MATH3001 Project Report
2024/25

UNIVERSITY OF LEEDS

High-dimensional data analysis and dimension reduction techniques

Cian Goodwin, 201608919

Supervisor: Jochen Voss
March 28, 2025

Contents

1 Introduction to High Dimensional Data Analysis

High-dimensional data analysis is an increasingly important topic that is relevant in almost every field of work. High dimensional data, sometimes called 'Big data' can produce extremely important insights however, finding them can be challenging. This is because when the data is in a high dimensional space it quickly becomes impossible to even comprehend the underlying patterns within the data set. This highlights the undeniable importance of dimension reduction techniques. Before investigating these techniques, it is vital that we are convinced of their importance by exploring possible scenarios where high dimensional data analysis is required.

1.1 Motivation for High Dimensional Data Analysis

'Big Data' is a broad term given to complex data sets which contain a huge amount of information due to the large number of features/variables. This causes an ongoing challenge when storing and attempting to develop insight from the data. Big data sets are typically stored in the form of an $(n \times p)$ data matrix, which we will often define as X.

$$X_{n\times p} = n \overbrace{\begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & \cdots & \vdots \\ x_{31} & \vdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & \cdots & \cdots & \cdots & x_{np} \end{pmatrix}}^{p}$$

Here each row of X corresponds to an observation and each column represents a different variable. The dimensionality of a data set is equal to its number of variables, p. In practice, data sets with a high number of dimensions are extremely common. Here are a few examples of fields that use high-dimensional data sets.

Medical data

Medical data in the UK contains around 68 million unique medical reports, one for each citizen. These medical reports are extremely important as the research carried out on this data could lead to diagnosis and treatment of otherwise unknown health issues. Every medical report tracks a large number of variables. For example name, age, height, weight and blood type. As advancements are constantly being made in the medical industry this number of variables will only increase creating an extremely high dimensional data set.

Sports data

The use of data in sports is on the rise as coaches attempt to optimise results, scouts search for future signings and fans analyse the performance of their favourite team/athlete. More specifically in football, statistics like number of goals, assists, passes and tackles are just a few of the vast range of variables a team may use to breakdown a players performance. Due to the increasing amount of money invested into sport, having the ability to turn this high dimensional data into insight is incredibly important.

Stock market data

When analysing the stock market prices of a sample of companies, we will track the value of each company daily. This leads to each day corresponding to a new dimension in the data set. In this

field there is huge financial risk, so the ability to produce insight and predict future trends using a high dimensional data set is extremely important.

1.2 Dimensionality reduction

A common misconception when first learning about high-dimensional data analysis is that more data or 'bigger' data sets leads to better insight. Although this is ultimately the case, at first the opposite is true. It can be more of a hindrance to your results. This is due to the 'Curse of dimensionality'.

1.2.1 The 'Curse of Dimensionality'

The "Curse of dimensionality" is an expression named by Richard E. Bellman in 1961 [1]. It refers to the fact that when the dimensionality of a space increases, the volume of this space increasing exponentially. This exponential increase in volume is easier to visualise with the aid of an example.

Suppose we have a continuous variable that ranges over a unit interval of length 1. This is a 1-dimension plane (a line of length 1). Now suppose that in order to understand the trend of this variable over the interval we need at least 10 evenly spaced data points. In other words we need no more than 0.1 distance between each data point. If we have two correlated variables of this description, the data is now on a 2-dimensional plane (unit square with sides of length 1). In order to understand the trends between both variables and still keeping a minimum of 0.1 distance between data points we would now need 100 points to fill the 2-dimensional plane. Extending this to 3-dimensions would require 1000 data points. The emerging pattern here is that for a k-dimensional space we would need 10^k data points. This very quickly becomes extremely large which highlights the exponential increase in volume caused by the growing number of dimensions.

This exponential increase in volume causes a lot of issues when trying to gain insight from a data set. Domingos [1] explains how many techniques used in low dimensional space become impossible when the number of dimensions increases, in turn highlighting the importance of dimensionality reduction.

1.2.2 Dimensionality Reduction Techniques:

There are many techniques of dimensionality reduction each with differing assumptions about the input data. However, most stem from similar underlying ideas. A key assumption which held in most dimensionality reduction techniques is that despite the input data being located in a high dimensional space, the relationships and correlations between variables causes the data to lie within a lower dimensional subspace.

In practice, perfect representations of the data in a lower dimension are very rare and likely impossible. Therefore optimising this reduction in dimensionality involves retaining, as well as possible, the important properties of the data set whilst discarding the arbitrary noise.

1.3 Report Outline

The goal of this report is to introduce a few of the most popular dimension reduction techniques. Each section is dedicated to a different technique where we will first explore the theory and motivations behind the method before applying them to real world datasets.

