A survey on Kernel PCA

Theoretical project for the Numerical Analysis for Machine Learning course

Luca Muscarnera

November 14, 2024

1 Introduction

1.1 Abstract

In recent years, the increase of importance of automated learning strategies - and the consequent physiological increment of complexity in data - required the construction of solid techniques in order to extract meaningful information from datasets. In fact, while the datum itself is certainly the building block of every statistical learning procedure, the intrinsic structure of data cloud may hide precious information regarding the phenomena of interest. Being able to characterize, in a sense, the nature of data (and the dynamics behind its generation) is of paramount importance in order to explain certain observed properties.

The main tool to extract this kind of structure from the data is the class of algorithms known as dimensionality reduction.

Dimensionality reduction is the transformation of high-dimensional data into a meaningful representation of reduced dimensionality. Ideally, the reduced representation should have a dimensionality that corresponds to the intrinsic dimensionality of the data, which may be defined as the minimum number of parameters needed to account for the observed properties of the data [1]

Note that, for the sake of generalization, we are talking about an abstract framework. We would like to decouple, in fact, the notion of dimensionality reduction from the specificity of each algorithm, highlighting the red thread that binds all the known techniques.

Among the possible different dimensionality reduction techniques, we will describe the Kernel PCA Algorithm, gently starting from the reasons behind its adoption and the origins of the method, describing the mathematical tools needed to explain in a formal and exhaustive way the algorithm and finally analyzing its possible usages, proving - with concrete use cases - the power of the presented instrument. We will also investigate the mathematics behind the method in a constructive manner; not reducing the narrative to the usage of mathematics as justification but instead using mathematics as a tool to reconstruct the method from its very fundations.

* * *

2 Dimensionality reduction in a Linear framework

2.1 Dimensionality reduction and Curse of Dimensionality [CoD]

In the introduction we presented the problem of dimensionality reduction as a resource for extracting important information from data, but we omitted how to characterize in a rigorous mathematical way this notion of importance, and in particular how reducing dimensionality of data actually helps in this task. To tackle this problem, we try to propose to following mathematical experiment which should make more clear how dimensions affect the "distribution of information" among the features, and therefore our ability to extract (and abstract) properties starting from data.

Among the simplest classification techniques available, K-NN plays a major role due to its intuitive interpretation and its non parametric nature. The main intuition behind this algorithm is that two objects that are near in the space of data are somehow related in terms of properties that characterize them. In 1999 the work of Beyer et al. [2] proved how, when dimensions start to increase, the significance of "being the closest" starts to vanish, since the "deviation" between closest distance and farthest distance shrinks.

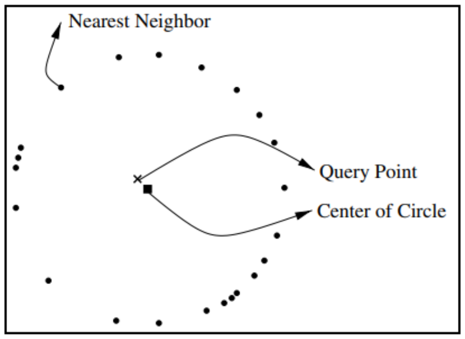


Figure 1: Visualization of the loss of significance of the notion of nearest neighbour, taken from *When Is "Nearest Neighbor" Meaningful?*[2]

Here, a simplified version of the aforementioned experiment is presented.
Let us, as a first step, introduce a metric to define a deviation among distances

Definition 2.1 (Distance deviation)**.** Given a collection of positive reals $\mathcal{D} = \{d_1, d_2, ..., d_n\}$, we can describe each number as its relative deviation from a reference distance in the following way

$$\sigma_{\bar{d}}(d) = \frac{d-\bar{d}}{\bar{d}}$$

where $\bar{d}$ denotes the reference distance, for example the sample mean distance in the collection $\mathcal{D}$ or the mean of the statistical distribution from where $\mathcal{D}$ is sampled.

Now, let us define a family of distributions $\mathscr{G} = \{\mathcal{G}_k\}_k$ with $\mathcal{G}_k = \mathcal{N}(\mathbf{0}_k, I_k)$ being the k-variate normal distribution. Let us define, now, the family of random variables $\{X_k\}_k$ and $\{Q_k\}_k$ such that $X_k, Q_k \sim \mathcal{G}_k$.
Consequentially, we define the family of random variables $\{Y_k\}_k$ such that $Y_k = \sigma_{\mathbb{E}[||X_k-Q_k||^2]}(||X_k - Q_k||^2)$

Theorem 1. Let $\alpha_n := var(Y_n)$ the sequence of variances of the aforementioned family of random variables. Then $lim_{n\to\infty}\alpha_n = 0$

Proof. (Sketch)

$$\begin{aligned}
&\alpha_n = \\
&= var(\tfrac{||X_n-Q_n||^2-\mathbb{E}[||X_n-Q_n||^2]}{\mathbb{E}[||X_n-Q_n||^2]}) \\
&= var(\tfrac{||X_n-Q_n||^2}{\mathbb{E}[||X_n-Q_n||^2]} - 1) \\
&= var(\tfrac{||X_n-Q_n||^2}{\mathbb{E}[||X_n-Q_n||^2]}) \\
&= \tfrac{1}{\mathbb{E}[||X_n-Q_n||^2]^2} var(||X_n - Q_n||^2) \\
&= \tfrac{1}{\mathbb{E}[\sum_i^n (x_i^{(n)}-q_i^{(n)})^2]^2} var(\textstyle\sum_i^n (x_i^{(n)} - q_i^{(n)})^2)
\end{aligned}$$

Since $\sum_i^n (x_i^{(n)} - q_i^{(n)})^2 \sim 2\sum_i^n (\mathcal{N}(0,1))^2 \sim 2\chi^2(n)$ and $\mathbb{E}[2\chi^2(n)] = 2n$, $var[2\chi^2(n)] = 8n$, we obtain

$$\begin{aligned}
&= \tfrac{8n}{4n^2} \\
&= \tfrac{2}{n}
\end{aligned}$$

which goes to 0 for $n \to \infty$ □

This basic examples offers a pedagogic perspective on our argumentation; when dimensionality increases extracting information becomes harder.

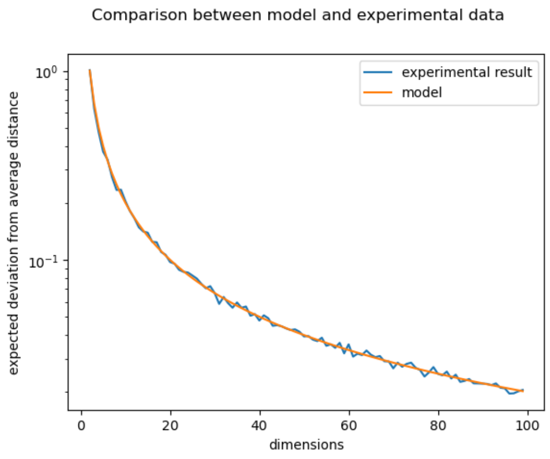


Figure 2: In this visualization an experiment was conducted to validate the previous theoretical result ; in orange we have the predicted value of the variance in deviation, while in blue the sample variance of the deviation

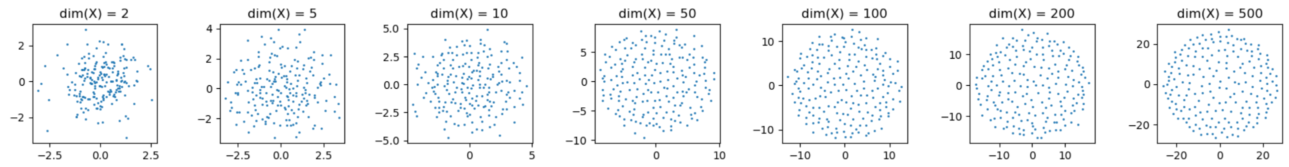


Figure 3: Here a visualization of the previous result is presented. Performing a Multidimensional Scaling [3] on normal samples in increasing dimension, the variability in the distances decreases, as can be noticed from the tendencies of data points to show low discrepancy [4]

2.2 PCA, a revisitation

Now that we exemplified the reasons behind the advantages of dimensionality reduction, we will try to offer a different perspective on the PCA algorithm, highlighting its implications and explaining how it perfectly fits our characterization of dimensionality reduction algorithms. In particular, let us recall the idea of treating a dimensionality reduction algorithm as a tool to produce a "denser" (in the sense of cardinality of features) dataset, still able to preserve certain properties, which – from a broad perspective – make it more similar to the original one. PCA pursues this idea, making a simple assumption ; if the new reduced dataset has its variance maximized, then it is offering a better description of the original dataset, or to be consistent with our tractation "it is more similar to it". Now we prove that maximizing variance in the reduced dataset minimizes the reconstruction error

