IMPERIAL

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

GROUP RESEARCH PROJECT

Kernel Principal Component Analysis

Author:
Alex Zhu
Jack Ye
Jiacheng Peng
Lang Chen
Qasim Salahuddin

Supervisor(s):
Qianqian Jiang

September 8, 2026

Abstract

In this report, we have investigated the mechanism and applications of the kernel method with respect to Principle Component Analysis, a commonly used dimension reduction method in the field of statistical learning. Through rigorous mathematical reasoning, we reconstruct the proof of standard (linear) PCA as a method of finding low dimensional subspace that maximizes the variance of the projected data, equavalently minimizes the average squared projection distance. Based on that, we introduce kernel PCA (KPCA) with a non-linear feature map, enabling the algorithm to deal with non-linear structures within original data. Regarding the application of KPCA, we experiment the algorithm on the datasets Olivetti Faces and MNIST. We explore on classification tasks with KPCA as a preprocessor, as well as data reconstruction via inverting the projection, and use different kernels to analyze the performance of KPCA and standard PCA. With explorations on fine-tuning kernel hyper-parameters and kernel ridge-regression, we demonstrate the effectiveness of KPCA, showcase the process of choosing optimal values of hyper-parameters, and give reasonable justifications on the reason of KPCA's effectiveness, and on the method of choosing kernels and fine-tuning.

Contents

1 Introduction

In modern statistics and machine learning, the dimension of data is growing exponentially, leading to the "curse of dimensionality". From financial indicators to high-resolution images, the high-dimensional data not only leads to enormous computational costs, but also introduces noise and redundant information, therefore affecting the generalization model. Thus, the technique of dimensionality reduction has become more and more important, for it maps data to a lower-dimensional space, while preserving most of the information of the original data.

Among all the dimensionality reduction methods, Principal Component Analysis (PCA), is the most widely used unsupervised learning algorithm. The history of PCA can date back to Karl Pearson's pioneering work on spatial line fitting in 1901[1], and was formalized and introduced into the field of statistics by Harold Hotelling in 1933[2]. The main idea of PCA is elegant: It finds directions that maximize the variance and minimize the mean squared error of the projected data, with the directions being orthogonal. Over more than a century, PCA has been a core part of multivariate statistical analysis, widely applied in image compression, factor modeling in financial risk management, and noise filtering in exploratory data analysis[3].

However, traditional PCA has a fatal limitation: PCA can only find the linear optimal subspace of the data in the original Euclidean space because it is essentially a linear geometric projection. In real world application, the data are not always linear. For such data, PCA cannot capture its intrinsic geometric structure, leading to dimensionality reduction failure or loss of key patterns.

To overcome this problem, Bernhard Schölkopf et al. proposed Kernel Principal Component Analysis (KPCA) in 1998[4]. They realized that if a nonlinear mapping Φ is used to map the data in the input space to a high-dimensional, or even infinite-dimensional feature space, and standard linear PCA is performed in this high-dimensional space, it captures the nonlinear principal components of the data in the original input space, meaning the new features are nonlinear functions of the original data features. Since all high-dimensional coordinate operations can be equivalently transformed into the computation of kernel functions in the original space, Kernel PCA has the ability to extract nonlinear features at low computational cost. Today, it has become a powerful tool for handling complex pattern recognition, nonlinear fault detection[5], and data clustering[6]. For example, it has been successfully applied to extract nonlinear features in facial recognition tasks [7], and to monitor complex industrial processes by identifying nonlinear sensory variations [8].

However, the practical implementation of KPCA has two well-known constraints: computational complexity and the pre-image problem. Since the algorithm requires the eigendecomposition of an $N \times N$ dense kernel matrix, it inherently demands $O(N^2)$ memory and $O(N^3)$ operations. This cubic time complexity becomes a significant computational bottleneck when dealing with modern large-scale datasets. Furthermore, because the nonlinear feature mapping Φ is never explicitly computed, projecting a low-dimensional embedding back into the original input space will be an ill-posed inverse problem.

Over the years, a lot of literature has been dedicated to resolving these issues. To make KPCA practically viable, the $O(N^3)$ bottleneck is typically managed using low-rank approximations like the Nyström method [9] or Random Fourier Features [10]. As for the pre-image problem, since exact inversion is impossible, practice relies on numerical estimations—most notably iterative optimization [11] or distance-preserving mappings [12].

This report aims to systematically explore the mathematical foundations and algorithmic evolution of PCA and its nonlinear generalizations. The structure is as follows: In Chapter 2 we will demonstrate the theoretical basis of standard PCA from two different perspectives: maximizing the variance and minimizing the mean squared error, and derive the dual representation

using singular value decomposition, and finally move on to the core part of the kernel method by introducing the feature map and kernel function, and kernel PCA. In Chapter 3, we turn to real world application, using standard PCA along with nonlinear PCA with different kernels, and we also discuss the merit and demerit of the algorithms, and consider directions of further improving the efficiency of kernel PCA.

2 Theory

This section introduces the mathematical foundations of Principal Component Analysis, starting from linear Principal Component Analysis (PCA) and extending to its nonlinear generalization, Kernel PCA. Standard PCA is a classical technique that finds low-dimensional linear representations of high-dimensional data. It can be understood through two equivalent geometric perspectives: maximizing the variance of projected data, or minimizing the mean squared reconstruction error. Both perspectives lead to the same solution: the principal components are the eigenvectors corresponding to the largest eigenvalues of the sample covariance matrix. However, standard PCA has a fundamental limitation — it can only capture linear patterns. When data lies on nonlinear curves or surfaces (e.g., circles, spirals, or intertwined moons), standard PCA fails to reveal the underlying structure. To overcome this, we consider mapping the data into a high-dimensional feature space, where linear functions in the feature space give a rich set of nonlinear functions in the original input space. Consequently, nonlinear patterns in the original input space might become linear in the feature space. Performing PCA in this space would solve the problem, but the feature space can be extremely high-dimensional (even infinite), making direct computation impossible. The key insight is the dual representation of PCA, derived via the Singular Value Decomposition (SVD), which expresses the eigenvectors of the sample covariance matrix in terms of its dual matrix, that is the kernel matrix. This kernel trick involves solely the kernel function, which is the inner product on the feature space that defines the entries of the kernel matrix, allowing us to implicitly perform PCA in the high-dimensional feature space without ever computing the explicit map. Combining all these methods together, we have the kernel PCA algorithm that can handle nonlinear structures.

2.1 Standard PCA

The goal of standard (linear) PCA is to find a low-dimensional representation of high-dimensional data that preserves as much of the original variation as possible. For example, consider a dataset of face images: each image has thousands of pixels, but the meaningful variation across faces may be captured by only a few basic features such as lighting, pose, or identity. PCA provides a tool to automatically discover these essential directions and reduce the overall dimensionality.

The widespread adoption of standard PCA stems from its computational efficiency and relative interpretability. Standard PCA is appropriate when the features exhibit strong linear relationship, meaning the data can be reasonably approximated by a flat geometric subspace (like a line or a flat plane).

Standard PCA finds some good new features (variables) which are linear combinations of original correlated features (variables); these good new features are uncorrelated and capture as much variation (information) of original data as possible. After successfully applying standard PCA, we obtain three core mathematical outputs. First, a set of orthogonal principal directions (the eigenvectors) that define the new optimal axes. Second, the corresponding eigenvalues, which quantify the exact proportion of variance captured along each axis, providing a clear metric for choosing the number of dimensions to retain. Finally, we obtain the

principal scores (new features), which serve as the new low-dimensional coordinates for our dataset. Together, these outputs yield a compressed representation of the data.

We introduce the mathematical foundation of standard PCA through two equivalent approaches: The first viewpoint is to seek directions that maximize the variance of the projected data, and the second is to find a subspace that minimizes the mean squared reconstruction error.

Throughout this chapter, we use uppercase letters for matrices and bold lowercase letters for column vectors.

2.1.1 The Centered Data and Covariance

Let $X \in \mathbb{R}^{N\times p}$ be the raw data matrix, where each row corresponds to one observation. Denote the i-th observation by $\mathbf{x}_i \in \mathbb{R}^p$, a column vector of p features (variables). There are N observations and p features. Thus X has N rows and p columns, and can be written as:

$$X = \begin{pmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_N^\top \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{Np} \end{pmatrix} \in \mathbb{R}^{N\times p}.$$

Here, the entry x_{ij} denotes the value of the j-th feature of the i-th observation.

In order to drop the bias introduced by the position of sample mean in the sample space, it is necessary to center the data with respect to its mean. By doing this, the PCA algorithm would consider only the spread of data along each direction, while not affected by the absolute location of data points. Let $\tilde{X}$ denote the centered data, where $\bar{\mathbf{x}} = (\bar{x}_1, \ldots, \bar{x}_p)^\top \in \mathbb{R}^p$ and $\bar{x}_j = \frac{1}{N}\sum_{i=1}^N x_{ij}$ and $\mathbf{1}_N = (1, \ldots, 1)^\top \in \mathbb{R}^N$.

$$\tilde{X} = X - \mathbf{1}_N \bar{\mathbf{x}}^\top = \begin{pmatrix} x_{11} - \bar{x}_1 & \cdots & x_{1p} - \bar{x}_p \\ \vdots & & \vdots \\ x_{N1} - \bar{x}_1 & \cdots & x_{Np} - \bar{x}_p \end{pmatrix}$$

Notation Declaration: To simplify notation, we assume without loss of generality for the remainder of this chapter that the data has been pre-centered. We drop the tilde notation. Hence, X and $\mathbf{x}_i$ strictly refer to the centered data matrix and centered observation vectors, respectively.

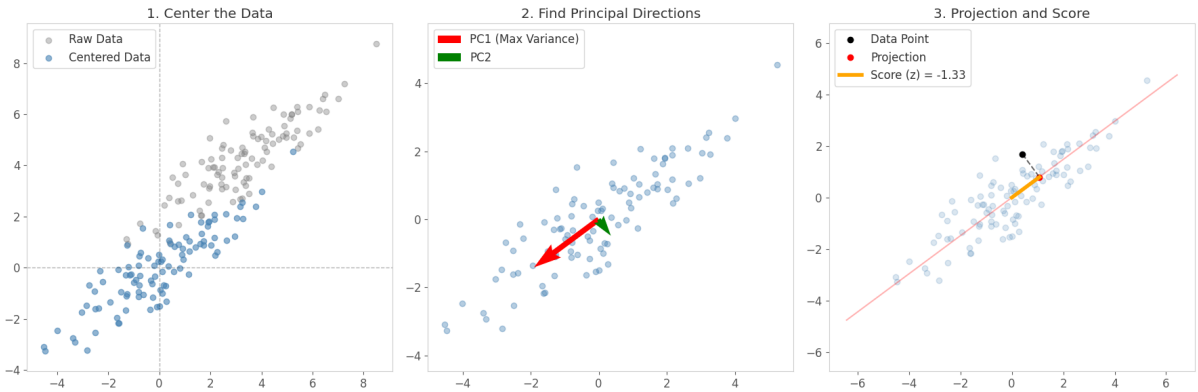

Figure 1 Geometric intuition of standard PCA. (1) The raw data is mean-centered around the origin. (2) The principal directions are identified, with the first principal component (PC1) capturing the maximum variance. (3) The orthogonal projection of a specific data point onto PC1 yields a scalar score z, its coordinate in the new 1D subspace.

From now on, we assume that the data matrix X is centered at the origin. The sample covariance matrix can then be written as:

$$S = \frac{1}{N} X^\top X \in \mathbb{R}^{p \times p} \tag{1}$$

In the follow sections, we will introduce two equivalent perspectives: Maximum Variance and Minimum MSE.

2.1.2 Maximum variance formulation

The goal is to project the data onto an $M < p$ dimensional subspace while maximising the variance of the projected data.

We aim to find the first principal component $\mathbf{u}_1$ such that the variance of data projected along this vector is maximized. Since we only need to consider the direction, the constraint of unit norm ($\|\mathbf{u}_1\| = 1$) is necessary.

To see why this constraint is needed, consider the orthogonal projection of a vector onto a line. Take $\mathbf{x} = (2,1)^\top$ and project it onto the line $\mathrm{span}\{(1,1)^\top\}$ (i.e., $y = x$). Using the unit direction vector $\mathbf{u} = \frac{1}{\sqrt{2}}(1,1)^\top$, the orthogonal projection is $(\mathbf{x}^\top \mathbf{u})\mathbf{u} = (1.5, 1.5)^\top$. If one mistakenly uses the non-unit vector $\mathbf{v} = (2,2)^\top$ and computes $(\mathbf{x}^\top \mathbf{v})\mathbf{v}$, the result is $(12,12)^\top$, which is not the orthogonal projection. Hence, imposing $\|\mathbf{u}_1\| = 1$ guarantees that the projection is orthogonal and that the variance $\mathbf{u}_1^\top S \mathbf{u}_1$ truly reflects the data spread.

The projection of a single data point $\mathbf{x}_i$ (centered, $\sum_{i=1}^N \mathbf{x}_i = \mathbf{0}$) onto $\mathbf{u}_1$ is the scalar $\mathbf{u}_1^\top \mathbf{x}_i$. Now if we apply the formula of sample variance,

$$\begin{aligned}
s^2(\mathbf{u}_1) &:= \frac{1}{N} \sum_{i=1}^N (\mathbf{u}_1^\top \mathbf{x}_i)^2 \\
&= \frac{1}{N} \sum_{i=1}^N (\mathbf{u}_1^\top \mathbf{x}_i)(\mathbf{x}_i^\top \mathbf{u}_1) \\
&= \mathbf{u}_1^\top \Big(\frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^\top\Big) \mathbf{u}_1 \\
&= \mathbf{u}_1^\top \Big(\frac{1}{N} X^\top X\Big) \mathbf{u}_1 \\
&= \mathbf{u}_1^\top S \mathbf{u}_1.
\end{aligned} \tag{2}$$

where S is the sample covariance matrix. Thus, the optimization problem is

$$\max_{\|\mathbf{u}_1\|=1} s^2(\mathbf{u}_1) = \max_{\|\mathbf{u}_1\|=1} \mathbf{u}_1^\top S \mathbf{u}_1$$

To solve this constrained optimization, introduce a Lagrange multiplier λ_1:

$$\mathcal{L}(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^\top S \mathbf{u}_1 - \lambda_1 (\mathbf{u}_1^\top \mathbf{u}_1 - 1).$$

Setting the derivative with respect to $\mathbf{u}_1$ to zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}_1} = 2S\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = 0 \implies S\mathbf{u}_1 = \lambda_1 \mathbf{u}_1.$$

This reveals that $\mathbf{u}_1$ must be an eigenvector of the covariance matrix S. Since the variance explained is $\mathbf{u}_1^\top S\mathbf{u}_1 = \mathbf{u}_1^\top(\lambda_1\mathbf{u}_1) = \lambda_1$, the first principal component is precisely the eigenvector corresponding to the largest eigenvalue.

The following derivation follows [13, pp. 497–501].

After the first principal component is found, the second principal component is obtained by repeating the same optimisation under the additional constraint that it must be orthogonal to the first. That is, we maximise $\mathbf{u}_2^\top S\mathbf{u}_2$ subject to $\|\mathbf{u}_2\| = 1$ and $\mathbf{u}_2^\top\mathbf{u}_1 = 0$.

