STAT3007/7007 Kernel PCA Tutorial

Lars Østberg Moan (s48270645)

1 Introduction

1.1 Introduction to PCA

Principle component analysis (PCA) is a widely used technique in machine learning applications, either as a way of visualizing high-dimensional data, as a step in feature engineering or as a way of lowering the dimensionality before training a model.

The core idea behind PCA is to come up with a new set of axis that represent the data "better" than the original axis. By "better" PCA does this by maximizing variance of the data along those new axis. However, more often than not, PCA only uses a subset of these new axis. Those that captures the highest variance and therefore can "explain" the data with fewer axis.

This method of using PCA to lower the dimensionality of the data can be explained in 6 easy steps.

- 1. Calculate the covariance matrix C.

$$C = \text{Cov}(X) = \frac{1}{N}(X - \bar{X})^T(X - \bar{X}) \tag{1}$$

  Where $\mathbf{X}$ is the dataset.

- 2. Compute the eigenvalues and corresponding eigenvectors of C. Eigenvalues λ from solving this:

$$\det(C - \lambda I) = 0 \tag{2}$$

  Eigenvectors $\mathbf{v}$ from solving this:

$$C\mathbf{v} = \lambda\mathbf{v} \tag{3}$$

- 3. The eigenvector with the highest corresponding eigenvalue will explain the most variance in the data. Therefore we choose the $\mathbf{n}$ eigenvectors with the highest eigenvalues. These $\mathbf{n}$ eigenvectors will be the column vectors of the projection matrix $\mathbf{M}$.
- 4. Transform the original data onto the new axis by projecting it onto $\mathbf{M}$.

$$\boldsymbol{X'} = \mathbf{XM} \tag{4}$$

1.2 Motivation for Kernel PCA

However as we can see in Equation 4 the projection is linear, therefore it works great if the data is linearly separable. Not so well for data that is inherently nonlinear. In Figure 2 an example of data that is not linearly separable is shown, and where PCA fails to separate the data. This is the motivation for Kernel PCA. The ability to separate data with nonlinear structures.

Note that in this example no dimensionality reduction was done, i.e all the principle components were used. Throughout this tutorial you will see examples of this. Where I apply a method, but with no reduction. This is mainly to make it easier to visualize, but it is also important to understand that these techniques can also be looked as a way of transforming onto a new set of axis that describes the variance better, i.e a way of separating the data.

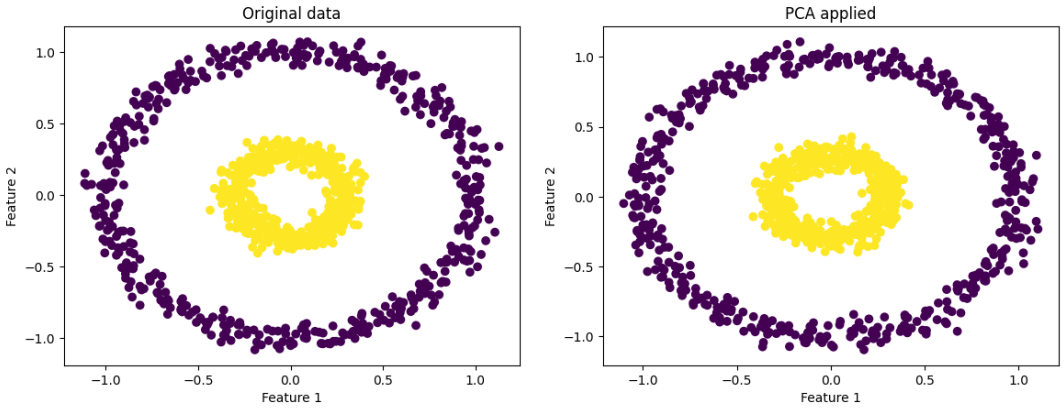

Figure 1: PCA applied to circular data. See: Listing 2

2 Kernel PCA

The main idea behind Kernel PCA is to project the data into a higher dimensional space where it might be linearly separable. A great example of this can be shown in Figure 2. Where the third dimension is defined as:

$$\mathbf{z} = \mathbf{x^2} + \mathbf{y^2} \tag{5}$$

It is clear that the data in $3D$ is clearly separable by a linear plane.

