Experimental Study of Kernel Methods in PCA, KMeans, MMD and LMNN Metric Learning

Ismail BACHCHAR [1]

Abstract

In this work, we implement and study different kernel methods for Principal Component Analysis (PCA) and KMeans. We provide their code implementation in Python and compare their results separately using different classical databases such as Moon, Swiss, and Circles datasets. Then, the Maximum Mean Discrepancy (MMD) distance is calculated over several domains from the Office/Caltech dataset using PCA and KPCA as feature transformation methods in order to capture the MMD shift. Finally, the Large Margin Nearest Neighbor (LMNN) metric learning method is presented to compare the 1-nearest neighbor algorithm performance and the MMD distance using Iris and Wine datasets in the original feature space and then in PCA and KPCA feature spaces.
All along these experiments, we do the benchmark study using Scikit-Learn code implementation alongside our own implementation of some methods in order to study their performance, such as execution time.

Code source: Advanced Machine Learning Kernel methods

Keywords— Machine Learning, Kernel Methods, KPCA, KMeans, MMD, LMNN, Metric Learning

1. Kernel Functions

1.1. The Concept of Kernel

The similarity measure, in input or output space, is the core of learning theory, therefore it should be chosen accordingly. Let's consider a similarity measure of the form:

$$\begin{aligned} k\colon \boldsymbol{X} \times \boldsymbol{X} &\to \mathbb{R} \\ (\boldsymbol{x}, \boldsymbol{x}') &\mapsto k(\boldsymbol{x}, \boldsymbol{x}') \end{aligned}$$

[1]Jean Monnet University, Saint-Etienne, France. Ismail BACHCHAR <isbachchar@gmail.com>.

that is, a function that, given two data points (vectors) $\boldsymbol{x}$ and $\boldsymbol{x}'$ in $\boldsymbol{X}$, returns a real number characterizing their similarity. This function k is called $\mathbf{kernel}$. With the properties:

- It's assumed that kernel k is symmetric, that is,

$$k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x}) \quad \forall \boldsymbol{x}, \boldsymbol{x}' \in \boldsymbol{X}$$

- kernel k is positive semi-definite (PSD)

$$\forall c_i \in \mathbb{R}, \boldsymbol{x}_i \in \boldsymbol{X}, \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0$$

- Kernels are like measures are (often) non-negative:

$$k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$$

- Kernels are expressing similarity measurement:

$$[k(\boldsymbol{x}, \boldsymbol{x}') \leq 1 \quad \text{and} \quad k(\boldsymbol{x}, \boldsymbol{x}') = 1] \Longleftrightarrow \boldsymbol{x} = \boldsymbol{x}'$$

1.2. Kernel matrix

In general, a kernel matrix (or Gram matrix) of a set of vectors (samples in high-dims) $(\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}) \in \boldsymbol{X}$ is defned to be a $n \times n$ matrix $\boldsymbol{K}$ as:

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1)\rangle & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \dots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{bmatrix}$$

For a well-defined kernel function, k.

1.3. Kernel Trick

Kernel-based methods sometimes need to transform the input data into some high-dimensional space, like in KPCA and KKMeans, for example, using a mapping function ϕ where $\phi : \mathbb{X} \longrightarrow \mathbb{H}$. Unfortunately, this high-dimensional mapping can seriously increase computation time, and sometimes we cannot explicitly calculate it when it mappes to infinity dimensions as in the Gaussian kernel.

To get around the computation cost problem, a kernel trick is used: $k(\boldsymbol{x}, \boldsymbol{x}\prime) = \phi(\boldsymbol{x})^T\phi(\boldsymbol{x}\prime)$, it only calculates the inner product between data points without even defining the mapping ϕ explicitly.
Given any algorithm that can be expressed solely in terms of dot products such as PCA, KMeans, and MMD, this trick allows us to construct different nonlinear versions of the algorithm.

1.4. Types of Kernel

Here is a list of common kernels: Let $\boldsymbol{x}$ and $\boldsymbol{x}'$ represent the feature vectors in input space $\boldsymbol{X}$

- *Linear Kernel*: is just the $dot - product$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle = \boldsymbol{x}^T\boldsymbol{x}'$$

- *Polynomial Kernel*: is useful for non-linear linear problems, it takes into account the interaction features. For degree-p polynomials, it is defined as

$$k(\boldsymbol{x}, \boldsymbol{x}') = (c + \boldsymbol{x}^T\boldsymbol{x}')^p$$

  where $c \geq 0$ is a free parameter representing the influence of higher-order versus lower-order terms in the polynomial.

- *Radial Basis Function (RBF)/Gaussian Kernel*: the most used kernel in machine learning. In particular, it is commonly used in support vector machine classification (SVM)

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2\right)$$

  where $\gamma = \frac{1}{2\sigma^2}$, and σ a free variable.

- *Sigmoid kernel*: defined as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \tanh\left(\gamma\boldsymbol{x}^T\boldsymbol{x}' + c\right)$$

  where $c \geq 0$ and $\gamma = \frac{1}{2\sigma^2}$ with σ a free variable.

- *Cosine kernel*: defined as:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{\boldsymbol{x}^T\boldsymbol{x}'}{||\boldsymbol{x}||||\boldsymbol{x}'||}$$

  where $c \geq 0$ and $\gamma = \frac{1}{2\sigma^2}$ with σ a free variable.

2. PCA and Kernel-PCA Comparison

2.1. Principal Component Analysis

PCA is a standard and popular statistical approach that tries to explain a large number of highly correlated variables (features of the data matrix) by a few components (new variables) that preserve important information, such as the variance. Final results of PCA are usually easy to interpret and make inferences with original variables because of the linear relationship between them.