In chapter 2 we discusses a matrix decomposition technique called the Singular Value Decompo-

sition (SVD) which is foundational to many important dimension reduction techniques. Chapter 3 investigates the most popular dimension reduction technique Principal Component Analysis (PCA). Here we explore its simplicity and versatility, whilst also highlighting some weaknesses of the technique caused by its strong assumptions. The remainder of the report is dedicated to explaining ways to overcome scenarios where PCA fails. Chapter 4 transitions to Multidimensional scaling (MDS). This technique provides the ability of dimension reduction when we are only given the dissimilarities between observations. Chapter 5 introduces Kernel PCA (K-PCA), which is an extension of standard PCA where kernel functions are used. This allows us to effectively reduce the dimensionality of data sets which exhibit non-linear relationships between variables.

2 Singular Value Decomposition

The aim of this chapter is to introduce a new technique called Singular Value Decomposition (SVD), which is in fact, a generalization of the more widely known technique eigen-decomposition. It builds upon the principals of eigen-decomposition, resulting in a similar process which works for all matrices. In particular SVD can be applied to non-square matrices. SVD is widely used in high dimensional data analysis since it provides the foundations for a lot of dimensionality reduction techniques such as, PCA and MDS.

The theory in this section builds upon the work of both Kalman in [2] and Brunton and Kutz in [3]. In section 3.1 we first explore the close relationship between SVD and eigen-decomposition. Section 3.2 provides a method for constructing the SVD of a given matrix. Section 3.3 explains the potential importance of the Economy SVD for when using this technique in practice. Finally, section 3.4 discusses an application of SVD on a data set using a simple example to demonstrate its purpose.

2.1 The relationship between SVD and Eigen-decomposition

Eigen-decomposition is one of the most fundamental and useful skills in linear algebra. It provides insight into the fundamental behavior of the matrix and can significantly simplify many matrix calculations which would otherwise require large amounts of computation. Eigen-decomposition is applied to square matrices in order to construct the unique description of that matrix in terms of its eigenvalues and eigenvectors. Consider an $n \times n$ matrix A, the unique eigen-decomposition of A is given by,

$$A = \Gamma \Lambda \Gamma^{-1},$$

where Γ is the matrix with the n eigenvectors of A as columns and Λ is the diagonal matrix with values corresponding to the eigenvalues of A. These are related so the i^{th} column of Γ is the eigenvector of A which has the i^{th} diagonal entry of Λ as its corresponding eigenvalue.

In the special case where A positive semi-definite (p.s.d), then its eigen-decomposition is called the spectral decomposition. Since A is p.s.d and consequently symmetric, its eigenvectors are pairwise orthogonal. Therefore, Γ is an orthogonal matrix and the unique spectral decomposition of A is of the form,

$$A = \Gamma \Lambda \Gamma^{T}.$$

Despite being extremely useful, as previously stated, eigen-decomposition is limited to square matrices. When we are dealing with data sets in practice, it is extremely rare for a data set to be

square as we will likely have a larger sample of observations, n, than variables, p. In this case, we use singular value decomposition. SVD allows us to describe any $(n \times p)$ matrix B in the form,

$$B = U\Sigma V^T, \qquad [2] \tag{2.1}$$

where, U and V are orthogonal matrices of sizes $(n \times n)$ and $(p \times p)$ respectively. Σ is a diagonal matrix with the same dimensions as B. The columns of U are called the 'left singular vectors' of B. The columns of V and called the 'right singular vectors' of B. Finally the diagonal entries of Σ are called the 'singular values' of B [2]. The singular values of B are ordered in terms of size. So, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$.

Dan Kalman in [2] gives a good explanation of how we can gain a deeper insight into what these decompositions show us. He explains that if we consider a symmetric $(n \times n)$ matrix A as a linear transformation which maps vectors in $\mathbb{R}^n$ to vectors in $\mathbb{R}^n$, then the spectral decomposition of A breaks this process down into its individual parts. First, Γ^T rotates the orthonormal basis of eigenvectors of A to align with the standard basis. Then Kalman explains how Λ "dilates and contracts these components according to the magnitude of their corresponding eigenvalues" [2]. Finally, Γ rotates the standard basis back into its basis of eigenvectors.

In the analogous case, an $(n \times p)$ matrix B describes a linear transformation of vectors from $\mathbb{R}^p$ to $\mathbb{R}^n$. Similarly, SVD breaks down this process into the 3 individual steps. V^T provides the basis vectors for the domain in $\mathbb{R}^p$ and rotates them to align with the respective standard basis. Σ acts like Λ in the previous example. It dilates and contracts these components whilst also discarding additional dimensions if $p > n$, and introducing new dimensions with scale 0 where $p < n$. Then finally U provides the new orthogonal basis vectors and rotates the standard basis to match this new basis in $\mathbb{R}^n$.

The close connection between SVD and spectral decomposition becomes even more important when faced with the problem of its construction. In fact, spectral decomposition plays a vital role in finding the SVD of a matrix. The following section will introduce techniques of constructing the SVD and work though a simple example.

2.2 Construction of the SVD

As defined in the previous section, the SVD breaks down a matrix transformation into 3 simple parts. The three steps in terms of an $(n \times p)$ matrix B are;

1. V^T rotates the right singular vectors which make up the columns in V to align with the standard basis.

2. Σ scales these components with respect to the corresponding singular values. Appending and discarding dimensions as required.

