UNIVERSITAT POLITÈCNICA DE CATALUNYA

MACHINE LEARNING II
GCED

DIMENSIONALITY REDUCTION PCA & KERNEL PCA

Authors:

Dani Gómez (48075432G)
Ricard Tarré (47995840S)

January 2022

Contents

1 Introduction

In this project we want to understand and analyze the usability of the **kernelized PCA** algorithm in addition to test it's performance, all starting from a generalization of the standard **PCA**.

To carry out this task, the first thing we are going to do is a theoretical comparison between both methods in order to understand the expected objectives when applying PCA and the advantage that its kernelized version offers us in certain situations.

After having understood the theoretical basis, we will study a method for evaluating the performance of Kernel PCA. We will present the theoretical concepts of the method, and show how to apply it for the different kernel functions chosen for practical analysis.

In the practical analysis we will first understand how the method described in the theoretical study works. With this purpose, toy datasets will be used. Once we have understood the potential of this method we will apply it to a real dataset.

2 Theoretical Analysis

2.1 PCA

The **Principal Component Analysis** is a multivariate technique used for dimensionality reduction. It's main objective when reducing dimensions is to preserve as much data variance as possible. We are going to explain step by step the procedure it follows in order to maximize the variance of the data projected:

1. Given n points $x_i \in \mathbb{R}^d$, we define the matrix X of points of size $n * d$. We then center this matrix around the mean of the points.
2. We now define the covariance matrix $C = X^T X$ such that $C = \sum_{i=1}^{d} (x_i)^T (x_i)$.
3. We want to find a new reducted set of uncorrelated variables Z that are linear combinations of the actual variables such that each $Z_j = a_j^T X$ and $\|a_j\| = 1$. It is proved that the a_j's that maximize the variance of Z_j are the eigenvectors of covariance matrix C.
4. If we want to reduce to w ($< d$) dimension we have to pick the w first Z_j's with higher variance. Given that $Var(Z_j) = Var(a_j^T X) = a_j^T Var(X) a_j = a_j^T C a_j = a_j \lambda_j a_j = \lambda_j$. Then we should pick the w eigenvectors with higher eigenvalues in decreasing order.
5. We can now compute the projected variables $Z_j = a_j^T X$ such that a_j is the eigenvector of C with the *j-th* biggest eigenvalue.

Having explained how **PCA** works, we are going to briefly expose some of it's advantages and disadvantages.

On the one hand, this is a fast and easy to compute algorithm that ""solves the issues of high dimensional data"". Nevertheless we should be careful in the balance between dimensions reduced and information lost. Among other disadvantages, the one that most involves us is that it assumes the features to be linearly correlated. That it is not a trivial thing as in real life we constantly make front to non-linear datasets which don't allow us to apply this algorithm.

2.2 Kernel PCA

The **Kernel PCA** is introduced to make front the limitations of classical **PCA** in terms of non-linear data.

We introduce a mapping function $\phi(x)$ that allows us to project data into a high feature dimensional space, also known as feature space, where it becomes linearly separable. Then we apply PCA in this space such that the projected variables are no longer linear with respect to the original ones, so we are achieving a non-linear dimensionality reduction in the original space.

We are going to show how to kernelize PCA step by step:

1. Given n points $x_i \in R^d$ and a feature map $\phi : x_i \to \phi(x_i)$ we define the matrix X' of the original points mapped in the feature space such that each column is $\phi(x_i)$.
2. Now we can compute the sample covariance matrix as $C = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \phi(x_i)^T$.

The theoretical idea is to compute classical PCA on this covariance matrix. Nevertheless, the feature space calculation together with the SVD is computationally very expensive. So the objective is to work in the original feature space without making computations in high dimensional space. This can be achieved with the **kernel trick**, such that knowing an existing kernel function $k(x_i, x_i')$, Aronszajn theorem assures there exists a mapping $\phi(x_i)$ such that $k(x_i, x_i') = \phi(x_i)^T \phi(x_i)$. We are going to show how to apply kernel trick in **KPCA** such that knowing an existing kernel function, there is no need to know the mapping function $\phi(x_i)$ and to make the computations in high dimensional space avoiding this way the expensive computational costs:

1. Having the covariance matrix C, the eigenvectors are $Cv_j = \lambda_j v_j$. Substituting C by it's previous definition we have $\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T v_j = \lambda_j v_j$. Defining eigenvectors as $v_j = \sum_{i=1}^{n} a_{ji}\phi(x_i)$ and substituting in the previous equations we have:
   (a) $\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T(\sum_{k=1}^{n} a_{jk}\phi(x_k)) = \lambda_j \sum_{k=1}^{n} a_{jk}\phi(x_k)$
   (b) $\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)(\sum_{k=1}^{n} a_{jk}k(x_i,x_k)) = \lambda_j \sum_{k=1}^{n} a_{jk}\phi(x_k)$
   (c) multiplying by $\phi(x_l)^T$ and substituting by kernel we have $\frac{1}{n}\sum_{i=1}^{n} k(x_l,x_i)(\sum_{k=1}^{n} a_{jk}k(x_i,x_k)) = \lambda_j \sum_{k=1}^{n} a_{jk}k(x_l,x_k)$
   (d) we can resume this in matrix form as $Ka_j = n\lambda_j a_j$
2. Observing the formula derivated in previous point $Ka_j = n\lambda_j a_j$, we realize that a_j are the eigenvectors of K with eigenvalues $n\lambda_j$.
3. So for a point x it's projection in component j will be $\phi(x)^T v_j = \sum_{i=1}^{n} a_{ji}\phi(x)^T\phi(x_i) = \sum_{i=1}^{n} a_{ji}k(x,x_i)$ where a_j is an eigenvector of Kernel matrix as seen in previous point.

We have applied the kernel trick such that we can calculate the projections of each point in each component without knowing the feature mapping or doing computations in high dimensional space. We only need to choose some existing kernel function, to compute the Kernel Matrix and calculate the eigenvectors of this matrix. Nevertheless, despite saving computational cost thanks to kernel trick, there is a fact that can make **Kernel PCA** much slower in some occasions. The fact is that in **PCA** we compute eigenvectors on the covariance matrix C of size $d * d$ (d is point dimension), while in **Kernel PCA** we compute it on kernel matrix K of size $n * n$ (n number of points). So if the number of points is much bigger than the dimensions, the cost increases notoriously. This fact also makes **Kernel PCA** a non-parametric method as it computes the optimum parameters with respect to the points instead of dimensions.

2.3 Kernel functions

We have shown how to kernelize the PCA algorithm for solving issues with non-linear data. As we explained in previous section, in order to avoid high computational costs, we applied kernel trick. For applying this trick it's necessary to choose a known kernel function and this is what we will discuss in this section.

We talk about known kernel functions because there is a large set of functions that meet the necessary properties (positive definite and symmetric) and that have been already used in ML for kernelizing different methods.

In following sections, experimental analysis, we are going to study the performance of **Kernel PCA** with both toy and real datasets. For this purpose, we must select a subset of known kernel functions to use them in the application of the algorithm. It's a hard challenge to decide a priori which kernel function is better for a real problem, as each function has it's properties such that it's performance depends on the type of data and the algorithm applied.

For kernelizing PCA we have chosen three typical and very known **kernel functions**. These are the **polynomial** to have a generalization of the linear case, the **Radial Basis Function** for being one of the most popular and the **sigmoid** one because it is used in neural networks. All three kernel functions are summarized below.

Kernel	**Function**	**Parameters**
Polynomial	$(< x, y > + c)^d$	$K \in \mathbb{R}, d \in \mathbb{N}$
RBF	$exp(-\frac{\lvert x-y\rvert^2}{2\sigma^2})$	$\sigma \in \mathbb{R}^+ \setminus \{0\}$
Sigmoid	$tanh(\alpha < x, y > + c)$	$\alpha, c \in \mathbb{R}$

Table 1: Kernel functions chosen to apply Kernel PCA algorithm.

3 Evaluating Kernel PCA performance

3.1 Introduction

At this point, we already know which datasets we are going to use plus the kernel functions we are going to study and compare, but in fact we have not proposed a way to evaluate which method or kernel function performs better results.

Selecting a way to evaluate the results is somewhat tricky because unlike standard PCA where one can use reconstruction error as the performance measure on the test set, in kernel PCA the problem is that reconstruction error is not comparable between different kernels. This is due to the fact that the reconstruction error is the distance measured in the target feature space and we know that different kernels correspond to different target spaces. However, if we are able to compute the reconstruction error in the original space, then we would be tackling this issue and therefore we could compare performance between different kernels in addition to non-kernel methods.

3.2 Reconstruction error in original space

Our goal is to find a way to compute the reconstruction error in the original space, but first we need to understand the logic of doing this.

For a given dataset, we reserve a point X as a test point. This test point X lives in the original space, while it's **KPCA** reconstruction lives in the target space. Given the reconstructed point Y, we want to find a point X' in the original space that when is mapped to the target space $\phi(X') : Y'$ it is as close as possible to Y. So that the distance between the found point X' and the real point X will be the reconstruction error computed in original space.

Finding this point X' is known as finding the pre-image of the reconstructed point, and of course we will need some method for computing this pre-image as it will be studied in the following point.

3.3 Computing pre-image

3.3.1 Theoretical concepts

Computing the exact pre-image of a reconstructed point given some kernel function, is generally an *ill-posed* problem, because no solution exists or there is no unique solution. So to tackle this problem, we seek an approximate solution of the pre-image, such that it's mapping is as close as possible to the real reconstructed point.