Let X be an n × d matrix (n: samples and d: variables). The goal of PCA is to project the data onto a space having dimensionality $r \leq d$ while maximizing the variance/distance of the projected data points.

To begin with, we'll consider the projection onto a one-dimensional space (r = 1). We can define the direction of this space using a d-dimensional vector u_1. Each data point $\boldsymbol{x}_n$ is then projected onto a scalar value :

$$\boldsymbol{u}_1^T\boldsymbol{x}_n = \sum_{i=1}^{d}\boldsymbol{u}_{1i}\boldsymbol{x}_{ni}$$

The mean of the projected data is $\boldsymbol{u}_1^T\bar{\boldsymbol{x}}$ where $\bar{\boldsymbol{x}}$ is the data points mean given by

$$\bar{\boldsymbol{x}} = \frac{\sum_{i=1}^{n}\boldsymbol{x}_i}{n}$$

and then the variance of the projected data is given by the squared distance between projected data points and the projected mean

$$\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{u}_1^T\boldsymbol{x}_i - \boldsymbol{u}_1^T\bar{\boldsymbol{x}})^2 = \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1$$

($\boldsymbol{u}_1$ is assumed to be a unit vector, $\boldsymbol{u}_1^T\boldsymbol{u}_1 = 1$, since we are only interested in its direction) where $\boldsymbol{S}$ is the data covariance matrix defined as

$$\boldsymbol{S} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T$$

Now the PCA objective function is

$$\max_{\boldsymbol{u}} \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1 \quad \text{s.t.} \quad \boldsymbol{u}_1 = 1$$

After introducing the Lagrange multiplier (λ_1), we get the equivalent unconstrained maximization problem

$$\max_{\boldsymbol{u}} \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1 + \lambda_1(1 - \boldsymbol{u}_1^T\boldsymbol{u}_1)$$

By setting the derivative with respect to $\boldsymbol{u}_1$ equal to zero, we see that this quantity has a stationary point when

$$\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1 \iff \boldsymbol{u}_1^T\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1$$

So the variance will be a maximum when we set $\boldsymbol{u}_1$ equal to the eigenvector having the largest eigenvalue λ_1 of the covariance matrix S. This eigenvector is known as the first principal component. Additional principal components (up to d) can be defined in an incremental fashion by choosing each new direction to maximize the projected variance and to be orthogonal to those already considered.

A classical way of doing PCA is to use Singular Value Decomposition. For simplicity, we consider that $\boldsymbol{X}$ is already centered, which is a required step in PCA.

The SVD of $\boldsymbol{X}$ is given by:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$$

where $\boldsymbol{U} = (\boldsymbol{u}_1, ..., \boldsymbol{u}_d)$ is an $n \times n$ orthogonal matrix satisfying $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}_n$, $\boldsymbol{V} = (\boldsymbol{v}_1, ..., \boldsymbol{v}_d)$ is a $d \times d$ orthogonal matrix satisfying $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}_d$, and $\boldsymbol{D} = diag(d_1, ..., d_d)$ is a $d \times d$ diagonal matrix holding singular values. The singular values are assumed to be ordered such that $d_1 \geq d_2 \geq ... \geq d_d \geq 0$.
Where singular values are related to the eigenvalues of covariance matrix $\boldsymbol{S}$ via $\lambda_i = \frac{d_i^2}{n-1}$.
Eigenvalues λ_i show variances of the respective principal components.

2.2. Kernel Principal Component Analysis

The K-PCA extends the classical PCA to a high dimensional feature space $\mathbf{H}$ using a mapping ϕ. It can extract up to d non-linear principal components without explicitly calculating the mapping itself. In other words, K-PCA is just PCA, but instead of extracting the principal components in the input space, they are extracted from the feature space.

Let $k : \mathbf{R}^d \times \mathbf{R}^d \longrightarrow \mathbf{R}$ with the property that there exist a map, $\phi : \mathbf{R}^d \longrightarrow \mathbf{H}$, into a dot product feature space $\mathbf{H}$ such that:

$$\forall \boldsymbol{x}, \boldsymbol{x\prime} \in \mathbf{R}^d \quad k(\boldsymbol{x}, \boldsymbol{x\prime}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x\prime}) \rangle = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x\prime})$$

Then, the covariance matrix in feature space is defined as:

$$\boldsymbol{C} = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T$$

As in PCA, the goal is to solve this eigendecomposition problem:

$$\boldsymbol{C}\boldsymbol{v} = \lambda \boldsymbol{v} \iff \boldsymbol{v} = \frac{1}{n\lambda} \sum_{i=1}^{n} \langle \phi(\boldsymbol{x}_i), \boldsymbol{v} \rangle \phi(\boldsymbol{x}_i)^T$$

We know that a eigenvector $\boldsymbol{v_j}$, associated with $\lambda_j \neq 0$, is a linear combination of features of space $\mathbf{H}$: $\boldsymbol{v_j} = \sum_{i=1}^{n} \alpha_i^j \phi(\boldsymbol{x}_i)$, where α_i^j means the i-th value of j-th vector.