Proof. (Analogy between variance maximization and error minimization and eigenvectors of the Covariance matrix.) Consider the following optimization problem

$$\text{Find } \mathbf{v} \text{ such that minimizes } \sum_i^N ||\mathbf{x^{(i)}} - \mathbf{v}^T\mathbf{x^{(i)}}\mathbf{v}||_2^2 \\ \text{Subject to } ||\mathbf{v}|| = 1$$

which aims to reduce the reconstruction error. We can exploit the definition of $||\cdot||_2^2$ to rerite it as

$$\text{Find } \mathbf{v} \text{ such that minimizes } \sum_i^N (\mathbf{x^{(i)}} - \mathbf{v}^T\mathbf{x^{(i)}}\mathbf{v})^T(\mathbf{x^{(i)}} - \mathbf{v}^T\mathbf{x^{(i)}}\mathbf{v}) \\ \text{Subject to } ||\mathbf{v}|| = 1$$

And from there, we can develop the calculations as

$$\text{Find } \mathbf{v} \text{ such that minimizes } \sum_i^N \mathbf{x^{(i)}}^T\mathbf{x^{(i)}} - 2\sum_i^N \mathbf{x^{(i)}}^T\mathbf{v}^T\mathbf{x^{(i)}}\mathbf{v} + \sum_i^N \underbrace{\mathbf{v}^T\mathbf{v}}_{||\cdot||\,=\,1} \mathbf{x^{(i)}}^T\mathbf{v}\mathbf{v}^T\mathbf{x^{(i)}} \\ \text{Subject to } ||\mathbf{v}|| = 1$$

And then

$$\text{Find } \mathbf{v} \text{ such that minimizes } \sum_i^N ||\mathbf{x^{(i)}}||_2^2 - \sum_i^N (\mathbf{x^{(i)}}^T\mathbf{v})^2 \\ \text{Subject to } ||\mathbf{v}|| = 1$$

Which is equivalent, due to indipendence of the term $||\mathbf{x^{(i)}}||_2^2$ from $\mathbf{v}$

$$\text{Find } \mathbf{v} \text{ such that maximizes } \sum_i^N ({\mathbf{x^{(i)}}}^T\mathbf{v})^2 \quad (\star)$$
$$\text{Subject to } ||\mathbf{v}|| = 1$$

Which is the variance of the projected point (due to the fact that projection of a zero mean dataset on a vector is linear). Finally we can rewrite $(\star)$ as

$$\sum_i^N ({\mathbf{x^{(i)}}}^T\mathbf{v})^2 = \sum_i^N \mathbf{v}^T\mathbf{x^{(i)}}{\mathbf{x^{(i)}}}^T\mathbf{v} = \mathbf{v}^T\left(\sum_i^N \mathbf{x^{(i)}}{\mathbf{x^{(i)}}}^T\right)^T\mathbf{v} \propto \underbrace{\mathbf{v}^T C(X)\mathbf{v}}_{\heartsuit}$$

with C(X) being the covariance matrix of the dataset. Hence, we also proved that finding the most significative eigenvector of the covariance matrix returns the first principal component, since its a well know fact from theory that maximizing $\heartsuit$ with the constraint of $\mathbf{v}$ having unitary norm is a problem equivalent to the largest eigenvalue problem. $\square$

2.3 World is not linear

While Eckart-Young theorem guarantees that PCA Projection offers the best linear dimensionality reduction (in terms of reconstruction error) it's fundamental to understand that this result is confined to the realm of linear methods.
The dynamics behind the generation of data in real world could be very far from being linear, thus the inability of classical PCA to capture non linear mechanics behind real world data.
In order to grasp the complexity of real world, dimensionality reduction methods should be able to incorporate this kind of complexity in some way.
In the following theoretical chapter we are going to describe the well-known Kernel Trick, an advanced machine learning technique to treat nonlinearity in a linear fashion, allowing us to extend the boundaries of the domain of linear techniques.

* * *

3 Kernel Trick

3.1 Feature Space Maps and Nonlinear Functionals

In order to introduce non linearity in a linear model, it is possible to use the so called **feature space maps**.

Definition 3.1. Let $\mathcal{X}, \mathcal{X}'$ be a couple of vector spaces such that $dim(\mathcal{X}) < dim(\mathcal{X}')$.A **feature space map** $\phi : \mathcal{X} \to \mathcal{X}'$ is a bijection that maps the first space in an higher dimensional vector space.

Again we preferred to give an abstract point of view for the sake of generality. Nonetheless the utility of this kind of mechanism is better understood with an example.
Consider, for instance, a set $\mathcal{X} \subseteq \mathbb{R}^2$; due to Riesz representation theorem every linear functional acting on $\mathbf{x} \in \mathcal{X}$ can be written as $L\mathbf{x} = w_1x_1 + w_2x_2 = \mathbf{w}^T\mathbf{x}$ with $\mathbf{w} \in \mathbb{R}^2$. Let us now define the feature space map ϕ as $\mathbf{x} \mapsto [1|\mathbf{x}]^T$, that is, roughly speaking, the map that concatenate a 1 to the vector.We will denote as $\mathcal{X}' \subset \mathbb{R}^3$ the image of $\mathcal{X}$ through ϕ.
Again, due to Riesz representation theorem every linear functional acting on $\mathbf{x}' \in \mathcal{X}'$ can be written as $L\mathbf{x}' = L\phi(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 = \mathbf{w}^T\mathbf{x}'$ with $\mathbf{w} \in \mathbb{R}^3$ and $\mathbf{x}$ the vector in $\mathcal{X}$ such that $\phi(\mathbf{x}) = \mathbf{x}'$. It is immediate to see how any linear functional defined on $\mathcal{X}'$ can be interpreted a nonlinear functional acting on $\mathcal{X}$; in fact $L[\cdot] = \mathbf{w}^T\phi(\cdot)$ acting on $\mathcal{X}$ is the composition of the maps $L \circ \phi$ which is non linear on the starting feature space. This idea, which alone introduce the possibility of applying linear models and methods in a non linear fashion, is still not a full description of the whole picture; the dimension of the image of the data space through the feature map could make in fact unfeasible the explicit computation of the embedding. In order to overcome this difficulty, the ideal situation would be represented by the possibility of performing computations that involve the images of vectors through the feature space map without the explicit need of representing those images, which would imply greater computational cost due to memory representation and inner product, which -in general- scales linearly with the size of vector representation.

3.2 The Trick

In many algorithms, like Ridge Linear Regression, we could operate in fact without an explicit representation of data but only figuring out how the scalar product behaves in the space where the dataset is represented. To exemplify this result we prove this statement in the context of ridge linear regression, highlighting scalar products as bilinear maps $\langle\cdot,\cdot\rangle$ from the cartesian square of the space of data to $\mathbb{R}$.

Proof. (Kernel Trick) Let $X = U\Sigma V^T$, as result of Singular Value Decomposition.
Let X_{test} the dataset on which we may want to perform prediction.

$$
\begin{aligned}
&X_{test}\left(X^TX+\lambda I\right)^{-1}X^T = \\
&= X_{test}\left(V\Sigma U^TU\Sigma V^T+\lambda V^TV\right)^{-1}V\Sigma U^T \\
&= X_{test}\left(V\Sigma^2V^T+\lambda V^TV\right)^{-1}V\Sigma U^T \\
&= X_{test}\left(V(\Sigma^2+\lambda I)V^T\right)^{-1}V\Sigma U^T \\
&= X_{test}V(\Sigma^2+\lambda I)^{-1}V^TV\Sigma U^T \\
&= X_{test}V(\Sigma^2+\lambda I)^{-1}\Sigma U^T \\
&= X_{test}V\Sigma(\Sigma^2+\lambda I)^{-1}U^T \\
&= X_{test}V\Sigma U^TU(\Sigma^2+\lambda I)^{-1}U^T \\
&= X_{test}V\Sigma U^T{U^T}^{-1}(\Sigma^2+\lambda I)^{-1}U^{-1} \\
&= X_{test}V\Sigma U^T\left(U(\Sigma^2+\lambda I)U^T\right)^{-1} \\
&= X_{test}V\Sigma U^T\left(U\Sigma^2U^T+\lambda I\right)^{-1} \\
&= X_{test}V\Sigma U^T\left(U\Sigma V^TV\Sigma U^T+\lambda I\right)^{-1} \\
&= X_{test}X^T\left(XX^T+\lambda I\right)^{-1}
\end{aligned}
$$

Hence $\hat{\mathbf{Y}} = X_{test}\left(X^TX+\lambda I\right)^{-1}X^T\mathbf{Y} = X_{test}X^T\left(XX^T+\lambda I\right)^{-1}\mathbf{Y}$ and we can derive

$$\hat{Y}_i = \sum_k \left(X_{test}X^T\right)_{ik}\left[\left(XX^T+\lambda I\right)^{-1}\mathbf{Y}\right]_k$$

By definition $\left(X_{test}X^T\right)_{ik} = \langle \mathbf{x}^{(\mathbf{i})}_{\mathbf{test}}, \mathbf{x}^{(\mathbf{k})}\rangle$ and the action of the matrix $\left(XX^T+\lambda I\right)$ on a generic vector $\mathbf{z}$ (that is the only needed information in order to solve the linear system induced by $\left(XX^T+\lambda I\right)^{-1}\mathbf{Y}$) can be expressed as $[\left(XX^T+\lambda I\right)\mathbf{z}]_i = \sum_k \left(XX^T\right)_{ik}z_k + \lambda z_i = \sum_k\langle \mathbf{x}^{(\mathbf{i})}, \mathbf{x}^{(\mathbf{k})}\rangle z_k + \lambda z_i$ proving how computing prediction only need the definition of a inner product. $\square$

Now the reasons behind the introduction of this tool may appear clear ; since in this framework there is no need for an explicit representation of the expanded feature space it is possible to overcome the possibly huge computational cost related to representing the image of the dataset through the feature map by just representing instead the scalar product in the new feature space.

