Kernel Principal Component Analysis

APRIL 2024

Contents

Abstract

The prevalence of large datasets characterized by a multitude of variables presents significant challenges in interpretation. For this, Principal Component Analysis (PCA) is a statistical technique for 'linear' dimensionality reduction.

$Kernel - PCA$ is a kernel based prominent non-linear extension of the classical dimensionality reduction technique. In this report we compare the performances of PCA with it kernel versions using different kernels including rbf, laplace, sigmoid.

We also perform this method on a dataset and see how the different kernels help us distinguish our distinct groups most effectively.

Chapter 1

Introduction

1.1 What is PCA?

Principal Component Analysis (PCA) is a widely utilized unsupervised technique aimed at reducing the dimensionality of the data. Initially conceptualized by Karl Pearson as a statistical method for identifying optimal lines or planes of fit within regression analysis, PCA has since found widespread application across diverse domains.

This process enables the extraction of essential information while discarding redundant or noisy features. PCA's versatility extends to machine learning, where it serves as a pivotal tool for dimensionality reduction and denoising tasks, including applications such as image reconstruction.

1.2 Classic Principal Component Analysis(PCA)

PCA (Hotelling, 1933) aims to reduce the dimension D of a high dimensional data $\{x_i\} \in \mathbb{R}^D$ into lower dimension $\{\mathbf{y_j}\} \in \mathbb{R}^k$ where $k < D$ and still preserve a huge percentage of the variation in data in the original space $\mathbb{R}^D$, in the final subspace $\mathbb{R}^k$.

We consider a dataset $\mathbf{x_i} \in \mathbb{R}^p \ \forall \ i = 1, ..., n$. We intend to project the data onto k-dimensional subspace where $k \ll p$. Then for any vector $\mathbf{x} \in \mathbb{R}^p$ we denote this projection with $\hat{\mathbf{x}} = \mathbf{Ax}$ where $\mathbf{A} = [\mathbf{u_1}, ..., \mathbf{u_k}]^T$ and $\mathbf{u_i}^T\mathbf{u_i} = 1 \ \forall \ i = 1, ..., k$. We want to maximize the variance of the projected data on $u_1, ...u_k$, which is the trace of the covariance matrix Σ.

We start by centering the dataset, which is done as

$$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i}$$

for simplicity of mathematical notation lets say, after centering we have the datapoints $\mathbf{x_i} \ \forall \ i = 1, ..., n$. Now we can calculate the covariance matrix as,

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i}\mathbf{x_i}^T$$

After which we can move on to the computation of eigenvalue-eigenvector pairs $(\lambda_j, \mathbf{u_j})$ using,

$$\Sigma \mathbf{u_j} = \lambda_j \mathbf{u_j}$$

Then the eigenvectors are sorted according to the decreasing values of their corresponding eigenvalues. After this we can calculate the j^{th} principal component score of $\mathbf{x_i}$ as

$$\hat{x}_{i,j} = \mathbf{u_j}^T \mathbf{x_i}$$

If the eigenvalues are ordered as $\lambda_\mathbf{1} \geq ... \geq \lambda_\mathbf{p} \geq \mathbf{0}$ (as Σ is symmetric p.s.d) and the corresponding eigenvectors as $\mathbf{u_1}, ..., \mathbf{u_p}$. Define $\Gamma = [\mathbf{u_1}, ..., \mathbf{u_p}]$ from which we can get the projection as

$$\hat{\mathbf{x}}_\mathbf{i} = \Gamma \mathbf{x_i}$$

$\forall\ i = 1, ..., n$, where $\hat{\mathbf{x}}_\mathbf{i} = [\hat{x}_{i,1}, ..., \hat{x}_{i,p}]$.

So now we know how we can obtain principal component scores of any random vector in $\mathbb{R}^p$ and based on the variance explained by the components (by looking at the relative magnitude of the eigenvalues) we can select a subspace of the original space accordingly.