So, Finding an eigenvector is equivalent to finding the coefficients of α^j.
By substituting $\boldsymbol{v_j}$ back into formula above, we get:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i)(\sum_{l=1}^{n} \alpha_l^j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_l)) = \lambda_j \sum_{l=1}^{n} \alpha_l^j \phi(\boldsymbol{x}_l)$$

Multiply by $\phi(\boldsymbol{x}_p)^T$ for $p \in 1, 2, ...N$:

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_p)^T \phi(\boldsymbol{x}_i)(\sum_{l=1}^{n} \alpha_l^j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_l))$$
$$= \lambda_j \sum_{l=1}^{n} \alpha_l^j \phi(\boldsymbol{x}_p)^T \phi(\boldsymbol{x}_l)$$
$$\frac{1}{n} \sum_{i=1}^{n} k(\boldsymbol{x}_p, \boldsymbol{x}_i)(\sum_{l=1}^{n} \alpha_l^j k(\boldsymbol{x}_i, \boldsymbol{x}_l)) = \lambda_j \sum_{l=1}^{n} \alpha_l^j k(\boldsymbol{x}_p, \boldsymbol{x}_l)$$

By plugging in the kernel matrix described in section 1.2 we get:

$$\boldsymbol{K}^2 \boldsymbol{\alpha^j} = n\lambda_j \boldsymbol{K}\boldsymbol{\alpha^j}$$
$$\boldsymbol{K}\boldsymbol{\alpha^j} = n\lambda_j \boldsymbol{\alpha^j}$$

Projection of $\boldsymbol{x}$ onto j-th principal component is given by:

$$\phi(\boldsymbol{x})^T \boldsymbol{v}_j = \sum_{i=1}^{n} \alpha_i^j \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}_i) = \sum_{i=1}^{n} \alpha_i^j k(\boldsymbol{x}, \boldsymbol{x}_i)$$

In global, K-PCA steps:

- Kernel choice
- Normalization (when data are not centered)
- Calculation of Gram matrix $\boldsymbol{K}$
- Solving the eigenvalue problem: $\boldsymbol{K}\alpha^j = \lambda_j \alpha^j$
- Project (new or old) data points as $\boldsymbol{y}_i = \sum_{i=1}^{n} \alpha_i^j k(\boldsymbol{x}, \boldsymbol{x}_i), \quad i \in 1, ..$

2.3. Experiment Settings

We perform a comparative study between our implementation and Scikit-Learn implementation of PCA and KPCA on a benchmark of classical datasets available in the Scikit-Learn library. The used datasets and their characteristics are represented in Table 1

Datasets	X shape	y shape	number of classes
moon	(1000, 2)	(1000,)	2
circles	(1000, 2)	(1000,)	2
classes	(1000, 20)	(1000,)	2
swiss	(1000, 3)	(1000,)	1000
iris	(150, 4)	(150,)	3

Table 1. Used Datasets Characteristics.

In this experiment, we use all the previously described kernels in section 1.4 while fixing the same number of components (lower dimensions) to 1 for Moon and Circles datasets as they have only 2 dimensions in the original space, and to 2 for the other datasets. Regarding the kernel parameters, the following values were used $\gamma = 0.008$, $c = 0$, $degree = 4$.

2.4. Experiment Results

Embedding Criterion

The first thing we observe from Figure 1 is that the results of our implementation of PCA and KPCA with the different kernels are identical to those obtained by the same methods implemented in Scikit-Learn.
As depicted in Figure 1 first 5 columns, the results of our-PCA, Sklearn-PCA, our-linearPCA, and Sklearn-linearPCA are identical. This is because the coef0 or c parameter of linear-PCA is set to 0, which means as if we are exactly doing PCA. We did this just to verify the implementations of PCA and KPCA as we used two different approaches. In the first one, we used a Singular Value Decomposition approach for PCA, whereas the EigenDecomposition approach was used in the second one.
For the 1-dimensionality embeddings (for Moon and Circles datasets), all methods found a good embedding that preserved the data structure, in this case, the class separation. Except for the polynomial-Kernel, it discriminated the Circles dataset second class (red class points in original data), as depicted in Figure 1 first 2 rows.
The sigmoid-PCA obtained the same embedding as PCA for the Classes dataset. But the two methods got slightly different results on the Swiss dataset. The results of the Swiss dataset are different from kernel to kernel except for the linear-PCA and kernel-PCA, in which they somehow got the same embedding. Lastly, for the Iris dataset, the linear-PCA, sigmoid-PCA, our implementation, and Sklearn implementation got the same results and they both projected the data from 4-dimensions to only 2 while preserving the 3-classes linearly separated.

The same experiment was reproduced with only one change for the Moon and Circles dataset dimensional parameter from $r = 1$ to $r = 2$. As shown in Figure 2 same results were observed for the comparison between kernel methods embeddings. But for the Moon and Circles datasets, we see that neither linear PCA nor kernel PCA obtained a low-dimensional embedding where the classes are perfectly linearly separable. This latter confirms the tradeoff between the dimensionality reduction criterion and structure preservation criterion, the class, in our case separation, in PCA and KPCA. To add another criterion to this tradeoff, we made a performance comparison between PCA and KPCA, as explained in the following section.

Execution Time Criterion

In this part, we ran the same experiment previously presented with the number of dimensions for Moon and Circles datasets set to $r = 1$ and for the rest of the datasets $r = 2$, and then we compare the execution time of our own implementation of PCA and KPCA with different kernels.
As shown in Table 2, no matter the data size, PCA runs in only some 1-2 milliseconds, whereas linear-PCA, even if it gives the same result for the embedding as PCA but still more costly in the execution time. This is caused by the kernelization step within the code of linear-PCA. All the kernels in KPCA have similar execution times but are very costly compared to PCA.

As a conclusion of this comparative study, we found out that PCA is very useful and fast when we have linearly separable data, or at least we only care about the dimensionality reduction criterion. For the KPCA, even if it is very costly in terms of execution time compared to PCA, it still provides rich embeddings regarding data structure preservation, such as class separation. KPCA is very useful again when the data is not linearly separable in the original space. We don't mind not hugely reducing the dimensionality but rather care about structure criteria.