Given a set of points of size n, the projection of a test point x is $\varphi = V^T\phi(x)$ (V is matrix of projection vectors). We want to find a point x' in the original space, such that it's mapping to target space $\phi(x')$ is as close as possible to φ. So we want to find the point x'^* that minimizes the function:

$f(x') = \|V^T\phi(x) - \phi(x')\|^2$ (9)

If we apply kernel trick to this function, we arrive that it's equivalent to:

$f(x') = k(x', x') - 2\sum_{i=1}^{n} \gamma_i k(x', x_i)$ (10),
where $\gamma_i = \sum_{j=1}^{ncomponents} \sum_{h=1}^{n} \alpha_{ji} k(x_h, x)\alpha_{jh}$ and α_j is eigenvector j of kernel matrix.

Finding this point that gives best approximation, is a solution for the situation where computing the pre-image is an *ill-posed* problem. However there exist some kernel functions that given the projection φ of a test point x, it exists an exact pre-image such that $\phi(x') = \varphi$. This happens when the kernel function is an invertible function of $\langle x_j, x_i \rangle$. If v is the invertible function such that $v(k(x_j, x_i)) = \langle x_j, x_i \rangle$, then the exact pre-image of a projected point is:

$x = \sum_{i=1}^{N} v(\sum_{i=1}^{n} \gamma_i k(e_j, x_i))e_j$, where $\{e_1, ..., e_n\}$ is any orthonormal basis in the input space, and γ_i is

as previously defined.

3.3.2 Choosing pre-image methods for kernel function selected

We have explained what is the purpose of pre-image and how to compute an estimation when kernel functions are not invertible. So given the kernel functions that we select in section *2.3*, we will choose different methods to find the closest approximation.

Polynomical kernel

For polynomical kernel, there only exists an invertible function of $\langle x_j, x_i \rangle$ when the degree-parameter D is odd. This function is $v = \sqrt[d]{(\bullet)} - K$.For the case of D even, if we apply $k(x,x) = exp(-\frac{|x-y|^2}{2\sigma^2})$ into function defined in (11) we compute the gradient and we set it to zero, we obtain the fixed-point algorithm expression:

$x^*_{t+1} = \sum_{i=1}^n \gamma_i \frac{\langle x^*_t, x_i \rangle + c}{\langle x^*_t, x^*_t \rangle + c}^{d-1}$

We will apply fixed point algorithm stopping when $\|x^*_{t+1} - x^*_t\| < 0.001$

RBF kernel

For RBF kernel, there does not exist any invertible function of $\langle x_j, x_i \rangle$. However, if we apply $k(x,x) = (< x, y > + c)^d$ into function defined in (11) and we compute the gradient, we obtain:

$\nabla f(x) = \frac{-2}{\sigma^2} \sum_{i=1}^n \alpha_i exp(-\frac{\|x-x_i\|^2}{2\sigma^2})(x - x_i)$

By setting this gradient to zero, we obtain the fixed-point algorithm expression:

$x^*_{t+1} = \frac{\sum_{i=1}^n \gamma_i k(x^*_t, x_i) x_i}{\sum_{i=1}^n \gamma_i k(x^*_t, x_i)}$

We will apply fixed point algorithm stopping when $\|x^*_{t+1} - x^*_t\| < 0.001$

Sigmoid kernel

For sigmoid kernel, exists an invertible function of $\langle x_j, x_i \rangle$. This function is $v = \frac{argtanh(\bullet) - c}{\alpha}$.

3.4 Algorithm for evaluating kernel performance

We have defined the methods for computing the pre-image of the three kernel functions chosen for the practical analysis. For each dataset and kernel function, we will chose the hyper-parameters that give the total minimum reconstruction error, and then we will compare the best reconstruction errors of the three kernel function. Given a dataset $D = \{x_1, ..., x_n\}$ and a set of parameters $P = \{p_1, ..., p_L\}$, the algorithm for choosing the hyper-parameters that led to minimum reconstructed error follows this steps:

1. Divide the dataset in 15 equal-sized grups of points, $D = G_1 \cup ... \cup G_{15}$
2. Set h = 1 and choose p_h
3. Set i = 1 and do following steps while $i < 15$:
   (a) Solve **kernel PCA** for $D - G_i$ with parameters p_h
   (b) Compute pre-images Z_j for points in G_i following the pre-image method assigned to kernel function in section *3.3*
   (c) Compute reconstruction error for each one of the points in G_i. For each $x_j \in G_i$, $E^h_{ij} = \|Z_j - X_j\|^2$

(d) Compute average reconstruction error for the points in G_i. $E_i^h = \frac{1}{|G_i|} \sum_{j=1}^{|G_i|} E_{ij}^h$

(e) i = i + 1

4. Compute 15-fold CV error, $E^h = \frac{1}{15} \sum_{k=1}^{15} E_k^h$
5. h = h + 1
6. If $h > L$ stop, else go to (2)
7. $h_{opt} = argmin(E^h)$

This entire algorithm has been implemented in *python* for each of the various kernel functions, and can be seen in the attached code we deliver.