Since S is symmetric, its eigenvectors corresponding to different eigenvalues are orthogonal. The eigenvector for the second largest eigenvalue automatically satisfies $\mathbf{u}_2^\top\mathbf{u}_1 = 0$ and captures the next highest variance. Hence the second principal component is the eigenvector associated with the second largest eigenvalue.

By induction, the j-th principal component is the eigenvector associated with the j-th largest eigenvalue, and the set of all p principal components forms an orthonormal basis of $\mathbb{R}^p$. This is equivalent to computing the full eigendecomposition of S:

$$SU = U\Lambda, \quad \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p), \quad \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_p \geq 0. \tag{3}$$

where $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_p] \in \mathbb{R}^{p\times p}$ is the orthogonal matrix of principal directions by spectral theorem.

In order to reduce the dimension of space and represent it in a lower-dimensional subspace, we keep only the first $M < p$ principal components, which are just the unit eigenvectors with the first M largest eigenvalues. From the eigen-decomposition of S, let $U_M = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M] \in \mathbb{R}^{p\times M}$ be the matrix of the first M principal directions. The projected data matrix is then:

$$Z = XU_M = \begin{bmatrix} \mathbf{x}_1^\top U_M \\ \vdots \\ \mathbf{x}_N^\top U_M \end{bmatrix} = \begin{bmatrix} z_{11} & \cdots & z_{1M} \\ \vdots & \ddots & \vdots \\ z_{N1} & \cdots & z_{NM} \end{bmatrix} \in \mathbb{R}^{N\times M} \tag{4}$$

where the (i, j)-th entry $z_{ij} = \mathbf{x}_i^\top\mathbf{u}_j$ is the projection score of the i-th data point onto the j-th principal component. For a new data point $\mathbf{x} \in \mathbb{R}^p$, its low-dimensional representation is $\mathbf{z} = U_M^\top\mathbf{x} \in \mathbb{R}^M$.

Remark. If the centred data lie in an r-dimensional subspace of $\mathbb{R}^p$, then S has rank r. Only the first r eigenvalues $\lambda_1, \ldots, \lambda_r$ are non-zero; the rest are zero. Their eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_r$ span that subspace, so projecting onto the first r principal components recovers the data exactly.[14, pp. 146]

Remark. In practice, eigenvalues beyond some M are small but not zero. This means the data have little variance in those directions. Treating that small variance as noise, we keep only the first M components. Removing these directions can improve the representation – PCA acts as a linear denoiser.[14, pp. 146]

Remark. Total variance is

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i^\top\mathbf{x}_i = \frac{1}{N}\sum_{i=1}^{N}\operatorname{Tr}(\mathbf{x}_i^\top\mathbf{x}_i) = \operatorname{Tr}\Big(\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^\top\Big) = \operatorname{Tr}(S) = \sum_{j=1}^{p}\lambda_j. \tag{5}$$

The fraction retained by the first M components is $\sum_{j=1}^{M}\lambda_j / \sum_{j=1}^{p}\lambda_j$. The proportion of variance captured by the projected data increases with the number of principal components. This is equivalent to minimising reconstruction error in Section 14.[14, pp. 147]

2.1.3 Minimum reconstruction error formulation

The goal is to approximate each data point by a representation using only a restricted number $M < p$ of variables, obtained by projecting the data onto a lower-dimensional subspace. Specifically, we aim to find an M-dimensional subspace such that the projected data minimize the average squared projection distance.

Let $\{\mathbf{w}_1, \ldots, \mathbf{w}_p\}$ be a complete orthonormal basis of $\mathbb{R}^p$, with $\mathbf{w}_i^\top \mathbf{w}_j = \delta_{ij}$, where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise. Since this basis is complete, any centred data point $\mathbf{x}_i \in \mathbb{R}^p$ can be represented as

$$\mathbf{x}_i = \sum_{j=1}^{p} (\mathbf{x}_i^\top \mathbf{w}_j)\mathbf{w}_j = \sum_{j=1}^{p} a_{ij}\mathbf{w}_j, \quad i = 1, \ldots, N, \tag{6}$$

where $a_{ij} = \mathbf{x}_i^\top \mathbf{w}_j$ (to see this, take the inner product with $\mathbf{w}_j$: $\mathbf{w}_j^\top \mathbf{x}_i = \sum_k a_{ik}\mathbf{w}_j^\top \mathbf{w}_k = a_{ij}$).

This representation corresponds to a rotation of the coordinate system to a new coordinate system defined by the basis vectors $\{\mathbf{w}_1, \ldots, \mathbf{w}_p\}$, and the original components $\{x_{i1}, \ldots, x_{ip}\}$ are replaced by the equivalent coordinates $\{a_{i1}, \ldots, a_{ip}\}$.

Since an M-dimensional subspace can be represented by the first M basis vectors, we therefore approximate each $\mathbf{x}_i$ by the first M basis vectors:

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{M} z_{ij}\mathbf{w}_j + \sum_{j=M+1}^{p} b_j \mathbf{w}_j, \tag{7}$$

where z_{ij} depend on the data point $\mathbf{x}_i$ and b_j are constants shared by all data points.

Therefore, we aim to minimize the mean squared error:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2$$

For fixed basis vectors, we minimize MSE with respect to z_{ij} and b_j. Using orthonormality, the squared norm expands as:

$$\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|^2 = \sum_{j=1}^{M}(a_{ij} - z_{ij})^2 + \sum_{j=M+1}^{p}(a_{ij} - b_j)^2,$$

where $a_{ij} = \mathbf{x}_i^\top \mathbf{w}_j$. Setting the derivative with respect to z_{ij} to zero gives:

$$z_{ij} = a_{ij} = \mathbf{x}_i^\top \mathbf{w}_j, \quad j = 1, \ldots, M. \tag{8}$$

Setting the derivative with respect to b_j to zero gives $-\frac{2}{N}\sum_{i=1}^{N}(a_{ij} - b_j) = 0$, hence:

$$b_j = \frac{1}{N}\sum_{i=1}^{N} a_{ij} = \left(\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\right)^\top \mathbf{w}_j = \mathbf{0}^\top \mathbf{w}_j = 0, \quad j = M+1, \ldots, p. \tag{9}$$

Therefore, for a fixed basis $\{\mathbf{w}_1, \ldots, \mathbf{w}_p\}$, by equations (6), (7), (8), (9) and orthonormality of the basis vectors, the mean square error is:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=M+1}^{p}(\mathbf{x}_i^\top \mathbf{w}_j)^2 = \sum_{j=M+1}^{p}\mathbf{w}_j^\top S \mathbf{w}_j,$$

where $S = \frac{1}{N} X^\top X$ is the covariance matrix, and the last equality is obtained by the same argument as in (2). Equivalently,

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i\|^2 - \sum_{j=1}^{M} \mathbf{w}_j^\top S \mathbf{w}_j, \tag{10}$$

Next, we need to minimising the mean square error over all M-dimensional subspaces, which is equivalent to:

$$\max_{\mathbf{w}_1,\ldots,\mathbf{w}_M} \sum_{j=1}^{M} \mathbf{w}_j^\top S \mathbf{w}_j, \quad \mathbf{w}_i^\top \mathbf{w}_j = \delta_{ij}, \quad i,j = 1,\ldots,M. \tag{11}$$

We first take $M = 1$ as an example. For a one-dimensional subspace spanned by a unit vector $\mathbf{w}_1$, the reconstructed point is $(\mathbf{w}_1^\top \mathbf{x}_i)\mathbf{w}_1$. The mean squared error becomes:

$$\begin{aligned}
\text{MSE}(\mathbf{w}_1) &:= \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - (\mathbf{w}_1^\top \mathbf{x}_i)\mathbf{w}_1\|^2 \\
&= \frac{1}{N} \sum_{i=1}^{N} (\|\mathbf{x}_i\|^2 - 2(\mathbf{w}_1^\top \mathbf{x}_i)^2 + (\mathbf{w}_1^\top \mathbf{x}_i)^2 \|\mathbf{w}_1\|^2) \\
&= \frac{1}{N} \sum_{i=1}^{N} (\|\mathbf{x}_i\|^2 - (\mathbf{w}_1^\top \mathbf{x}_i)^2) \qquad (\text{since } \|\mathbf{w}_1\| = 1) \\
&= \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i\|^2 - \frac{1}{N} \sum_{i=1}^{N} (\mathbf{w}_1^\top \mathbf{x}_i)^2 \\
&= \underbrace{\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i\|^2}_{\text{constant}} - \mathbf{w}_1^\top S \mathbf{w}_1.
\end{aligned} \tag{12}$$

The first term is the total variance of the data and does not depend on $\mathbf{w}_1$. Therefore, minimising the mean squared error is equivalent to maximising the projected variance:

$$\min_{\|\mathbf{w}_1\|=1} \text{MSE}(\mathbf{w}_1) \iff \max_{\|\mathbf{w}_1\|=1} \mathbf{w}_1^\top S \mathbf{w}_1.$$

Therefore, from the derivation in Section 2.1.2, $\mathbf{w}_1$ is exactly the eigenvector $\mathbf{u}_1$ of S with respect to the largest eigenvalue λ_1.

We now extend the above reasoning to an arbitrary M-dimensional subspace. We will show that the maximum is achieved when $\mathbf{w}_1, \ldots, \mathbf{w}_M$ are exactly the eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_M$ of S with respect to the largest M eigenvalues $\lambda_1, \ldots, \lambda_M$.

Let $W_M = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_M]$ be the $p \times M$ matrix with orthonormal columns. Then

$$\sum_{j=1}^{M} \mathbf{w}_j^\top S \mathbf{w}_j = \sum_{j=1}^{M} \text{Tr}(S \mathbf{w}_j \mathbf{w}_j^\top) = \text{Tr}(S W_M W_M^\top) = \text{Tr}(W_M^\top S W_M).$$

Write the spectral decomposition of the covariance matrix as $S = U \Lambda U^\top$, where $U = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_p]$ is orthogonal ($U^\top U = I_p$) and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_p)$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$.

Define $\widetilde{W} = U^\top W_M$. Because U is orthogonal, $\widetilde{W}$ also has orthonormal columns: $\widetilde{W}^\top \widetilde{W} = I_M$. Then

$$\mathrm{Tr}(W_M^\top S W_M) = \mathrm{Tr}(\widetilde{W}^\top \Lambda \widetilde{W}) = \sum_{i=1}^{p} \lambda_i \left(\sum_{j=1}^{M} (\widetilde{W}_{ij})^2 \right).$$

Set $a_i = \sum_{j=1}^{M} (\widetilde{W}_{ij})^2$ for $i = 1, \ldots, p$. The properties of $\widetilde{W}$ imply $0 \le a_i \le 1$ (each row of $\widetilde{W}$ has squared Euclidean norm at most 1, because extending $\widetilde{W}$ to a full orthogonal matrix gives row norms exactly 1) and $\sum_{i=1}^{p} a_i = M$ (each column of $\widetilde{W}$ has unit norm).

Thus the maximisation problem reduces to

$$\max \sum_{i=1}^{p} \lambda_i a_i \quad \text{subject to} \quad 0 \le a_i \le 1, \ \sum_{i=1}^{p} a_i = M. \tag{13}$$

Since the eigenvalues are ordered decreasingly, the maximum is achieved by taking the largest possible weights on the largest eigenvalues: set $a_1 = a_2 = \cdots = a_M = 1$ and $a_{M+1} = \cdots = a_p = 0$. Hence

$$\sum_{j=1}^{M} \mathbf{w}_j^\top S \mathbf{w}_j \le \lambda_1 + \lambda_2 + \cdots + \lambda_M = \sum_{i=1}^{M} \lambda_i.$$

This upper bound is attained when we choose $\mathbf{w}_j = \mathbf{u}_j$ for $j = 1, \ldots, M$ (i.e., our arbitrary basis aligns perfectly with the first M eigenvectors of S). Indeed, then $\widetilde{W}$ is the matrix whose first M rows form I_M and the rest are zero, so $a_i = 1$ for $i \le M$ and $a_i = 0$ otherwise, giving the sum $\sum_{i=1}^{M} \lambda_i$.

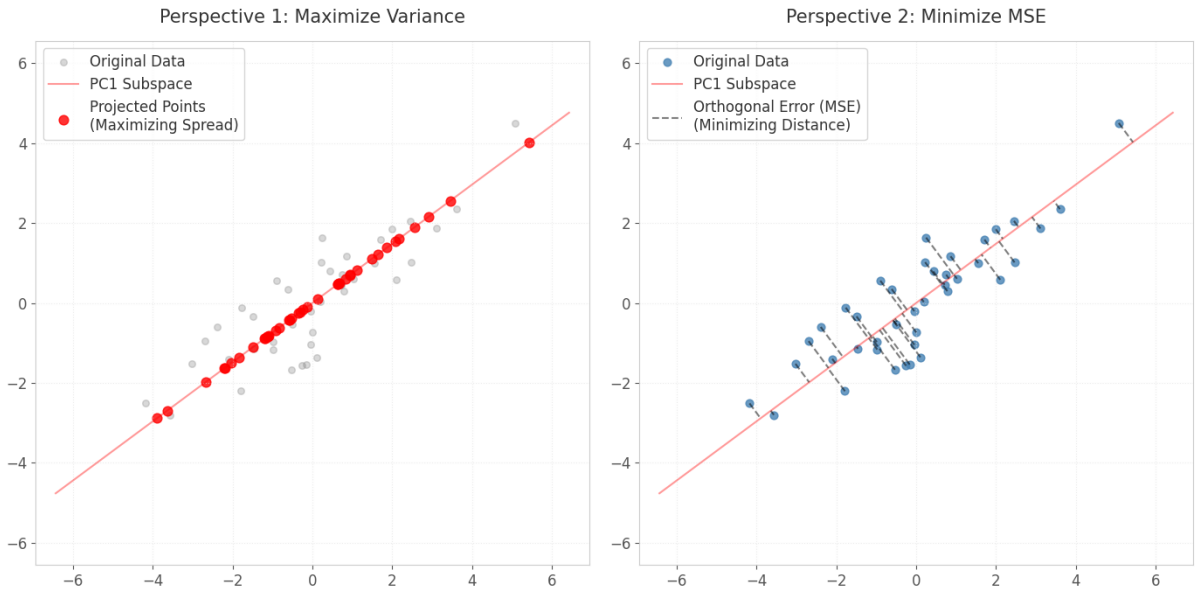

Figure 2 The geometric equivalence of standard PCA perspectives. (Left) PCA seeks a direction that maximizes the variance (spread) of the projected points on the principal axis. (Right) This is mathematically equivalent to minimizing the mean squared reconstruction error, which corresponds to the sum of the squared lengths of the orthogonal projection lines (dashed).

Thus the minimum-error perspective leads to exactly the same eigenvalue problem as the maximum-variance viewpoint (see Section 2.1.2) and [13, pp. 497–501]. The corresponding

value of the error measure is

$$\text{MSE}_{\min} = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 - \sum_{j=1}^{M}\lambda_j = \sum_{j=M+1}^{p}\lambda_j, \tag{14}$$

where we used $\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i\|^2 = \frac{1}{N}\sum_{i=1}^{N}\text{Tr}(\mathbf{x}_i^\top\mathbf{x}_i) = \text{Tr}(\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^\top) = \text{Tr}(S) = \sum_{j=1}^{p}\lambda_j$.

This derivation follows the same line of reasoning as in [13, pp. 497–501] and [14, Proposition 6.12, pp.148-150].