3. KMeans and Kernel-KMeans Comparison

3.1. KMeans

KMeans is an unsupervised algorithm that is a member family of the partitioning clustering algorithms. This family of methods groups the data into a set of size k of groups/clusters while satisfying the following criteria:

- (1) each group contains at least one point, and
- (2) each point belongs to exactly one group

KMeans reach that objective by minimizing a Euclidian distance between points of the same cluster and the mean of their cluster's centroid:

$$\sum_{i=1}^{k} \sum_{j=1}^{|C_i|} ||x_j - \mu_i||^2$$

where $\mu_i = \frac{1}{|C_i|} \sum_{l=1}^{|C_i|} x_l$ is the mean of cluster i.

3.2. Kernel KMeans

As in KPCA, Kernel KMeans is a non-linear version of the KMeans algorithm. Instead of working with the data points

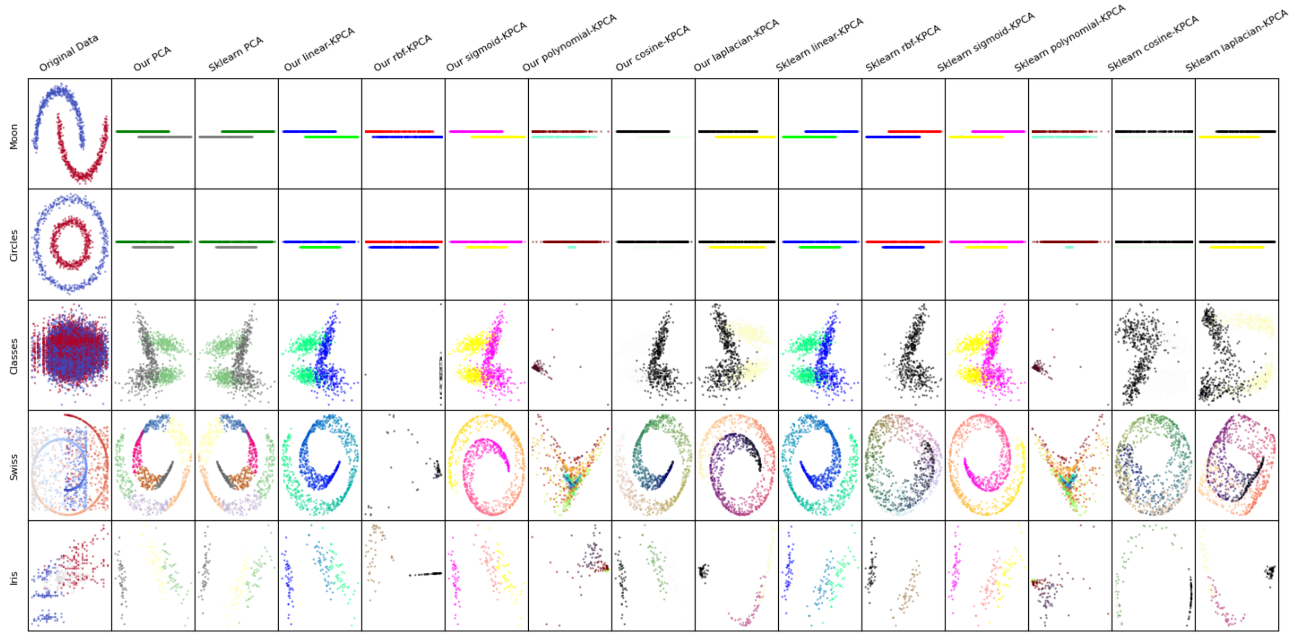

Figure 1. PCA and KPCA embeddings using our and Sickit-Learn implementations with r=1 dimensions

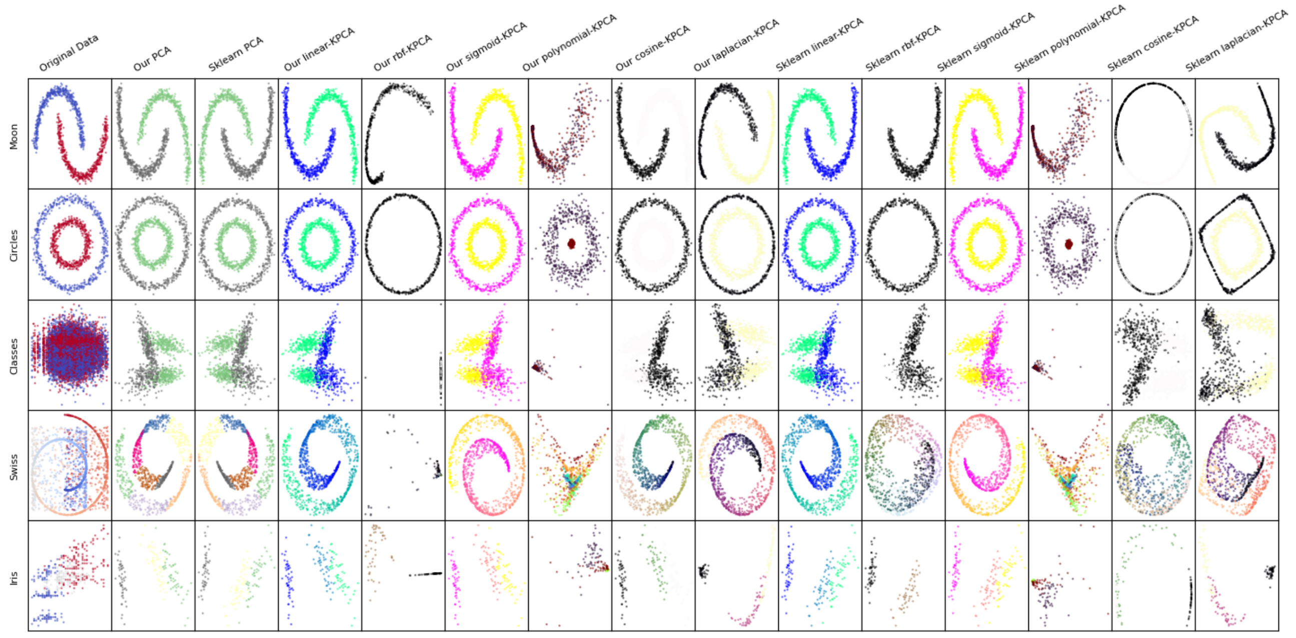

Figure 2. PCA and KPCA embeddings using our and Sickit-Learn implementations with r=2 dimensions