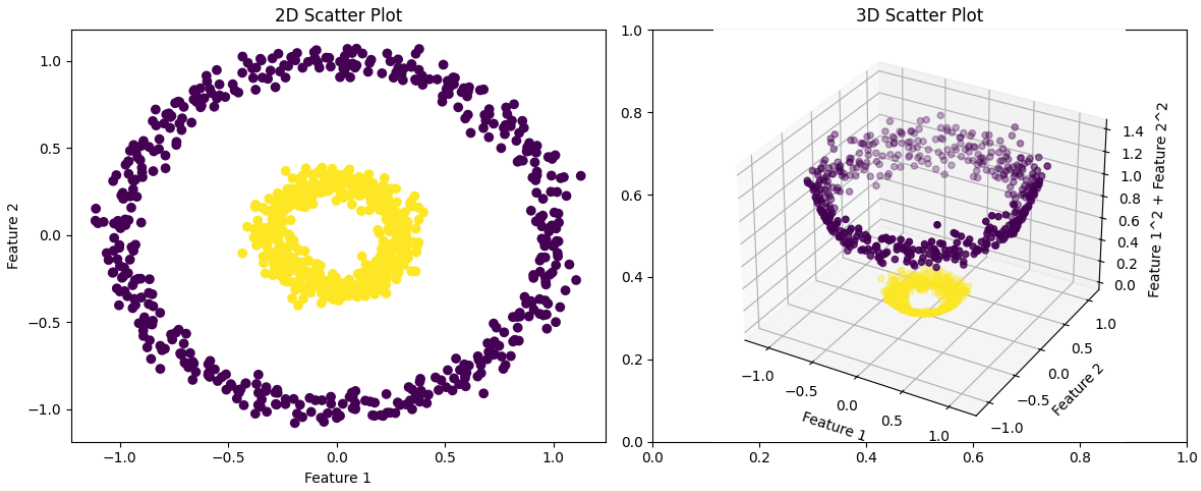

Figure 2: Projecting the data from $2D \to 3D$. See: Listing 1

2.1 Theory

As shown in the motivation kernel PCA utilizes the power of mapping the data into a higher dimension. This mapping is done using an arbitrary function $\mathbf{f}$.

Since this function is not specified, most often cross-validation is used to find the best function for our data. Most of the time, one of these are used:

- Polynomial: $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T}\mathbf{x}' + \mathbf{c})^d$
- Gaussian RBF: $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$

- Sigmoid: $K(\mathbf{x}, \mathbf{x}') = \tanh(\alpha \mathbf{x}^T \mathbf{x}' + \mathbf{c})$
- Cosine: $K(\mathbf{x}, \mathbf{x}') = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\| \|\mathbf{x}'\|}$

2.2 Applying Kernel-PCA

- 1. Computing the N-by-N kernel matrix $\mathbf{K}$ using the kernel trick.
  Given a dataset $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ where each $\mathbf{x}_i$ is a data point in the original feature space, the kernel matrix $\mathbf{K}$ is defined as

$$K_{ij} = \langle f(\mathbf{x}_i), f(\mathbf{x}_j) \rangle \tag{6}$$

  where $\langle \cdot, \cdot \rangle$ specifies the inner product in the high-dimensional space resulting of $f(\mathbf{x}_i)$.

  Using the kernel trick, we can express this inner product in terms of the kernel function k:

$$K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \tag{7}$$

  This is a very powerful result, because it means that instead of needing to explicitly calculating the higher dimensional feature vectors $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$ and then computing their inner product, we can directly compute the inner product using the kernel function k. *Kernel principal component analysis* 2023

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{8}$$

- 2. Centering $\mathbf{K} \rightarrow \overline{\mathbf{K}}$.
  Note that the kernel matrix $\mathbf{K}$ might not be centered, i.e have zero-mean. Therefore it needs to be centered before one can proceed with Kernel-PCA.
- 3. Compute the eigenvalues and eigenvectors of the centered kernel matrix $\overline{\mathbf{K}}$
- 4. Project the data $\mathbf{X}$ onto the new principle components by using Equation 4. But now $\mathbf{M}$ is defined by the eigenvectors of $\overline{\mathbf{K}}$.