4 Understanding reconstruction error

Many people uses Kernel PCA not for dimensionality reduction but for data shaping. So they can, for example, transform non-linear separable data into linear separable data without reducing dimension. This is not really the goal of Kernel PCA or PCA. The goal of this algorithms is to reduce the dimensions of data maintaining the maximum quality and information possible. This is what **reconstruction error** measures, the quality loss when reducing dimensions. However, until now we have only seen the mathematics and the way to compute reconstruction error in different cases. Now we want to understand how this error works before applying the algorithm to a real dataset.

To understand it's behaviour, we will practice on three 2-D toy datasets, exaplained in the following section, computing the reconstruction error using PCA and Kernel PCA. But in order to apply the last one with each of the three kernel functions, we first have to carry out a cross validation testing different values of the hyper-parameters and thus decide which one behaves better to evaluate its reconstruction error.

4.1 Toy Datasets chosen

For the first part of our analysis we worked with toy datasets, easy to generate and manage. Those were generated using the *sklearn* python library; one linearly separable set and two that are not, which would demonstrate the importance of kernelizing PCA algorithm.

For the linearly separable case we chose Blobs Dataset and we initially generated a set with 1000 points. For this set, our decision was not to separate the clusters of both classes completely to leave a small margin of miss-classified points.

For the non-linearly separable case we selected the Circles and Moons Datasets and generated both sets with 1500 points. We thought that these sets could be a suitable visual representation for what we want to convey.

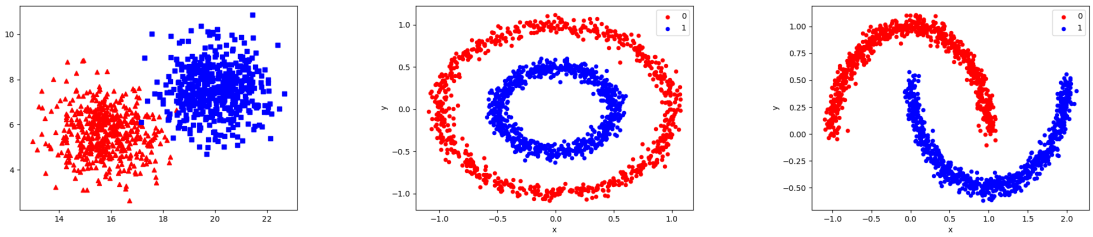

Figure 1: Plot of Blobs (**left**), Circles (**mid**) and Moons (**right**) datasets.

Remark: Once these Datasets were built and tested, we realized that working with them required a very high computational cost that we could not afford. Therefore, for learning purposes, we decided to reduce the Blobs dataset size by 80% and the remaining ones (Circles and Moons) by 90% in order to compute the optimal values of the hyper-parameters of each kernel. Always keeping the proportions balanced. Later on, for the KPCA calculation we would use the entire Datasets again.

4.2 Circles

We start by analyzing the circles dataset and then the moons one, to leave the linear dataset last analyzed.

Before making comparisons between different kernel functions, we have to select the hyper-parameters that benefit the most and behave better for each of them kernels.

RBF Reconstruction Error

For the RBF kernel, we only have σ as a parameter. We carry out a cross-validation on six different values, less than and greater than 1. The following table summarizes the reconstruction error obtained by applying the algorithm on each value of σ:

	$\sigma = \frac{1}{10}$	$\sigma = \frac{1}{5}$	$\sigma = \frac{1}{3}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 4$
Circles	1.1717	1.1727	1.0678	1.094	1.3317	1.3743

Table 2: RBF kernel reconstruction error for Circles dataset.

We observe how the error obtained is very similar for each execution, letting us understand that for this kernel function a value for σ close to one works fairly well. Even so, we highlight $\sigma = \frac{1}{3}$ for being the best result.

Polynomial Reconstruction Error

On the other hand, for the polynomial kernel we need to select the value of two hyper-parameters. We execute the same cross-validation and obtain the following results:

	$d = 4,\ c = 1$	$d = 3,\ c = 2$	$d = 2,\ c = 2$	$d = 2,\ c = 1$	$d = 3,\ c = 0$	$d = 2,\ c = 0$
Circles	1.8921	2.7649	3.8298	1.1841	0.8046	0.6974

Table 3: Polynomial kernel reconstruction error for Circles dataset.

From the results we conclude that the higher the value of c, the higher the error. Furthermore, values for d close to 1 also yield the lowest reconstruction error. Therefore, we highlight the pair of hyper-parameters $d = 2,\ c = 0$.

Sigmoid Reconstruction Error

For our last kernel, we repeat the previous selection process for two hyper-parameters and we get the following:

	$\alpha = 3,\ c = 2$	$\alpha = 3,\ c = 1$	$\alpha = 2,\ c = 1$	$\alpha = 1,\ c = 1$	$\alpha = 1,\ c = 0$	$\alpha = \frac{1}{2},\ c = 0$
Circles	0.9441	0.7292	0.8151	1.1370	0.6126	7.8863

Table 4: Sigmoid kernel reconstruction error for Circles dataset.