To do so we introduce a new mathematical object

Definition 3.2. Let $\mathcal{X}$ be a set and $\mathcal{Y}$ be a Hilbert space, equipped with the inner product $\langle\cdot,\cdot\rangle_{\mathcal{Y}}$. A **kernel** $k:\mathcal{X}\times\mathcal{X}\to\mathbb{R}$ is a function such that $\exists\phi:\mathcal{X}\to\mathcal{Y}$ such that $\forall(x,x')\in\mathcal{X}\times\mathcal{X}$ it is true that $k(x',x) = \langle\phi(x),\phi(x')\rangle_{\mathcal{Y}}$. [5]

For instance, if we consider the feature map $\phi^*(\mathbf{x}) := [1|\mathbf{x}]$ it is possible to represent $\langle\phi^*(\mathbf{x}),\phi^*(\mathbf{y})\rangle$ inheriting the inner product from the space $\mathbb{R}^{dim[\mathbf{x}]+1}$ that is

$$k_{\phi^*}(\mathbf{x},\mathbf{y}) = \langle\phi^*(\mathbf{x}),\phi^*(\mathbf{y})\rangle = [1|\mathbf{x}]^T[1|\mathbf{y}] = 1+\mathbf{x}^T\mathbf{y}$$
(1)

3.2.1 Computational advantage of the kernel approach

To give an intuition on the advantage given by the use of a kernel, it's possible to consider the class of **polynomial kernels** and their implicitly defined feature maps.
Let us define

$$k_2 := \left(1+\mathbf{x}^T\mathbf{y}\right)^2$$

and let ϕ_2 be the induced feature map linked to kernel k_2.
We can then rewrite

$$
\begin{aligned}
&\left(1+\mathbf{x}^T\mathbf{y}\right)^2 \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \left(\mathbf{x}^T\mathbf{y}\right)^2 \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \left(\textstyle\sum_i^n x_iy_i\right)^2 \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \left(\textstyle\sum_i^n x_iy_i\right)\left(\textstyle\sum_j^n x_jy_j\right) \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \textstyle\sum_i^n\sum_j^n x_iy_ix_jy_j \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \textstyle\sum_i^n\sum_j^n x_ix_jy_iy_j \\
&= 1 + 2\ \mathbf{x}^T\mathbf{y} + \textstyle\sum_{(i,j)}\left(\mathbf{x}\otimes\mathbf{x}\right)_{ij}\left(\mathbf{y}\otimes\mathbf{y}\right)_{ij} \\
&= 1 + \sqrt{2}\mathbf{x}\cdot\sqrt{2}\mathbf{y} + \left(\mathbf{x}\otimes\mathbf{x}\right):\left(\mathbf{y}\otimes\mathbf{y}\right) \\
&= \underbrace{1}_{\mathcal{O}(1)}\cdot 1 + \underbrace{\sqrt{2}\mathbf{x}}_{\mathcal{O}(n)}\cdot\sqrt{2}\mathbf{y} + \underbrace{flat\left(\mathbf{x}\otimes\mathbf{x}\right)}_{\mathcal{O}(n^2)}\cdot flat\left(\mathbf{y}\otimes\mathbf{y}\right)
\end{aligned}
$$
(♠)

Starting from ♠ we can imagine the feature map images as vector with dimension of $n^2 + n + 1$, hence scalar product on the explicit feature map would be $\mathcal{O}(n^2)$.
This result can be easily generalized in higher dimension, pointing out that the explicit computation of k_p through the feature map ϕ_p has complexity $\mathcal{O}(n^p)$
A special mention is reserved to the **radial kernel**

$$k_\sigma(\mathbf{x}, \mathbf{y}) = exp\left(-\frac{||\mathbf{x}-\mathbf{y}||^2}{\sigma^2}\right)$$

for which the implicit feature map is infinite dimensional [6] , making kernel methods the only possible way to exploit this special feature map.

* * *

4 A gentle encounter with the Mathematics of Kernel PCA

4.1 Theoretical building blocks

Until now we built the foundations of our description of the kernel PCA Method. We introduced concepts like kernel trick and dimensionality reduction, which pose the basic building blocks necessary to explain in a correct way this algorithm. Nonetheless, in order to "glue" everything together we need something more; we need to build reasoning in an axiomatic fashion, in order to gradually build the logic of the algorithm defending our tractation from possible flaws.
The first results that we are going to present may appear as unrelated, but are – in fact – of great utility in explaining how – and most of all why – Kernel PCA works.

We start presenting a basic result of Linear Algebra

Lemma 2. Let $M \in \mathbb{R}^{N \times N}$ and let $X = \{\mathbf{x^{(k)}}\}_k^q \subset \mathbb{R}^N$ a collection of q vectors such that $M = \sum_k^q \mathbf{x^{(k)}}\mathbf{x^{(k)}}^T$. Then if a vector $\mathbf{v}$ is a non-degenerate eigenvector of M it can be expressed as a linear combination of vectors in X.

Proof. (Sketch)

$$M = \sum_k^q \mathbf{x^{(k)}}\mathbf{x^{(k)}}^T \text{ by Hypothesis}$$
$$M = \sum_k^N \lambda_k \mathbf{v^{(k)}}\mathbf{v^{(k)}}^T \text{ by Spectral Decomposition}$$

Hence

$$\sum_k^q \mathbf{x^{(k)}}\mathbf{x^{(k)}}^T = \sum_k^N \lambda_k \mathbf{v^{(k)}}\mathbf{v^{(k)}}^T$$

Which implies

$$\sum_k^q \mathbf{x^{(k)}}\mathbf{x^{(k)}}^T \mathbf{v^{(x)}} = \sum_k^N \lambda_k \mathbf{v^{(k)}}\mathbf{v^{(k)}}^T \mathbf{v^{(x)}} \quad \forall \mathbf{v^{(x)}} \in eig(M)$$

By orthogonality of eigenvectors we obtain

$$\sum_k^q \mathbf{x^{(k)}}\mathbf{x^{(k)}}^T \mathbf{v^{(x)}} = \sum_k^N \lambda_k \mathbf{v^{(k)}}\delta_{kx} \quad \forall \mathbf{v^{(x)}} \in eig(M)$$
$$\sum_k^q \mathbf{x^{(k)}} \underbrace{\mathbf{x^{(k)}}^T \mathbf{v^{(x)}}}_{\in \mathbb{R}} = \lambda_x \mathbf{v^{(x)}} \quad \forall \mathbf{v^{(x)}} \in eig(M)$$
$$\frac{1}{\lambda_x}\sum_k^q \mathbf{x^{(k)}}\gamma_{xk} = \mathbf{v^{(x)}} \quad \forall \mathbf{v^{(x)}} \in eig(M)$$

Therefore every non degenerate eigenvector of M belongs to $span(X)$ □

This result is fundamental for the following argumentation. In fact, in principle, in order to represent the element of a $N-$dimensional vector space in a field $\mathbb{K}$ we would need N objects in the same field. The possibility to reduce the amount of information necessary to represent eigenvectors (and thus principal components) is crucial for the implementation of the method; in fact, as we have seen in previous discussion, the dimensionality of the feature mapping often grows very rapidly with respect to the expressivity of the map, being even infinite in some peculiar cases [7][8], causing dimension of eigenvectors of covariance matrix to explode.

Now, we introduce another result, that allows us not only to define a way to express the solution of an eigenvalue problem but also a way to solve it even if the dimensionality of the eigenvectors is very high.

Lemma 3. Let X be a dataset of samples $(\mathbf{x^{(1)}}, ..., \mathbf{x^{(N)}})$ such that $\sum_i^N \mathbf{x^{(i)}} = \mathbf{0}$. Let $C(X)$ be the covariance matrix associated with X. Then, in order to check if a vector $\mathbf{v}$ satisfy the problem $C(X)\mathbf{v} = \lambda(\mathbf{v})\mathbf{v}$ is sufficient to check that $\mathbf{x}^T C(X)\mathbf{v} = \lambda(\mathbf{v})\mathbf{x}^T\mathbf{v} \;\; \forall \mathbf{x} \in span(X)$.