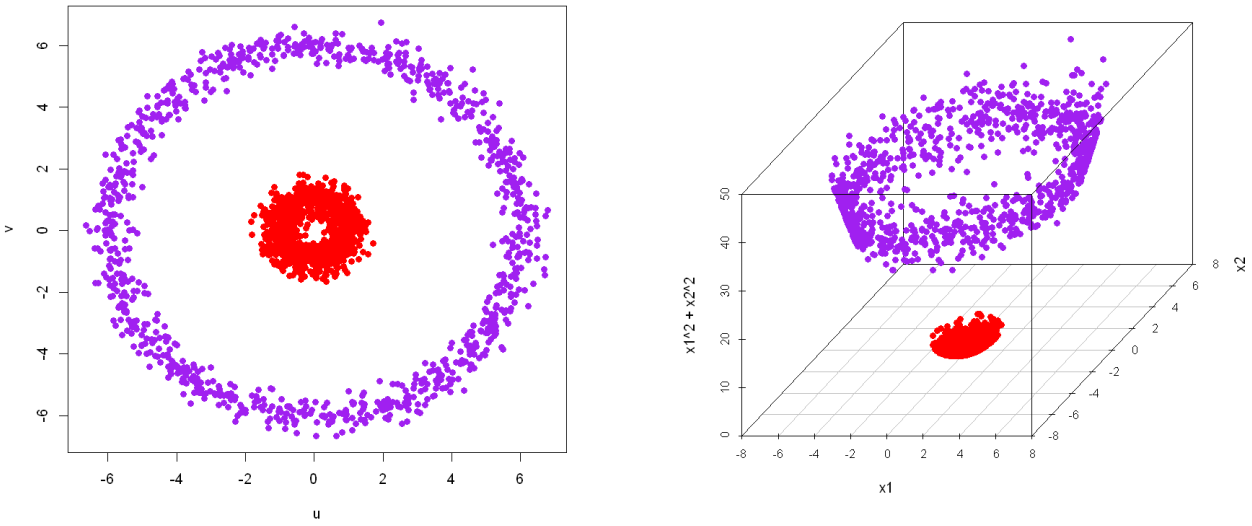
1.3 Examples where Classic PCA fails

Example1 : Consider a set of points in the $\mathbb{R}^2$ that are generated randomly using the following R code,

```
set.seed(123)
theta = runif(1000, -pi,pi)
x = cos(theta) + rnorm(200, 0, 0.3)
y = sin(theta) + rnorm(200, 0, 0.3)
u = 6*cos(theta) + rnorm(200, 0, 0.3)
v = 6*sin(theta) + rnorm(200, 0, 0.3)

plot(u, v, col='purple',pch = 19)
points(x, y , col='red',pch=19)
```

The corresponding graph produced is displayed in figure 1.1(a) below,



(a) 2D plot

(b) 3D plot with a new axis defined

Figure 1.1: Scatterplot of points generated using above R code

Here we can see a collection of two points here, namely the red points which form a smaller circle and purple points which form a larger circle. We can easily distinguish between the two set of points here, however if we perform a PCA of these points then what we get isn't desired.

```
prcomp(data.frame(x=x1,y=x2))

## Standard deviations (1, .., p=2):
## [1] 3.065787 3.036757
##
## Rotation (n x k) = (2 x 2):
##          PC1       PC2
## x  0.5188615 0.8548583
## y -0.8548583 0.5188615
```

The corresponding eigenvalues are the more or less the same, so we cannot drop either of the linear components here which means there has been no benefit of doing PCA here. This can be easily understood as the variance along both the axes is the same, so this type of output is what was expected.

So what we do now is define another axis here, namely $z = x^2 + y^2$, and plot the same points using the following code and get the figure 1.1(b) as shown above,

```
x1=c(x,u)
x2=c(y,v)
groups <- rep(c("red", "purple"), each = length(x))
scatterplot3d(x1,x2, x1^2+x2^2,color=groups,pch=19)
```

From the get go, you can understand that there is a clear distinction now of the two groups in the z-axis.

Note: For sake of simplicity we chose the radius of the larger circle to be 6, we could have to chosen a smaller radius say, 2, but then we would have to change the new axis defined appropriately so as to have clear separation among the two groups of points.