2.2 Dual Representation and SVD

Linear PCA works with the $p \times p$ covariance matrix $S = \frac{1}{N}X^\top X$, which is computationally feasible when p is small. However, when p is huge (e.g., millions), directly diagonalizing S becomes prohibitively expensive. For example, when dealing with images[15], an image of the size of 256×256 lies in the space $\mathbb{R}^p$ with $p = 65536$. Therefore, the covariance matrix would have $65,536 \times 65,536 = 4,294,967,296$ entries. Nevertheless, N can be quite small instead if the dataset does not contain many images.

Fortunately, there is an alternative formulation that works with the $N \times N$ matrix $K = XX^\top$. This is called the dual representation. It is computationally advantageous when the number of observations N is much smaller than the number of features p (i.e., $N \ll p$). (see e.g., [15]) In this setting, storing and diagonalizing the $N \times N$ Kernel matrix is far more efficient than working with the $p \times p$ covariance matrix. However, this advantage disappears if N is also very large — in that case, the dual representation becomes expensive as well. The discussion for large N is discussed in [10].

More importantly, the dual representation shows that PCA only needs inner products between data points. This is a key insight that will be used later to extend PCA beyond linear patterns. The following derivation follows Chapter 6 of *Kernel Methods for Pattern Analysis* [14].

2.2.1 From Covariance Matrix to Kernel Matrix

Recall the sample covariance matrix $S = \frac{1}{N}X^\top X$ and its eigen-decomposition $SU = U\Lambda$, where $U = (\mathbf{u}_1, \ldots, \mathbf{u}_p)$ is orthogonal and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_p)$.

Substitute S into the eigenvalue equation, we have

$$SU = \frac{1}{N}X^\top X\, U = U\Lambda$$

and hence

$$X^\top X\, U = U(N\Lambda) = U\Lambda'.$$

Multiplying both sides on the left by X, we get

$$XX^\top X\, U = X\, U\Lambda'.$$

Therefore, the columns of XU are eigenvectors of $XX^\top$, with corresponding eigenvalues given by the diagonal entries of Λ', provided that these columns are nonzero.

Define the Kernel matrix to be $K = XX^\top \in \mathbb{R}^{N\times N}$, and let

$$V = XU = (\mathbf{v}_1, \ldots, \mathbf{v}_p) \in \mathbb{R}^{N\times p}. \tag{15}$$

Substitute into the above equation:

$$KV = V\Lambda'$$

Notice that V is now a matrix of eigenvectors of K with eigenvalues defined as the diagonal entries of Λ'. Taking one column $\mathbf{v}_j$ of V, we can see that

$$K\mathbf{v}_j = \lambda'_j \mathbf{v}_j$$

where λ'_j is the j-th diagonal entry of Λ'. Again, since we only need the direction of eigenvectors, we shall normalize all $\mathbf{v}_j$ for $j = 1, \ldots, p$,

$$\|\mathbf{v}_j\|^2 = \|X\mathbf{u}_j\|^2 = \mathbf{u}_j^\top X^\top X \mathbf{u}_j = \lambda'_j \, \mathbf{u}_j^\top \mathbf{u}_j = \lambda'_j \implies \|\mathbf{v}_j\| = \sqrt{\lambda'_j} \tag{16}$$

Before proceeding to the dual representation, it is essential to consider the dimensions and the rank of these matrices. The matrix $X^\top X$ is of size $p \times p$, while $K = XX^\top$ is $N \times N$. By fundamental linear algebra, both matrices share the exact same non-zero eigenvalues. The number of strictly positive eigenvalues is bounded by the rank of the data matrix, such that $r = \mathrm{rank}(X) \leq \min(N, p)$.

If $p > N$, the covariance matrix $X^\top X$ is not full rank ($\mathrm{rank}(X^\top X) \leq N$) and has at least $p - N$ zero eigenvalues. Conversely, if $N > p$, K will contain at least $N - p$ zero eigenvalues. Because our normalization step requires dividing by $\sqrt{\lambda'_j}$, the mapping between $\mathbf{u}_j$ and $\mathbf{v}_j$ is strictly valid only for the r principal directions where the eigenvalues are strictly positive ($\lambda'_j > 0$). For zero eigenvalues, the data has no variance along those directions, and they are inherently discarded during dimensionality reduction.

Assuming $\lambda'_j > 0$, the normalised $\mathbf{v}_j$ is:

$$\hat{\mathbf{v}}_j = \frac{1}{\sqrt{\lambda'_j}} X\mathbf{u}_j \tag{17}$$

Note that the hat notation here represents normalized $\mathbf{v}_j$. From now on, we assume that $\mathbf{v}_j$ is normalized with the expression above. Now we can represent every $\mathbf{u}_j$ in terms of normalized $\mathbf{v}_j$:

$$X\mathbf{u}_j = \sqrt{\lambda'_j}\, \mathbf{v}_j$$

Multiply both sides on the left by $X^\top$:

$$X^\top X\mathbf{u}_j = \sqrt{\lambda'_j}\, X^\top \mathbf{v}_j$$

But $X^\top X\mathbf{u}_j = \lambda'_j \mathbf{u}_j$. Substituting this gives

$$\lambda'_j \mathbf{u}_j = \sqrt{\lambda'_j}\, X^\top \mathbf{v}_j$$

Dividing both sides by λ'_j (assuming $\lambda'_j > 0$) yields the dual representation of the principal components:

$$\mathbf{u}_j = \frac{1}{\sqrt{\lambda'_j}} X^\top \mathbf{v}_j \tag{18}$$

2.2.2 Dual Representation of $\mathbf{u}_j$ and its Projection

From the normalized dual relation (18) derived earlier, where where $X \in \mathbb{R}^{N\times p}$ is the data matrix with rows $\mathbf{x}_1^\top, \ldots, \mathbf{x}_N^\top$, $\mathbf{v}_j$ is the normalized eigenvector of the Kernel matrix $K = XX^\top$, and $\lambda'_j = N\lambda_j$ is the corresponding eigenvalue. Expanding this expression in terms of individual data points gives:

$$\mathbf{u}_j = \frac{1}{\sqrt{\lambda'_j}} \begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_N \end{pmatrix} \begin{pmatrix} v_{1j} \\ \vdots \\ v_{Nj} \end{pmatrix} = \frac{1}{\sqrt{\lambda'_j}} \sum_{t=1}^{N} \mathbf{x}_t v_{tj}, \quad \mathbf{x}_t \in \mathbb{R}^{p\times 1}.$$

Define the dual coefficients as:

$$\alpha_{tj} = \frac{1}{\sqrt{\lambda'_j}} v_{tj}. \tag{19}$$

Then the principal component direction can be written as:

$$\mathbf{u}_j = \sum_{t=1}^{N} \alpha_{tj} \mathbf{x}_t. \tag{20}$$

This shows that each principal component is a linear combination of the data points. The coefficients α_{tj} are determined by the eigenvectors of the Kernel matrix $K = XX^\top$.

For a centred point $\mathbf{x}_i$, its projection onto $\mathbf{u}_j$ is:

$$z_{ij} = P_{\mathbf{u}_j}(\mathbf{x}_i) = \mathbf{u}_j^\top \mathbf{x}_i = \sum_{t=1}^{N} \alpha_{tj} (\mathbf{x}_t^\top \mathbf{x}_i). \tag{21}$$

Thus, the projection depends only on the dot products between the data points. No explicit $\mathbf{u}_j$ is required.

So far, we have expressed PCA entirely in terms of dot products $\mathbf{x}_i^\top \mathbf{x}_j$ between data points. This is a key insight: if we can replace the standard dot product with a more general similarity measure, we might be able to extend PCA to capture nonlinear patterns.

In the next section, we will replace the dot product $\mathbf{x}_i^\top \mathbf{x}_j$ with a kernel function:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle,$$

where ϕ maps the data into a higher-dimensional feature space. This substitution yields Kernel PCA, which can discover nonlinear structures that linear PCA cannot.

A fundamental result from Mercer's theorem, guarantees that any continuous, symmetric, positive definite kernel function can be expressed as such an inner product in some (possibly infinite-dimensional) feature space. This ensures that the kernel trick is mathematically sound.

2.3 Kernel PCA

Kernel PCA extends standard PCA to capture nonlinear patterns. It provides a rich set of nonlinear features. The new features are nonlinear function of the original features (variables). Instead of working directly in the original input space $\mathbb{R}^p$, we map the data into a higher-dimensional (possibly infinite-dimensional) feature space, conduct standard PCA in the feature space. In Section 2.2, we have seen that standard PCA can be expressed entirely in terms of dot products $\mathbf{x}_i^\top \mathbf{x}_j$ between data points. Since PCA in feature space also depends only on inner

products $\langle \phi(x_i), \phi(x_j) \rangle$ between data points, these can be computed directly via a kernel function $k(x, y) = \langle \phi(x), \phi(y) \rangle$ without ever evaluating ϕ explicitly. **Mercer's theorem** guarantees that for any continuous, symmetric, positive definite kernel function, there exists feature map ϕ into a feature space such that the kernel function can be expressed as an inner product.

2.3.1 Feature Maps, Kernel Trick, and Kernel Methods

Definition 2.1. A **linear function** between vector spaces V and W over $\mathbb{R}$ is a function $f : V \to W$ satisfying

$$f(\alpha \mathbf{u} + \beta \mathbf{v}) = \alpha f(\mathbf{u}) + \beta f(\mathbf{v}), \qquad \forall \mathbf{u}, \mathbf{v} \in V, \ \forall \alpha, \beta \in \mathbb{R}.$$

When $W = \mathbb{R}$, such an $f : V \to \mathbb{R}$ is also called a **linear function** on V.

Definition 2.2. An **inner product space** over $\mathbb{R}$ is a vector space V equipped with a map $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, called an **inner product**, such that for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and $\alpha, \beta \in \mathbb{R}$:

(i) (Symmetry) $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$;

(ii) (Linearity) $\langle \alpha \mathbf{u} + \beta \mathbf{v}, \ \mathbf{w} \rangle = \alpha \langle \mathbf{u}, \mathbf{w} \rangle + \beta \langle \mathbf{v}, \mathbf{w} \rangle$;

(iii) (Positive-definiteness) $\langle \mathbf{u}, \mathbf{u} \rangle \geq 0$, with equality if and only if $\mathbf{u} = \mathbf{0}$.

Let $\mathcal{H}$ be an inner product space, a **feature map** is a function $\phi : \mathbb{R}^p \to \mathcal{H}$ that transforms each input vector $\mathbf{x}$ into a new vector $\phi(\mathbf{x})$ in $\mathcal{H}$, we call $\mathcal{H}$ the feature space. ($\mathcal{H}$ may be the infinite-dimentional Hilbert space.)

For the fixed $\mathbf{w} \in \mathcal{H}$,

$$f_{\mathbf{w}}(\mathbf{z}) = \langle \mathbf{w}, \mathbf{z} \rangle_{\mathcal{H}}, \quad \forall z \in \mathcal{H}$$

is a linear function on the feature space $\mathcal{H}$.

The core philosophy of kernel methods is that **linear functions** in the feature space $\mathcal{H}$, when composed with the (typically nonlinear) **feature map** ϕ, give a rich set of **nonlinear functions** on the original input space.

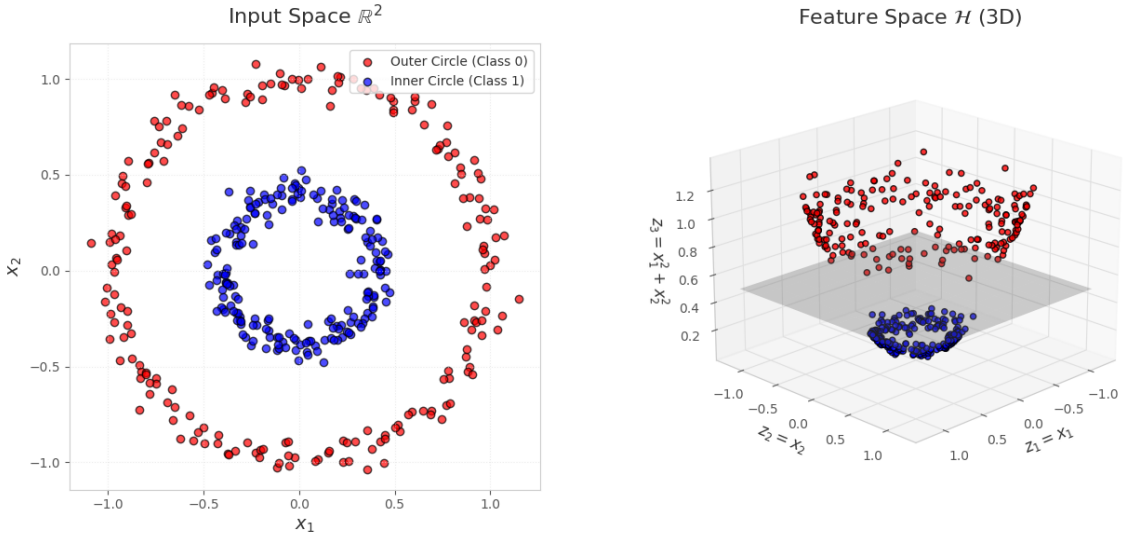

Figure 3 Geometric intuition of a feature map $\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$. (Left) Two concentric circles in $\mathbb{R}^2$ that are not linearly separable. (Right) The same data under ϕ, in $\mathcal{H} \subset \mathbb{R}^3$, where the two classes become linearly separable by a plane.

Example of figure 3: Circular curved structure and lifting map. Take a 2D input $\mathbf{x} = [x_1, x_2]^\top$. Consider a dataset consisting of two concentric circles, which cannot be separated by any straight line in the standard 2D plane. We can apply a feature map that maps the data into a 3D space by adding a squared radial dimension:

$$\phi(\mathbf{x}) = [x_1, x_2, x_1^2 + x_2^2]^\top.$$

In this 3D feature space $\mathcal{H}$, the inner circle (having a smaller radius squared) is mapped to a lower height on the third axis, while the outer circle is mapped to a higher height. A standard 2D plane can now perfectly slice between the two structures.

Concretely, in the right panel of Figure 3 , the gray plane is exactly the level set $f_{\mathbf{w}}(\mathbf{z}) = c$ of the linear function $f_{\mathbf{w}}(\mathbf{z}) = \langle \mathbf{w}, \mathbf{z} \rangle_{\mathcal{H}} = z_3$ with $\mathbf{w} = (0, 0, 1)^\top$. Composing $f_{\mathbf{w}}$ with the feature map ϕ gives

$$(f_{\mathbf{w}} \circ \phi)(\mathbf{x}) = x_1^2 + x_2^2,$$

a nonlinear function on $\mathbb{R}^2$ whose level set $(f_{\mathbf{w}} \circ \phi)(\mathbf{x}) = c$ is exactly the circle separating the two classes in the input space.

Kernel Trick. In general, the feature space $\mathcal{H}$ may be very high-dimensional(or even infinite-dimensional), making it computationally infeasible to evaluate $\phi(\mathbf{x})$ explicitly for every data point. This is computationally infeasible. Fortunately, many algorithms, including the dual formulation of PCA (Section 2.2), do not require $\phi(\mathbf{x})$ itself; they depend on the data only through pairwise inner products $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$.