Kernels Datasets	PCA	linear-KPCA	rbf-KPCA	sigmoid-KPCA	polynomial-KPCA	cosine-KPCA	laplacian-KPCA
moon	0.0	1.2018	1.4304	1.3026	1.1632	1.1757	1.2
circles	0.001	1.3163	1.3504	1.2662	1.1097	1.2751	1.1665
classes	0.002	1.3387	0.9914	1.252	1.197	1.2801	1.3474
swiss	0.001	1.3245	1.3779	1.0626	1.166	1.3197	1.2198
iris	0.0	0.0474	0.0342	0.0369	0.0386	0.0385	0.0369

Table 2. Comparison between PCA and different Kernel PCA execution time (in secs).

in the original space, KKMeans maps them to a higher-dimensional feature space $\boldsymbol{H}$ using a mapping ϕ. KMeans is a good candidate to be kernelized as it uses the Euclidian distance, which introduces the dot product, and therefore no need to know the mapping ϕ explicitly; hence the kernel trick discussed in section 1.3.
The distance $||x_i - \mu_i||^2$ will be replaced by $||\phi(x_i) - \phi(\mu_i)||^2$, and let $k(x,y) = \langle\phi(x), \phi(y)\rangle$:

$$\begin{aligned} ||\phi(x_j) - \phi(\mu_i)||^2 &= \langle\phi(x_j), \phi(x_j)\rangle + \\ &\langle\phi(\mu_i), \phi(\mu_i)\rangle - 2\langle\phi(x_j), \phi(\mu_i)\rangle \\ &= k(x_j, x_j) + k(\mu_i, \mu_i) + 2k(x_j, \mu_i) \end{aligned}$$

By calculating the Gram matrix using any pre-defined kernel of the data matrix, the objective function of KKmeans becomes:

$$\sum_{i=1}^{k}\sum_{j=1}^{|C_i|} k(x_j, x_j) + k(\mu_i, \mu_i) + 2k(x_j, \mu_i)$$

3.3. Experiment Settings

We perform a comparative study between KMeans and Kernel KMeans on a benchmark of classical datasets available in the Scikit-Learn library. The used datasets and their characteristics are represented in Table 1

In this experiment, we use all the previously described kernels in section 1.4 while fixing the same number of clusters $k = 2$ for all the datasets. Regarding the kernel parameters, the following values were used $\gamma = 0.06$, $c = 2$, $degree = 3$.

3.4. Experiment Results

Clusters Visualization and Silhouette Score
As shown in Figure 3, all kernels KMeans seem to obtain the same result as obtained by classical KMeans (second column). This is supported by the Silhouette score shown in the Table 3, where all kernels seem to obtain the same score for the same dataset.
The worst obtained score is for the Classes dataset (second row). This could be explained by the fact that this dataset has 20 features which could be a lot for the KMeans algorithm to handle.

Execution Time
For the execution time criterion for KMeans and KKMeans as depicted in Table 4 we see that KMeans is very fast than its kernel version, which is not surprising as we know that the second one requires the kernelization step of input data.

4. MMD Distance Comparison using PCA and KPCA Feature Transformations

4.1. Maximum Mean Discrepancy Distance

A basic definition of the MMD distance is that it is the distance between two distributions, more precisely, between their feature means.
Given two probability distributions $\boldsymbol{P}$ and $\boldsymbol{Q}$ over an instance space $\boldsymbol{X}$, the MMD can be defined as follows:
$MMD(P, Q) = ||\mathbb{E}_{x\sim P}[x] - \mathbb{E}_{y\sim Q}[y]||$

4.2. Kernel Maximum Mean Discrepancy Distance

As in previous kernelized methods, PCA and KPCA, since MMD measures a distance, it is also a good candidate for kernelization.
First, we map x to $\phi(x)$, then we calculate the squared distance in order to introduce the dot product
Let $k(x,y) = \langle\phi(x), \phi(y)\rangle$

$$\begin{aligned} MMD^2(P,Q) &= ||\mathbb{E}_{x\sim P}[\phi(x)] - \mathbb{E}_{y\sim Q}[\phi(y)]||^2 \\ MMD^2(P,Q) &= \langle\mathbb{E}_{x\sim P}[\phi(x)], \mathbb{E}_{x\sim P}[\phi(x)]\rangle \\ &+\langle\mathbb{E}_{y\sim Q}[\phi(y)], \mathbb{E}_{y\sim Q}[\phi(y)]\rangle \\ &-2\langle\mathbb{E}_{x\sim P}[\phi(x)], \mathbb{E}_{y\sim Q}[\phi(y)]\rangle \end{aligned}$$

Then:

$$\begin{aligned} MMD^2(P,Q) = \mathbb{E}_{x,\sim P,x,\sim P}[k(x,x)] + \mathbb{E}_{y,\sim Q,y,\sim Q}[k(y,y)] \\ -2\mathbb{E}_{x,\sim P,y,\sim Q}[k(x,y)] \end{aligned}$$