Now we can perform linear PCA on these axes and obtain the results accordingly ,

```
prcomp(data.frame(x=x1,y=x2,z=x1^2+x2^2))

## Standard deviations (1, .., p=3):
## [1] 17.602849  3.064689  3.036427
##
## Rotation (n x k) = (3 x 3):
##            PC1          PC2          PC3
## x  0.004633342  0.500780730 -0.865561779
## y -0.002754636 -0.865561394 -0.500795253
## z  0.999985472 -0.004704664  0.002630973
```

Looking at the eigenvalues we can now say there is major variance along the new axis, namely $z = x^2 + y^2$, which accounts for $\sim 75\%$ of the total variance.

So we see that sometimes we are not able to reduce the complexity of the data with the constraint of 'linearity'. Thus we have to ponder into the realm of non-linearity to devise new axes to better represent the variances of the components of the random vectors. One of the special cases of Non-linear PCA is the Kernel Principal Component Analysis which makes use of the theory of the kernels to produce new dimensions without explicitly requiring a map from $\mathbb{R}^p$ to $\mathbb{R}^m$ (m $>$ p).

Chapter 2

Kernel Principal Component Analysis (kernel-PCA)

Kernel methods have revolutionized many areas of machine learning by providing a powerful framework for handling non-linear relationships in data. The "kernel trick" is a key concept in kernel methods, enabling algorithms to be generalized to non-linear problems through the substitution of a kernel function for the inner product. This approach has led to significant advancements in various machine learning tasks, including classification, regression, dimensionality reduction, and clustering.

The essence of the kernel trick lies in the fact that many machine learning algorithms can be expressed in terms of inner products between data points, typically represented as a Gram matrix. These kernel functions effectively capture the similarities or dissimilarities between pairs of data points in a high-dimensional feature space, without explicitly transforming the data into that space. This implicit mapping allows algorithms to effectively handle non-linear relationships in the input data.

2.1 Kernel and its Properties

2.1.1 What is a kernel?

Consider the input data points $\mathbf{x_i} \in \mathbb{R}^p$, $i = 1, ..., n$. We start by giving the definition of a feature map,

Definition: A map $\Phi : \mathbb{R}^p \to \mathbb{R}^m$ is called a $feature\ map$ if,

$$\Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), ..., \phi_m(\mathbf{x}))^T \in \mathbb{R}^m \ (m > p), \tag{2.1}$$

where atleast one of the ϕ_j's is a non-linear map.

Given these points $\mathbf{x_1}, ..., \mathbf{x_n}$ we can non-linearly transform them into an m-dimensional $feature\ space$ as $\mathbf{\Phi(x_1)}, ..., \mathbf{\Phi(x_n)}$.

We now give the definition of a kernel,

Definition: A map $\kappa : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is called a Kernel (also called a positive definite kernel)

if it is symmetric, i.e, $\kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{y}, \mathbf{x}) \ \forall \ \mathbf{x}, \mathbf{y} \in \mathbb{R}^p$ and it is positive semi-definite i.e

$$\sum_{i=0}^{n}\sum_{j=0}^{n} c_i c_j \kappa(\mathbf{x_i}, \mathbf{x_j}) \geq 0 \tag{2.2}$$

$\forall \ c_1, ..., c_n \in \mathbb{R} \ \& \ \mathbf{x_1}, ..., \mathbf{x_n} \in \mathbb{R}^p$.

Proposition: Given a feature map $\Phi : \mathbb{R}^p \to \mathbb{R}^m$ if we define $\kappa : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ s.t,

$$\kappa(\mathbf{x}, \mathbf{z}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle \ \forall \ x, z \in \mathbb{R}^p \tag{2.3}$$

then, κ is a kernel.

Proof:

1. $\kappa(x, y) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{z}) \rangle = \kappa(y, x)$ (symmetirc)
2. $\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n} c_i c_j \kappa(\mathbf{x_i}, \mathbf{x_j}) = \sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n} c_i c_j \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle = || \sum\limits_{i=0}^{n} c_i \mathbf{\Phi}(\mathbf{x_i}) || \geq 0$
   hence, positive definite.

Hence, κ is a valid kernel.