3. U rotates the standard basis to align with the left singular vectors which make up the columns of U.

In order to construct the SVD of a given matrix B, we need to find an pairwise orthogonal set of vectors (the right singular vectors) where after undergoing the transformation described by B map to a new set of pairwise orthogonal vectors (the left singular vectors). That is, an orthogonal set of vectors which retains its orthogonality under the transformation of B. If we were to choose any random orthogonal set of vectors to be the right singular vectors there is no guarantee that the set of their respective images is also orthogonal. Consider the matrix,

$$
\begin{aligned}
B^TB = \left(U\Sigma V^T\right)^T\left(U\Sigma V^T\right) &= V\Sigma^TU^TU\Sigma V^T && \text{(Since } U \text{ is orthogonal)}\\
&= V\Sigma^T\Sigma V^T && \text{(Since } \Sigma \text{ is diagonal)}\\
&= VDV^T.
\end{aligned}
$$

Where the columns of V are the orthogonal, right singular vectors of B. And D is the diagonal matrix with entries corresponding to the squares of the singular values. Also we know that the product of a matrix and its transpose is always symmetric and therefore p.s.d. So, B^TB is p.s.d. With this in mind the unique spectral decomposition of B^TB is,

$$B^TB = \Gamma_1\Lambda\Gamma_1^T.$$

The uniqueness of this spectral decomposition implies that the decomposition of B^TB we derived above by using the SVD of B is exactly equal to this spectral decomposition of B^TB. So, $V \equiv \Gamma_1$ and $D \equiv \Lambda$. So, the right singular vectors of B are exactly equal to the eigenvectors of B^TB. A similar technique can be used to find the left singular vectors. Consider the matrix,

$$
\begin{aligned}
BB^T = \left(U\Sigma V^T\right)\left(U\Sigma V^T\right)^T &= U\Sigma V^TV\Sigma^TU^T && \text{(Since } V \text{ is orthogonal)}\\
&= U\Sigma\Sigma^TU^T && \text{(Since } \Sigma \text{ is diagonal)}\\
&= UDU^T.
\end{aligned}
$$

Again, BB^T is symmetric and p.s.d. therefore has a unique spectral decomposition given by,

$$BB^T = \Gamma_2\Lambda\Gamma_2^T.$$

Using the same argument as before, $U \equiv \Gamma_2$ and $D \equiv \Lambda$. So the left singular vectors of B are exactly equal to the eigenvectors of BB^T. In both cases we have that $D \equiv \Lambda$. Recall, the entries of D are the squares of the singular values of B and, the entries of Λ are the eigenvalues of B^TB and BB^T. So, we also have that the singular values are the square root of the eigenvalues of either B^TB or BB^T. These sets of eigenvalues are equal except the larger of the two matrices has extra eigenvalues of zero to fill the surplus diagonal elements.

2.3 The Economy SVD

In practice, when faced with an $(n \times p)$ data matrix X, it is almost always the case that $n > p$. That is there are more observations than variables. This can result in X and Σ to be very 'tall and thin'. Since Σ is a diagonal matrix, we can identify that the final $n - p$ rows of Σ will all be 0. Therefore, we can use this feature to easily create a reduced form for the SVD.

Consider again equation (2.1), the definition of SVD. Let $\mathbf{u}_i$ and $\mathbf{v}_i$ correspond to the i^{th} left and right singular vectors of X respectively. Also let σ_i represent the i^{th} singular value of X. Hence, the SVD of X looks like so,

$$
X_{n\times p} = U_{n\times n}\Sigma_{n\times p}V^T_{p\times p} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \cdots & \mathbf{v}_1^T & \cdots \\ \cdots & \mathbf{v}_2^T & \cdots \\ & \vdots & \\ \cdots & \mathbf{v}_p^T & \cdots \end{bmatrix} \tag{2.2}
$$

When the matrix multiplication in equation 2.2 expanded, $\mathbf{u}_i$ is scaled by σ_i. However, there are only p singular values causing the final $n-p$ rows to be rows of 0. Therefore, the final $n-p$ left singular vectors are scaled by 0 meaning they don't contribute to the composition of X. To simplify 2.2 we can truncate both U and Σ to remove these redundant columns and rows.

$$X_{n\times p} = \widehat{U}_{n\times p}\widehat{\Sigma}_{p\times p}V^T_{p\times p} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_p \\ \vdots & \vdots & & \vdots \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p \end{bmatrix} \begin{bmatrix} \cdots & \mathbf{v}_1^T & \cdots \\ \cdots & \mathbf{v}_2^T & \cdots \\ & \vdots & \\ \cdots & \mathbf{v}_p^T & \cdots \end{bmatrix} \tag{2.3}$$

Equation (2.3) is the economy SVD. Both $\widehat{U}$ and $\widehat{\Sigma}$ represent the truncated matrices. It is useful in practice as it allows us to express X in terms of three matrices all with the same rank as the original matrix. Consequently, the economy SVD is much easier to store as the redundant components are removed.