From the previous results we can see that this kernel is the one that generally behaves the best since it obtains the lowest errors regardless of the values of the hyper-parameters, except for $\alpha < 1$. Even so, and as in all cases, we highlight the pair $\{\alpha = 1,\ c = 0\}$.

Comparison

We conclude that the best reconstruction error is 0.6126 given by the sigmoid kernel function with values $\{\alpha = 1, c = 0\}$. Nevertheless, this error is not enough to face the one achieved by the classical PCA execution, that gave us 0.001, which is quite minor.

This are the plots of the projected data in both cases:

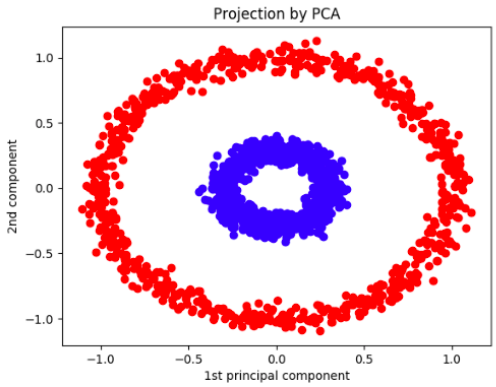

Figure 2: Circles PCA projection

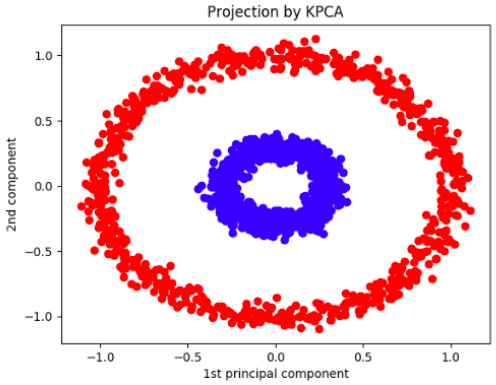

Figure 3: Circles KPCA projection

We can see in both plots that the reconstruction is practically perfect, which means that almost no quality of the data has been lost. Fact that we could deduce from the low value of the reconstruction error obtained.

If instead of evaluating performance of kernel PCA with the proposed reconstruction error, we evaluate it with supervised techniques for classification in two classes, the perception of the results would be very different. In fact, if we apply SVM to the resulting projection in order to classify points in red or blue classes, the combination that gives the lowest classification error is RBF kernel with $\sigma = \frac{1}{5}$. Here is the projected data on this last implementation:

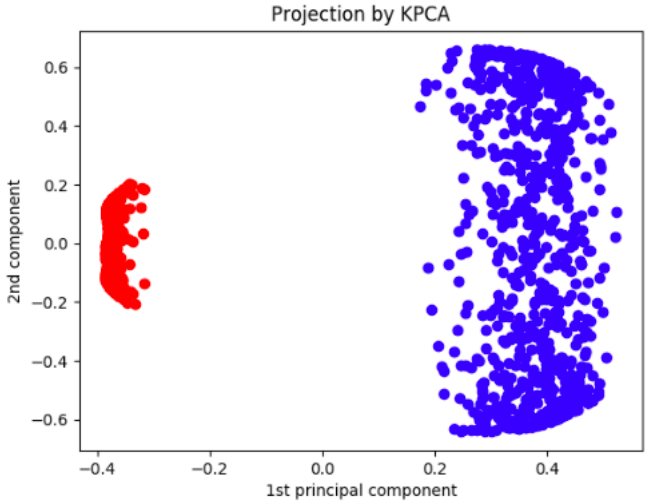

Figure 4: Circles KPCA projection with RBF

We can observe that data is now linearly separable, reason why SVM performs without error. Nevertheless the quality of data has been loss, which causes a reconstruction error of 1.17 in this case. This reconstruction error may seem low for the way in which data has changed, but it is still low because range of axis have not changed therefore distance error is low. We will observe other cases in further datasets that this same situation occurs but instead the error goes very high.

4.3 Moons

We continue with the moons dataset and repeat the procedure seen above about choosing parameters before comparing kernels.

RBF Reconstruction Error

	$\sigma = \frac{1}{10}$	$\sigma = \frac{1}{5}$	$\sigma = \frac{1}{3}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 4$
Moons	0.8051	0.8187	0.8112	0.8503	1.1170	1.2886

Table 5: RBF kernel reconstruction error for Moons dataset.

In the table above we can see the results of the cross-validation carried out with the hyper-parameter σ. From this we observe results similar to those obtained on the same kernel with the Circles dataset. That is, an average low error for all the chosen σ values, of which the lowest error is given by $\sigma = \frac{1}{10}$.

Polynomial reconstruction error

	$d = 4,\ c = 1$	$d = 3,\ c = 2$	$d = 2,\ c = 2$	$d = 2,\ c = 1$	$d = 3,\ c = 0$	$d = 2,\ c = 0$
Moons	15.9495	5.9005	5.1800	3.6068	0.6720	0.7582

Table 6: Polynomial kernel reconstruction error for Moons dataset.