2.1.2 Types of kernels

Below we have listed a few commonly used kernels,

- **Linear Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i^T x_j})$$
- **Polynomial Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i^T x_j} + c)^d$$
where $c \in \mathbb{R}$ is the offset and d is the degree of the polynomial.
- **RBF(Radial Basis Function) Kernel**
This is also known as the **Gaussian Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = exp(-\frac{1}{2\sigma^2}||\mathbf{x_i} - \mathbf{x_j}||^2)$$
- **Laplacian Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = exp(-\sigma||\mathbf{x_i} - \mathbf{x_j}||)$$
- **Sigmoidal Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = tanh\Big(\alpha(\mathbf{x_i^T x_j}) + c\Big)$$
here also α and c are respectively the scale and the offset parameters respectively.
- **Cosine Kernel**
$$\kappa(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x_i^T x_j}}{||\mathbf{x_i}||||\mathbf{x_j}||}$$

Later we will make use all these kernels on various datasets and see how each of them performs by conjuring new axes.

2.2 Kernel PCA

To carry out linear PCA in the feature space, we follow a similar procedure, but instead of directly computing the covariance matrix in the input space, we work with the covariance matrix estimated in the feature space. This means that we compute the covariance matrix using the feature vectors obtained by mapping the original data into the feature space.

2.2.1 PCA on the Feature map

We have the feature vectors of our input vectors in the feature space as $\Phi(\mathbf{x_1}), ..., \Phi(\mathbf{x_n})$. To compute the Covariance matrix Σ we need to take the centered versions of these vectors. So for sake of simplicity we assume that $\sum_{i=1}^{n} \Phi(\mathbf{x_i}) = 0$. Thus we have the covariance matrix as,

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} \Phi(\mathbf{x_i}) \Phi(\mathbf{x_i})^T \tag{2.4}$$

The eigenequation of this matrix is $\Sigma \mathbf{v} = \lambda \mathbf{v}$, where $\mathbf{v}$ is the eigenvector corresponding to the eigenvalue $\lambda \geq 0$ (by equation(2.3)) of Σ. Now take inner-product with $\Phi(\mathbf{x_i})$ to get,

$$\langle \Phi(\mathbf{x_i}), \Sigma \mathbf{v} \rangle = \lambda \langle \Phi(\mathbf{x_i}), \mathbf{v} \rangle \tag{2.5}$$

$\forall i = 1, ..., n$. We can get following from the equation (2.4)

$$\Sigma \mathbf{v} = \frac{1}{n} \sum_{i=1}^{n} \Phi(\mathbf{x_i}) \langle \Phi(\mathbf{x_i}), \mathbf{v} \rangle \tag{2.6}$$

So all solutions $\mathbf{v}$ with nonzero eigenvalue λ are contained in the span of $\Phi(\mathbf{x_1}), ..., \Phi(\mathbf{x_n})$. Implying there exist coefficients, $\alpha_i, i = 1, ..., n$, such that

$$\mathbf{v} = \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x_i}) \tag{2.7}$$

Substituting this and (2.6) in (2.5) we get

$$\frac{1}{n} \sum_{j=1}^{n} \alpha_i \sum_{k=1}^{n} \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_k}) \rangle \langle \Phi(\mathbf{x_k}), \Phi(\mathbf{x_j}) \rangle = \lambda \sum_{k=1}^{n} \alpha_k \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_k}) \rangle \tag{2.8}$$

for all $i = 1, ..., n$. Define the $(n \times n)$-matrix $\mathbf{K} = (K_{ij})$, where

$$K_{ij} = \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x_j}) \rangle$$

Note that $\mathbf{K}$ will usually be a huge matrix. Then, the eigenequation (2.8) as

$$\mathbf{K}^2 \boldsymbol{\alpha} = n \lambda \mathbf{K} \boldsymbol{\alpha} \tag{2.9}$$

where $\boldsymbol{\alpha} = (\alpha_1, ..., \alpha_n)^T$, or this can be written as

$$\mathbf{K}\boldsymbol{\alpha} = \tilde{\lambda}\boldsymbol{\alpha} \tag{2.10}$$

where $\tilde{\lambda} = n\lambda$. Note that we can express the eigenvalues and eigenvectors, $(\tilde{\lambda}, \boldsymbol{\alpha})$ of kernel matrix $\mathbf{K}$ in terms of those, $(\lambda, \mathbf{v})$, for covariance matrix Σ.