Proof. (Sketch)
Let $W = span(X)$ and $W^\perp = \{\mathbf{v} : \mathbf{v}^T\mathbf{w} = 0 \; \forall \mathbf{w} \in span(X)\}$. Every vector $\mathbf{z}$ can be expressed as $\mathbf{z} = \mathbf{z}_W + \mathbf{z}_{W^\perp}$, with $\mathbf{z}_W \in W, \mathbf{z}_{W^\perp} \in W^\perp$ highlighting the two orthogonal components. Suppose that we want to test two vectors $\mathbf{x}, \mathbf{x}'$ such that $\mathbf{x} = \mathbf{a}_W + \mathbf{b}_{W^\perp}$ and $\mathbf{x}' = \mathbf{a}_W + \mathbf{c}_{W^\perp}$, which means that their $W^\perp$ component do not coincide. Hence

$$\begin{aligned}
&\mathbf{x}^T C(X)\mathbf{v} = \\
&= (\mathbf{a}_W + \mathbf{b}_{W^\perp})^T C(X)\mathbf{v} \\
&= \mathbf{a}_W^T C(X)\mathbf{v} + \overbrace{\underbrace{\mathbf{b}_{W^\perp}^T}_{\in W^\perp} \underbrace{C(X)\mathbf{v}}_{\in W}}^{=0} \\
&= \mathbf{a}_W^T C(X)\mathbf{v} \\
&= \mathbf{a}_W^T C(X)\mathbf{v} + \overbrace{\underbrace{\mathbf{c}_{W^\perp}^T}_{\in W^\perp} \underbrace{C(X)\mathbf{v}}_{\in W}}^{=0} \\
&= (\mathbf{a}_W + \mathbf{c}_{W^\perp})^T C(X)\mathbf{v} \\
&= \mathbf{x}'^T C(X)\mathbf{v}
\end{aligned}$$

And in an analogous way

$$\lambda \mathbf{x}^T\mathbf{v} = \lambda \mathbf{x}'^T\mathbf{v}$$

Therefore

$$\mathbf{x}^T C(X)\mathbf{v} = \lambda \mathbf{x}^T\mathbf{v} \iff \mathbf{x}'^T C(X)\mathbf{v} = \lambda \mathbf{x}'^T\mathbf{v}$$

Suppose now to build the equivalence relationship $\sim_*$ such that $\mathbf{x} \sim_* \mathbf{y}$ if and only if their projection on W coincides. The previous proof guarantees us that testing on a vector, tests implicitly also for vectors that are equivalent to it. It's easy to verify that the class of representants of the equivalence relationship coincides with $span(X)$ since every vector is equivalent, in the sense that we defined, to a vector with null $W^\perp$ component. Since we also proved that testing one vector automatically implies the verification of the equality also for vectors which are equivalent to it, we proved that is sufficient to test the equality on $span(X)$. Moreover, due to the linearity of the problem, it is sufficient to test the equality on a **basis** of $span(X)$, which -by definition- may be the set X itself. $\square$

4.2 Construction of the algorithm

Let us now call Φ the dataset obtained mapping each sample of the initial dataset X through a feature map ϕ and $C(\Phi)$ as the covariance matrix of the newly crafted dataset Φ.
We now present a chain of equivalent problems that brings us from the initial eigenvector problem in an high dimensional feature space to a feasible one, that is the one actually solved by the kernel PCA algorithm.

Proof. (Construction of the algorithm)
Starting from the classical eigenvalue problem for the covariance matrix

$$C(\Phi)\mathbf{v} = \lambda\mathbf{v}$$

By **Lemma 3** we obtain

$$\iff \phi(\mathbf{x^{(i)}})^T C(\Phi)\mathbf{v} = \lambda\phi(\mathbf{x^{(i)}})^T\mathbf{v} \quad \forall i$$

Hence, by definition of Covariance Matrix

$$\iff \phi(\mathbf{x^{(i)}})^T \left(\frac{1}{N}\sum_j \phi(\mathbf{x^{(j)}})\phi(\mathbf{x^{(j)}})^T\right)\mathbf{v} = \lambda\phi(\mathbf{x^{(i)}})^T\mathbf{v} \quad \forall i$$

$$\iff \frac{1}{N}\phi(\mathbf{x^{(i)}})^T \left(\sum_j \phi(\mathbf{x^{(j)}})\phi(\mathbf{x^{(j)}})^T\mathbf{v}\right) = \lambda\phi(\mathbf{x^{(i)}})^T\mathbf{v} \quad \forall i$$

Then, expliciting the eigenvalue in the coordinates system induced by the image of the samples through the feature map

$$\iff \frac{1}{N}\phi(\mathbf{x^{(i)}})^T \sum_j \phi(\mathbf{x^{(j)}})\phi(\mathbf{x^{(j)}})^T \underbrace{\left(\sum_k \alpha_k\phi(\mathbf{x^{(k)}})\right)}_{\textbf{Lemma 2}} = \lambda\phi(\mathbf{x^{(i)}})^T \underbrace{\left(\sum_k \alpha_k\phi(\mathbf{x^{(k)}})\right)}_{\textbf{Lemma 2}} \quad \forall i$$

And finally, developing the calculations exploiting the definition of the kernel function

$$\begin{aligned}
&\iff \frac{1}{N}\sum_j\sum_k \alpha_k\phi(\mathbf{x^{(i)}})^T\phi(\mathbf{x^{(j)}})\phi(\mathbf{x^{(j)}})^T\phi(\mathbf{x^{(k)}}) = \lambda\sum_k \alpha_k\phi(\mathbf{x^{(i)}})^T\phi(\mathbf{x^{(k)}}) \quad \forall i\\
&\iff \frac{1}{N}\sum_j\sum_k \alpha_k k(\mathbf{x^{(i)}},\mathbf{x^{(j)}})k(\mathbf{x^{(j)}},\mathbf{x^{(k)}}) = \lambda\sum_k \alpha_k k(\mathbf{x^{(i)}},\mathbf{x^{(k)}}) \quad \forall i\\
&\iff \frac{1}{N}\sum_j\sum_k K_{ij}K_{jk}\alpha_k = \lambda\sum_k K_{ik}\alpha_k \quad \forall i\\
&\iff \frac{1}{N}\sum_j K_{ij}\sum_k K_{jk}\alpha_k = \lambda\sum_k K_{ik}\alpha_k \quad \forall i\\
&\iff \frac{1}{N}\sum_j K_{ij}[K\mathbf{a}]_j = \lambda[K\mathbf{a}]_i \quad \forall i\\
&\iff \frac{1}{N}[KK\mathbf{a}]_i = \lambda[K\mathbf{a}]_i \quad \forall i\\
&\iff KK\mathbf{a} = N\lambda K\mathbf{a}\\
&\iff K^2\mathbf{a} = N\lambda K\mathbf{a}\\
&\iff K\mathbf{a} = N\lambda\mathbf{a}
\end{aligned}$$

□

Solving this eigenvalue problem is hence possible to obtain the coefficients of the linear combination over the images of samples through the feature map that yields the eigenvector of the covariance matrix, which is by definition a principal component.

4.3 Computing projections

We now prove how to use the information obtained with the solution of the eigenvalue problem in order to compute projections over the principal components.

Proof. (Computation of projections)

$$\begin{aligned}&[\Phi \mathbf{v}^{(\mathbf{j})}]_i = \\ &= \phi(\mathbf{x}^{(\mathbf{i})})^T \mathbf{v}^{(\mathbf{j})} \\ &= \phi(\mathbf{x}^{(\mathbf{i})})^T \textstyle\sum_k \alpha_k^{(j)} \phi(\mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} \phi(\mathbf{x}^{(\mathbf{i})})^T \phi(\mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} k(\mathbf{x}^{(\mathbf{i})}, \mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} K_{ik} \\ &= [K\mathbf{a}^{(\mathbf{j})}]_i\end{aligned}$$

Hence

$$\Phi \mathbf{v}^{(\mathbf{j})} = K\mathbf{a}^{(\mathbf{j})}$$

Let us now define a matrix A such that $A_{ij} := \alpha_i^j$, which contains - as columns - the vector of coefficients that express the whole spectrum of eigenvectors. Then, being V the matrix that has as columns the eigenvectors of the covariance matrix we get

$$\Phi V = KA$$

$\square$

This proof, which explain how to project the training set that generated the covariance matrix can be generalized also for computing the projection of unseen data points.

