta\eta}^+ (\boldsymbol{K}_{\eta\cdot} \boldsymbol{\beta} + \mu \mathbf{1}_{n1}) \tag{E.8}$$

Appendix F

Gradient of the kernel PCA projection with the kernel expansion framework

The constraints of the constrained pre-image are given by the kernel PCA projections over the first p principal components. This can be computed in terms of the kernel vector as

$$\boldsymbol{y}_{x,p} = \boldsymbol{\Lambda}_{1:p1:p}^{-1/2} \boldsymbol{U}_{1:p\cdot}^{T} \boldsymbol{k}_{X,x} \tag{F.1}$$

where $\Lambda_{1:p1:p}$ is a square submatrix formed with the first p rows and columns of Λ and $\boldsymbol{U}_{1:p\cdot}^{T}$ is the $p \times m$ matrix formed with the first p rows of $\boldsymbol{U}$.

We are going to compute the Jacobian matrix of $\boldsymbol{y}_{x,p}$ since its rows are the gradients of its functional components.

$$\boldsymbol{\nabla y}_{x,p} = \boldsymbol{\Lambda}_{1:p1:p}^{-1/2} \boldsymbol{U}_{1:p\cdot}^{T} \boldsymbol{\nabla k}_{X,x} \tag{F.2}$$

Recall that $\boldsymbol{k}_{X,x} = \boldsymbol{B}\boldsymbol{h}_x + \boldsymbol{b}$, where $\boldsymbol{B}$ and $\boldsymbol{b}$ are the coefficients of the kernel expansion computed by the Kernel Ridge Regression algorithm, and $\boldsymbol{h}_x$ is the auxiliary Gaussian kernel vector of width σ_h.

$$\boldsymbol{\nabla k}_{X,x} = \boldsymbol{B}\mathrm{Diag}(\boldsymbol{h}_x)(\boldsymbol{X} - x\boldsymbol{1}_{1m})^T \tag{F.3}$$

Appendix G

Notation

$\mathcal{X}$	Input space
d	Dimension of the input space
$\mathcal{H}$	Feature (Kernel) space
φ	Kernel mapping $\varphi : \mathcal{X} \to \mathcal{H}$ such that $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$
x_i	Training points
m	Number of training points
X, X_m	Training set $X = \{x_1, \ldots, x_m\}$. X_m indicates the number of points
Φ	Mapping of the training set $\Phi = \{\varphi(x_1), \ldots, \varphi(x_m)\}$
k	Kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
$\boldsymbol{k}_x$	Kernel vector from x towards the training set $\boldsymbol{k}_x = [k(x, x_1), \ldots, k(x, x_m)]^T$
$\boldsymbol{k}_{X,x}, \boldsymbol{k}_{X_m,x}$	Kernel vector for a data dependent kernel $k_{X,x}$
$\boldsymbol{K}$	Kernel matrix. $\boldsymbol{K}_{ij} = k(x_i, x_j)$ with $i, j = 1, \ldots, m$
$\boldsymbol{K}_X, \boldsymbol{K}_{X_m}$	Kernel matrix w.r.t. training set X, X_m
$\lambda_i, \boldsymbol{u}_i$	ith eigenvalue, eigenvector pair of $\boldsymbol{K}$, in decreasing order of the eigenvalue
$\boldsymbol{\Lambda}, \boldsymbol{U}$	Eigenvalues and eigenvector matrices of $\boldsymbol{K}$
r	Rank of $\boldsymbol{K}$
σ, σ_k	Scale parameter of Gaussian or Diffusion Maps kernels
h	Auxiliary kernel used for extending data dependent kernels
σ_h	Scale parameter of the Gaussian auxiliary kernel
α_i	Coefficients of a linear combination in the Feature Space
$\boldsymbol{\alpha}$	Vector of coefficients
q	Number of principal components
p_i	ith principal component
$\mathbb{P}_q \varphi(x)$	Projection of $\varphi(x)$ over the span$\{p_1, \ldots, p_q\}$
$\boldsymbol{y}_x$	Vector with the kernel PCA projections of $\varphi(x)$ ($\boldsymbol{y}_x \in \mathbb{R}^m$)
$\varphi_R, \mathcal{H}_R$	Reproducing Kernel Hilbert Space mapping and Feature Space
$\varphi_M, \mathcal{H}_M$	Mercer's mapping and Feature Space
$p(x)$	Probability density function of the input space
T_k	Linear operator associated to the kernel $T_k : L_2 \to L_2$
γ_j, φ_j	jth eigenvalue-eigenfunction pair of T_k