Denote the ordered eigenvalues of $\mathbf{K}$ by $\tilde{\lambda}_1 \geq ... \geq \tilde{\lambda}_n \geq 0$ with associated eigenvectors $\boldsymbol{\alpha_1}, ..., \boldsymbol{\alpha_n}$, where $\boldsymbol{\alpha_i} = (\alpha_{i1}, ..., \alpha_{i_n})^T$. If we require that $\langle \mathbf{v_i}, \mathbf{v_i} \rangle = 1, i = 1, ..., n$, then, using the expansion as in (2.7) for $\mathbf{v_i}$ and the eigenequation (2.8), we have that

$$1 = \sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_{ij}\alpha_{ik}\langle \Phi(\mathbf{x_j}), \Phi(\mathbf{x_k}) \rangle \tag{2.11}$$

$$= \sum_{j=1}^{n}\sum_{k=1}^{n} \alpha_{ij}\alpha_{ik}K_{jk} \tag{2.12}$$

$$= \langle \boldsymbol{\alpha_i}, \mathbf{K}\boldsymbol{\alpha_i} \rangle = \tilde{\lambda}_i \langle \boldsymbol{\alpha_i}, \boldsymbol{\alpha_i} \rangle, \tag{2.13}$$

which determines the normalization of the vectors $\boldsymbol{\alpha_1}, ..., \boldsymbol{\alpha_n}$.

Now if $\mathbf{x} \in \mathbb{R}^p$ is a test point, then the *nonlinear principal component scores* of $\mathbf{x}$ corresponding to Φ are given by the projection of $\Phi(\mathbf{x}) \in \mathbb{R}^m$ onto the eigenvectors $\mathbf{v_k} \in \mathbb{R}^m$,

$$\langle \mathbf{v_k}, \Phi(\mathbf{x}) \rangle = \lambda_k^{-1/2} \sum_{i=1}^{n} \alpha_{ki} \langle \Phi(\mathbf{x_i}), \Phi(\mathbf{x}) \rangle \tag{2.14}$$

Now using the kernel trick explained above, the nonlinear principal component scores of $\mathbf{x}$ can be expressed as

$$\langle \mathbf{v_k}, \Phi(\mathbf{x}) \rangle = \lambda_k^{-1/2} \sum_{i=1}^{n} \alpha_{ki} K(\mathbf{x_i}, \mathbf{x}), \quad k = 1, ..., n \tag{2.15}$$

2.2.2 Centering of Feature vectors

We made the assumption that $\sum_{i=1}^{n} \Phi(\mathbf{x_i}) = 0$, but it may not be always true. To go around this we can apply the transformation to the kernel matrix, $\mathbf{K}$ as

$$\tilde{\mathbf{K}} = \mathbf{HKH}$$

where $\mathbf{H} = \mathbf{I}_n - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$. The resulting matrix $\tilde{\mathbf{K}}$ corresponds to starting with a centered Φ as required in the above theory.

Chapter 3

Application and Problems

Now that we have introduced the theory of Kernels and its generalization into nonlinear PCA, we can move on to apply these techniques and see how we can benefit from this.

3.1 Dimensionality Reduction and Pattern Recognition

We pick a dataset that has a non-linear structure in higher dimensions. The idea is to reduce the dimensions of the data while also transforming the data using the kernel method so that we are able to interpret the dataset much more easily. Here, we pick the *Spirals* Dataset present in the 'RSSL' package of R. In this we will take 1000 samples which will have three variables denoting its position in $\mathbb{R}^3$ space and a variable *Class* denoting the group to which this observation belongs.