Definition 2.3. A **kernel** is a function κ that for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^p$, satisfies,

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}},$$

where ϕ is a feature map from $\mathbb{R}^p$ to an inner product space $\mathcal{H}$,

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in \mathcal{H}.$$

The **kernel trick** is the observation that $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ can often be evaluated directly from inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ between data points, without ever constructing $\phi(\mathbf{x}_i)$ or $\phi(\mathbf{x}_j)$. For example, recall the feature map $\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$ from Figure 3. The inner product in $\mathcal{H}$ is

$$\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}} = \mathbf{x}^\top \mathbf{y} + \|\mathbf{x}\|^2 \|\mathbf{y}\|^2,$$

Kernel Methods. A **kernel method** is an algorithm that accesses the data only through pairwise inner products $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, with the inner product is evaluated by a kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$. Different kernels induce different feature spaces $\mathcal{H}$ and capture different patterns, e.g. the linear kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$ (recovers the original algorithm), the polynomial kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j)^2$ (quadratic interactions), and the RBF kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$ (flexible, can separate concentric circles or spirals) — yet the computational cost depends on the sample size N, not on dimension of feature space $\mathcal{H}$.

In summary, a feature map ϕ sends the data into feature space $\mathcal{H}$, where linear functions in feature space, when composed with ϕ, become nonlinear functions of the original input in $\mathbb{R}^p$; the kernel trick evaluates the resulting inner products $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ directly, without forming ϕ; and kernel methods apply this substitution to the algorithm that depends only on such inner products. We next make this concrete for PCA: Section 2.3.2 derives PCA in feature space and shows that it reduces to the eigenvalue problem on the kernel matrix.

2.3.2 PCA in Feature Space: From Covariance to Kernel Matrix

Performing PCA in feature space $\mathcal{H}$ requires solving an eigenvalue problem for the covariance matrix S of the mapped data in feature space $\mathcal{H}$ (often referred to as the covariance operator when $\mathcal{H}$ is infinite-dimensional) . Since ϕ is never computed explicitly, S itself cannot be formed directly. We show that this eigenvalue problem of the sample covariance matrix (operator) can be reduced to an $N \times N$ eigenvalue problem of the kernel matrix K. The derivation follows Schölkopf et al. [4].

Let $\mathbf{x}_i \in \mathbb{R}^p$, $i = 1, \ldots, N$ be the data points, ϕ be a feature map from $\mathbb{R}^p$ to an inner product space $\mathcal{H}$ (may be an infinite-dimensional Hilbert space),

$$\phi : \mathbf{x} \longmapsto \phi(\mathbf{x}) \in \mathcal{H}.$$

Assuming the mapped data in the feature space $\mathcal{H}$ is strictly centered, i.e., $\sum_{i=1}^{N} \phi(\mathbf{x}_i) = \mathbf{0}$, the sample covariance matrix in feature space is given by

$$S = \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \tag{22}$$

If $\mathcal{H}$ is infinite-dimensional, $\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top$ is understood as the linear operator mapping $\mathbf{u} \in \mathcal{H}$ to $\langle \phi(\mathbf{x}_i), \mathbf{u} \rangle_{\mathcal{H}} \, \phi(\mathbf{x}_i)$; we use the outer product notation for convenience [4]. We now have to find the eigenvalues $\lambda > 0$ and the eigenvectors $\mathbf{u} \in \mathcal{H}$ satisfying the eigenvalue equation:

$$S\mathbf{u} = \lambda \mathbf{u}. \tag{23}$$

Substituting the definition of S into (23) yields:

$$\frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \mathbf{u} = \lambda \mathbf{u} \implies \mathbf{u} = \frac{1}{N\lambda} \sum_{i=1}^{N} \big(\phi(\mathbf{x}_i)^\top \mathbf{u}\big)\phi(\mathbf{x}_i).$$

Since $\phi(\mathbf{x}_i)^\top \mathbf{u}$ is simply a scalar, this equation reveals that all solutions $\mathbf{u}$ with $\lambda > 0$ must lie in the span of the mapped data points $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_N)$. Thus, we can represent the eigenvector $\mathbf{u}$ as a linear combination of the feature vectors:

$$\mathbf{u} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i), \tag{24}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_N]^\top$ is a column vector of coefficients.

To find $\boldsymbol{\alpha}$, we substitute (22) and (24) back into the original eigenvalue equation $S\mathbf{u} = \lambda\mathbf{u}$:

$$\frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \left(\sum_{j=1}^{N} \alpha_j \phi(\mathbf{x}_j) \right) = \lambda \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i).$$

To eliminate the explicit feature map ϕ, we multiply both sides from the left by $\phi(\mathbf{x}_k)^\top$ for any $k \in \{1, \ldots, N\}$. The left-hand side becomes

$$\phi(\mathbf{x}_k)^\top \cdot \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \left(\sum_{j=1}^{N} \alpha_j \phi(\mathbf{x}_j) \right)$$
$$= \frac{1}{N} \sum_{i=1}^{N} \underbrace{\phi(\mathbf{x}_k)^\top \phi(\mathbf{x}_i)}_{\text{scalar}} \cdot \sum_{j=1}^{N} \alpha_j \underbrace{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)}_{\text{scalar}},$$

and the right-hand side becomes

$$\lambda \sum_{i=1}^{N} \alpha_i \underbrace{\phi(\mathbf{x}_k)^\top \phi(\mathbf{x}_i)}_{\text{scalar}}.$$

Defining the $N \times N$ kernel matrix $K = (K_{ij})_{1 \le i,j \le N}$, where $K_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, the equation becomes, for each $k = 1, \ldots, N$,

$$\frac{1}{N} \sum_{i=1}^{N} K_{ki} \sum_{j=1}^{N} \alpha_j K_{ij} = \lambda \sum_{i=1}^{N} \alpha_i K_{ki},$$

which in matrix form reads

$$\frac{1}{N} K^2 \boldsymbol{\alpha} = \lambda K \boldsymbol{\alpha} \tag{25}$$

Since K is symmetric and positive semi-definite, removing K from both sides only affects the eigenvectors corresponding to the zero eigenvalue (which do not contribute to the principal components). Thus, the problem simplifies to solving the eigenvalue problem for the kernel matrix:

$$K\boldsymbol{\alpha} = N\lambda\boldsymbol{\alpha}. \tag{26}$$

This result shows that the eigen-decomposition of the covariance matrix S on the mapped data in $\mathcal{H}$ is equivalent to the eigen-decomposition of the $N \times N$ kernel matrix K, where $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. The latter requires no explicit evaluation of ϕ: all computations reduce to pairwise kernel evaluations, and the problem size is determined by the number of samples N, not by the dimension of the feature space $\mathcal{H}$.

The eigenvectors $\boldsymbol{\alpha}$ of K are determined only up to a scalar multiple. We fix the scale by requiring the corresponding vector $\mathbf{u} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i)$ in $\mathcal{H}$ to be a unit vector. Recall that the **norm** on $\mathcal{H}$ is induced by the inner product,

$$\|\mathbf{u}\|_{\mathcal{H}} := \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{H}}},$$

so the normalization condition is $\|\mathbf{u}\|_{\mathcal{H}} = 1$. Since

$$\|\mathbf{u}\|_{\mathcal{H}}^2 = \langle \mathbf{u}, \mathbf{u} \rangle_{\mathcal{H}} = \sum_{i,j=1}^{N} \alpha_i \alpha_j \underbrace{\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}}_{= K_{ij} := \kappa(\mathbf{x}_i, \mathbf{x}_j)} = \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} = N\lambda \|\boldsymbol{\alpha}\|^2$$

the normalization condition $\|\mathbf{u}\|_{\mathcal{H}} = 1$ translates to

$$\|\boldsymbol{\alpha}\| = \frac{1}{\sqrt{N\lambda}}. \tag{27}$$

With the normalized $\boldsymbol{\alpha}$ obtained, the principal direction $\mathbf{u} = \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i)$ is fully determined. For any new data point $\mathbf{x} \in \mathbb{R}^p$, its principal component is defined as the projection onto $\mathbf{u}$:

$$z = P_{\mathbf{u}}(\phi(\mathbf{x})) = \langle \mathbf{u}, \phi(\mathbf{x}) \rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}_i),\ \phi(\mathbf{x}) \right\rangle_{\mathcal{H}} = \sum_{i=1}^{N} \alpha_i\, \kappa(\mathbf{x}_i, \mathbf{x}). \tag{28}$$

This result shows that the projection of any data point $\mathbf{x}$ onto the principal direction $\mathbf{u}$ in $\mathcal{H}$ depends only on kernel evaluations $\kappa(\mathbf{x}_i, \mathbf{x})$ between $\mathbf{x}$ and the training data — $\phi(\mathbf{x})$ is never computed explicitly.

Remark. The principal component $z = P_{\mathbf{u}}(\phi(\mathbf{x}))$ is an instance of the core philosophy of kernel methods. Specifically, $P_{\mathbf{u}} : \mathcal{H} \to \mathbb{R}$ is a linear function on feature space. Its composition with the feature map ϕ gives

$$z = (P_{\mathbf{u}} \circ \phi)(\mathbf{x}) = P_{\mathbf{u}}(\phi(\mathbf{x}))$$

which is a nonlinear function of $\mathbf{x}$ from $\mathbb{R}^p$ to $\mathbb{R}$, since ϕ is nonlinear. The principal component of Kernel PCA is therefore precisely a linear function in feature space composed with a nonlinear feature map — yielding a nonlinear feature of the original input. Finally, Kernel PCA gives the new features which are nonlinear function of the original features (variables).

Since Kernel PCA performs standard PCA in $\mathcal{H}$, all mathematical and statistical properties of linear PCA carry over, with statements now concerning $\mathcal{H}$ rather than $\mathbb{R}^p$. Assuming eigenvectors are sorted in descending order of eigenvalue, the following hold for any $M \in \{1, \ldots, N\}$:

1. the first M principal components carry more variance than any other M orthogonal directions in $\mathcal{H}$;
2. the mean-squared approximation error in representing the data by the first M principal components is minimal;
3. the principal components are uncorrelated;

Additionally, for kernels that depend only on inner products or distances in $\mathbb{R}^p$ — such as the RBF kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2)$ and the polynomial kernel $\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y})^d$, the extracted features do not depend on the choice of orthonormal coordinate system used to represent the input data [4].

Building upon this mathematical foundation, the practical implementation of Kernel PCA can be summarized in the following computational steps.

2.3.3 The Kernel PCA Algorithm

Step 1: Input

Given training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^p$, a chosen kernel function κ and the number of principal components M to retain.

Step 2: Compute the raw kernel matrix

Compute the $N \times N$ kernel matrix K by evaluating the kernel function on all pairs of training points:

$$K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j), \qquad i, j = 1, \ldots, N. \tag{29}$$

Step 3: Center the kernel matrix

Let $\phi(\mathbf{x}_i)$ be the image of the i-th data point in feature space $\mathcal{H}$. Since the derivation in Section 2.3.2 assumes centered data in $\mathcal{H}$, we must account for the feature space mean. Let $\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{N}\sum_{k=1}^{N} \phi(\mathbf{x}_k)$ denote the centered feature vector. The centered kernel matrix $\tilde{K}$ is defined by

$$\tilde{K}_{ij} = \langle \tilde{\phi}(\mathbf{x}_i), \tilde{\phi}(\mathbf{x}_j) \rangle_{\mathcal{H}}.$$

Expanding using the definition of $\tilde{\phi}$:

$$\tilde{K}_{ij} = \left\langle \phi(\mathbf{x}_i) - \frac{1}{N}\sum_{k=1}^{N}\phi(\mathbf{x}_k),\ \phi(\mathbf{x}_j) - \frac{1}{N}\sum_{l=1}^{N}\phi(\mathbf{x}_l) \right\rangle_{\mathcal{H}}$$
$$= K_{ij} - \frac{1}{N}\sum_{k=1}^{N}K_{kj} - \frac{1}{N}\sum_{l=1}^{N}K_{il} + \frac{1}{N^2}\sum_{k,l=1}^{N}K_{kl},$$

where $K_{ij} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle_{\mathcal{H}} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is the uncentered kernel matrix from Step 2. In matrix form:

$$\tilde{K} = K - \mathbf{1}_N K - K\mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N, \tag{30}$$

where $\mathbf{1}_N \in \mathbb{R}^{N\times N}$ is the matrix with all entries equal to $\frac{1}{N}$.

Step 4: Eigendecomposition of the centered kernel matrix

Solve the eigenvalue problem

$$\tilde{K}\mathbf{v}_j = \lambda_j \mathbf{v}_j, \qquad j = 1, \ldots, N, \tag{31}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$ and $\|\mathbf{v}_j\| = 1$. Retain the first M eigenvectors corresponding to the M largest positive eigenvalues.

Step 5: Compute the dual coefficients

For $j = 1, \ldots, M$, set

$$\boldsymbol{\alpha}_j = \frac{1}{\sqrt{\lambda_j}}\mathbf{v}_j. \tag{32}$$

This rescaling ensures $\|\boldsymbol{\alpha}_j\| = \frac{1}{\sqrt{\lambda_j}}$, satisfying the normalization condition derived in Section 2.3.2.

Step 6: Project the data

For any data point $\mathbf{x} \in \mathbb{R}^p$, its image $\phi(\mathbf{x})$ in $\mathcal{H}$ is projected onto the j-th principal direction $\mathbf{u}_j$. The resulting principal component score is (cf. Eq. (28))

$$z_j(\mathbf{x}) = P_{\mathbf{u}_j}(\phi(\mathbf{x})) = \sum_{i=1}^{N}\alpha_{ij}\,\kappa(\mathbf{x}_i, \mathbf{x}), \qquad j = 1, \ldots, M. \tag{33}$$

where α_{ij} is the i-th entry of $\boldsymbol{\alpha}_j$ from Step 5. This formula applies to any data point — training or test — and requires only the kernel function κ and the coefficients $\boldsymbol{\alpha}_j$; $\phi(\mathbf{x})$ is never computed explicitly.

Training data.

For the N training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the kernel evaluations $\kappa(\mathbf{x}_i, \mathbf{x}_l)$ are already encoded in the centered kernel matrix $\tilde{K}$ from Step 3. The score vector $\mathbf{z}_j \in \mathbb{R}^N$ collecting all training scores on the j-th principal component is therefore

$$\mathbf{z}_j = \tilde{K}\boldsymbol{\alpha}_j, \qquad j = 1, \ldots, M,$$

and the full transformed training data matrix $Z_{\text{train}} \in \mathbb{R}^{N \times M}$ is

$$Z_{\text{train}} = \tilde{K} A, \tag{34}$$

where $A = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M] \in \mathbb{R}^{N \times M}$ and $(Z_{\text{train}})_{lj} = z_j(\mathbf{x}_l)$.

Test data. For n new test points $\{\mathbf{x}_1^*, \ldots, \mathbf{x}_n^*\}$, the kernel evaluations between test and training points are not contained in $\tilde{K}$. A new $n \times N$ kernel matrix must be computed:

$$K_{li}^* = \kappa(\mathbf{x}_l^*, \mathbf{x}_i), \qquad l = 1, \ldots, n, \; i = 1, \ldots, N.$$

The transformed test data matrix $Z_{\text{test}} \in \mathbb{R}^{n \times M}$ is then

$$Z_{\text{test}} = K^* A, \tag{35}$$

where $(Z_{\text{test}})_{lj} = z_j(\mathbf{x}_l^*)$. Note that K^* depends on the kernel function κ itself, not just the training kernel matrix $\tilde{K}$ — the kernel function must remain available at test time.