2.4 SVD for Data Compression

One major use of SVD is for data compression or reduced rank matrix approximation. Kalman [2] describes the application well. Originally, storing X needs np entries, one for each number in the matrix. Kalman [2] describes how we best approximate X with lower rank matrices. As a consequence providing the ability to store a good approximate of X using as little as $n+p$ entries.

Recall that the rank of a matrix X is equal to the number of linearly independent rows or columns that make up X. In terms of the SVD, as we saw in equation 2.3, the rank of X is equal to the number of non-zero singular values. Let $\text{rank}(X) = k$. Equipped with this knowledge of the rank of X and equation 2.2, we can express the SVD of in terms of a linear combination of rank-1 matrices.

$$X = U\Sigma V^T = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \qquad [2] \tag{2.4}$$

Hence, X is the linear combination of k rank-1 matrices. Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k$, the rank one matrices which correspond to the higher singular values will contribute more to the composition of X. So, if we wanted to produce the best rank-1 approximation of X then this would be the first term of the sum in equation (2.4).

$$X \simeq \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T,$$

is the best rank-1 approximation of X. Kalman [2] describes how this approximation needs only $n+p$ entries to store. This is extremely useful when dealing with real world data where values of n and p are extremely large. However, rather unsurprisingly, rank-1 approximations often don't produce a good representation of the data. By using the same argument of the order of singular values, we can create the best rank-r approximation of X by using the first 'r' rank-1 matrices of the sum in equation (2.4). This approximation can be stored using only $r(n+p)$ entries.

Choosing the best rank to use for an approximation comes down to 2 opposing factors. The first is the amount of storage needed to hold the approximation, where a smaller storage size is more desirable. The second is maximising the level the accuracy of the approximation. This can be measured by the proportion of singular values used to create the approximation. Improving either of these two factors will negatively impact the other so, finding a satisfactory balance is very important.

2.4.1 An example of using SVD for data compression on a greyscale image

Consider the following 119 pixel greyscale image in Figure 1,

Figure 1: A greyscale image of the numbers 0, 1, 2 and 3 in white on a black background.

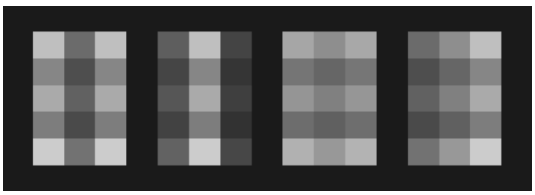
A greyscale image can be easily described by a data matrix X where pixel (i, j) of the image is represented by entry (i, j) in X. These entries are on the continuous interval $[0, 1]$ where 0 refers to a black pixel and 1 refers to a white pixel. The greyscale image in Figure 1 is described in this way by the following data matrix,

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

X has dimensions (7×17) so uses $7 \times 17 = 119$ entries, one for each pixel. The non-zero singular values of X are,

$$\begin{bmatrix} 5.8389075 & 1.8248816 & 1.4463905 & 0.9878305 & 0.7135209 \end{bmatrix}.$$

So, the rank of X is 5 and a rank-5 approximation of X produces the image in Figure 1 perfectly. Using SVD, we can find the best reduced rank approximations of X for each rank lower than 5. The images produced by these approximations, the number of entries used to store each image and the total accuracy of the respective images can be seen below.



(a) Rank-1, 24 entries, 54% accuracy

(b) Rank-2, 48 entries, 71% accuracy

(c) Rank-3, 72 entries, 84% accuracy

(d) Rank-4, 96 entries, 93% accuracy

Figure 2: 4 greyscale images produced by varying levels of reduced rank approximation of X.

In Figure 2 there is a clear increase in accuracy as the rank of approximation increases. However, it can also be seen that each subsequent increase in rank causes a smaller improvement to the approximation. Each increase in rank also causes an increase of 24 entries that need to be stored.

Figure 2a is the best rank-1 approximation of X. The first singular value of X is 5.8389075 which accounts for 54% of the total sum of all singular values. The image produced in Figure 2a highlights the areas of the numbers 0, 1, 2 and 3 rather than the detail of individual numbers so, this is a poor representation of the original image. Despite being slightly more accurate, in Figure 2b it is still hard to make out the numbers 2 and 3. Figure 2d is a very good approximation of the original image however the required number of entries has only decreased by 23 which is a very minimal compression. A good choice in this case would be the rank-3 approximation in Figure 2c as, it produces an image with 84% accuracy whilst significantly reducing the number of entries needed form 119 to 72.

For a larger scale example, Brunton and Kutz in [3] carried out this process of data compression on a greyscale image with 2000×1500 pixels. They showed how this image can be compressed using a rank-100 approximation to produce an image with 80% accuracy which uses 11.67% of the original storage.