In this case, with the Moons dataset, we see that for a high degree of the exponent d the error rises a lot. On the other hand, we get the same results as before as far as small error for low c values is concerned. The couple with better performance is $\{d = 3,\ c = 0\}$.

Sigmoid reconstruction error

	$\alpha = 3,\ c = 2$	$\alpha = 3,\ c = 1$	$\alpha = 2,\ c = 1$	$\alpha = 1,\ c = 1$	$\alpha = 1,\ c = 0$	$\alpha = \frac{1}{2},\ c = 0$
Moons	1.3013	0.9850	0.9148	1.4181	0.8182	0.6764

Table 7: Sigmoid kernel reconstruction error for Moons dataset.

Finally, with the Sigmoid kernel we do not observe any strange or atypical value. For this dataset we see that this time it is not the kernel that works best. The hyper-parameter combination with the lowest value for the error is $\alpha = \frac{1}{2},\ c = 0$. Which is surprising since for the previous dataset this pair had been the one with the highest error.

Comparison

It's clearly seen that the lowest reconstruction error is 0.6720 for polynomial kernel function applied with $\{d = 3, c = 0\}$. Although, in the same way as before, it is still not low enough to even match the error obtained with classic PCA, that has a value of 0.001.

The projected data has the following shape if we represent the plot of the first two principal components:

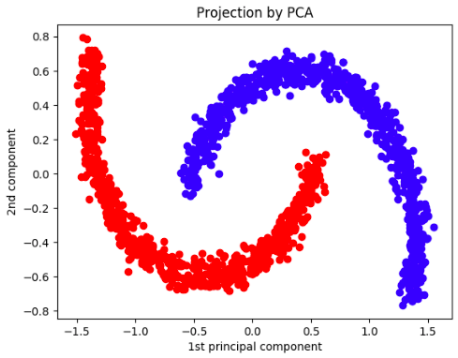

Figure 5: Moons PCA projection

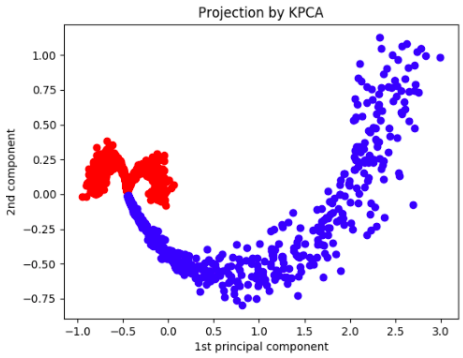

Figure 6: Moons best KPCA projection

We can observe in the plot of PCA projection (5) that despite the data has changed of place (can be observed on the plot in section 4.1), the patterns followed are the same and the quality has not suffered any lost. Despite this change of distribution, the reconstruction error has been zero, which supports the trustfulness of this measure. In KPCA (6), the pattern of the blue class has been maintained, but not the one of the red class, reason why the reconstruction error is higher.

It stands out among all errors recorded, the highest one obtained for polynomial kernel applied with degree = 4 and c = 1, that has a value of 15.94. We will plot the reconstruction and try to understand why this happens:

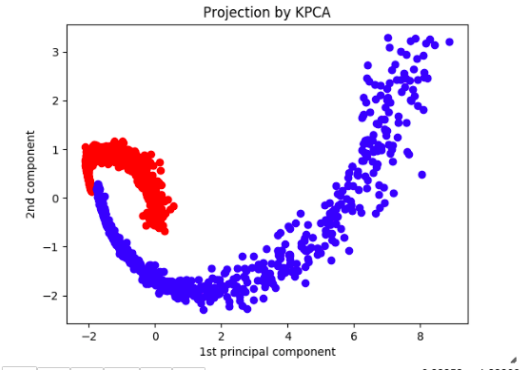

Figure 7: Moons Dataset projection with highest error

We can observe in the previous figure that the pattern is not far from the one in Figure 6 (6). It is a few worse for the red class but the pattern of blue class is still well represented. So, we arrive to the conclusion that the reason why the error difference is so big in this case is because the change of the axis scale. This suggest that sometimes the reconstruction error might seem bigger when trying some kernel functions but in reality the patterns of data have been maintained and the real problem is the scale.

4.4 Blobs

Finally we are going to check how the kernel PCA behaves with a linearly separable dataset.

RBF Reconstruction Error

	$\sigma = \frac{1}{10}$	$\sigma = \frac{1}{5}$	$\sigma = \frac{1}{3}$	$\sigma = 1$	$\sigma = 2$	$\sigma = 4$
Blobs	9.4208	8.9745	2.3792	2.4616	1.4632	3.1178

Table 8: RBF kernel reconstruction error for Blobs dataset.

For the first kernel, we see in the previous table how the lowest error is found for σ values close to one, both above and below. Specifically, the lowest one is given by $\sigma = 2$.