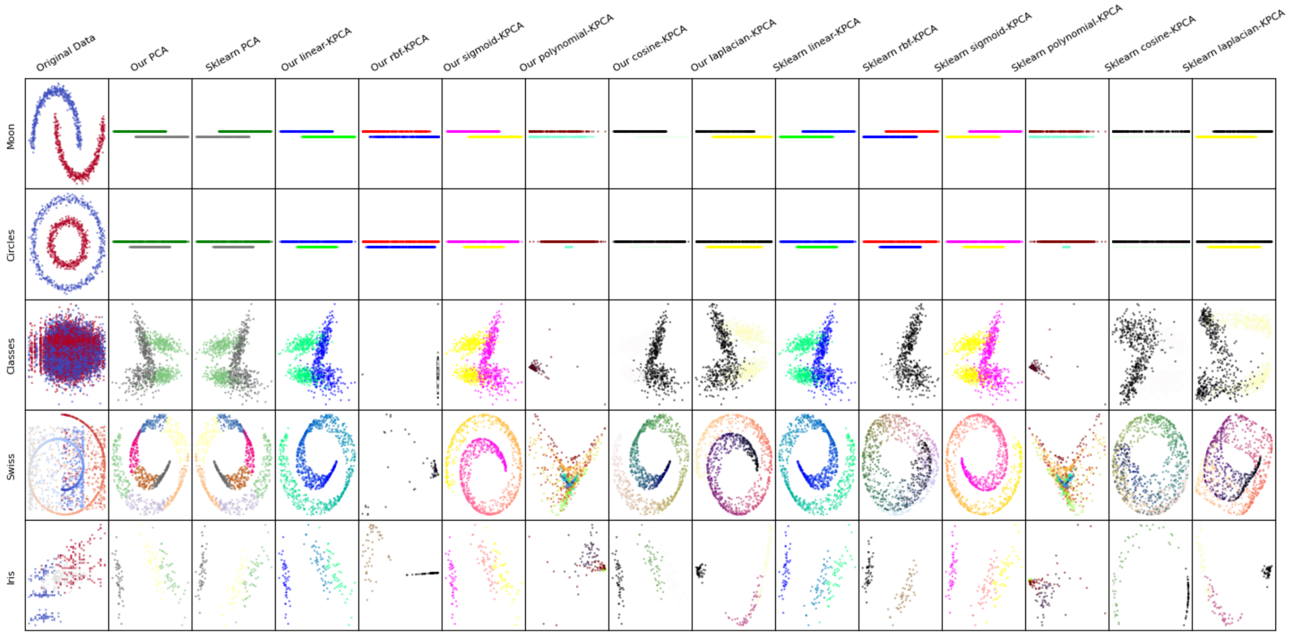

Figure 3. KMeans and Kernel KMeans results

	Kmeans	linear	rbf	sigmoid	polynomial	cosine	laplacian
moon	0.494	0.494	0.494	0.492	0.426	0.493	0.494
circles	0.351	0.354	0.351	0.349	0.355	0.354	0.348
classes	0.087	0.087	0.086	0.069	0.086	0.058	0.081
swiss	0.272	0.224	0.131	0.259	0.216	0.269	0.251
iris	0.681	0.650	0.671	0.161	0.516	0.650	0.657

Table 3. A comparison of Kmeans and kernel Kmeans silhouette score.

	Kmeans	linear	rbf	sigmoid	polynomial	cosine	laplacian
moon	0.0565	0.2057	0.1582	0.1441	0.1482	0.1202	0.224
circles	0.0812	0.1129	0.2301	0.1498	0.224	0.1482	0.18
classes	0.1103	0.1693	0.3524	0.2538	0.2174	0.2802	0.4858
swiss	0.1057	0.2414	0.2107	0.2237	0.2229	0.1262	0.2574
iris	0.005	0.002	0.003	0.0066	0.004	0.003	0.003

Table 4. A comparison of Kmeans and kernel Kmeans execution time (in sec).

4.3. Experiment Settings

In this part, the objective is to compare the MMD distance between the different domains of the Office/Caltech dataset. We calculate the distance in the original feature space after a PCA projection and then after a Gaussian kernel PCA projection using. We project the following dimensions $[1, 2, 4]$. We work on the Office/Caltech dataset, which contains GoogleNet1024, surf, and CaffeNet4096. Each dataset provides the extracted FTS -Fourier Transform Spectrometer- of images from four domains: Caltech10, Amazon, Webcam, and DSLR.

In the following parts, we compare the MMD distances between every pair domain of every dataset alongside after PCA transformation from our implementation and Scikit-Learn implementation in order to verify that our code works as well, then KPCA with RBF kernel and same default gamma parameter provided by Sklearn $(1/features)$. We couldn't run the experiment with our implementation of KPCA due to the expansive required execution time.

4.4. Experiment Results

As depicted in the Table 5 and Table 6, 7 where each table groups the MMD distances of a specific dataset, the MMD distances are in all the three cases have big values only in the original space, without using any feature transformation. Surprisingly, when doing this later as a preprocessing step, the MMD calculated in the new feature spaces, whether it's PCA and KPCA (not only RBF kernel), is always near zero. Actually, we have no explanation for this behavior!

5. LMNN Metric Learning Comparative Study

In this part, we choose to work with the Large Margin Nearest Neighbor metric learning algorithm available in the Python library metric_learn. As they define it:
LMNN learns a Mahalanobis distance metric in the kNN classification setting. The learned metric attempts to keep close k-nearest neighbors from the same class while keeping examples from different classes separated by a large margin. This algorithm makes no assumptions about the distribution of the data.