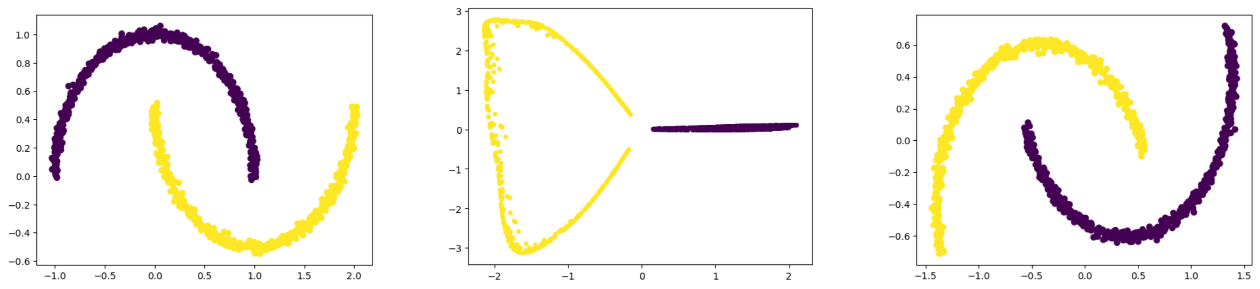
Proof. (Computation of projections for a generic query)
Let X^{query} a collection of data points, and Φ^{query} their image through the feature map ϕ

$$\begin{aligned}&[\Phi^{(query)} \mathbf{v}^{(\mathbf{j})}]_i = \\ &= \phi({\mathbf{x}^{(\mathbf{i})}}_{query})^T \mathbf{v}^{(\mathbf{j})} \\ &= \phi({\mathbf{x}^{(\mathbf{i})}}_{query})^T \textstyle\sum_k \alpha_k^{(j)} \phi(\mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} \phi({\mathbf{x}^{(\mathbf{i})}}_{query})^T \phi(\mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} k({\mathbf{x}^{(\mathbf{i})}}_{query}, \mathbf{x}^{(\mathbf{k})}) \\ &= \textstyle\sum_j \alpha_k^{(j)} K_{ik}^{query} \\ &= [K^{query}\mathbf{a}^{(\mathbf{j})}]_i\end{aligned}$$

With K^{query} defined such that $K_{ij}^{query} = k({\mathbf{x}^{(\mathbf{i})}}_{query}, \mathbf{x}^{(\mathbf{j})})$.
Hence following the previous argumentation we get

$$\Phi^{query} V = K^{query} A$$

$\square$



(a) Moons dataset

(b) Projection of Moons dataset over its two first kernel principal components

(c) Projection of Moons dataset over its linear principal components

Figure 4: In this example we projected the famous synthetic Moon Dataset against its principal components, obtained starting from a radial kernel.As can be noticed, kernel principal components appear to hold information regarding the cluster structure in the data (note that projected data would be linearly separable with just the first principal component), which is not "detected" by classical PCA. *Dataset source : Scikit Learn Moons Dataset [9]*

4.4 Zero centered feature space maps

Recalling the definition of covariance matrix that we gave at the beginning of our discussion, the previous results hold only if

$$\sum_i \phi(\mathbf{x^{(i)}}) = \mathbf{0}$$

In fact, if the feature space map ϕ does not center the data then the matrix $\frac{1}{N}\sum_i \phi(\mathbf{x^{(i)}})\phi(\mathbf{x^{(i)}})^T$ is not a valid estimate of the covariance matrix.
In fact a valid estimate of covariance matrix C of a dataset X should be defined such that

$$\begin{aligned}
&C(X)_{ij} = \\
&= \tfrac{1}{N}\textstyle\sum_k (X_{ki} - \tfrac{1}{N}\sum_q X_{qi})(X_{kj} - \tfrac{1}{N}\sum_p X_{pj}) \\
&= \tfrac{1}{N}\textstyle\sum_k (X_{ki} - \tfrac{1}{N}\sum_q X_{qi})(X_{kj} - \tfrac{1}{N}\sum_p X_{pj}) \\
&= \tfrac{1}{N}\underbrace{\sum_k X_{ki}X_{kj}}_{(X^TX)_{ij}} \underbrace{- \frac{1}{N}\sum_k X_{kj}\frac{1}{N}\sum_q X_{qi} - \frac{1}{N}\sum_k X_{ki}\frac{1}{N}\sum_p X_{pj} + \frac{1}{N}\sum_k \frac{1}{N^2}\sum_q X_{qi}\sum_p X_{pj})}_{\text{ERROR THAT VANISHES IF } \sum_q X_{qf}=0 \quad \forall f}
\end{aligned}$$

Moreover, there are a lot of examples where the aforementioned condition is not respected; it's sufficient to think of the $\phi^*(\mathbf{x}) = [1|\mathbf{x}]^T$ feature space map. Given any dataset X the resulting dataset Φ has always mean of the first featur equal to 1, making the hypothesis fail.

The idea proposed by the work of Smola et al.[10] is to build kernels in order to make their implicit feature maps **zero-centering**.
More specifically, given a feature map ϕ, associated with the kernel k, we aim to produce a feature map $\tilde{\phi}$ such that

$$\tilde{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \tfrac{1}{N}\textstyle\sum_j^N \phi(\mathbf{x^{(j)}})$$

Computing the generic dot product among two vectors $\mathbf{x}, \mathbf{y}$ we obtain

$$\tilde{\phi}(\mathbf{x})^T\tilde{\phi}(\mathbf{y}) = \left(\phi(\mathbf{x}) - \tfrac{1}{N}\textstyle\sum_j^N \phi(\mathbf{x^{(j)}})\right)^T \left(\phi(\mathbf{y}) - \tfrac{1}{N}\textstyle\sum_j^N \phi(\mathbf{x^{(j)}})\right)$$

Which can be simplified in

$$= \phi(\mathbf{x})^T\phi(\mathbf{y}) - \tfrac{1}{N}\textstyle\sum_j^N \phi(\mathbf{x^{(j)}})^T\phi(\mathbf{y}) - \tfrac{1}{N}\sum_j^N \phi(\mathbf{x^{(j)}})^T\phi(\mathbf{x}) + \tfrac{1}{N^2}\sum_i^N\sum_j^N \phi(\mathbf{x^{(j)}})^T\phi(\mathbf{x^{(i)}})$$

Therefore, applying the kernel trick

$$\tilde{k}(\mathbf{x},\mathbf{y}) = k(\mathbf{x},\mathbf{y}) - \tfrac{1}{N}\textstyle\sum_j^N k(\mathbf{x^{(j)}},\mathbf{y}) - \tfrac{1}{N}\sum_j^N k(\mathbf{x^{(j)}},\mathbf{x}) + \tfrac{1}{N^2}\sum_i^N\sum_j^N k(\mathbf{x^{(i)}},\mathbf{x^{(j)}}) \tag{2}$$

It was therefore possible to build a kernel that induces a zero centering feature map (and so it is suitable for computing the sample covariance matrix on the image of the dataset through the feature map $\tilde{\phi}$) starting from a kernel that induces a feature map ϕ that produces a feature expanded implicit dataset with unknown sample mean.

* * *

5 Kernel PCA Algorithm

5.1 Pseudocode

Using the tools that we built during this discussion, we can finally assemble the full pseudocode for projecting a dataset over its first m principal components.

Algorithm 1 Full projection pseudocode for the first m kernel principal components, with k as kernel

```
function PROJECT(X,m,k)
    initialize $K$ such that $K_{ij} := k(\mathbf{x^{(i)}}, \mathbf{x^{(j)}})$
    $\tilde{K} \leftarrow K - \frac{1}{N}K\mathbf{1}\mathbf{1}^T - \frac{1}{N}\mathbf{1}\mathbf{1}^TK + \frac{1}{N^2}\mathbf{1}^TK\mathbf{1}\mathbf{1}\mathbf{1}^T$        ▷ centered Gram matrix
    $\tilde{k}(\mathbf{x},\mathbf{y}) :=$ centering of $k$
    $\{(\lambda_i, \mathbf{a^{(i)}})\}_{i=1}^m \leftarrow$ first $m$ eigencouples (eigenvalue magnitude order) for $\tilde{K}$
    let $A$ such that $A_{ij} := \mathbf{a^{(j)}}_i$
    return $KA$
end function
```

Note that in the second line we are defining the centered Gram Matrix in a more compact and efficient way, with respect to evaluating the kernel $\tilde{k}$ over every possible couple in the dataset.