$\mathcal{H}^m, \mathcal{H}^m(X)$	Empirical kernel Feature Space $\mathcal{H}^m = \text{span}\{\varphi(x_1), \ldots, \varphi(x_m)\}$
ξ, ψ	Elements in $\mathcal{H}$ or $\mathcal{H}^m$
$\boldsymbol{\alpha}_\xi$	$\boldsymbol{\alpha}$-vector representation of $\xi \in \mathcal{H}^m$
$\boldsymbol{k}_\xi$	kernel vector representation of $\xi \in \mathcal{H}^m$. $\boldsymbol{k}_x$ is a short notation for $\boldsymbol{k}_{\varphi(x)}$
$\boldsymbol{y}_\xi$	Kernel PCA representation of $\xi \in \mathcal{H}^m$. $\boldsymbol{y}_x$ is a short notation for $\boldsymbol{y}_{\varphi(x)}$
φ_m	Empirical kernel map: $\varphi_m(x) = \mathbb{P}_{\mathcal{H}^m}\varphi(x)$
k_X, k_{X_m}	Data dependent kernel functions: for a given X, $k_X : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
z_i	Low dimensional representation of x_i
Z, Z_m	Set of low dimensional representations of the training set X
β_i, b	Coefficients of the kernel expasion for extending a data dependent kernel
$\boldsymbol{\beta}$	Extension coefficients vector (only the β_i's)

Bibliography

[1] D. M. Allen. The relationship between variable selection and data augmentation and a method for prediction. *Technometrics*, 16:125–7, 1974.

[2] P. Arias, G. Randall, and G. Sapiro. Errata on Arias *et al.* `http://iie.fing.edu.uy/~parias/errata`.

[3] P. Arias, G. Randall, and G. Sapiro. Connecting the out-of-sample and pre-imagen problems in kernel methods. In *Proc. of the IEEE Conference on CVPR*, 2007.

[4] C. Baker. *The numerical treatment of integral equations*. Clarendon Press, Oxford, 1977.

[5] G.H. Bakır, J. Weston, and B. Schölkopf. Learning to find pre-images. In *Advances in NIPS 16*, 2004.

[6] G.H. Bakır, A. Zien, and K. Tsuda. Learning to find graph pre-images. In *Pattern Recognition: Proceedings of the 26th DAGM Symposium*, 2004.

[7] G. Baudat and F. Anuar. Kernel-based methods and function approximation. In *Proceedings IJCNN*, volume 2, pages 1244–1249, 2001.

[8] M. Belkin and P.Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in NIPS 14*, 2002.

[9] M. Belkin, V. Sindhwani, and P. Niyogi. Manifold regularization: a geometric framework for learning from examples. Technical report, University of Chicago, 2004.

[10] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16(10):2197–2219, 2004.

[11] Y. Bengio, J-F. Paiement, and P. Vincent. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps and Spectral Clustering. In *Advances in NIPS 16*, 2004.

[12] G. Blanchard, O. Bousquet, and L. Zwald. Statistical properties of kernel Principal Components Analysis. *Machine Learning*, 66(2-3):259–294, 2007.

[13] B. E. Boser, I. M. Guyon, and V. Vapnik. A training algotithm for optimal margin classifiers. In *Proc. of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, ittsburgh, PA, 1992. MIT Press.

[14] M. Brand. Charting a manifold. In *Advances in NIPS 16*, 2002.