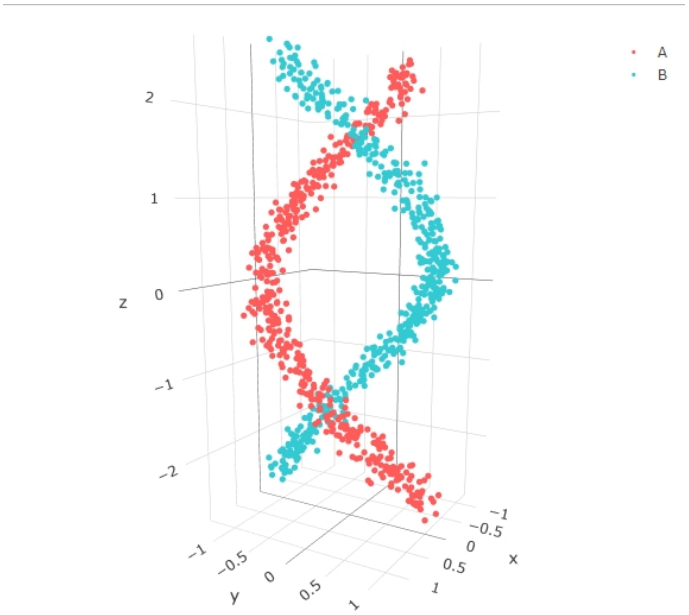
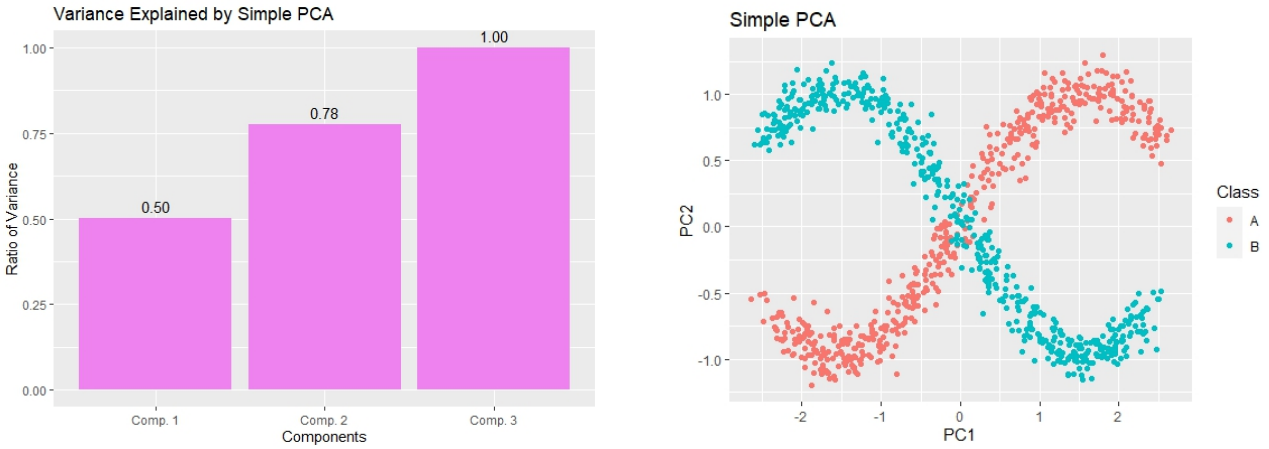


Figure 3.1: Original Graph of Spirals Dataset

Note: We chose a different random state(specifically 123), this is important as different datasets lead to different Principal Components in Kernel version.