3 Experiments

We now apply this theory in practice to a number of experiments. An advantage of Kernel PCA is that its application on a predictive modelling problem can be completely separated from the Machine Learning algorithm used. This modularity allows us to focus our efforts on investigating the effects of various kernel functions applied on datasets prior to a learning algorithm, and the following experiments aim to analyse this in detail.

All analysis took place in Python using Scikit-learn[16]. A link to the Github repository containing code used to run experiments and further details can be found in Appendix A.

3.1 Olivetti Faces

We start with a problem involving the Olivetti Faces dataset[17]. This is a widely used dataset, and provides a good example showcasing the capabilities of Kernel PCA. We aim to apply PCA in conjunction with a classification algorithm on a predefined training set, to then attempt to successfully classify items in a predefined test set. Afterwards, we attempt to reconstruct images after PCA has been performed on the data set, to visually see what the effects of PCA are.

3.1.1 Dataset overview

The dataset consists of 10 greyscale images of 40 different people (400 in total). Each image is stored as a 64x64 pixel square, equivalent to a 4096-dimension vector (see Figure 4). Yet it is clear that the images lie on a much smaller subspace - indeed we cannot have any random combination of pixels, but only pixels resembling a human face. Moreover, this subspace is complex; head rotations or changes in lighting clearly have a non-linear effect on the pixel structure. This hints that applying Kernel PCA upon the data before any attempt at classification would be fruitful.

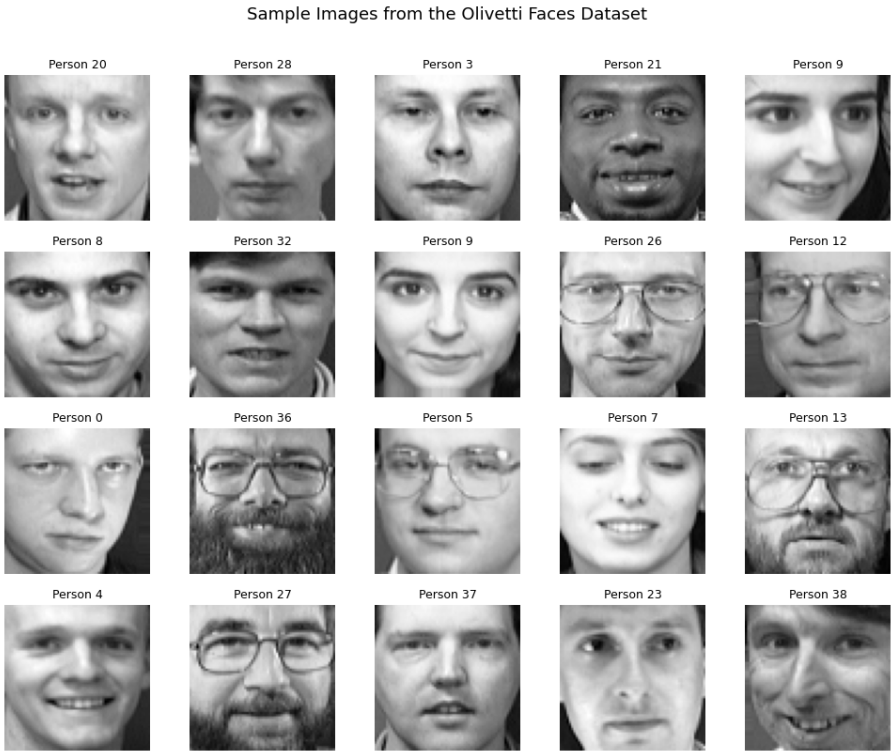

Figure 4 Sample images from the Olivetti Faces dataset

3.1.2 Classification

We perform the following algorithm on the dataset as part of the experiment.

Algorithm 1 Classification of Olivetti faces via kNN, with Kernel PCA

Input: Data points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ with $x_i \in \mathbb{R}^{4096}, y_i \in \{1, 2, ..., 40\}$ and a Kernel function $\kappa(\mathbf{x}, \mathbf{x}')$

Output: Test classification accuracy

1: Split data points 80/20 into training and test sets (stratifying by person). Denote these sets as

$$X_{\text{train}} = \{(x_i^{\text{train}}, y_i^{\text{train}})\}_{i=1}^{n}, \qquad X_{\text{test}} = \{(x_j^{\text{test}}, y_j^{\text{test}})\}_{j=1}^{n_{\text{test}}}.$$

2: Compute the kernel matrix:

$$K_{ij} = \kappa(x_i^{\text{train}}, x_j^{\text{train}})$$

3: Centre the kernel matrix:

$$\tilde{K} = K - \frac{1}{N}\mathbf{1}\mathbf{1}^\top K - \frac{1}{N}K\mathbf{1}\mathbf{1}^\top + \frac{1}{N^2}(\mathbf{1}^\top K\mathbf{1})\mathbf{1}\mathbf{1}^\top,$$

4: Solve the eigenvalue problem involving $\tilde{K}$ to find eigenvectors α_r:

$$\tilde{K}\alpha_r = \lambda_r \alpha_r, \qquad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0.$$

where $\mathbf{1} = (1, \ldots, 1)^\top \in \mathbb{R}^N$ is a column vector of ones (as before).

5: Define

$$A_m = (\alpha_1, ..., \alpha_m)$$

as the projection matrix with $m \leq n$ components

6: Compute

$$Z_{\text{train}} = \tilde{K} A_m$$

where

$$Z_{\text{train}} = (z_1^{\text{train}}, \ldots, z_n^{\text{train}})^\top.$$

is the matrix whose rows contain the (transposed) projected training data, with respect to the chosen kernel function

7: Compute the test kernel matrix:

$$K_{ij}^{\text{test}} = \kappa(x_j^{\text{test}}, x_i^{\text{train}})$$

Then centre it the same way the training kernel matrix was centred, and obtain $\tilde{K}_{\text{test}}$

8: Compute

$$Z_{\text{test}} = \tilde{K}_{\text{test}} A_m$$

where

$$Z_{\text{test}} = (z_1^{\text{test}}, \ldots, z_n^{\text{test}})^\top.$$

is the matrix whose rows contain the (transposed) projected test data, with respect to the chosen kernel function

9: Finally we train a k-Nearest Neighbours (kNN) classifier with the projected training data and original labels. We test it on the projected test data, and we calculate a test accuracy percentage by counting the number of test items correctly classified as a percentage of the size of the whole test set (which is 80)

---

k-Nearest Neighbours (kNN)

We now provide a brief mathematical formulation of the algorithm performed by a kNN classifier. Given a training set

$$\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}, x_i \in \mathbb{R}^m, y_i \in \{1, 2, ..., C\}$$

a k-Nearest Neighbours (kNN) classifier will predict the classification of a new data point x by first identifying the k nearest training points to x, under Euclidean distance (at least in our case, a different metric could also be used):

$$d(x, x_i) = ||x - x_i||_2$$

Let $N_k(x)$ denote the set of indices correspoding to the k training samples with the smallest such distance. The class prediction is then given to be the majority label amongst all of these neighbours:

$$\widehat{y} = \arg \max_{c \in \{1,\ldots,C\}} \sum_{i \in N_k(x)} \mathbf{1}(y_i = c)$$

($\mathbf{1}(\cdot)$ is the indicator function, taking value 1 when its argument is true and 0 otherwise). In our experiment we set $k = 5$ - we only consider the 5 nearest neighbours. This is a commonly used default choice. An odd value of k is often used to avoid ties in the decision stage, and as our dataset is fairly small a conservative value of $k = 5$ is reasonable.

kNN is a good classifier choice in an experiment evaluating different kernels as the decision rule directly reflects the geometry of the projected space, rewarding kernels who separate the data well. It also has no learned parameters, allowing us to focus on adjusting the PCA side of the experiment rather than also having to tune a learning parameter in a classifier.

We use the following kernel functions in the above algorithm:

- **Linear:** $\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$

- **Polynomial:** $\kappa(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + r)^d$
- **RBF (Gaussian):** $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$
- **Sigmoid:** $\kappa(\mathbf{x}, \mathbf{x}') = \tanh\left(\gamma \langle \mathbf{x}, \mathbf{x}' \rangle + r\right)$
- **Cosine:** $\kappa(\mathbf{x}, \mathbf{x}') = \frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\| \|\mathbf{x}'\|}$

Note that d, r and γ are hyperparameters which will affect the accuracy and numerical stability of each kernel. For maximum accuracy these hyperparameters need to be tuned, however for this simple problem we will take heuristic estimates of $d = 3, r = 0$ and $\gamma = 10^{-3}$ for the RBF and Polynomial kernel functions, and $\gamma = 10^{-4}$ for the sigmoid function.

We now present some findings from the classification tests.

Figure 5 shows the test accuracy for each kernel function with 5, 10, 20, 40, 60, 80 and 100 components. It is clear that standard PCA is outperformed or at least matched at every component size, with the sigmoid kernel giving the highest test accuracy overall. 40 components seems to be an optimum number of components - the test accuracy of almost all 5 kernels are maximised at this point. Table 1 shows this more closely.

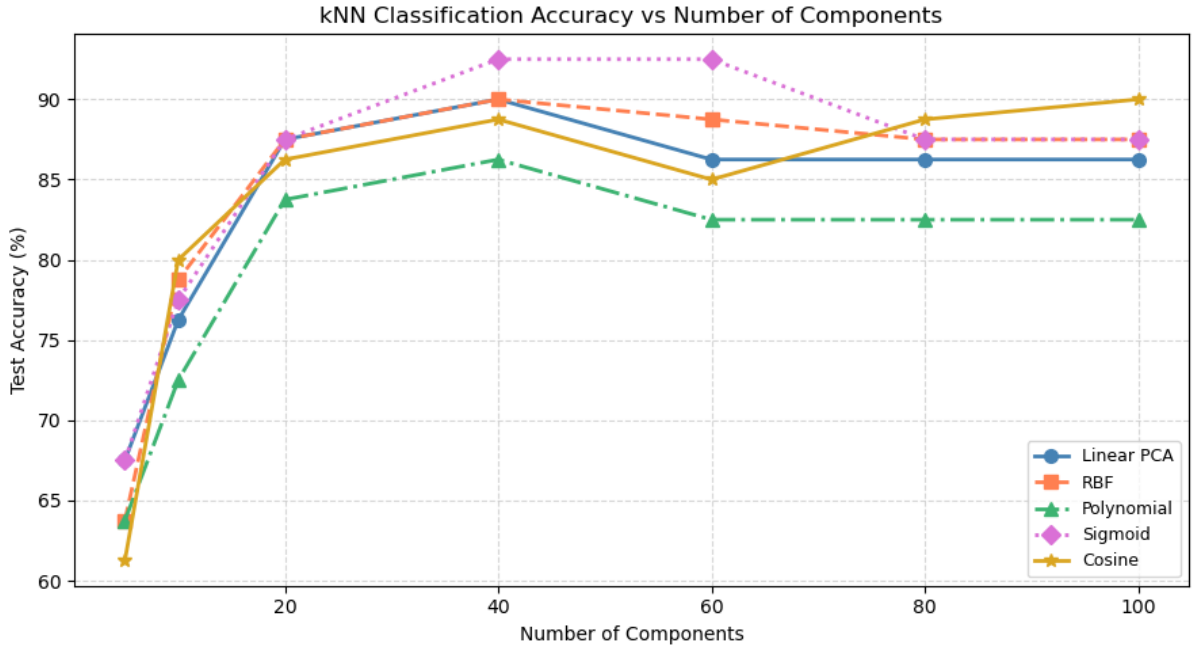

Figure 5

Table 1 kNN Classification Accuracy after PCA (40 components)

Method	Test Accuracy (%)
Linear PCA	90.0
RBF	90.0
Polynomial	86.2
Sigmoid	92.5
Cosine	88.8

Remark. It is of interest that after 40 components, the accuracy of the tests seem to drop in some cases - one might expect this to be strictly increasing. This is explained by the fact that

while early components capture directions of large variance, which correspond to meaningful structures within the data, later components may begin to start capturing noise, therefore reducing the overall accuracy. kNN is particularly susceptible to this, as every dimension included contributes equally in training its decision rule.

A modified experiment was also ran which first ran a grid search to find optimal $d, r$and γ parameters for the kernel functions. Test accuracies calculated with the new optimal parameters improved results slightly - the sigmoid kernel now gave a 93.75% test accuracy, with other kernels either slightly improving or remaining the same. The overall conclusion of standard PCA being outperformed by certain kernel function PCA's remained the same.

3.1.3 Reconstruction

Linear Reconstruction We now take advantage of the fact that our dataset consists of images - we are able to visually see the effects of transformations or reconstructions of the data. In this section we attempt to reconstruct test images via different numbers of principal components, found using Linear PCA and RBF PCA. This quality of the reconstructed images will give an idea on how effective such a dimensionality reduction algorithm is.

First we look at a reconstruction using Linear PCA. The following algorithm reconstructs a test image using m principal components.

Algorithm 2 Principal Component Reconstruction of Test Images (Linear PCA)

Input: The set of training faces $\{x_i^{\text{train}}\}_{i=1}^n$, and a test image x^{test} to reconstruct. Number of components in the reconstruction is also required.

Output: Reconstructed test image $\widehat{x}^{\text{test}}$

1: Compute the sample mean of the training data:

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

2: Compute the covariance matrix:

$$S = \frac{1}{n}\sum_{i=1}^{n} (x_i^{\text{train}} - \bar{x})(x_i^{\text{train}} - \bar{x})^\top$$

3: Find eigenvectors $v_1, v_2, ..., v_m$ corresponding to the first m eigenvalues with $\lambda_1 \geq \lambda_2 \geq ....\lambda_m \geq 0$. Define

$$V_m = (v_1, v_2, ..., v_m)$$

4: Compute

$$\widehat{x}^{\text{test}} = \bar{x} + V_m V_m^\top (x^{\text{test}} - \bar{x})$$

This is equivalent to projecting the original data via the eigenvector matrix, then projecting back via the transposed eigenvector matrix. We now have a reconstructed image, using m components.

Figure 6 shows algorithm 2 applied to 5 different test images, reconstructed with 10 through 60 components (in jumps of 10). We can see that 10 components are already enough to start bringing out features of individual people, and every successive addition of 10 components increases accuracy further. Identifiability seems to peek at about 40 components - at this stage we can see clear resemblances between each original photo and 40 component reconstruction. However after this stage, we start to see noise entering the data - images start becoming more

blurry, and we start losing some of that identifiability (although there are some features which do become more prominent only after 40 components). Indeed, we saw that our test accuracy for standard PCA peaks at 40 components.