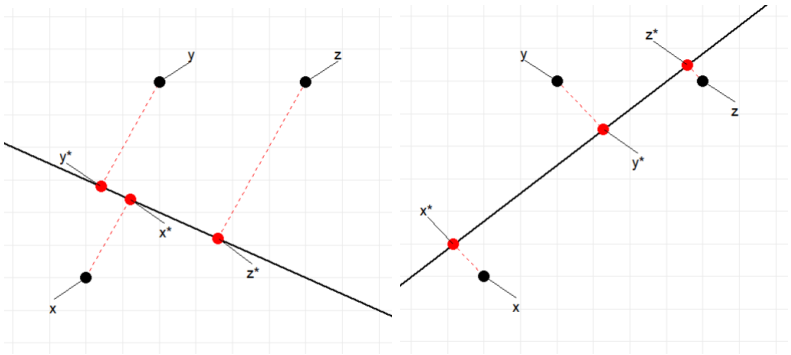
3 Principal Component Analysis

Principal component analysis (PCA) is one of the oldest and most fundamental techniques in multivariate analysis and dimension reduction. Discovered independently by both Pearson (1901) and Hotteling (1933) [4], PCA makes use of eigen-decomposition to find a linear transformation which maximizes the variation retained from a data set when its dimensionality is reduced. Despite being around 100 years old, PCA is still relevant today and has many important applications in a wide range of fields.

This chapter builds upon the work of both Shlens [5] and Jolliffe [4]. We will first build some intuition behind PCA using a simple example in section 3.1. After this in section 3.2 we define a formal method for carrying out PCA. We will then explore the close relationship between PCA and SVD, resulting in a new method for PCA which involves the SVD. In section 3.4 we will apply PCA to a real world data set and develop some insight from the resulting plot. Finally, we finish this section by explaining some of the limitations of PCA.

3.1 General Aim

When reducing the dimensionality of a data set it is likely to be impossible to retain 100% of the information. To visualise this consider Figure 3a.



(a) A poor choice of direction. (b) The best choice of direction.

Figure 3: Two plots with the same 3 non-collinear data points x, y and z but with differing choices of 1-dimensional planes. In both plots the points are projected onto the chosen 1-dimensional plane, these projections are labeled x*, y* and z* respectively.

This is a simple 2-dimensional plot containing 3 non-collinear data points x, y and z. The information that we would like to retain is the variation in the data, this is represented by the distances between the individual data points. In order to reduce this plot to 1-dimension, we would have to project the points onto a straight line which passes through the 2-dimensional plane. The respective projections of these points in Figure 3a are labeled x*, y* and z* and are in red. However by doing this, the specific distances between the points x, y and z differ to the respective distances between their projections x*, y* and z* in the 1 dimensional plane. For example in Figure 3a the distance between y and z is smaller than the distance between x and z but, when we project these points onto the 1-dimensional plane the distance between y* and z* is greater than the distance between x* and z*. This is due to the loss of information when we reduced the dimensionality of our plot.

This inspires the question, can we minimize this loss of information by choosing a specific 1-dimensional plane which best retains the variation within the set of data points when we project the data onto it? In fact this turns out to be true. Consider Figure 3b which contains the same data points x, y and z, however in this plot I chose the 1-dimensional plane to be a simple linear regression model informally known as the 'line of best fit'. Here, the variations between data points are much better represented by the variations between their projections x*, y* and z* in the 1-dimensional plane. In other words, it is possible find the directions within the higher dimensional plot which exhibit the maximum possible variation from the original data set when the dimensionality is reduced which will in turn, minimize the loss of information.

This is the essence of PCA. A technique of dimension reduction for a data set of inter correlated variables with the goal of retaining the maximum possible variation in the data [4]. This is achieved in the form of a linear transformation of the original correlated variables to a new set of uncorrelated variables called the Principal Components (PCs) [4]. The PCs are the directions previously mentioned which exhibit the maximum variation in the data set. The previous example is an extremely simplified version of PCA in which we reduced the dimensionality of a data set consisting of 3 points from 2 dimensions to 1 dimension. In regard to this example, the use of PCA is actually not very helpful at all. No extra insights about the data have been found as a result of this reduction in dimensionality. However, the power of PCA becomes more apparent when we consider higher dimensional data sets. PCA can "reveal some of the sometimes hidden simplified structures that often underlie a high dimensional data set"[5].

In practice it is not uncommon to be faced with a data set with a extremely large number of both dimensions, p, and data points, n. When transferring this technique to data sets in higher dimensions it is vital that the following assumptions are made [5].

1. It is important to retain as much variation as possible.
2. Variables are inter-correlated. Hence, there are linear relationships between variables.
3. The PCs are orthogonal and are ordered in terms of importance from most variation to least variation. PCk is the PC with the k^{th} most associated variance.

This ordering of the PCs is key as we can expect the directions with maximum associated variance to provide insight into the important trends in the data set. Whereas, we can treat the directions of smaller associated variance as noise or random error. PCA allows us to display the data set in terms of only the first k PCs. Consequently, reducing the dimensionality to k whilst retaining the maximum possible variation from the original dataset and ignoring the noise.