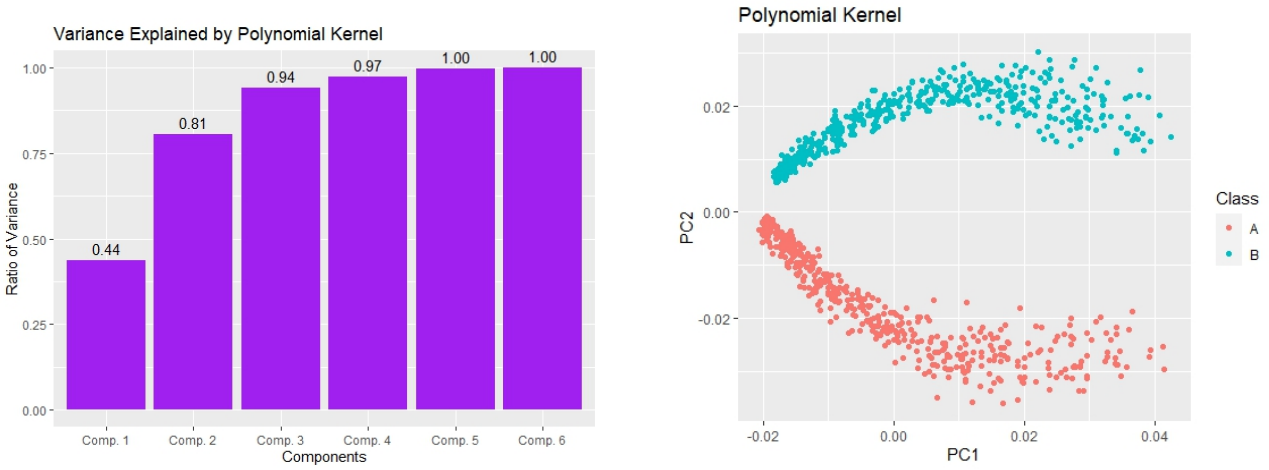
3.1.1 Simple PCA

We have the dataset in 3D space as shown in figure 3.1. Now we move to applying PCA and Kernel PCA on this dataset. For this, first we apply simple linear PCA to see how much we benefit from it. It is expected that it wont be able to capture much of the variance least of all not be able to separate the two groups.



(a) Variance Explained (b) Principal Components

Figure 3.2: Simple PCA results



(a) Variance Explained (b) Principal Components

Figure 3.3: Polynomial Kernel PCA results

3.1.2 Polynomial Kernel PCA

Since this isn't able to produce anything beneficial, we make use of non-linear PCA for this problem, namely with the Polynomial Kernel and the RBF/Gaussian Kernel. For the Polynomial kernel, we

are using the parameters as, $degree, d = 2$ and $offset, c = -75$. Corresponding to this we get,

We can see clearly with just 2 Principal Components we were able to explain more than 80% of the total variance. On top of that we have a clear boundary between the groups as shown in the Figure 3.3(b), hence classifying the two groups using a linear boundary.

For reference, we used the following type of code to plot these graphs,

```
kpc <- kpca(~., data = as.data.frame(d2), kernel="polydot",
                                          kpar=list(degree=2, offset=-75),
                                          features=0)
p2 <- as.data.frame(kpc@pcv) |> ggplot(aes(V1,V2,color=d2_1$Class)) +
                                geom_point() +
                                labs(color='Class', title = 'Polynomial Kernel') +
                                xlab('PC1') + ylab('PC2')
p2
s2 <- cumsum(kpc@eig)/sum(kpc@eig)
data2 <- data.frame(Comp = paste("Comp.", 1:length(s2)),Variance_Explained= s2)
ggplot(data2, aes(x = Comp, y = s2)) +
    geom_bar(stat = "identity", fill = "purple") +
    geom_text(aes(label = sprintf("%.2f", s2)), vjust = -0.5) +
    labs(x = "Components", y = "Ratio of Variance") +
    ggtitle("Variance Explained by Polynomial Kernel")
```

3.2 Denoising Images

The idea is that the variance of the noise is going to be very low, and the actual structure is going to be encaptured by the principal components. So after transforming the dataset into the principal components we are basically getting rid of all the small redundant features.

Here we have a version of the famous MNIST dataset, which is a collection of handwritten images of single digit numbers from 0 to 9. We will apply PCA as well as Kernel PCA, with kernel being 'RBF', to see which of the following is able to perform better at this task.

3.2.1 The Dataset

To begin with, we have 1000 such handwritten samples as X_train_noisy, where each sample is a 16×16 pixels images stored as a vector of 256 variables where each component has value being between 0 and 1. Also to see how well PCA works we have testing dataset, namely X_test_noisy, which contains 100 such samples. So we simply load the dataset and see what the images are, as shown in Figure 3.5. (We have taken the test set for the figure)