2.3 Application and examples of Kernel-PCA

To follow the code that is used to create these plots and all the other plots in this tutorial, please have a look at: `https://github.com/larsmoan/Kernel-PCA`

2.3.1 Circular data

As shown in Figure 1 regular PCA was not able to separate the data from the circular dataset. In Figure 3 we can see how Kernel PCA is able to separate the data into linearly separable clusters. Another observation from this plot is how important the choice of kernel function is. Whilst the RBF is very successful at separating the data, the two other kernel functions are not. This shows the importance of choosing an appropriate kernel function, in practice this is often done using cross validation with hyper-parameter search.

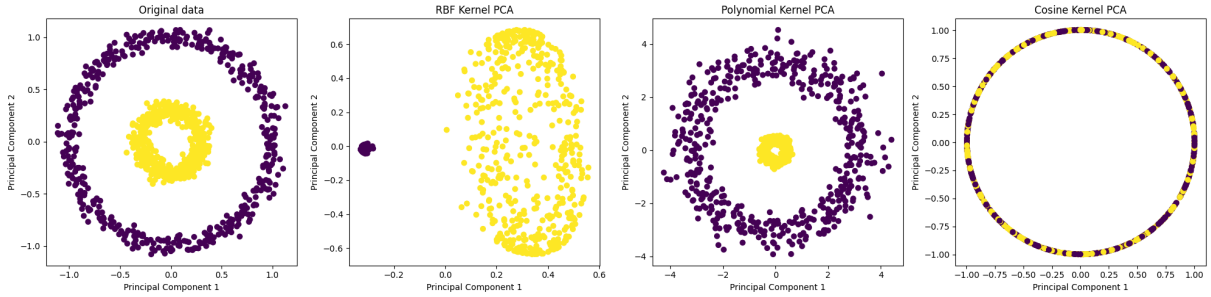

Figure 3: Kernel-PCA applied to circular dataset with different kernel funtions. See: Listing 5

2.3.2 Moon data

Another type of dataset that is often used to "benchmark" nonlinear dimensinality reduction techniques is the moon dataset. In the plots below we can see the performance of Kernel-PCA on this type of dataset. Once again is the RBF the best kernel function, and enables us to seperate the data with a line after applying Kernel-PCA.

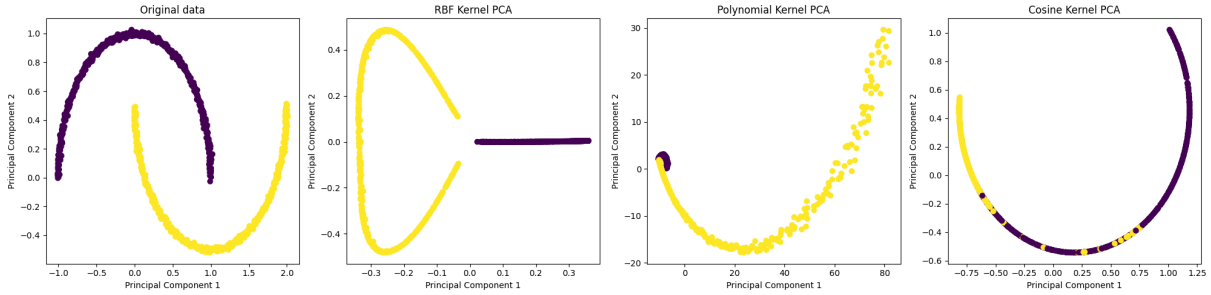

Figure 4: Kernel PCA applied to moon data with different kernel functions. See: Listing 5

2.3.3 Limitations of Kernel PCA

The main disadvantage of Kernel PCA is it's inability to reconstruct the data that is transformed. This can be shown in the Figure 5. Normal PCA is able to perfectly reconstruct the data if the number of components is equal to the original number of features. Kernel PCA applied with RBF however is not even close to reconstructing the data precisely. *Sci-kit learn Kernel PCA* 2023