3.2 The Method of PCA

3.2.1 Centering and scaling the data

Before carrying out PCA, it is vital that we center the data matrix X. Let X be an uncentered data matrix. Then the centered matrix $\widehat{X}$ is,

$$\widehat{X} = X - \overline{X},$$

where $\overline{X}$ is the matrix with entries equal to the respective column mean in X. This subtracts the respective column mean from each entry of X. Consequently, $\widehat{X}$ has column means of 0. Note that this does not affect the variation in the data set, it simply shifts the p-dimensional cloud of data points to be centered at the origin. This makes computation of the covariance matrix much easier.

Typically it is also useful to scale the data set before carrying out PCA. This means setting the variance of each variable to 1. Without this step, variables with larger variations will dominate when finding the directions of maximum variation.

Due to the importance of these steps we will assume all data sets have already been both centered and scaled.

3.2.2 Deriving the Principal Components

Principal components (PCs) are defined to be oriented in the direction of maximum variation in a data set $X_{n\times p}$. Let Σ be the $(p \times p)$ covariance matrix of X. Now suppose $\underline{a}$ is a p-dimensional vector. We are trying to find the direction of $\underline{a}$ which maximizes the variation in X. So we are maximizing,

$$\mathsf{Var}(\underline{a}^T X) = \underline{a}^T \mathsf{Var}(X)\underline{a} = \underline{a}^T \Sigma \underline{a}, \tag{3.1}$$

with respect to $\underline{a}$. We must also apply the constraint that $\underline{a}^T\underline{a} = 1$. This ensures that we are focused on the direction of $\underline{a}$. Without this constraint, our value for equation (3.1) could always be increased by a multiplying $\underline{a}$ by a scalar greater than 1. So we are now maximizing,

$$\frac{\underline{a}^T \Sigma \underline{a}}{\underline{a}^T \underline{a}}, \tag{3.2}$$

with respect to $\underline{a}$. Since Σ is p.s.d, we can find its spectral decomposition such that $\Sigma = \Gamma\Lambda\Gamma^T$. Γ is the orthogonal matrix with columns containing the eigenvectors of Σ and Λ is the diagonal matrix with entries corresponding to the eigenvalues of Σ. If we make the substitution of $\underline{b} = \Gamma^T\underline{a}$, we can manipulate (3.2) like this,

$$\frac{\underline{a}^T \Sigma \underline{a}}{\underline{a}^T \underline{a}} = \frac{\underline{a}^T \Gamma\Lambda\Gamma^T \underline{a}}{\underline{a}^T \underline{a}} = \frac{\underline{a}^T \Gamma\Lambda\Gamma^T \underline{a}}{\underline{a}^T \Gamma\Gamma^T \underline{a}} = \frac{\underline{b}^T \Lambda \underline{b}}{\underline{b}^T \underline{b}}.$$

Λ is the diagonal matrix with entries, λ_i, which correspond to the eigenvalues of Σ. Now order the eigenvalues so that, $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_p$.

$$\frac{\underline{a}^T\Sigma\underline{a}}{\underline{a}^T\underline{a}} = \frac{\underline{b}^T\Lambda\underline{b}}{\underline{b}^T\underline{b}} = \frac{\sum_{i=1}^{p}\lambda_i b_i^2}{\sum_{i=1}^{p} b_i^2} \leq \lambda_1 \frac{\sum_{i=1}^{p} b_i^2}{\sum_{i=1}^{p} b_i^2} = \lambda_1.$$

So (3.2) is bounded above by λ_1. This bound is achieved when $\begin{bmatrix}1 & 0 & \dots & 0\end{bmatrix}^T$. Therefore by reversing our previous substitution,

$$\underline{a} = \Gamma\underline{b} = \begin{bmatrix}\gamma_1 & \gamma_2 & \dots & \gamma_p\end{bmatrix} \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix} = \gamma_1.$$

$\underline{a} = \gamma_1$ is the solution to maximizing (3.2). γ_1 is the eigenvector corresponding to λ_1, the largest eigenvalue of Σ. So we have shown that the direction of maximum variance in $X_{n\times p}$, PC1 is in the direction of γ_1. Since PC2 must be orthogonal to PC1, we can show using a similar method that PC2 is in the direction of γ_2. In general, the i^{th} PC is in the direction of the eigenvector which corresponds to the i^{th} largest eigenvalue of Σ.

3.2.3 Calculating the PC scores

Recall that the goal of PCA was to find a linear transformation of the variables of a data set X which produces a new orthogonal and uncorrelated set of variables called the principal components. These PCs are also required to be ordered in terms of the proportion of total variance from the data set that they explain. We now know that the directions of the principal components are equal to the eigenvectors of the centered covariance matrix of the data set X. These eigen vectors are typically called the 'loadings' of X. The entries in the loadings of X contain the coefficients of the linear transformation we are trying to find. So let L be the loading matrix which has columns corresponding to the loadings of X. Consider,

$$Y = XL.$$

Since as we described the loadings in L are the eigenvectors of Σ_X, covariance matrix of the variables in X, L is orthogonal and is simply a rotation of the standard basis to the new PCs. So Y describes the same data set as X but, instead in terms of the principal components. The entries of Y are called the PC scores, entry (i, j) contains the score of observation i on PCj. Using these scores we can now create a plot of the data set in terms of the first k PCs. The coordinates of observation i in our plot will be the first k columns of the score matrix, Y, after carrying out PCA. Due to the original assumptions this plot is the best possible representation of X in k-dimensional space.