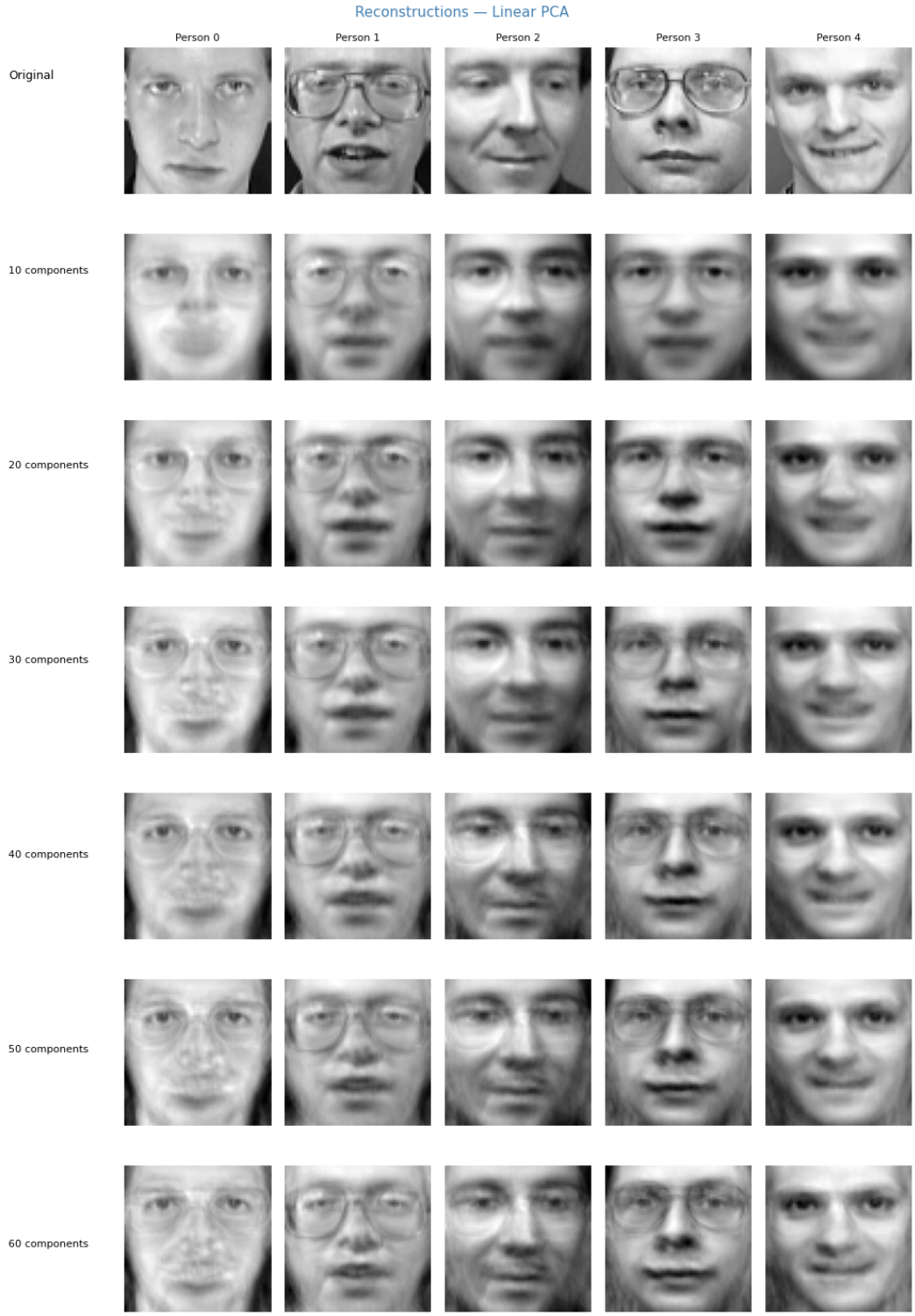

Figure 6 Row 1 shows 5 images taken from the dataset. Each successive row shows the linear reconstruction of an image with 10, 20, 30, 40, 50 and 60 principal components respectively.

These are impressive results - remember that this dataset was initially stored on a 4096-dimension space, yet with only 40 components we have a strikingly effective reconstruction.

But we can do even better - now we consider a reconstruction with Kernel PCA performed with an RBF kernel.

Kernel Reconstruction In the linear reconstruction we did, we were able to construct an image with m components directly via the $V_m V_m^\top$ matrix. However we run into a problem when attempting a kernel reconstruction - the principal directions now live in the feature space rather than the sample space, and since the mapping ϕ is not explicitly known (it may not even be surjective), an exact inverse can no longer be found. To work around this, we approximate the inverse via a method called Kernel ridge regression (see, e.g.,[18]).

We aim to learn an approximate reconstruction map

$$\widehat{x} : \mathbb{R}^M \to \mathbb{R}^p, \qquad z \mapsto \widehat{x}(z)$$

Consider a target $y_i \in \mathbb{R}$. We now define a new mapping

$$\phi_2 : \mathbb{R}^M \to \mathcal{H}_2$$

with a corresponding kernel function

$$\kappa_2(z, z') = \langle \phi_2(z), \phi_2(z') \rangle$$

Note the use of the subscript to emphasise that this is a new kernel. Its purpose is to learn the inverse reconstruction map from projected coordinates back to image space. We now use linear regression on the training pairs (z_i, y_i), seeking a linear predictor

$$\widehat{y}(z) = w^\top \phi_2(z)$$

As $\mathcal{H}_2$ may be high-dimensional, a least-squares fit alone may overfit, so we add a ridge penalty. We obtain the cost function

$$\frac{1}{2} \sum_{i=1}^{n} (y_i - w^\top \phi_2(z_i))^2 + \frac{\mu}{2} |w|^2$$

with $\mu > 0$ as a regularisation parameter. After setting the partial derivative with respect to w to 0, we obtain

$$w = (\mu I + \Phi_2 \Phi_2^\top)^{-1} \Phi_2 y \tag{36}$$

where

$$\Phi_2 = (\phi_2(z_1), ..., \phi_2(z_n))$$

and

$$y = (y_1, ..., y_n)^\top$$

Note we cannot compute w explicitly like this, as $\Phi_2 \Phi_2^\top$ is an operator on a very large (possibly infinite) space. However, we can utilise the same kernel trick we have been using throughout this report - we rewrite the solution so that it only involves inner products between training points. By dual representation of w in (36), it follows that

$$w = \Phi_2 (\mu I_n + \Phi_2^\top \Phi_2)^{-1} y \tag{37}$$

Looking at the above equation, we see that w must lie in the column-span of Φ_2, hence there exist coefficients α such that

$$w = \Phi_2 \alpha$$

Substituting into our prediction rule we obtain

$$\widehat{y}(z) = \alpha^\top \Phi_2^\top \phi_2(z) = \alpha^\top \kappa_2(z)$$

where

$$\kappa_2(z) = (\kappa_2(z_1, z), ...\kappa_2(z_n, z))^\top$$

To determine α, we substitute $w = \Phi_2\alpha$ into our original equation for w, and after manipulation we obtain

$$\alpha = (K_2 + \mu I)^{-1} y$$

where $K_2 = \Phi_2^\top \Phi_2$. this is an $n \times n$ matrix, which we can compute. Up till now we have been considering scalar-valued y. To translate this to our vector-based image setting, we define

$$B = (K_2 + \mu I)^{-1} X_{\text{train}}$$

where X_{train} is as before. Finally, we obtain

$$\widehat{x}(z) = B^\top \kappa_2(z)$$

where $\kappa_2(z)$ is our kernel vector as seen [18].

We now use this theory to reconstruct an image with an RBF kernel as both the forward and inverse kernel. The following algorithm describes this process.

Algorithm 3 Principal Component Reconstruction of Test Images (Kernel PCA)

Input: The set of training faces $\{x_i^{\text{train}}\}_{i=1}^n$, and a test image x^{test} to reconstruct. Number of components in the reconstruction is also required. We use an RBF kernel with $\gamma_1 = 10^{-3}$. We set $\alpha = 3 \times 10^{-5}$ and $\gamma_2 = 1$ in the inverse RBF kernel

Output: Reconstructed test image $\widehat{x}^{\text{test}}$

1: Using an RBF kernel, perform Kernel PCA using m principal components on our training data to obtain projected coordinates (in the feature space) denoting these as

$$Z_{\text{test}} = (z_1^{\text{test}}, \ldots, z_n^{\text{test}})^\top.$$

This is a matrix whose rows contain the (transposed) projected test data, with respect to an RBF kernel function

2: We then construct an inverse-kernel matrix to perform ridge-regression:

$$(K_2)_{ij} = \kappa_2(z_i, z_j)$$

3: Compute

$$B = (K_2 + \mu I_n)^{-1} X^{\text{train}}$$

where $X^{\text{train}} = (x_1^{\text{train}}, x_2^{\text{train}}, ..., x_n^{\text{train}})^\top$

4: Now project the test image x^{test} (using, again, an RBF kernel considering m principal components) to obtain z^{test}

5: Compute the kernel vector

$$k_2(z) = (\kappa_2(z_1, z), ..., \kappa_2(z_n, z))^\top$$

6: Finally reconstruct the image as

$$\widehat{x}^{\text{test}} = B^\top k_2(z)$$

Figure 7 shows algorithm 3 applied to 5 test images, with 10 through 60 components (in jumps of 10). At first glance, these look very similar to our linear reconstructions - remember that we only had slight differences in test accuracies between the linear and RBF kernel classification results due to our small sample size, so this is to be expected. However upon looking closer, there are a few interesting observations.

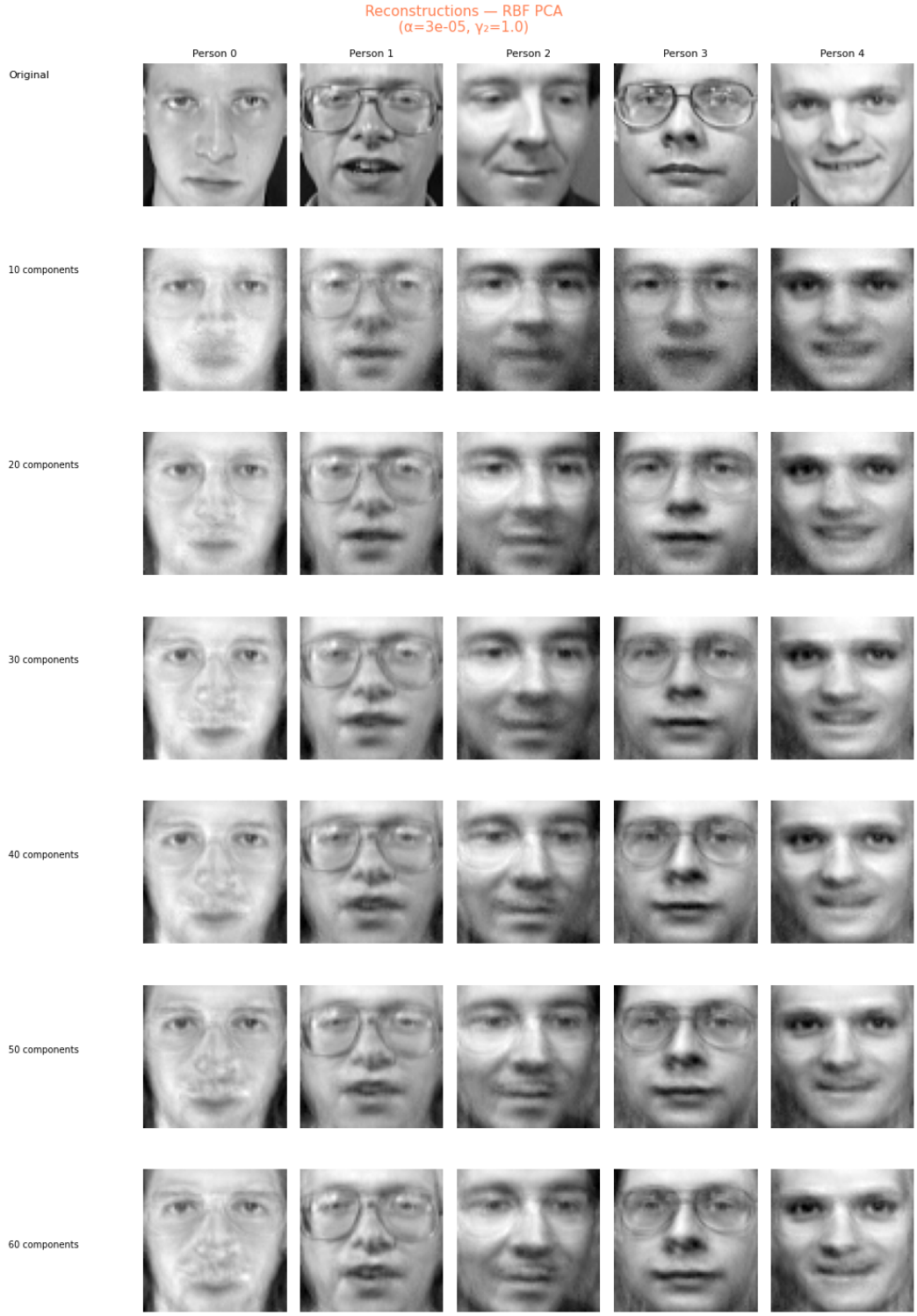

Figure 7 Row 1 shows 5 images taken from the dataset. Each successive row shows the RBF reconstruction of an image with 10, 20, 30, 40, 50 and 60 principal components respectively.

RBF reconstructions of Person 0 seem to include less background noise giving the illusion that he is wearing glasses, compared to the linear reconstruction. Additionally, unlike the linear reconstructions, there is less noise after 40 components in the RBF reconstruction. This is consistent with our previous classification results, where RBF starts to slightly outperform linear PCA after 40 components (see Figure 5). Overall, the results here seem to be better in likeliness to the original images than the linear reconstructions in 6.

3.2 Image Classification on MNIST Dataset

In order to analyze the performance of Kernel PCA in detail, and delve deeper into the choice of kernels and fine-tuning, we choose the classic MNIST dataset and perform image classification, which is larger in size comparing to Olivetti Faces, and is more explanatory since there are less number of classes with more distinction between classes.

3.2.1 Dataset overview

The MNIST dataset consists of 70,000 grayscale images of 10 hand-written digits (0-9), each being a 28×28 image, where each pixel has a 0-255 grayscale value. In this experiment, we chose 5,000 images as the training set, and 2,000 images to test the classification algorithm. Mathematically, every data point of the training set lies in the 784-dimensional space:

$$X = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_{5000}^\top \end{pmatrix}, \quad \mathbf{x}_i \in \mathbb{F}_{256}^{784} \quad i \in 1, 2, \ldots, 5000$$

Approximately, we can use $\mathbb{R}^p$ with $p = 784$. Looking at the image below, we can see that the distinction between numbers 3 and 8 or between 1 and 4 require the algorithm to capture non-linear geometric features to draw a difference correctly, suggesting that Kernel PCA can be applied on this classification task.