[15] M. L. Braun. *Spectral Properties of the Kernel Matrix and their Relation to Kernel Methods in Machine Learning*. PhD thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, 2005.

[16] M. Brookes. The matrix reference manual. Online `http://www.ee.ic.ac.uk/hp/staff/dmb/matrix/intro.html`, 2005.

[17] G. C. Cawley and N. L. Talbot. Reduced rank kernel ridge regression. *Neural Processing Letters*, 16:293–302, 2002.

[18] G. C. Cawley and N. L. Talbot. Fast exact leave-one-out cross-validation of sparse least-squares support vector machines. *Neural Networks*, 17(10):1467–75, 2004.

[19] F. R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (CBMS Regional Conference Series in Mathematics)*. American Mathematical Society, 1997.

[20] R. R. Coifman and S. Lafon. Diffusion maps. *Applied Computational Harmoninc Analysis. Special Issue on Diffusion Maps and Wavelets*, 21:5–30, July 2006.

[21] R. R. Coifman and S. Lafon. Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions. *Applied Computational Harmoninc Analysis. Special Issue on Diffusion Maps and Wavelets*, 21:31–52, July 2006.

[22] J. A. Costa and A. O. Hero. Geodesic entropic graphs for dimension and entropy estimation in manifold learning. *IEEE Trans. on Signal Processing*, 52(8):2210–2221, 2004.

[23] T. Cox and M. Cox. *Multidimensional Scaling*. Chapman & Hall, London, 1994.

[24] D. Cremers. Dynamical statistical shape priors for level set based tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1262–1273, August 2006.

[25] D. Cremers, T. Kohlberger, and C. Schnörr. Nonlinear shape statistics via kernel spaces. In *Pattern Recognition (Proc. DAGM)*, volume 2191 of *LNCS*, pages 269–276, Munich, Germany, Sept. 2001. Springer.

[26] S. Dambreville, Y. Rathi, and A. Tannenbaum. Statistical shape analysis using kernel PCA. In *IS&T/SPIE Symposium on Electronic Imaging*, 2006.

[27] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–75, 2005.

[28] A. Elgammal. Nonlinear generative models for dynamic shape and dynamic appearance. In *Proc. of the 2004 Conference on CVPR Workshop*, volume 12, 2004.

[29] P. Etyngier, F. Ségonne, and R. Keriven. Shape priors using manifold learning techniques. In *Proc. of the 25th ICCV*, 2007.

[30] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE Transactions on PAMI*, 25(2):214–225, 2004.

[31] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.

[32] J. Gower. Adding a point to vector diagrams in multivariate analysis. *Biometrika*, 55(3):582–585, November 1968.

[33] T. Graepel. Kernel matrix completion by semidefinite programming. In *Artificial Neural Networks - ICANN*, 2002.

[34] J. H. Ham, D. D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. In *Proc. of the 21st ICML*, 2004.

[35] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406):502–16, 1989.

[36] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001.

[37] H. Hoffmann. *Unsupervised Learning of Visuomotor Associations*. PhD thesis, Universitt Bielefeld, Technische Fakultt, 2005.

[38] H. Hoffmann. Kernel PCA for novelty detection. *Pattern Recognition*, 40(3):863–74, 2007.

[39] I. T. Jolliffe. *Principal Component Analysis*. Springer, New York, 2nd edition, 2002.

[40] P. Khurd, S. Baloch, R. Gur, C. Davatzikos, and R. Verma. Manifold learning techniques in image analysis of high-dimensional diffusion tensor magnetic resonance images. In *Proc. of the IEEE Conference on CVPR*, 2007.

[41] V. Koltchinskii. Asymptotics of spectral projections of some random matrices approximating integral operators. *Progress in Probability*, 43:191–227, 1998.

[42] V. Koltchinskii and E. Giné. Random matrix approximation of spectra of integral operators. *Bernoulli*, 6(1):113–167, 2000.

[43] J. T. Kwok and I.W. Tsang. The pre-image problem in kernel methods. *IEEE Transactions on Neural Networks*, 15(6):1517–1525, 2004.