Proof. (Verification that $\tilde{K}_{ij} = \tilde{k}(\mathbf{x^{(i)}}, \mathbf{x^{(j)}})$) We defined

$$\tilde{K} := K - \tfrac{1}{N}K\mathbf{1}\mathbf{1}^T - \tfrac{1}{N}\mathbf{1}\mathbf{1}^T K + \tfrac{1}{N^2}\mathbf{1}^T K \mathbf{1}\mathbf{1}\mathbf{1}^T$$

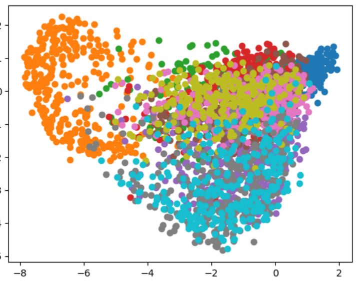
Hence

$$\begin{aligned}
\tilde{K}_{ij} &= K_{ij} - \left(\tfrac{1}{N}K\mathbf{1}\mathbf{1}^T\right)_{ij} - \left(\tfrac{1}{N}\mathbf{1}\mathbf{1}^T K\right)_{ij} + \tfrac{1}{N^2}\mathbf{1}^T K \mathbf{1}\\
\tilde{K}_{ij} &= K_{ij} - \tfrac{1}{N}\left(\sum_q K_{iq}[\mathbf{1}\mathbf{1}^T]_{qj}\right)_{ij} - \tfrac{1}{N}\left(\sum_p [\mathbf{1}\mathbf{1}^T]_{iq}K_{qj}\right)_{ij} + \tfrac{1}{N^2}\sum_q\sum_p \mathbf{1}_q K_{qp}\mathbf{1}_p\\
\tilde{K}_{ij} &= K_{ij} - \tfrac{1}{N}\left(\sum_q K_{iq}\right)_{ij} - \tfrac{1}{N}\left(\sum_p K_{qj}\right)_{ij} + \tfrac{1}{N^2}\sum_q\sum_p K_{qp}\\
\tilde{K}_{ij} &= K_{ij} - \tfrac{1}{N}\left(\sum_q K_{iq}\right)_{ij} - \tfrac{1}{N}\left(\sum_p K_{jq}\right)_{ij} + \tfrac{1}{N^2}\sum_q\sum_p K_{qp}\\
\tilde{K}_{ij} &= k(\mathbf{x^{(i)}}, \mathbf{x^{(j)}}) - \tfrac{1}{N}\left(\sum_k k(\mathbf{x^{(i)}}, \mathbf{x^{(k)}})\right) - \tfrac{1}{N}\left(\sum_k k(\mathbf{x^{(j)}}, \mathbf{x^{(k)}})\right) + \tfrac{1}{N^2}\sum_q\sum_p k(\mathbf{x^{(q)}}, \mathbf{x^{(p)}})\\
\tilde{K}_{ij} &= \tilde{k}(\mathbf{x^{(i)}}, \mathbf{x^{(j)}})
\end{aligned}$$

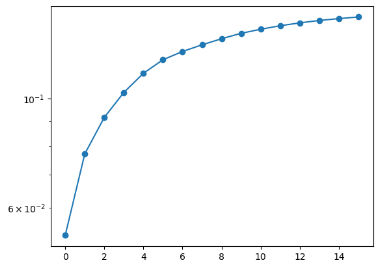
□

5.2 Is the projection really able to hold information?

After a lot of formal discussion, we are able to use the kernel PCA algorithm to perform data embedding in a smaller vector space.
At this point of the discussion, is important to offer a real proof of concept of the possibilities offered by the algorithm, in order to understand the semantics of the embedding and to exploit the power of the method at the fullest.
In the two next sections we are going to investigate two interesting use cases, *Denoising* and *Sample Generation*, which show a direct example of the ability of kernel PCA to incorporate complex information about inner structures inside the data inside lower dimensional spaces.



(a) Projection of the MNIST [11] dataset



(b) Explained variance diagram

Figure 5: Explained variance for a part of the MNIST [11] dataset, computing the explained variance diagram exploiting that the identity $\sum_i \lambda(\Phi^T\Phi)_i^2 = \mathrm{Tr}[\Phi^T\Phi\Phi^T\Phi] = Tr[\Phi\Phi^T\Phi\Phi^T] = Tr[K^2] = Tr[K^TK] = \sum_i\sum_j K_{ij}^2$

* * *

6 Kernel PCA - Denoiser

One of the most famous uses of the Kernel PCA algorithm is denoising. The problem of denoising can be explained effectively by just its formalization

Definition 6.1. (Observation Vector) Let $\mathbf{x}$ be a vector (that we denote as **observation**) such that exists an unknown vector $\tilde{\mathbf{x}}$ (that we denote as **truth**) for which $\mathbf{x} = \tilde{\mathbf{x}} + \epsilon$, with ϵ a generic source of noise.

Definition 6.2. (Denoising) Let $\mathbf{x}$ be a an observation. The task of extracting the truth vector $\tilde{\mathbf{x}}$ associated with it, is denoted as a denoising task.

Until now, the reasons why kernel PCA is useful in this context may appear vague.
Recalling the work of Mika et al [12], where an useful insight on the link between information and noise is give, we would like to recall the idea of **explained variance**. In particular, it is a well known fact from theory that principal components linked to largest eigenvalues in terms of absolute value tend to represent composite features that capture most of the intrinsic variance of the dataset (see the "PCA revisited" section for a short proof explaining this concept). Hence, from an informatical theoretical perspective, principal components capture different amounts of information, which are

"proportional" (in a sense) to the magnitude of their associated eigenvalue. Starting from this assumption we have built the heuristics that motivates denoising ; since noise adds (virtually) no information to the dataset, it is likely that it gets captured by least significant principal components, leaving the first ones unaltered.
Let us reasons, without any loss of generality, on the reconstruction on the first component only. In classical PCA, then would be sufficient to reconstruct the projections using the following technique

$$\hat{\mathbf{x}} = \underbrace{\mathbf{v}^T\mathbf{x}}_{\textsc{Project}_{\mathbf{v}(\mathbf{x})}} \quad \mathbf{v}$$

In an attempt to generalize this notion to kernel PCA, the flaw appears immediatly; applying the same reasoning we would be reconstructing the image of the sample in the feature map, not the original sample in the dataset.In fact, $\mathbf{v}$ in the context of kernel pca, as we proved, belongs to $span(\phi(X)))$ (with X being the dataset and ϕ the feature map implicitly defined by the kernel) which is in general different from $span(X)$ which is the space where the original data lies.

The solution proposed by Mika et al. [12] is to treat denoising as an optimization problem, which is called **pre-image** problem

$$\text{Find } \mathbf{x} \text{ minimizing } ||\phi(\mathbf{x}) - \mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v}||_2^2 \ \diamondsuit$$

In other terms, since we know that feature maps are bijective and we have not an explicit way to represent the image of samples through a feature map we try to find , in the original feature space, the data point which minimizes the distance between its image and the projection. Developing the calculations, in fact, explicit expressions of the feature map ϕ disappear.

Proof. (Explicit form the optimization problem $\diamondsuit$)

$$\begin{aligned}
&||\phi(\mathbf{x}) - \mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v}||_2^2 \\
&= (\phi(\mathbf{x}) - \mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v})^T(\phi(\mathbf{x}) - \mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v}) \\
&= \phi(\mathbf{x})^T\phi(\mathbf{x}) - 2\phi(\mathbf{x})^T\mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v} + (\mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v})^T\mathbf{v}^T\phi(\mathbf{x}^*)\mathbf{v} \\
&\sim \phi(\mathbf{x})^T\phi(\mathbf{x}) - 2\phi(\mathbf{x})^T\mathbf{v}\mathbf{v}^T\phi(\mathbf{x}^*) \\
&= \phi(\mathbf{x})^T\phi(\mathbf{x}) - 2\phi(\mathbf{x})^T\left(\textstyle\sum_i \alpha_i\phi(\mathbf{x^{(i)}})\right)\left(\textstyle\sum_j \alpha_j\phi(\mathbf{x^{(j)}})^T\right)\phi(\mathbf{x}^*) \\
&= \phi(\mathbf{x})^T\phi(\mathbf{x}) - 2\textstyle\sum_i\sum_j \alpha_i\alpha_j\phi(\mathbf{x})^T\phi(\mathbf{x^{(i)}})\phi(\mathbf{x}^*)^T\phi(\mathbf{x^{(j)}}) \\
&= \underbrace{k(\mathbf{x},\mathbf{x})}_{\star} - 2\textstyle\sum_i\sum_j \alpha_i\alpha_j k(\mathbf{x},\mathbf{x^{(i)}})k(\mathbf{x}^*,\mathbf{x^{(j)}}) \\
&= k(\mathbf{x},\mathbf{x}) - 2\textstyle\sum_i \alpha_i \quad \underbrace{\left(\displaystyle\sum_j \alpha_j k(\mathbf{x}^*,\mathbf{x^{(j)}})\right)}_{\textsc{Projection of } \phi(\mathbf{x}^*) \textsc{ on } \mathbf{v}} \quad k(\mathbf{x},\mathbf{x^{(i)}}) \\
&= k(\mathbf{x},\mathbf{x}) - 2\textstyle\sum_i \alpha_i\beta k(\mathbf{x},\mathbf{x^{(i)}})
\end{aligned}$$

□

Which, for projection on multiple kernel principal components, can be generalized as follows [12]

$$= k(\mathbf{x},\mathbf{x}) - 2\textstyle\sum_i\sum_j \alpha_i^j\beta_j k(\mathbf{x},\mathbf{x^{(i)}})$$

Minimizing this quantity can yield to a **pre-image** able to solve the reconstruciton problem.
A special mention is deserved by those kernels for which the induced norm is constant (e.g. the radial kernel induces the norm in the expanded feature space $||\phi(\mathbf{x})||_2 = \sqrt{\phi(\mathbf{x})^T\phi(\mathbf{x})} = \sqrt{k(\mathbf{x},\mathbf{x})} = \sqrt{exp\left(-\frac{||\mathbf{x}-\mathbf{x}||^2}{\sigma^2}\right)} = 1\ \forall\mathbf{x}$), making thus the term $\star$ constant. In this case we can build a simpler optimization problem

Proof. (Alternative form of the optimization problem, for kernel inducing constant norm $\diamondsuit$)

$$\begin{aligned}
&= k(\mathbf{x},\mathbf{x}) - 2\textstyle\sum_i \alpha_i \sum_j \beta_j k(\mathbf{x},\mathbf{x^{(i)}}) \\
&\sim -2\textstyle\sum_i \alpha_i \sum_j \beta_j k(\mathbf{x},\mathbf{x^{(i)}})
\end{aligned}$$

□

Which yields to the following **fixed point iteration schema** [12]

$$\mathbf{x}_{t+1} = \frac{\sum_i\sum_j \beta_j\alpha_i^j k(\mathbf{x}_t,\mathbf{x^{(i)}})\mathbf{x^{(i)}}}{\sum_i\sum_j \beta_j\alpha_i^j k(\mathbf{x}_t,\mathbf{x^{(i)}})}$$

Note that, particular attention has to be reserved to the condition $k(\mathbf{x},\mathbf{x}) = const$, since kernels may lose this property after centralization.