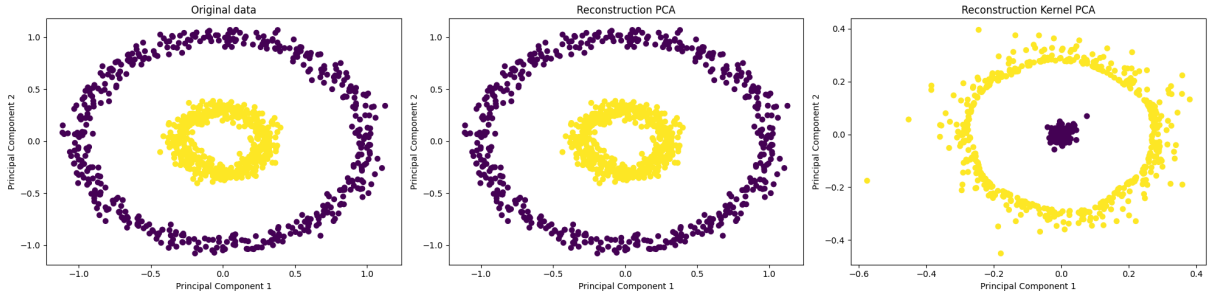

Figure 5: Kernel PCA's inability to reconstruct data precisely. RBF is used for Kernel PCA in the example. See: Listing 6

3 Questions

3.1 Q1

Why is the reconstruction using Kernel-PCA not exact?
Your answer doesn't have to include maths, but some explanation is needed.

3.2 Q2

As we saw in section subsection 2.2 centering the kernel matrix $\mathbf{K} \rightarrow \overline{\mathbf{K}}$ is needed before we can compute the eigenvalues and eigenvectors.
How is this done? Show your answer using mathematical expressions in matrix form.

3.3 Q3

It is stated that if RBF is used then the resulting higher dimensional space is infinitely dimensional, how can that be?
Your answer doesn't have to include maths, but some explanation is needed.

4 Solutions

4.1 Q1

This is because of the non-linearity introduced by the kernel function. In many cases it is not possible to obtain an analytical back-projection and therefore we are not able to obtain an exact reconstruction. This is because of the fact that the kernel function is in most cases not analytically invertible. Therefore the scikit-learn method for Kernel PCA applies a **Kernel Ridge** method for approximating the back projection.

4.2 Q2

This is the formula for centering the kernel matrix $\mathbf{K}$.

$$K' = K - \mathbf{1_N}K - K\mathbf{1_N} + \mathbf{1_N}K\mathbf{1_N} \tag{9}$$

gotten from *Kernel principal component analysis* 2023. Where $\mathbf{1_N}$ is the same dimension as the kernel matrix K but with all values equal to $\frac{\mathbf{1}}{\mathbf{N}}$.

4.3 Q3

This is a rather advanced topic and I advise you to read up on Hilbert spaces if you are more interested in this.

The main reason for why the higher dimensional space is infinite when using RBF is because of how the kernel matrix $\mathbf{K}$ scales with new additions to the dataset. Each time a new sample is added to $\mathbf{X}$ the rank of $\mathbf{K}$ scales by 1. Therefore the dimension of the higher dimensional space is said to be reliant on the number of samples $\mathbf{m}$ and therefore also infinite, since there are no limitations as to the size of our dataset. *Radial Basis function kernel* 2023

Appendix A.

Code used for the tutorial is shown below, can be more easily followed by having a look at the GitHub repository: `https://github.com/larsmoan/Kernel-PCA`.

5 Code

Listing 1: 2D → 3D plot of the circular data

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles, make_moons
from sklearn.decomposition import PCA, KernelPCA

x_circle, y_circle = make_circles(n_samples=1000, factor=0.3, noise=0.05, random_state=0)
x_moon, y_moon = make_moons(n_samples=1000,
                  noise=0.01,
                  random_state=0)


sum_squared = x_circle[:, 0]**2 + x_circle[:, 1]**2

x_3d = np.column_stack((x_circle, sum_squared))

fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# 2D scatter plot
axes[0].scatter(x_circle[:, 0], x_circle[:, 1], c=y_circle)
axes[0].set_xlabel('Feature 1')
axes[0].set_ylabel('Feature 2')
axes[0].set_title('2D Scatter Plot')

# 3D scatter plot
ax = fig.add_subplot(122, projection='3d')
ax.scatter(x_3d[:, 0], x_3d[:, 1], x_3d[:, 2], c=y_circle)
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 1^2 + Feature 2^2')
ax.set_title('3D Scatter Plot')

plt.tight_layout()
plt.show()
```

Listing 2: Applying regular PCA

```
pca = PCA(n_components=2, )
x_pca = pca.fit(x_circle).transform(x_circle)
plt.scatter(x_pca[:, 0], x_pca[:, 1], c=y_circle)
plt.title('PCA applied')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