5.1. Experiment Settings

For this last part, we choose to work on the Iris and Wine classical datasets available in the Scikit-Learn library as both have the same number of labels/classes, 3. The objective is to compare the 1-Nearest Neighbors accuracy across different KPCA projections in the cases of transforming and not transforming the data by LMNN.
The second study concerns making the same approach, but instead of 1-NN accuracy, we compare the MMD distance between every pair of the datasets' labels, such as label 0 with label 1 (0-1). In both cases, the LMNN is tested on a separated test_set that has not been used for the training.

5.2. Experiment Results

1NN Accuracy Comparison

As presented in Table 8 we see that the 1NN accuracy decreased when the LMNN transformation was applied. This decrease is big on the original space but less effective when the data have been transformed using KPCA. This change is heavily affecting the Iris dataset. This could mean that the LMNN doesn't help the Iris dataset to cluster its labels, or it could not transform the data into another feature space that satisfies points of the same cluster near each other and far from other points. **MMD Distance Comparison**
In this part, we compare the MMD distance in between datasets' labels in different feature spaces. As shown in Table 9, the MMD distance increases when calculated between different Iris dataset labels when no KPCA transformation is used, such as in the first row. But for the Wine dataset labels, the MMD distance decreases.
The only case when we get MMD values in the same range between labels before and after applying the LMNN is when the RBF-PCA with 3 dimensions is applied on the Wine dataset (row 6).

Code and Data

The code implementation of this project is available on GitHub (Advanced Machine Learning Kernel methods). Presented in a reproducible manner so that the same results will be obtained. The code is organized into folders, where every folder contains a .py with the methods implementations and a notebook file with the experiment's code. For example, the LMNN metric method is available in the folder Metric Learning which contains ml_lmnn.py and results_LMNN.ipynb notebook.
Within the repository, we provide the Office/Caltech dataset, a well-known dataset in the domain adaptation field. It contains images' extracted FTS -Fourier Transform Spectrometer- features. The other used datasets, such as Iris, Wine, and Circles, are available through the Scikit-Learn library.

6. Conclusion

In this project, we used several kernels and their corresponding parameters to examine as many combinations as we can, but not all of the results are presented in this writing. Instead, the code of the experiments alongside the implementations of the algorithms is available on GitHub for further analysis.

	Caltech10_amazon	Caltech10_webcam	Caltech10_dslr	amazon$_{w}ebcam$	amazon_dslr	webcam_dslr
CaffeNet4096	115.12646484375	295.362060546875	431.4677734375	389.96240234375	539.395263671875	177.2685546875
CaffeNet4096 PCA (1)	8.282490084342775e-16	-5.66048974171885e-16	-2.254717404464567e-15	-2.585826924962953e-15	-2.7719942793549874e-15	1.0676183062461301e-16
CaffeNet4096 PCA$_s(1)$	7.865228832244128e-16	1.0708129985300334e-15	-1.750583181753629e-15	1.497052359894327e-15	1.0240853893325701e-15	-1.5023860843956285e-15
CaffeNet4096 PCA (2)	-8.380705477586108e-16	-8.42733054773637e-16	-2.1673257203662705e-15	7.620594329593097e-18	5.849057887345095e-16	-1.7668293920761526e-15
CaffeNet4096 PCA_s (2)	-2.440902695085051e-16	3.6124536880278914e-16	2.5198961188739055e-15	-1.148409577169897e-15	9.293955142056659e-16	2.231963419537944e-15
CaffeNet4096 PCA (4)	5.819886525431791e-16	-2.491133429639282e-16	-2.573777215543655e-15	-3.11149490963089e-15	4.929727860164388e-16	-1.5893394704290488e-16
CaffeNet4096 PCA_s (4)	1.2093048168121995e-15	-1.3253861727975097e-15	-2.5502367576006835e-16	-6.798105077173381e-16	-7.934800802063751e-16	-1.5456598976260237e-16
CaffeNet4096 rbf-KPCA (1)	-1.2494184669741293e-20	-8.700682185550472e-20	1.6755408631058207e-19	-9.810298049341206e-20	2.536859260353638e-19	7.299231993339358e-20
CaffeNet4096 rbf-KPCA (2)	-1.3979700612578757e-19	6.918033143755209e-20	9.03514874554691e-20	-8.395133247727479e-20	4.67947193055275e-20	1.6118649295719977e-19
CaffeNet4096 rbf-KPCA (4)	7.413191633942823e-20	2.3488903692884914e-19	-1.7210545670842665e-20	-4.2043290131841066e-20	1.0893971657037267e-19	-1.9353407221716475e-19

Table 5. Pair domains MMD distance for CaffeNet4096 dataset in original space and after PCA and KPCA projections.