Figure 8 Examples of hand-written numbers in MNIST.[19]

3.2.2 Methodology

To perform the classification task, we chose linear Support Vector Classifier (SVC) as the classification algorithm that does not itself have the capability of handling non-linear features. Geometrically, SVC attempts to find a hyperplane that slices data of different classes by maximizing the distance between the hyperplane (also known as the margin) and the nearest data

points. Define the hyperplane as

$$\mathbf{w}^\top \mathbf{x} + \mathbf{b} = 0 \quad \mathbf{w}, \mathbf{x}, \mathbf{b} \in \mathbb{R}^p \tag{38}$$

with learnable parameters $\mathbf{w}$ as the normal vector and $\mathbf{b}$ as the bias. SVC uses Gradient Descent[20] to solve the optimization problem of the loss function below:

$$\min \mathcal{L}(\mathbf{w}, \mathbf{b}) = \min_{\mathbf{w},\mathbf{b}} \left[\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \max\{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + \mathbf{b})\} \right] \tag{39}$$

where N is again the number of data points, and $y_i \in \{-1, +1\}$ is the binary class of the label of the i-th data point. The first term of the loss function aims to maximize the margin. The margin boundary is defined via the equations

$$\mathbf{w}^\top \mathbf{x} + \mathbf{b} = \pm 1 \tag{40}$$

Taking two points x_+ and x_- on opposite boundaries,

$$\begin{aligned} \text{margin} &= (x_+ - x_-)^\top \frac{\mathbf{w}}{\|\mathbf{w}\|} \\ &= (1 - b - (-1 - b)) \frac{1}{\|\mathbf{w}\|} \\ &= \frac{2}{\|\mathbf{w}\|} \end{aligned}$$

Since minimizing $\|\mathbf{w}\|^2$ is equivalent to minimizing $\|\mathbf{w}\|$, the first term corresponds to the goal of SVC. The second term is known as a hinge loss, which penalizes incorrect classification and having data points inside the margin. In addition, C is a hyper-parameter that controls the weight between maximizing margin and preventing false results.

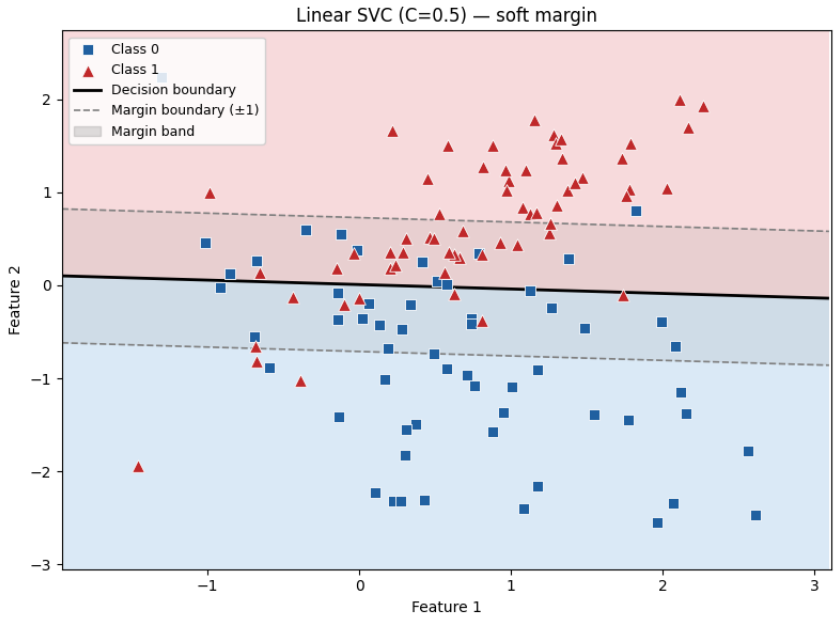

Figure 9 An illustration of how SVC works. The blue squares and red triangles correspond to data points labelled as different classes. The data points lying inside and on the margin are called the "support vectors", as they geometrically support the hyperplane and fix it at this position. With a soft margin ($C < 1$), SVC allows data points inside its margin, enhancing robustness while potentially reduces accuracy.

For classification tasks with $p > 2$ classes, it is a common practice to train p SVC classifiers each distinguishing one class from all others.

To test the performance of Kernel PCA in this experiment, we apply it first on the training set to find the principle components, and project data $X_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times p}$ onto the principle components to obtain $Z_{\text{train}} \in \mathbb{R}^{N_{\text{train}} \times M}$. We then train SVC on Z_{train}, and test the classification with all of the test data $X_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times p}$ also projected onto the principle components as $Z_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times M}$. Moreover, to optimize the effectiveness of Kernel PCA, we explored a series of settings to fine-tune kernels with hyper-parameters.

In this experiment, the following kernels are used:

- **Polynomial Kernel:** $\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^\top \mathbf{x}' + c)^d, \quad \gamma, c \geq 0,\ d \in \mathbb{N}$
- **Gaussian (RBF) Kernel:** $\kappa(\mathbf{x}, \mathbf{x}') = \exp\big(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\big), \quad \gamma > 0$
- **Cosine Similarity Kernel:** $\kappa(\mathbf{x}, \mathbf{x}') = \dfrac{\mathbf{x}^\top \mathbf{x}'}{\|\mathbf{x}\|\,\|\mathbf{x}'\|}$

Notice that Polynomial Kernel and RBF Kernel both have tunable parameters. For Polynomial Kernel, γ controls the scale of the inner product before taking the d-th power, which is important for regulating the magnitude of kernel function when both N and d are large, preventing numerical overflow; c is the bias that introduces all terms with degree lower than d when it is non-zero, which is able to decide whether to include more linear features or take only d-degree non-linear features; finally, d is the degree of the kernel, which is simply the degree of non-linearity the kernel can capture. For RBF Kernel, there is only one parameter γ, which controls the bandwidth or the spread of the bell-shaped curve generated by the kernel. When γ is small, the spread would be large, so that neighbouring data points would result in almost the same kernel function value; when γ is large, data points are considered more isolated, making the kernel matrix more diagonal. The goal of the experiment is to search among a series of values of these parameters under different conditions, so that we obtain a best configuration and we may speculate the reason why it is optimal.

Here is a summary table for the value of tuned parameters and settings used in the entire grid search of the experiment.

Remark. Note that a degree-1 polynomial kernel is equivalent to linear kernel with a linear transformation. In addition, some large γ's of Polynomial Kernel for specific degrees would cause numerical instability and obstructs computation, therefore not all γ's have a test result for each degree.

target	parameter	values
the entire KPCA	number of components (M)	$16, 32, 64, 128, 256, 512, 1024$
RBF Kernel	bandwidth (γ)	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$
Polynomial Kernel	degree (d)	$1(\text{linear}), 2, 3, 4, 5$
Polynomial Kernel	weight (γ)	$1/784, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$
Polynomial Kernel	bias (c)	$0, 1$

Table 2 Experiment setting summary table ($1/784$ is the default value of γ defined as the reciprocal of number of features, or $1/p$)

It is of importance to clarify that linear PCA can run on different number of components M, but cannot exceed the original number of features $p = 784$, while Kernel PCA can extract more

components than p due to the large dimensionality of feature space. The experiment algorithm is described in detail in the following pseudocode (Algorithm 4).

Algorithm 4 Experiment of Classification on MNIST

Input: Size of training set N_{train}, size of testing set N_{test}, numbers of principle components M to try, kernel κ, hyper-parameters of kernel (γ, c, d, etc.)

for κ be none (without KPCA), Linear (standard PCA), RBF, Polynomial and Cosine **do**
  for variations in parameters $M, \gamma, c, d, \ldots$ **do**
    Obtain X_{train} and X_{test} according to the sizes by sampling randomly from MNIST dataset.
    if κ is defined **then**
      Apply Kernel PCA. Let $X_{\text{train}} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_{N_{\text{train}}}^\top \end{bmatrix} \in \mathbb{R}^{N_{\text{train}} \times p}$, $K = X_{\text{train}} X_{\text{train}}^\top$. Project the train data to obtain $Z_{\text{train}} = KA \in \mathbb{R}^{N_{\text{train}} \times M}$ as given in (**??**).
      Train SVC on Z_{train}.
      for $i = 1, \ldots, N_{\text{test}},\ j = 1, \ldots, M$ **do**
        Let $X_{\text{test}} = \begin{bmatrix} \mathbf{x}_1'^\top \\ \vdots \\ \mathbf{x}_{N_{\text{test}}}'^\top \end{bmatrix} \in \mathbb{R}^{N_{\text{test}} \times p}$, project the test data to obtain $Z_{\text{test}} \in \mathbb{R}^{N_{\text{test}} \times M}$, where $z'_{ij} = \sum_{t=1}^{N_{\text{train}}} \alpha_{tj} \kappa(\mathbf{x}_t, \mathbf{x}'_i)$ as given in (**??**).
      end for
      for $i = 1, \ldots, N_{\text{test}}$ **do**
        Let $Z_{\text{test}} = \begin{bmatrix} \mathbf{z}_1'^\top \\ \vdots \\ \mathbf{z}_{N_{\text{test}}}'^\top \end{bmatrix}$, classify each $\mathbf{z}'_i$ via SVC.
      end for
    else
      Train SVC on X_{train}.
      for $j = 1, \ldots, N_{\text{test}}$ **do**
        Classify $\mathbf{x}'_j$ via SVC
      end for
      Calculate summary statistics for training and testing
  end for
end for

3.2.3 Results and Evaluation

After running through the experiments, it has been shown that comparatively optimal configurations for the RBF Kernel and Polynomial Kernel is as following:

kernel (κ)	**number of components** (M)	**parameters**	**train accuracy**	**test accuracy**
Polynomial (1st)	1024	$d = 2, \gamma = 0.01, c = 1$	99.92%	92.35%
RBF	1024	$\gamma = 0.001$	96.74%	89.40%

Table 3 The best configuration of RBF Kernel out of 35 tested combinations, and that of Polynomial Kernel out of 252 combinations of parameter values.

In the grid search of different combinations of parameters, it is shown that the performance of Kernel PCA is essentially influenced by the choice of parameters, as well as the choice of kernels.

For the RBF kernel, we can see from figure 10 that the optimal choice of inverse bandwidth γ is around the scale of 10^{-3}. Geometrically, the bandwidth $\sigma = 1/\sqrt{2\gamma}$ controls the influence of data points that are of different distance from the data point of interest. If σ is small, points are considered far away from each other, causing over-fit to occur; on the other hand, if σ is large, then points would all look the same, resulting in under-fit. Beside that, in figure 10 it is apparent that RBF does better with more principle components, but the overall performance is not significantly improved when M exceeds p.

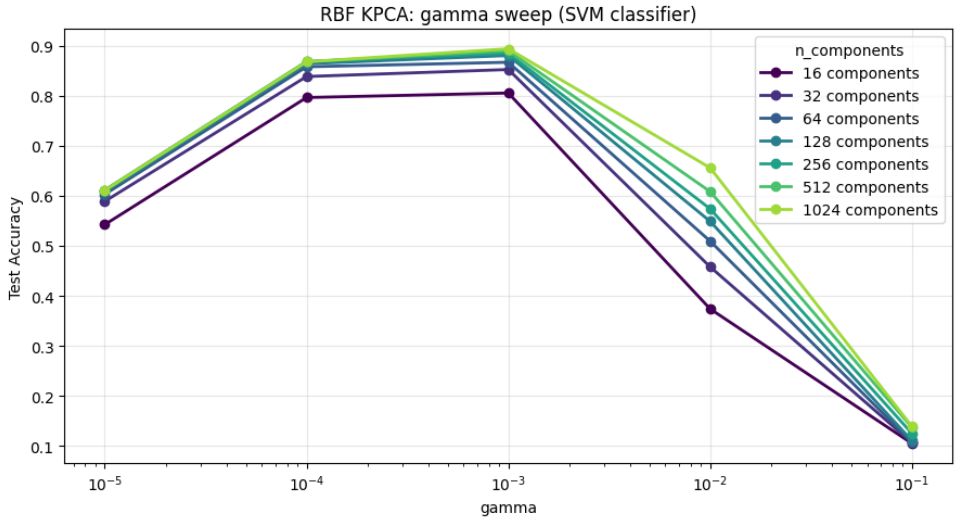

Figure 10 Grid search result of RBF on γ and M

For the Polynomial kernel, it has been demonstrated in 252 trials that a non-zero bias $c = 1$ works better than $c = 0$ in general, as the best train accuracy under $c = 0$ is 91.00%, while the best result under $c = 1$ is 92.35%. For each combination, the classification result is also better in almost all cases when $c = 1$. One possible reason for this result is that $p = 784$ is already a large dimensionality, which could make the class boundary between different digits become relatively linear, even though the data itself is non-linearly shaped.

Consequently, it is crucial to guarantee that linear features are captured by the kernel. When $c = 0$, only the non-linear feature of degree d would be captured, while if $c = 1$, all lower degrees would be considered instead. This can also suggest a reason why RBF kernel sometimes performs worse than other kernels (including linear PCA under some cases).

Figure 11 shows the result of grid search with $c = 1$ as a heat map. We observe that for each degree, increase in M always results in a better test accuracy. Moreover, the kernels on higher degrees rely even more on large number of principle components. This indicates that the Polynomial kernel needs large dimensionality to gain a variety of structures in the original space, so as to distinguish data points labelled with different classes. This is possibly due to the limitation of Polynomial kernel that it can only capture non-linear features of certain degrees, while kernels like RBF can capture entire families of non-linear features. However, this can also be an advantage considering its selectivity.

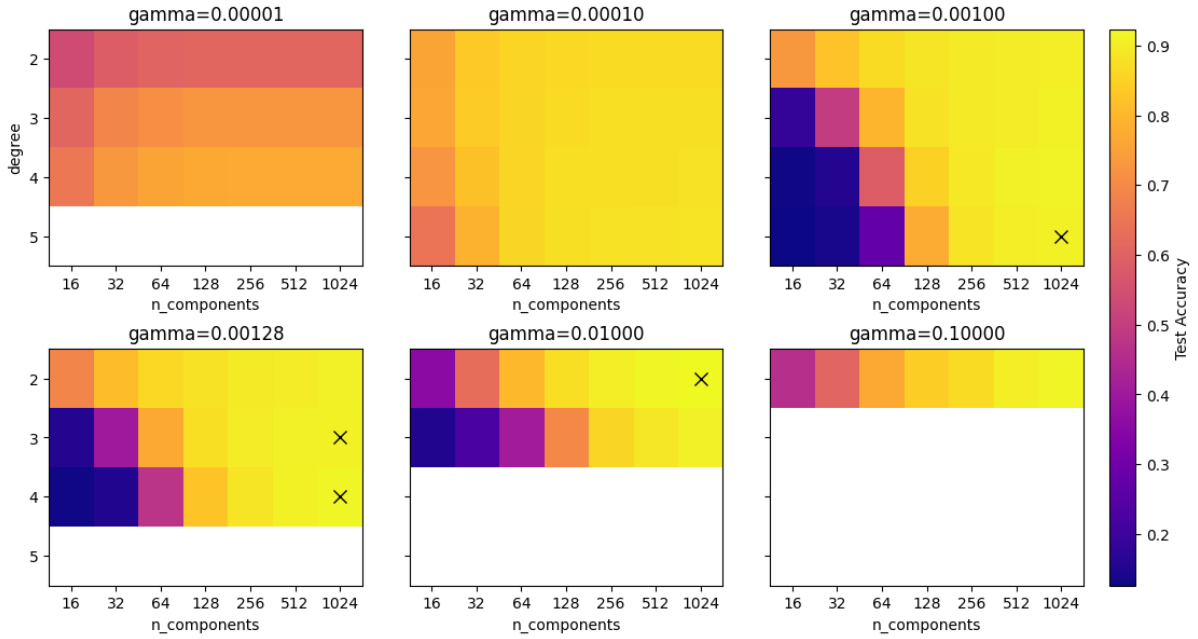

Figure 11 A heat map showing the test results of Polynomial Kernel under variations in M and d for each tested γ **under the assumption that** $c = 1$. The white spaces in the plots means no result is gathered due to numerical instability, suggesting that such combinations are invalid. The black markers indicate the optimal combination for each degree. There are significant trends showing that increasing M has a positive impact on test accuracy.