Listing 3: Applying Kernel PCA

```
gamma=9
kernel_pca = KernelPCA(n_components=2, kernel="rbf", gamma=gamma)
x_kernel_pca = kernel_pca.fit(x_circle).transform(x_circle)
plt.title(f"Kernel PCA applied using radial basis function and gamma: {gamma}")
```

```
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.scatter(x_kernel_pca[:, 0], x_kernel_pca[:, 1], c=y_circle)
plt.show()
```

Listing 4: Comparing kernel functions with Kernel PCA

```
def plot_kernel_pca(x, y, hyperparams: dict):
    gamma = hyperparams.get('gamma', 9)
    degree = hyperparams.get('degree', 8)
    coef0 = hyperparams.get('coef0', 0.5)


    kernel_pca_rbf = KernelPCA(n_components=2,
                               kernel="rbf",
                               gamma=gamma)
    x_kernel_pca_rbf = kernel_pca_rbf.fit_transform(x)

    kernel_pca_poly = KernelPCA(n_components=2,
                                kernel='poly',
                                degree=degree)
    x_kernel_pca_poly = kernel_pca_poly.fit_transform(x)


    kernel_pca_cosine = KernelPCA(n_components=2, kernel='cosine')
    x_kernel_pca_cosine = kernel_pca_cosine.fit_transform(x)

    fig, axes = plt.subplots(1, 4, figsize=(20, 5))

    for ax, data, title in zip(axes, [x, x_kernel_pca_rbf, x_kernel_pca_poly,
    x_kernel_pca_cosine],
                               ["Original data",
                                "RBF Kernel PCA",
                                "Polynomial Kernel PCA",
                                "Cosine Kernel PCA"]):
        ax.scatter(data[:, 0], data[:, 1], c=y)
        ax.set_title(title)
        ax.set_xlabel("Principal Component 1")
        ax.set_ylabel("Principal Component 2")

    plt.tight_layout()
    plt.show()
```

Listing 5: Comparing kernel functions on circular and moon data

```
hyperparams = {
    'gamma': 9,
    'degree': 8,
    'coef0': 0.5
}
plot_kernel_pca(x_circle, y_circle, hyperparams)

#Test it with the half moon data as well
hyperparams = {
    'gamma': 19,
    'degree': 8,
    'coef0': 0.9
}
plot_kernel_pca(x_moon, y_moon, hyperparams)
```

Listing 6: Recunstruction for PCA and Kernel PCA

```
#Check the reconstruction with Kernel PCA compared to vanilla PCA

kernel_pca = KernelPCA(n_components=2, kernel="rbf", gamma=14, fit_inverse_transform=True,
    alpha=0.8)
x_kernel_pca = kernel_pca.fit(x_circle).transform(x_circle)
x_kernel_pca_reconstructed = kernel_pca.inverse_transform(x_kernel_pca)

#Vanilla PCA
pca = PCA(n_components=2)
x_pca = pca.fit(x_circle).transform(x_circle)
x_pca_reconstructed = pca.inverse_transform(x_pca)

fig, axes = plt.subplots(1, 3, figsize=(20, 5))
for ax, data, title in zip(axes, [x_circle, x_pca_reconstructed, x_kernel_pca_reconstructed],
                           ["Original data",
                           "Reconstruction PCA",
                           "Reconstruction Kernel PCA"]):
    ax.scatter(data[:, 0], data[:, 1], c=y_circle)
    ax.set_title(title)
    ax.set_xlabel("Principal Component 1")
    ax.set_ylabel("Principal Component 2")

plt.tight_layout()
plt.show()
```

References

Kernel principal component analysis (2023). Accessed on 06/05-2024. URL: https://en.wikipedia.org/wiki/Kernel_principal_component_analysis.

Radial Basis function kernel (2023). Accessed on 06/05-2024. URL: https://en.wikipedia.org/wiki/Radial_basis_function_kernel.

Sci-kit learn Kernel PCA (2023). Accessed on 06/05-2024. URL: https://scikit-learn.org/stable/auto_examples/decomposition/plot_kernel_pca.html.