3.2.4 Choosing a suitable dimensionality

Now that we have the loadings and PC scores, we must choose the dimensionality of our reduction. Typically, k is chosen to be 2 allowing us to create a 2D plot, called a biplot, of PC1 against PC2 however the choice of k can vary depending on the data set and the purpose of the dimension reduction.

A more intuitive choice of k would be to base it on the desired proportion of total variation

retained after the reduction. The total variation in a data set is equal to the sum of the variance for each variable. Note that this is equal to the trace of the covariance matrix, Σ_X. It is true that the trace of any matrix is equal to the sum of its eigenvalues. We know that each eigenvalue of Σ_X is paired with a unique eigenvector which we have shown are the loadings of X. So, the proportion of the variance shown by each PC, is equal to the proportion of total variance found in the respective eigenvalue. Therefore, the first k PCs exhibit,

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{p} \lambda_i}, \tag{3.3}$$

percent of the total variance from the original p-dimensional data set. Taking equation (3.3) into account, we can now determine the required k needed to retain a given level of variation.

It can also be helpful to create a scree plot. A scree plot, is a graph of PC against corresponding proportion of accounted variance. The ordering of the PCs means this is a decreasing graph so the later PCs can be disregarded. An example of a scree plot can be seen in section 4.4 when we apply PCA to a data set.

3.3 The close relationship between PCA and SVD

PCA has an extremely close relationship to SVD. Jolliffe [4] describes how SVD can give us great insights into how PCA works. Schlens [5] also explains how SVD can be extremely useful when carrying out PCA, by contributing to a more efficient method. In order to understand this consider the covariance matrix of a centered data set X,

$$\Sigma_X = \frac{1}{n} X^T X.$$

We derived that the loadings are the eigenvectors Σ_X, therefore they are also the eigenvectors of $X^T X$. From chapter 3 of this report, we can also find the SVD of X.

$$X = U \Sigma V^T$$

Note that here Σ is the diagonal matrix containing the singular values of X, whilst Σ_X is the covariance matrix of X. As defined, the right singular vectors which make up V are the eigenvectors of $X^T X$. Hence the right singular vectors of X are also the loadings of X. Using this lets multiply both sides of this SVD on the right by V.

$$XV = U \Sigma V^T V$$

V contains the loading vectors, $V = L$. Also V is orthogonal so $V^T V = \mathbb{I}$. Therefore,

$$Y = XV = U\Sigma \implies Y\Sigma^{-1} = U.$$

Y contains the PC scores of the observations. So, the matrix of left singular vectors, U, contains the scaled PC scores. This method of PCA using the SVD is a more efficient and actually provides some more information about the data.

3.4 Applying PCA to a data set

In this section we will apply PCA to a real world dataset with the goal of effectively reducing its dimensionality and gaining insight about otherwise hidden correlations between variables. The data set I have chosen is taken from the Coffee Quality Institute (CQI). It contains a sample of 207 types of coffee beans. The original database contains a wide range of information about each bean for example, the region of origin, harvest year and processing method. In order to apply PCA to this data set, I have focused on the numerical data. I reduced the dataset down to 7 variables.

- Aroma
- Flavour
- Aftertaste
- Acidity
- Body - Viscosity of coffee
- Balance - How well different flavours of the coffee blend
- Overall

All 7 variables a recorded as a score in there respective category out of 10. More information on the data set, can be found at https://database.coffeeinstitute.org/coffees/arabica. We can expect strong correlations between the chosen variables so PCA is a good choice of dimensionality reduction technique here.

Since we have scaled this data set, the total variance is equal to $p = 7$. After applying PCA we obtain the following eigenvalues of the covariance matrix,

$$\begin{bmatrix} 5.81743420 & 0.40490520 & 0.22641794 & 0.19765108 & 0.13603074 & 0.12124865 & 0.09631217 \end{bmatrix}$$

Using equation (3.3) we can show that the cumulative sum of variation shown is,

$$\begin{bmatrix} 83.11\% & 88.89\% & 92.13\% & 94.95\% & 96.89\% & 98.62\% & 100\% \end{bmatrix}.$$

Hence, 83.11% of the total variation in the CQI data set is found in just PC1 and a further 5.78% accounted for in PC2. So, if we set $k = 2$ we can create a biplot which accounts for 88.89% of the total variation in the original data set. We can also show this using a scree plot.