This trend is even more noticeable in the following example in table 4 of a series of trials under $c = 0$, since there are no low-degree feature being captured for each case, worsening the robustness of the algorithm under the change of M.

	Number of Components ($\gamma = 0.01, c = 0$)						
Degree	16	32	64	128	256	512	1024
2	19.00%	29.85%	55.25%	79.50%	88.00%	90.15%	91.00%
3	14.70%	18.55%	27.65%	54.60%	74.60%	86.25%	88.15%

Table 4 An example of Polynomial Kernel relying on the number of components when $\gamma = 0.01$, $c = 0.0$. There is a clear trend that the test accuracy is highly impacted by M, and higher M's are favoured.

Finally, the choice of γ is also vital for tuning the Polynomial kernel. As shown in figure 12, both RBF and Cosine kernel have kernel matrices that are naturally normalized, since the value of their kernel functions would never exceed 1. Meanwhile, although linear PCA has the kernel matrix with the largest magnitude, it scales linearly, and it is also not necessary to use kernel matrix in computation for the linear case. Nevertheless, different values of γ can have a significant impact on the performance of the Polynomial kernel, since it is the only way to regularize the kernel matrix. We can see that $\gamma = 0.01$ keeps the entries within the magnitude of around 10, which is numerically stable.

Regarding the kernel matrix plot itself, there are interesting traits that can be found on the sorted index with respect to digit classes. For instance, the entries of digit 1 is brighter than others for the RBF kernel, which matches the projection plot in figure 13, which we will discuss later. In addition, the diagonal entries (the entries $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ with $\mathbf{x}_i$ and $\mathbf{x}_j$ being in the same

class) have larger values in all four matrices, indicating that the kernel is functional, as it does measure data points of the same class geometrically as "closer" points.

Number of components (M)	Linear	RBF	Polynomial	Cosine	None
16	85.20%	80.55%	35.90%	83.35%	—
32	88.20%	85.25%	63.00%	88.40%	—
64	87.70%	86.70%	80.05%	88.90%	—
128	88.45%	88.05%	87.40%	89.70%	—
256	88.15%	88.50%	90.35%	89.45%	—
512	88.45%	89.00%	91.90%	89.60%	—
1024	—	89.40%	92.35%	89.60%	—
N/A	—	—	—	—	88.65%

Table 5 Test accuracy by kernel and number of components with tuned best parameters. More precisely, RBF has $\gamma = 0.001$, Polynomial has $d = 2, \gamma = 0.01, c = 1$. Note that Linear PCA cannot go up to $M = 1024$, and that "None" refers to no PCA used before classification.

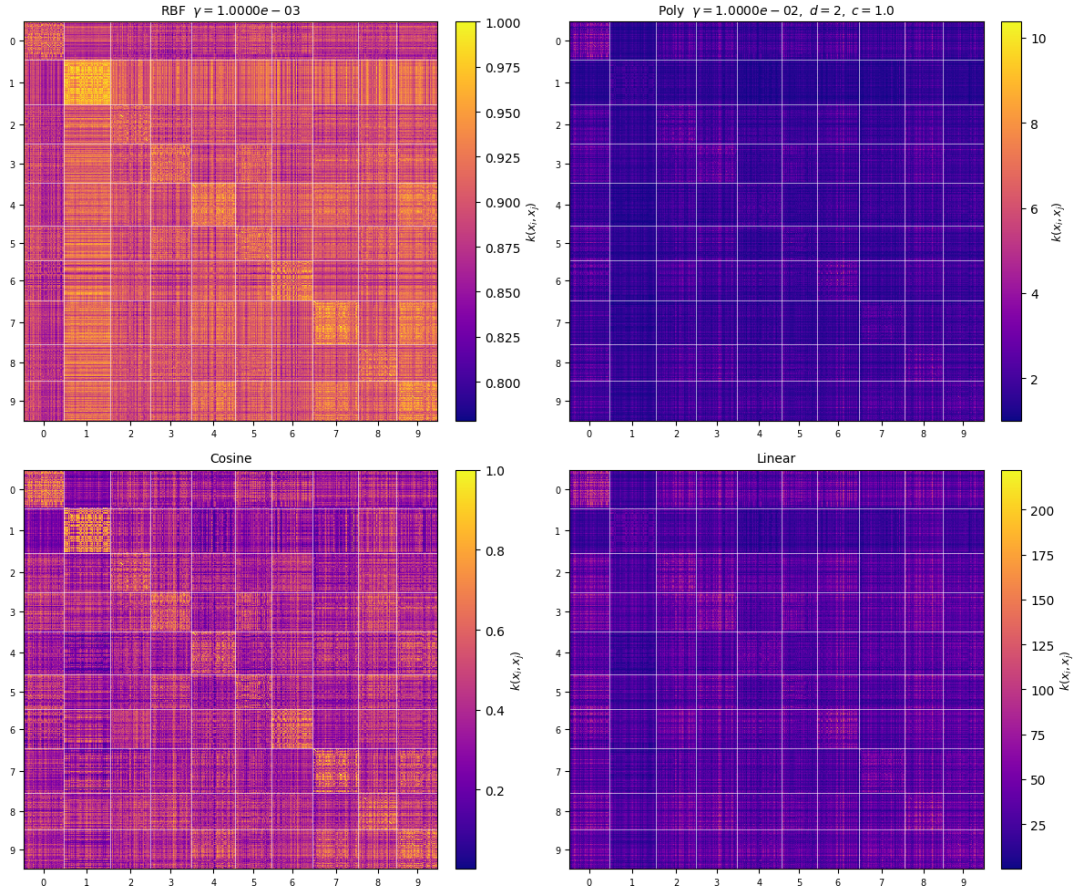

Figure 12 Heat map plots of kernel matrices sorted by classification labels of 4 different kernels used.

To evaluate the results, we can treat the test accuracy (88.65%) of SVC with no PCA as a baseline performance. Looking at table 5, from the column of linear kernel, it is demonstrated that linear PCA does not fundamentally improve the performance of classification algorithm, since the class boundary may already be relatively linear, while the non-linearity data structure

cannot be handled by linear PCA. Nevertheless, it has been powerful at representing the original data in very low dimension. Linear PCA outperforms all other kernels at $M = 16$, and achieves a similar test accuracy as plain SVC that works on $p = 784$ dimensions.

In comparison, the Cosine kernel achieves a better result at moderately low M. With only 32 principle components, Cosine KPCA reaches the baseline accuracy and outperforms linear PCA. It is robust under different M's, and gives a visible improvement to the classifier's performance up to around 1%.

Furthermore, the tuned RBF kernel has a similar performance as Cosine kernel, but is outperformed since the class boundaries do not contain abundant non-linear features, which hinders RBF kernel's powerfulness of capturing non-linear features. Instead, the more selective Polynomial kernel achieves a high peak performance of 92.35% at $M = 1024$ after tuning. This suggests that degree-2 non-linear feature is enough to gain the ability to distinguish digits of different shapes in the original data. However, the captured features are only distinguishable when projected to a high-dimensional space spanned by principle components. The following figure 13 can also give a hint on what actually happens within kernel PCA.

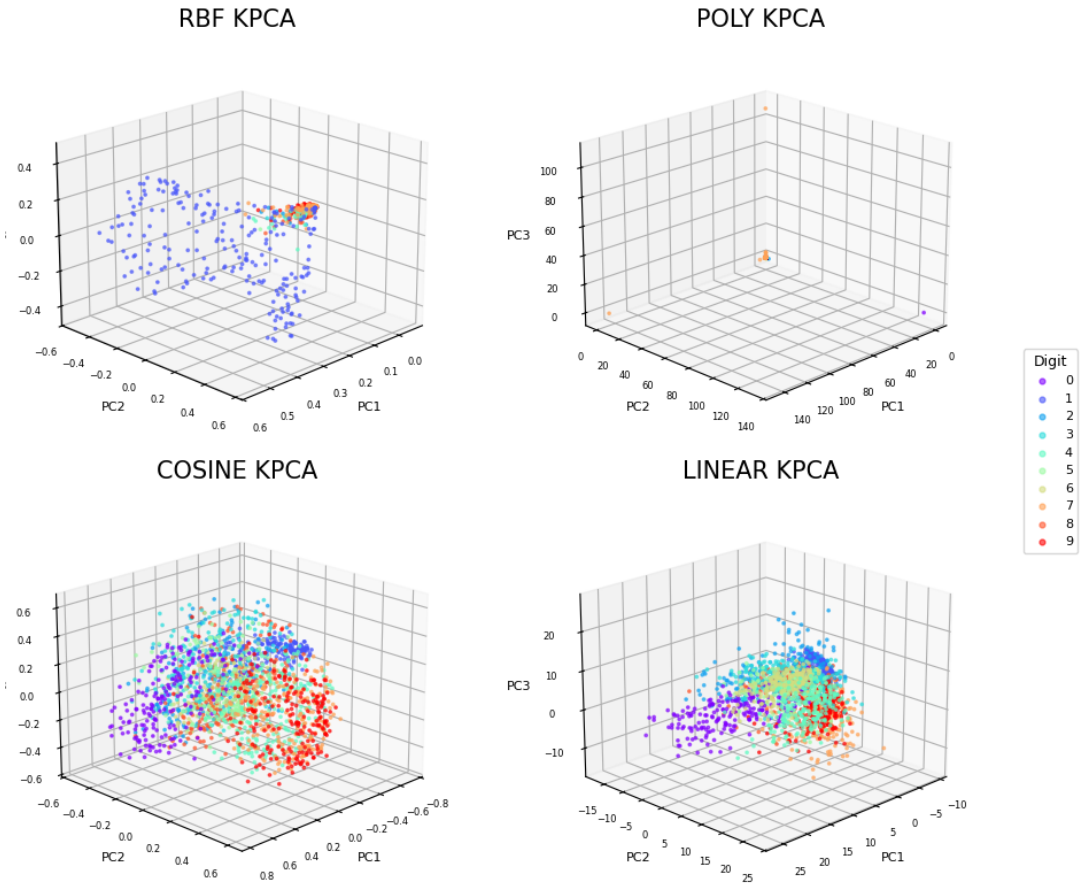

Figure 13 Visualization of the Kernel PCA by projecting data onto the first 3 kernel principle components with tuned parameters. Notice that RBF draws a clear distinction between digit 1 and all other digits, since 1 is distinctively linearly-shaped comparing to others; on the other hand, the polynomial kernel depends greatly on large M, so it "collapses" into clotted dots.

In a word, we have demonstrated the importance of fine-tuning hyper-parameters of kernels as they have a strong impact on the final performance, and we have showcased that Kernel PCA is capable of improving the performance of classification by extracting essential information,

separating entangled clusters and exposing a variety of curved structures in the original data. It is also worth noting that the choice of kernel matters much in application. For this experiment on MNIST, due to the simplicity of the form of data, it is preferable to use kernels like Cosine kernel that does not need fine-tuning to get a robust, acceptable performance; at the same time, it can be fruitful to fine-tune a more targeted kernel like Polynomial kernel to achieve a peak performance while using simple classifiers. The choice of kernel should be made according to both the nature of the data being researched on, and the purpose of applying Kernel PCA (for instance, to optimize performance or to lower computational cost by preprocessing into lower dimensions).

4 Conclusion & Future Focus

In this report, we have investigated Principal Component Analysis as well as Kernel Principal Component Analysis from both practical and theoretical perspectives. We began with formulating PCA, and demonstrated how the notion of principal components arise naturally from a maximum variance and also a minimum reconstruction error viewpoint. The idea of a dual representation was then shown, which allowed us to formulate the algorithm in terms of inner products between data points.

We then used this idea to introduced kernel methods and the 'kernel trick', which allowed PCA to be performed without knowing explicitly what the non-linear feature map was. We derived the Kernel PCA algorithm, which was then used in some practical applications.

As a visual example, we conducted experiments on the Olivetti Faces dataset, by performing classification via a k-Nearest Neighbours, and also reconstruction using a kernel ridge regression algorithm. In both these cases we saw that Linear PCA could be outperformed by PCA performed with a select choice of kernel function, highlighting the power of this dimensionality-reduction technique.

In order to further investigate the mechanism of kernel PCA and the issues of choosing and fine-tuning kernels, we experimented on MNIST dataset by comparing the performance of different kernels as a preprocessor for SVC classifier, as well as grid searching on different values of parameters. With the test accuracy of SVC classification with no PCA preprocessing as a baseline, we concluded that kernel PCA is capable of both achieving a similar performance as the baseline while drastically reducing data dimensions, and reaching a peak performance that is significantly better after tuning and under certain circumstances. A good use of kernel PCA depends on the purpose of applying it and the underlying structure of data, and is essentially associated with the choice of kernel and hyper-parameters. In this experiment, for instance, we found that for data without sophisticated, high-order non-linear structures, using a Polynomial kernel can be more effective than many other kernels. However, it is also hard to fine-tune and not effective on low dimensions.

To sum up, we have formulated the mathematical proof of linear and kernel PCA from the most basic motivations, and have showcased the effectiveness of PCAs by concrete experiment results, and further explored on visualizing the effect and optimizing the performance. Looking ahead, future research could explore larger-scale experimental evaluations, derive stronger theoretical justifications for hyperparameter selection, and extend the application of PCA-based methods to a wider range of unsupervised learning problems, including clustering.

A GitHub Repository

The code used to perform experiments and plot graphs can be found in the GitHub repository: https://github.com/Deuter1um/M2RG20-Kernel-Method.git

B Use of Generative AI

Generative AI was used to assist with the development of code used to run various experiments associated with the project. It was also used for brief assistance with LaTeX queries, and with general aid in understanding theoretical results. All the contents of this report were created originally by all group members.

References

[1] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[2] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417, 1933.

[3] I. T. Jolliffe. *Principal component analysis*. Springer, 2nd edition, 2002.

[4] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[5] S. W. Choi, C. Lee, J.-M. Lee, J. H. Park, and I.-B. Lee. Fault detection and identification of nonlinear processes based on kernel pca. *Chemometrics and intelligent laboratory systems*, 75(1):55–67, 2005.

[6] M. Girolami. Mercer kernel-based clustering in feature space. *IEEE transactions on neural networks*, 13(3):780–784, 2002.

[7] M.-H. Yang, N. Ahuja, and D. Kriegman. Face recognition using kernel eigenfaces. In *Proceedings 2000 international conference on image processing (Cat. No. 00CH37101)*, volume 1, pages 37–40. IEEE, 2000.

[8] J.-M. Lee, C. Yoo, S. W. Choi, P. A. Vanrolleghem, and I.-B. Lee. Nonlinear process monitoring using kernel principal component analysis. *Chemical engineering science*, 59(1): 223–234, 2004.

[9] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. *Advances in neural information processing systems*, 13, 2000.

[10] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[11] S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. *Advances in neural information processing systems*, 11, 1998.

[12] J.-Y. Kwok and I.-H. Tsang. The pre-image problem in kernel methods. *IEEE transactions on neural networks*, 15(6):1517–1525, 2004.

[13] C. M. Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024.

[14] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[15] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3 (1):71–86, 1991.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] scikit-learn developers. The olivetti faces dataset. https://scikit-learn.org/0.19/datasets/olivetti_faces.html, 2018. Accessed: 2026-06-01.

[18] M. Welling. Kernel ridge regression, 1991. Lecture notes.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] S. R. Sain. The nature of statistical learning theory, 1996.