Proof. (Loss of constant norm after centering of kernels)

$$\nabla_{\mathbf{x}}\tilde{k}(\mathbf{x},\mathbf{x}) = \underbrace{\nabla_{\mathbf{x}}k(\mathbf{x},\mathbf{x})}_{=0} - \tfrac{1}{N}\textstyle\sum_j^N \nabla_{\mathbf{x}}k(\mathbf{x^{(j)}},\mathbf{x}) - \tfrac{1}{N}\textstyle\sum_j^N \nabla_{\mathbf{x}}k(\mathbf{x^{(j)}},\mathbf{x}) + \underbrace{\frac{1}{N^2}\sum_i^N\sum_j^N \nabla_{\mathbf{x}}k(\mathbf{x^{(i)}},\mathbf{x^{(j)}})}_{=0}$$

$$= -\tfrac{2}{N}\textstyle\sum_j^N \nabla_{\mathbf{x}}k(\mathbf{x^{(j)}},\mathbf{x})$$
$$\neq 0$$

□

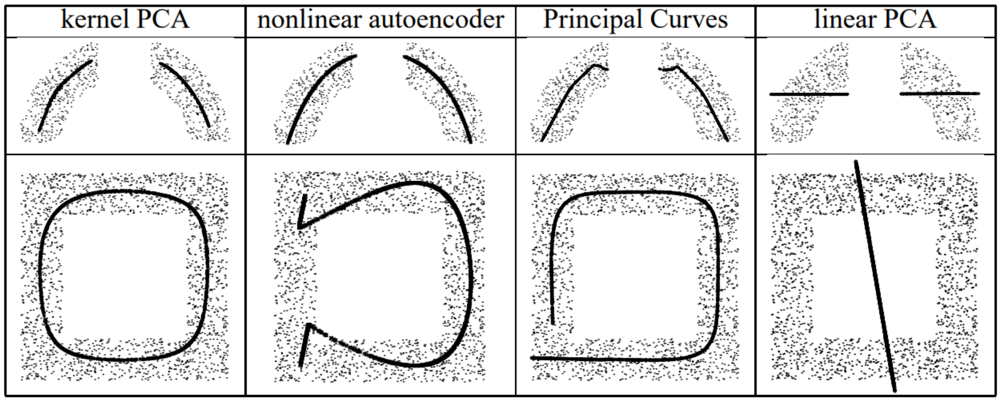


Figure 6: Different approaches for dimensionality reduction are used for denoising. The bold lines represent denoised samples, that are obtained starting from the application of various dimensionality reduction techniques ; note that only kernel PCA was able to recover fully the circular structure of the data in the second row. *Source [12]*

7 Kernel PCA - a Small Generative Experiment

To finalize our discussion, we want to offer a perspective on an unusual but yet interesting application of kernel PCA. Initially, when we introduced the problem of dimensionality reduction, we talked about the idea of encoding data in a way that preserves the ”largest amount of information” on the initial dataset.This notion of ”amount of information” was then explained in the previous section, showing how kernel PCA was indeed able to encode information in a way that preserved meaningful features of the input data. A spontaneous question may arise from this consideration : if it is possible to map data in a smaller space through kernel PCA, what kind of encoding may describe data that is not yet known? More precisely, we may ask ourselves if it is possible to find in the **latent space** induced by kernel PCA the distribution that represent the mapping of the data distribution in **feature space** through the projection operator defined by the kernel PCA algorithm. This kind of approach may be in fact useful for defining a **generative model** based on the encoding of kernel PCA ; learning the distribution of encoded samples we could try to produce an inverse mapping that remaps the distribution of encoded samples into the feature space and thus sampling from the obtained distribution, generating new samples. This tecnhique was initially proposed by the work of Winant et al. [13], showing how the embedding induced by kernel PCA was able to encode complex composite features in data, such as concepts related to shapes and human faces. Here, we will propose a variation of their algorithm, focusing on the possibility of sampling from the predicted data distribution and to perform interpolation between samples, to study the inherent semantics of the latent space of the MNIST [11] dataset embedding obtained through kernel PCA.

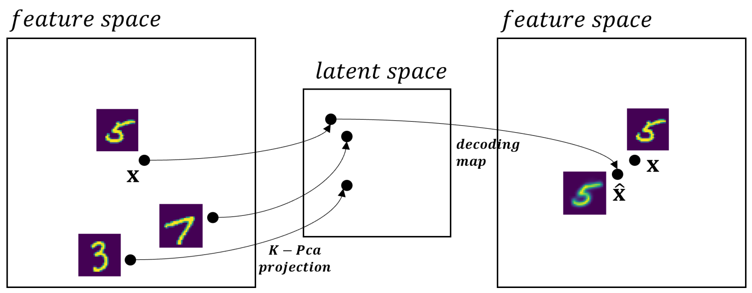


Figure 7: Conceptual representation of the encoder-decoder interpretation of the kernel PCA. Note that, for the sake of generality, the **decoding map** is a generic function from the latent space to the feature space. Even denoising in some sense could be framed in this schema *Source of the datapoints: MNIST dataset[11]*

Algorithm 2 Generation of new samples

```
function GENERATE(steps)
    $X_{proj} \leftarrow$ PROJECT($X$)                                          ▷ Embedding of the dataset
    Find $\mathscr{D}$ such that $||\mathscr{D}(X_{proj}) - X||_F^2$ is minimized
    $\pi(\mathbf{x}) := \frac{1}{N}\sum_i^N k_\pi^{(\sigma)}(\mathbf{x} - \mathbf{x_{proj}^{(i)}})$                 ▷ Estimated distribution with KDE
    $\mathbf{x} \leftarrow$ LANGEVIN$_\pi(\mathbf{x}, steps)$                              ▷ Random sampling from $\pi$
    return $\mathscr{D}(\mathbf{x})$                         ▷ Decoding of the random sample in latent space
end function
```

The pseudocode , even if simple, need some explanation in order to be understood. The first two instructions of the function, in particular, are responsible of emulating the **encoder-decoder** structure using the kernel PCA as an encoder and a regression method to build the decoder (in our experiments kernel regression was used) Finally, an estimate of the distribution in latent space is computed using **Kernel Density Estimation**[14], using then such estimate as an input for the **Langevin sampling algorithm**[15]. Here we report, for reader convenience, the pseudocode of the Langevin iteration, but we invite the reader to refer to [15] in order to have a deeper insight on this procedure.

Algorithm 3 Langevin Dynamics Sampling

```
Let $\xi \sim \mathcal{N}(\mathbf{0}, I)$
function LANGEVIN($\pi$,$\tau$,steps)
    $\mathbf{x} \leftarrow ...$                                                        ▷ Initial guess
    for $i \in [1, steps]$ do
        $\mathbf{x} \leftarrow \mathbf{x} + \tau\nabla\{log[\pi(\mathbf{x})]\} + \sqrt{2\tau}\xi$
    end for
    return x
end function
```

Results of the aforementioned experiment were encouraging. More specifically the experiment was conducted with a latent space of dimension equal to 15, using both for the kernel PCA encoder both for the kernel regression model a Gaussian Kernel. The algorithm was able to generate samples not present in the training set, as can be seen in the following visualization.