	Caltech10_amazon	Caltech10_webcam	Caltech10_dslr	amazon_webcam	amazon_dslr	webcam_dslr
GoogleNet1024	23.4381103515625	66.9559326171875	95.701416015625	84.83489990234375	118.3509521484375	41.06561279296875
GoogleNet1024 PCA (1)	1.9139351314786057e-16	-1.2146422089027707e-16	-7.272992345062915e-17	7.197094112567055e-17	3.9800244740700514e-16	1.6516492162952193e-16
GoogleNet1024 PCA_s (1)	-1.959496194256426e-16	9.578462742675417e-17	-7.027703902709192e-17	-1.6664774876288934e-17	1.200833387573839e-16	-5.757675336741626e-16
GoogleNet1024 PCA (2)	2.0366805594943686e-16	3.4501459729454805e-16	-4.241979473586239e-16	-2.388191311558865e-18	9.211291586089165e-17	-4.725428495852632e-16
GoogleNet1024 PCA_s (2)	3.0542919823066897e-17	6.413442140473848e-16	2.3249050463477235e-17	-3.432350021380469e-16	6.141892961452116e-18	7.935540023875868e-16
GoogleNet1024 PCA (4)	7.546152107334088e-16	4.777195683690828e-16	9.61810866518928e-17	5.201352272091014e-16	-7.003270452576789e-16	-6.00790009973229e-16
GoogleNet1024 PCA_s (4)	4.8218196330589884e-17	-3.305468407195436e-16	5.823602931035202e-17	4.134607536906433e-16	7.951091484850595e-16	3.3066924820981866e-16
GoogleNet1024 rbf-KPCA (1)	-5.604254002482921e-20	-2.9540447024214703e-19	9.353285529483403e-21	-2.778687686116786e-19	-4.188131224271412e-20	3.3758694361180553e-20
GoogleNet1024 rbf-KPCA (2)	-1.1727680148234032e-19	-2.580226421470728e-19	-2.289454212614352e-19	1.2466714923836088e-19	-2.6993182635458793e-19	-1.0159837228823717e-19
GoogleNet1024 rbf-KPCA (4)	-2.6881657342212645e-19	-1.2207441996283157e-19	4.178826154035423e-19	4.712300549391436e-19	2.321862657156929e-21	5.571342626597372e-19

Table 6. Pair domains MMD distance for GoogleNet1024 dataset in original space and after PCA and KPCA projections.

	Caltech10_amazon	Caltech10_webcam	Caltech10_dslr	amazon_webcam	amazon_dslr	webcam_dslr
surf	9.47858599390716	12.767228056985516	5.443264352493301	17.50231507534879	11.716756947309761	4.278696739977065
surf PCA (1)	2.2662331686810976e-16	-7.335529312003234e-17	2.5784672228990084e-17	1.3012661707605222e-16	-1.3875587053490724e-16	-2.3175941650417678e-17
surf PCA_s (1)	-2.7330784903229273e-16	1.7701399288091215e-16	1.197315251547489e-16	1.5311111653553566e-16	-1.1343945797442008e-16	2.8865426483432746e-16
surf PCA (2)	1.9620070315107016e-16	-1.9960397922497525e-16	1.9480767013810857e-16	-1.1582789353097914e-16	3.332131013625755e-16	2.1473023418947337e-16
surf PCA_s (2)	1.229514950776669e-16	-2.9196917850725856e-17	-1.611087394140478e-16	3.523265998780814e-18	1.312637244110757e-16	-4.1847877846166825e-17
surf PCA (4)	3.064113014007008e-17	-1.5051487757357633e-17	-7.385612851131842e-17	3.087447446285533e-16	-7.385713930984515e-17	1.3754327331360692e-16
surf PCA_s (4)	6.508829196071869e-17	2.0370021798365473e-17	4.8798131740984935e-17	-2.0869487663129346e-16	-1.5730143790907215e-16	7.511205397036452e-17
surf rbf-KPCA (1)	1.0900377946525287e-19	-1.3651646646988625e-19	-2.451801367843391e-19	6.897449169954716e-20	-2.499327534046805e-19	-6.4000570162885846e-21
surf rbf-KPCA (2)	3.8277415702502276e-20	1.2689383873141622e-19	-2.6006529796126557e-21	-1.404507060306706e-19	1.405962279227751e-19	-3.6694846318623137e-19
surf rbf-KPCA (4)	-1.081728058940885e-19	-5.145307141050148e-20	1.7902277141072936e-19	-1.5505029978634685e-19	-2.727362161864288e-20	1.1958425263918034e-19

Table 7. Pair domains MMD distance for surf dataset in original space and after PCA and KPCA projections.

Datasets	1NN os	1NN after lmnn
Iris	1.0	0.4667
Wine	0.6481	0.3704
Iris rbf-KPCA (2)	0.8889	0.5556
Wine rbf-KPCA (2)	0.3704	0.3704
Iris rbf-KPCA (3)	0.9111	0.3333
Wine rbf-KPCA (3)	0.3704	0.3333
Iris rbf-KPCA (4)	0.9778	0.6889
Wine rbf-KPCA (4)	0.3704	0.3704

Table 8. 1NN accuracy comparison between datasets and their KPCA transformation before and after LMNN projection.

Datasets	0-1 os	0-2 os	1-2 os	0-1 lmnn	0-2 lmnn	1-2 lmnn
Iris	10.489003348071265	22.253143598615935	2.333943783068804	46.991613121901935	95.88935532823876	9.027050976659723
Wine	361161.67985665216	213240.27073315647	19412.325194987352	184.36903704034614	83.97250098322752	26.292380984508327
Iris rbf-KPCA (2)	1.4540678188333527	1.5359100554085456	0.42513871539000014	83.36610878864359	83.34799940703058	8.866836843941945
Wine rbf-KPCA (2)	0.0014203300885756503	0.004227686720609728	0.004800002196953085	1.4257339729074243	4.336984612555827	1.1458078063897261
Iris rbf-KPCA (3)	1.4586554858084195	1.5423288502572015	0.42529210517914984	71.27702220769993	7.229123349796222	46.00290846756947
Wine rbf-KPCA (3)	0.0027805382079268966	0.00530392819481051	0.004816606843744713	0.003612161412078778	0.006797124314059067	0.0060906946539115835
Iris rbf-KPCA (4)	1.4589467808136543	1.5459678218579462	0.4312815116555431	19.965530796471725	6.38856867649004	18.568500250749977
Wine rbf-KPCA (4)	0.005141182383392 6165	0.006910802003975307	0.004888867559175523	1.2520009103012362	0.2600706422178728	0.46811342278061985

Table 9. MMD distance comparison between different labels (0-1) of datasets in original space (os) and after LMNN projection of train-datapoints only