Figure 3.4: Noisy Images

3.2.2 PCA and Kernel PCA to reduce the noise

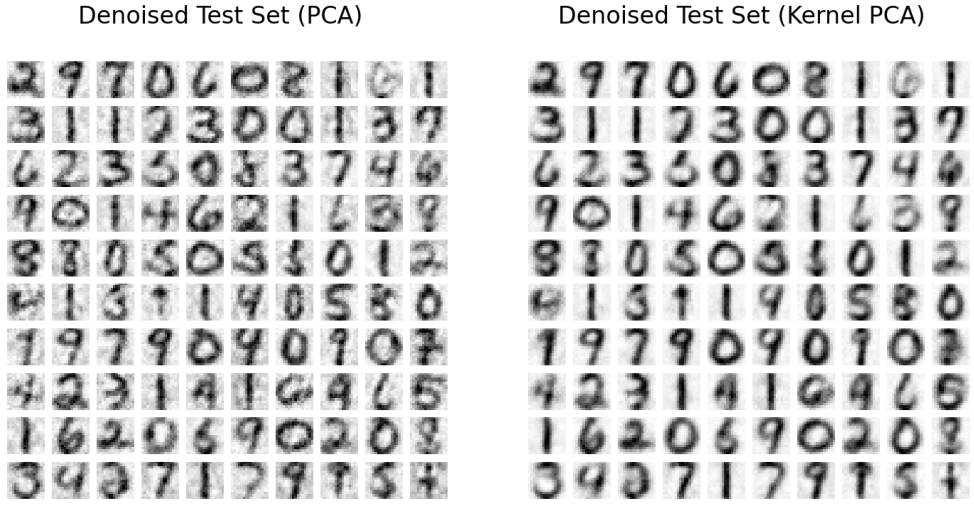
We can see that the images are not very clear and it leads to problems while training models for classification of these images. So we first apply PCA to the training dataset and obtain the Principal Components, , in machine learning language as we say fitting the training data. After obtaining the Principal Components we apply it to the training set to transform the dataset into a form where we are removing noise that is stored in higher order Principal Components. Then since all that noise is lost while making this transformation, we will transform it back to the original components so as to the images which is deprived of all the noise that was stored in the higher order principal components. Thus achieving the task of denoising completely.

```
pca = PCA(n_components=40)
pca.fit(X_train_noisy)
X_hat_pca = pca.inverse_transform(pca.transform(X_test_noisy))
```

This same process will be applied to kernel PCA as well with kernel being the 'RBF' kernel with σ value set to 0.01.

```
kernel_pca = KernelPCA(n_components=40, kernel="rbf", gamma=0.01,
                       fit_inverse_transform=True, alpha=0.1)
kernel_pca.fit(X_train_noisy)
X_hat_kpca = kernel_pca.inverse_transform(kernel_pca.transform(X_test_noisy))
```

After denoising the test images using PCA and Kernel PCA, the resulting images are shown in the Figure 3.4 and Figure 3.6 (a) and (b) respectively.



(a) PCA (b) Kernel PCA

Figure 3.5: Denoised images for the corresponding type of PCA

From the two figures it is evident that the Kernel PCA is performing better at the task of denoising the images.

3.3 Advantages and Disadvantages of PCA

Advantages of PCA:

- **Non-linearity:** Kernel PCA can capture complex, non-linear relationships in the data that traditional PCA cannot handle. This is achieved by implicitly mapping the data into a higher-dimensional space using kernel functions.
- **Dimensionality reduction:** Similar to PCA, kernel PCA can be used for dimensionality reduction while preserving the essential information in the data. It identifies the principal components in the transformed feature space, which can be used to reduce the dimensionality of the data while retaining as much variance as possible.

- **Feature extraction:** Kernel PCA can be used for feature extraction in cases where the original features are not suitable for analysis or when dealing with high-dimensional data. By projecting the data onto a lower-dimensional space, it can highlight the most informative features for subsequent analysis.

Disadvantages of PCA :

- **Computational complexity:** Kernel PCA can be computationally expensive, especially when dealing with large datasets or using complex kernel functions. The computation of the kernel matrix and its eigenvectors can become prohibitive for very high-dimensional data.
- **Parameter tuning:** Kernel PCA involves the selection of hyperparameters such as the choice of kernel function and its parameters (e.g., the width of the Gaussian kernel). The performance of kernel PCA may be sensitive to the choice of these parameters, and finding the optimal values can be challenging.
- **Interpretability:** Like traditional PCA, interpreting the principal components obtained from kernel PCA may not be straightforward, especially when using non-linear kernel functions. This can limit the ability to interpret the results in terms of the original features of the data.
- **Overfitting:** Kernel PCA can be prone to overfitting, especially when using high-dimensional feature spaces or complex kernel functions. Careful regularization and model selection techniques are required to avoid overfitting and ensure generalization to unseen data.

Bibliography

[1] Alan Julian Izenman. *Modern Multivariate Statistical Techniques.* Springer, 2008.

[2] Inge Koch. *Analysis of Multivariate and High-Dimensional Data.* Cambridge University Press, 2014.

[3] Bernhard Scholkopf. *Kernel Pricnipal Component Analysis.* Max-Planck-Institut f. biol. Kybernetik, 1997.