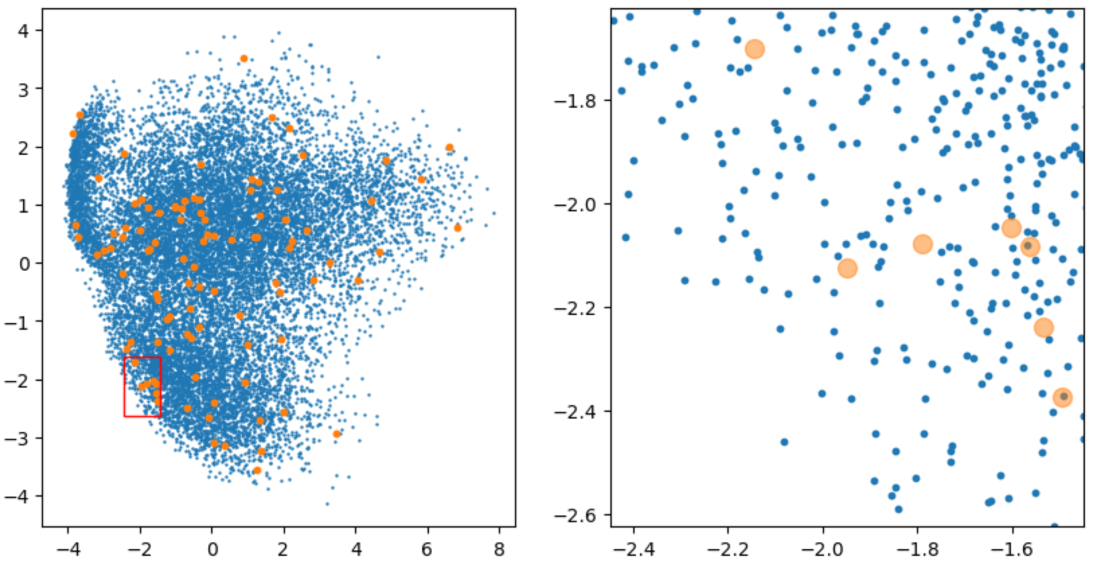


Figure 8: Visualization of the Langevin sampling on the latent space induced by the Kernel PCA Projection (newly generated samples are pointed out in orange) On the right, a detail of the latent space is highlighted, showing how the sampling effectively collected points not present in the output of the embedding.

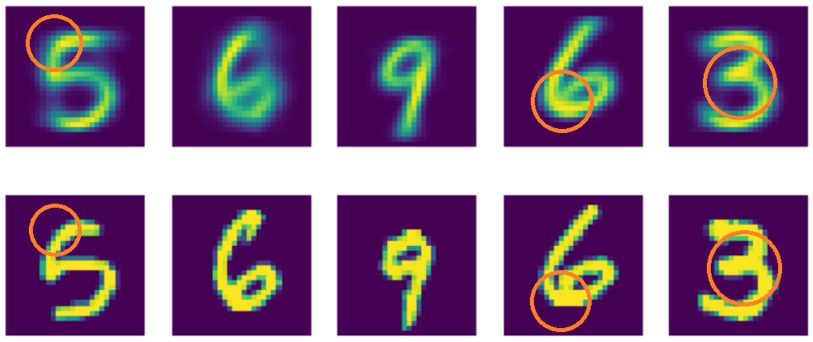


Figure 9: Visualization of generated samples, paired with the closest (in sense of **Froebenius Norm**) sample in the training set. Some dissimilarities are highlighted. *Source of data: MNIST dataset[11]*

In order to investigate the **semantics** of the Latent Space vectors, interpolation experiments where also conducted, highlighting how moving countinuosly in the latent space reflected in a ”continuous transformation” of generated samples

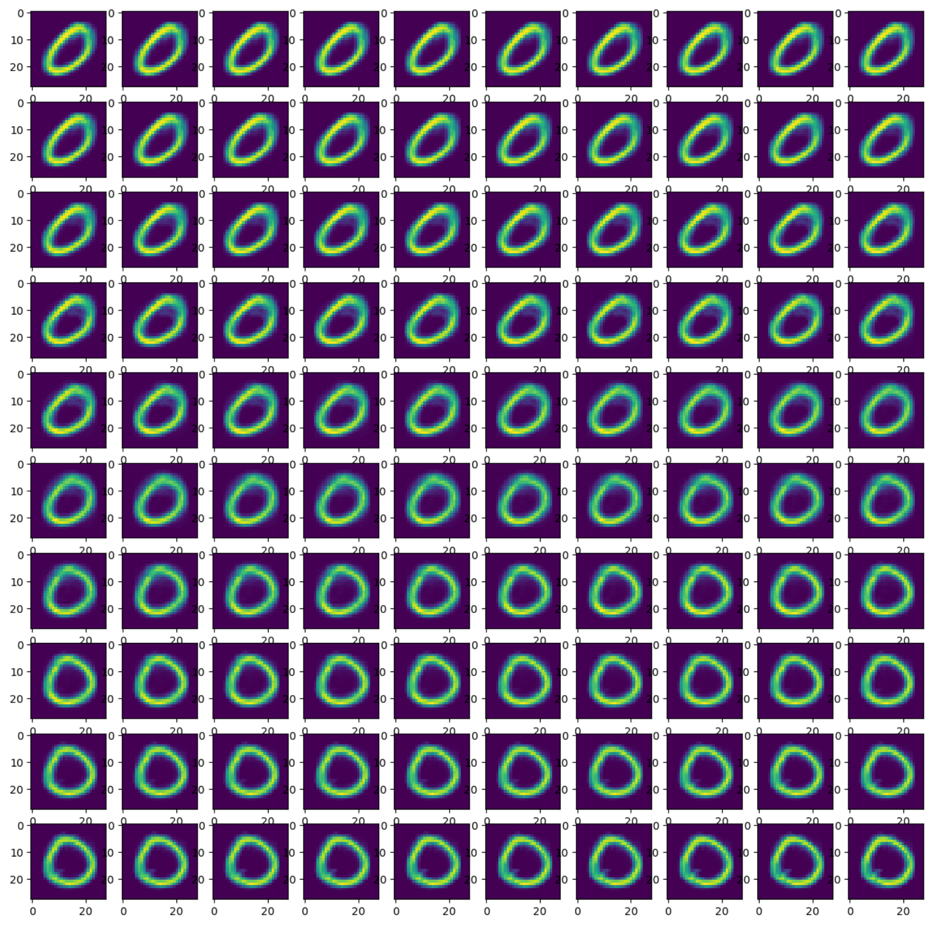


Figure 10: Upper-left and Lower-right samples are the decoding of embeddings obtained from samples in the training set. The in between samples represent the progression of decoded images obtained moving from one latent space representation to the other. *Source of data: MNIST dataset[11]*

Offering, therefore, a proof of concepts of the inner semantics of kernel principal components, and of their linear combination.

* * *

8 Conclusions

In the previous discussion, we tried to offer an explanation of what is kernel PCA, and how it represents a powerful tool in Data Science.
Some of the possibilities were presented, although they do not represent the whole spectrum of kernel PCA capabilities. With our work we wanted to highlight how kernel PCA (and in general kernel methods), even if not a state-of-the-art technique, can give us precious insights. Nonetheless, it is still not totally clear how the observed good generalization is obtained in a space with infinite dimensions, contradicting the curse of dimensionality. To conclude our tracctation we would like, in particular, to refer to the work [16] of Belkin et al, where this apparent contradiction is investigated. In their particular, the authors analyzed the similarity between kernel methods and overparametrized neural network, showing how overfitted kernel methods tend to build generalization, again apparently violating the theory regarding the curse of dimensionality.
This results were again brought to relevance by more modern results like the double descent [17], highlighting the great importance of building a stronger understanding of high dimensional kernel methods, which in the special case of kernel PCA may reveal complex structure inside data, helping researchers in science,medicine and engineering.

References

[1] Laurens van der Maaten, Eric Postma, and Jaap van den Herik. Dimensionality reduction: A comparative review. 2009.

[2] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? 1999.

[3] Warren S. Torgerson. Multidimensional scaling: I. theory and method. 1952.

[4] Harald Niederreiter. Low-discrepancy and low-dispersion sequences. 1987.

[5] Thomas Hofmann, Bernhard Scholkopf, and Alexander J. Smola. Kernel methods in machine learning. 2008.

[6] Christopher M. Bishop. *Pattern Recognition and Machine Learning.*

[7] Bernhard Scholkopf, Kah-Kay Sung, Chris J. C. Burges, Federico Girosi, Partha Niyogi, Tomaso Poggio, and Vladimir Vapnik. Comparing support vector machines with gaussian kernels to radial basis function classifiers.

[8] Bernhard Scholkopf, Sebastian Mika, Chris J. C. Burges, Philipp Knirsch, Klaus-Robert Muller, Gunnar Ratsch, and Alexander J. Smola. Input space versus feature space in kernel-based methods. 1999.

[9] Scikit learn moons dataset. `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html`.

[10] Bernhard Scholkopf, Alexander Smola, and Klaus-Robert Muller. Nonlinear component analysis as a kernel eigenvalue problem. 1998.

[11] LeCun Yann, Cortes Corinna, and Burges CJ. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[12] Sebastian Mika, Bernhard Scholkopf, Alex Smola, Klaus-Robert Muller, Matthias Scholz, and Gunnar Ratsch. Kernel pca and de-noising in feature spaces. 1998.

[13] David Winant, Joachim Schreurs, and Johan A. K. Suykens. Latent space exploration using generative kernel pca.

[14] Yen-Chi Chen. A tutorial on kernel density estimation and recent advances. 2017.

[15] Florentin Coeurdoux and Nicolas Dobigeon. Normalizing flow sampling with langevin dynamics in the latent space. 2023.

[16] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. 2018.

[17] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. 2019.