A remarkable fact is that this is the only dataset in which errors are so high for this kernel function, which makes us think about the relationship it has with linear separability.

Polynomial Reconstruction Error

	$d = 4,\ c = 1$	$d = 3,\ c = 2$	$d = 2,\ c = 2$	$d = 2,\ c = 1$	$d = 3,\ c = 0$	$d = 2,\ c = 0$
Blobs	2226705492	1143.6464	187216.18	161623.10	901.6288	774945.89

Table 9: Polynomial kernel reconstruction error for Blobs dataset.

Without a doubt, this combination of dataset and kernel function used for the execution has been the one that has surprised us the most. For all pairs of hyper-parameters the error rises to scales that we had not yet observed.

We do not highlight any pair, we conclude that this is not a good kernel to run KPCA on the Bloobs dataset.

Sigmoid Reconstruction Error

	$\alpha = 3,\ c = 2$	$\alpha = 3,\ c = 1$	$\alpha = 2,\ c = 1$	$\alpha = 1,\ c = 1$	$\alpha = 1,\ c = 0$	$\alpha = \frac{1}{2},\ c = 0$
Blobs	9.8584	9.5040	9.5465	9.6780	8.6535	7.8863

Table 10: Sigmoid kernel reconstruction error for Blobs dataset.

For the sigmoid kernel the error is quite similar in each pair of hyper-parameters. Even so, the lowest is again for $\{\alpha = \frac{1}{2},\ c = 0\}$, although it is not comparable with the one obtained by RBF kernel.

Comparison

We can see that the best reconstruction error is 1.4632 for RBF kernel function applied with $\sigma = 2$. Again, the error obtained by applying PCA continues to be magnitudes of a lower order than the best obtained with KPCA.

This are the graphics of the projected data in the previous case:

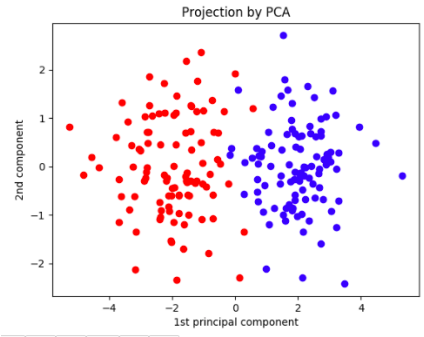

Figure 8: Blobs PCA projection

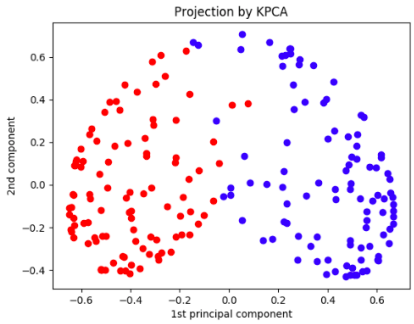

Figure 9: Blobs best KPCA projection

We can observe in PCA projection (8) that in the same way that it happened in Moons dataset, data has changed of place but the patterns followed are the same ones and the quality of data has not been lost. Instead, in KPCA projection (6), the data has also changed it is original distribution but the patterns are not the same.

Among all the errors obtained, it out-stands the one of more than two million for polynomial kernel applied with degree = 4 and c = 1. In the projection following image we can see how another time the problem is the scale, and the patterns of the data are not so bad. If we remember the conclusions from Moons dataset, the biggest error was obtained from Polynomial kernel too and for the same hyper-parameters. So from this, we can suspect that is the high degree applied what makes the scale increase so much.

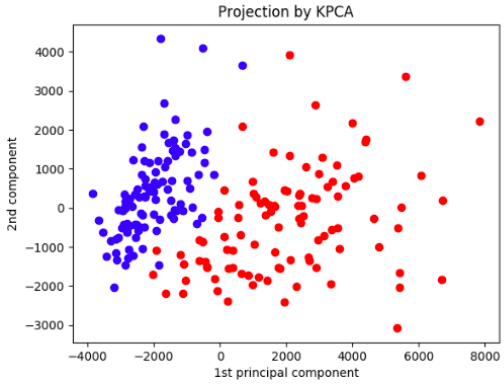

Figure 10: Blobs Dataset projection with highest error

4.5 Conclusions from reconstruction error

For all the datasets tested, we have seen how classical PCA performed without error and noticed that this could be related to the change in the patterns and scale of the data that KPCA applies. We have proven that when there are no big changes in the scale, reconstruction error works well for recognising if the patterns of the data are well maintained. We have also proven that this error is not a good measure if we want to evaluate if data is well prepared for a supervised algorithm.

It may seem in this previous analysis that kernel PCA it's not useful as it has been outperformed by classical PCA in all cases, but the reason behind this is that in this experiment we are applying kernel functions without reducing dimensions, so it was expected that patterns of data would change and therefore reconstruction error increased. We have to reduce dimensions to show how in non-linear datasets kernel PCA works better that PCA with respect of maintaining the quality and information of data when reducing dimensions.