[44] S. Lafon. *Diffusion Maps and Geometric Harmonics*. PhD thesis, Yale University, 2004.

[45] S. Lafon, Y. Keller, and R. R. Coifman. Data fusion and multicue data matching by diffusion maps. *IEEE Transactions on PAMI*, 28(11):1784–1797, November 2006.

[46] E. Levina and P.J. Bickel. Maximum likelihood estimation of intrinsic dimension. In *Advances in NIPS 17*, 2005.

[47] S. Mika, B. Schökopf, A. Smola, K. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In *Advances in NIPS 10*, 1998.

[48] A. Y. Ng, M.I Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in NIPS 14*, 2001.

[49] E. J. Nyström. Über die praktische Auflösung von Integralgleichungen mit Anwendung auf Randwertaufgaben. *Acta Mathematica*, 16:185–204, 1930.

[50] V. Roth. The generalized LASSO. *IEEE Trans. on Neural Networks*, 15(1), 2004.

[51] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[52] L. K. Saul, K. Q. Weinberger, J. H. Ham, F. Sha, and D. D. Lee. Spectral methods for dimensionality reduction. In B. Schölkopf, O. Chapelle, and A. Zien, editors, *Semisupervised Learning*. MIT Press, 2006.

[53] B. Schölkopf and A. Smola. *Learning with Kernels. Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, USA, 2002.

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

[55] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. *Advances in kernel methods: support vector learning*, pages 327–352, 1999.

[56] J. Shawe-Taylor and N. Cristianini. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[57] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, MA, USA, 2004.

[58] J. Shawe-Taylor, C. Williams, N. Cristianini, and J. Kandola. On the eigenspectrum of the Gram matrix and the generalization error of kernel PCA. *IEEE Transactions on IT*, 51(7):2510–22, 2005.

[59] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on PAMI*, 22(8):888–905, 2000.

[60] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[61] J. Sun, S. Boyd, L. Xiao, and P. Diaconis. Fastest mixing of markov chain on a graph and a connection to a maximum variance unfolding problem. *SIAM Review*, 48(4):681–99, 2006.

[62] Y. Teh and S. Roweis. Automatic alignment of local representations. In *Advances in NIPS 15*, 2003.

[63] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[64] K. Tsuda, S. Akaho, and K. Asai. The em algorithm for kernel matrix completion with auxiliart data. *Journal of Machine Learning Research*, 4:67–81, 2003.

[65] R. Urtasun, D. Fleet, and P. Fua. 3d people tracking with gaussian process dynamical models. In *Conference on Computer Vision and Pattern Recognition*, June 2006.

[66] L. Vandenberghe and S. Boyd. *Semidefinite Programming*. SIAM Review 38, 1996.

[67] K. Q. Weinberger, B. D. Packer, and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proc. of the 10th International Workshop on Artificial Intelligence and Statistics*, Barbados, January 2005.

[68] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proc. of the IEEE Conference on CVPR*, volume 2, pages 988–995, Washington D.C., 2004.

[69] K. Q. Weinberger and L. K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[70] K. Q Weinberger, F. Sha, Q. Zhu, and L. K. Saul. Graph laplacian methods for large-scale semidefinite programming, with an application to sensor localization. In *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[71] Y. Weiss. Segmentation using eigenvectors: a unifying view. In *Proc. of the 17th ICCV*, pages 975–982, 1999.

[72] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in NIPS 13*, 2001.

[73] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. Technical Report CSE-02-019, Department of Computer Science & Engineering, Pennsylvania State University, 2002.

[74] W.-S. Zheng and J.-H. Lai. Regularized locality preserving learning of pre-image problem in kernel PCA. In *Proceedings of the 18th ICPR*, volume 2, 2006.

[75] W.-S. Zheng, J.-H. Lai, and P.C. Yuen. Weakly supervised learning on pre-image problem in kernel methods. In *Proceedings of the 18th ICPR*, volume 2, 2006.