5 Application of algorithm to a real dataset

Having understood how reconstruction error works, we will prove the power of kernel PCA with a real world dataset.

The dataset we have chosen after long research and testing with many others, is Money dataset, that is a 4-D dataset that has 1373 observations, which is good for us to have a similar measure to the ones we have been using previously.

The purpose now is to reduce to two dimensions the data with classical PCA and Kernel PCA and measure the reconstruction error to evaluate the performance. We will apply the three kernel functions presented in previous sections with different hyper-parameters. For this study, we will use fewer hyper-parameter combinations in the cross-validation due to the size of the real dataset.

RBF Reconstruction Error

	$\sigma = \frac{1}{5}$	$\sigma = \frac{1}{2}$	$\sigma = 1$	$\sigma = 3$	$\sigma = 5$
Money	7.65	8.068	8.214	9.595	8.817

Table 11: RBF kernel reconstruction error for Money dataset.

Polynomial Reconstruction Error

	$d = 2,\ c = 1$	$d = 3,\ c = 2$	$d = 2,\ c = 0$	$d = 4,\ c = 1$	$d = 2,\ c = 2$
Money	39580.57	1168.62	136035.96	2408274427.33	32916.17

Table 12: Polynomial kernel reconstruction error for Money dataset.

Sigmoid Reconstruction Error

	$\alpha = 2,\ c = 2$	$\alpha = \frac{1}{2},\ c = 0$	$\alpha = 1,\ c = 0$	$\alpha = 3,\ c = 1$
Money	7.898	7.111	6.246439	7.50405

Table 13: Sigmoid kernel reconstruction error for Money dataset.

In general, we see how the mean of errors for each combination of hyper-parameters in each kernel function is higher than those obtained for the toy datasets. The polynomial kernel also stands out for its low performance and the best error is given by the pair of values $\{\alpha = 1,\ c = 0\}$ with the Sigmoid kernel.

Comparison

Once we have presented the errors for all combinations of Kernel PCA, we compute the error of classical PCA that in this case is 6.33. For the combination of Sigmoid kernel with hyper-parameters $\{\alpha = 1, c = 0\}$, the reconstructed error its lightly smaller than for classical PCA with a value of 6.24.

If we observe the errors in the tables we can see how RBF and Sigmoid have in general similar errors to the ones presented in PCA. However, the errors for polynomial kernels are very high, probably because of the scale problem observed and commented in the toy datasets section.

Here we have the images of both PCA and KPCA projections obtained with the previously discussed reconstruction errors.

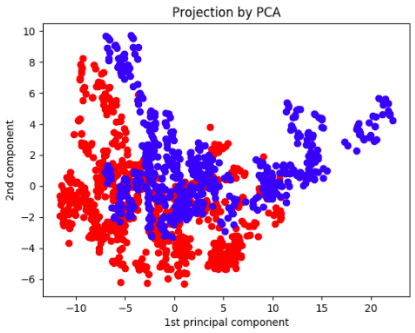

Figure 11: Money PCA projection

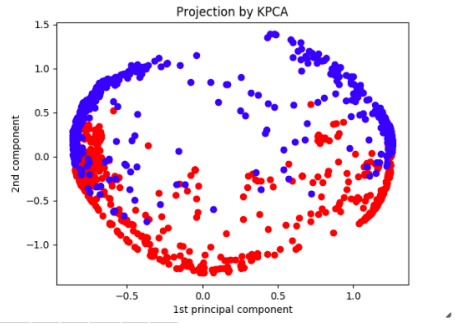

Figure 12: Money best KPCA projection

6 Conclusions

The most important conclusions that this study has given us are the following.

On the one hand, we have obtained less relevant conclusions such as the fact it has not been as easy as we thought at the beginning to evaluate the behavior of the KPCA, since we did not expect that by changing to feature space we were altering the possible comparisons. Talking about kernel functions used in the study, we can also say that the Polynomial has been the one that has given the worst performance.

On the other hand, and the most remarkable fact that has made us learn the most is about the usability of the reconstruction error. We have seen that using reconstruction error is not a good measure for deciding if Kernel PCA performs well for preparing data for a supervised method like SVM or other classification algorithms. Instead, it is, in fact, a good measure for deciding if performs well the reduction of dimensions for a certain dataset, this is, losing as little information as possible in the process and maintaining data quality.

7 References

[1] Paul Honeine and Cédric Richard, *The pre-image problem in kernel-based machine learning*
hal.archives-ouvertes.fr/hal-01965582/document

[2] Md. Ashad Alam and Kenji Fukumizu, *HYPERPARAMETER SELECTION IN KERNEL PRINCIPAL COMPONENT ANALYSIS*
https://thescipub.com/pdf/jcssp.2014.1139.1150.pdf

[3] https://www.cs.mcgill.ca/ dprecup/courses/ML/Lectures/ml-lecture13.pdf

[4] https://stats.stackexchange.com/questions/131142/how-to-choose-a-kernel-for-kernel-pca