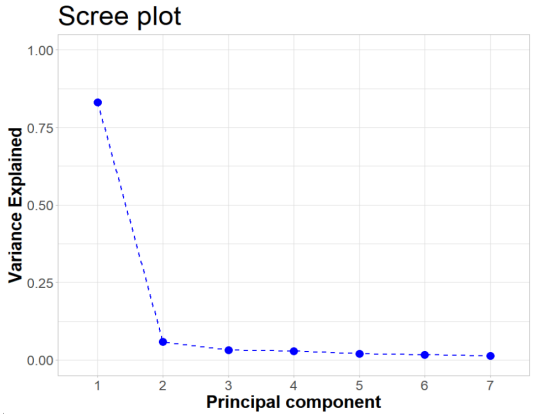


Figure 4: Scree plot showing the proportion of variance explained by each PC.

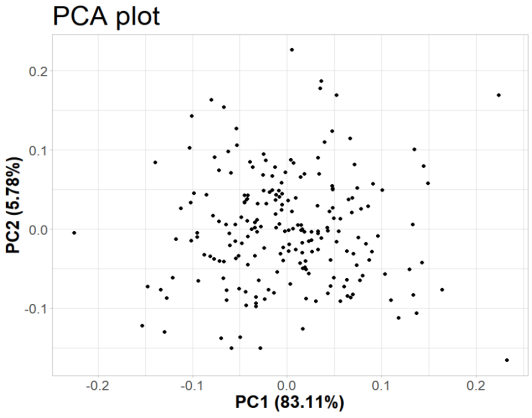


Figure 5: PCA biplot of beans data using first two PCs.

Figure 4 shows that after the first 2 PCs very little variation is added so $k = 2$ is a good choice. In Figure 5, we can see the cloud of data points plotted in respect to the first 2 PCs. It is important

to remember that the vast majority of variation is seen in PC1. In order to interpret what this tells us it can be useful to look at the loadings which make up these PCs.

$$\begin{array}{c} \\ \text{PC1} = \\ \\ \text{PC2} = \end{array} \begin{array}{c} \begin{array}{ccccccc} \text{Aroma} & \text{Flavor} & \text{Aftertaste} & \text{Acidity} & \text{Body} & \text{Balance} & \text{Overall} \end{array} \\ \begin{bmatrix} -0.36 & -0.39 & -0.39 & -0.37 & -0.35 & -0.39 & -0.39 \end{bmatrix} \\ \begin{bmatrix} -0.63 & -0.22 & -0.14 & 0.20 & 0.68 & 0.18 & -0.05 \end{bmatrix} \end{array}$$

In PC1 we can see that the weights for each variable are all very close to being equal and they all share the same sign. Therefore PC1, which exhibits 83.11% of the variation, explains the quality of the beans. Giving high quality beans negative scores and low quality beans positive scores. However, PC2 is dominated by the weights for Aroma and Body variables which have similar size but oposite sign. Hence, PC2 compares coffee beans which produce better scented coffee (negative scores) to coffee beans which produce coffee with higher viscosity (positive scores).

3.5 Limitations of PCA

Although PCA can be extremely effective at reducing the dimensionality of data set and removing the noise, unfortunately PCA is not a perfect technique. When applied to the wrong data sets the insights it produces can be wrong. In fact in some cases, PCA can't be applied at all. When faced with either of these issues different dimension reduction techniques are required.

For example, imagine we were faced with a data set containing only the respective distances between all the cities in a country. If there is a large number of cities then this data set is of a very high dimensionality. Reducing this down to a 2-dimensional plot to create a map of the cities in the country is impossible to do using PCA because we have no initial inter correlated variables.

Another issue with PCA is caused by the heavy dependence on the assumption of linearity of relationships between variables. If the relationship between variables is non-linear then PCA breaks down. The insight that PCA reveals about this data is likely not accurate and useless.

The remainder of this report will focus on introducing 2 new methods of dimension reduction which will counteract these two common issues. The next chapter introduces Multidimensional scaling which can be used on dissimilarity data. Whilst in chapter 6 the focus moves to Kernel PCA which provides a technique used to approach data with variables that exhibit non-linear relationships.

4 Multidimensional Scaling

Multidimensional scaling (MDS) is another popular technique for dimension reduction. It is similar to PCA in the sense that both techniques have the goal of best representing a high dimensional data set in a lower dimensional form. However, the input data for each technique is very different. MDS overcomes a limitation of PCA by taking input data in the form of a dissimilarity matrix, $\Delta_{(n\times n)} = (\delta_{ij})$. Here δ_{ij} is some measure of the dissimilarity between object i and j.

Joseph Kruskal [6] provides a good example which helps develop some intuition for the purpose of MDS. Imagine you are provided with the distance between the cities of a country and you are asked to produce an accurate map of the country [6]. Kruskal explains how although there exists geometric techniques to do this MDS provides a more computationally efficient method for creating this spatial map.

The versatility of MDS is a main reason for its popularity. MDS can be applied to many varying forms of dissimilarity data. In the previous example by Kruskal, the dissimilarities were distances between cities therefore we can use Metric MDS (MMDS). However, dissimilarities aren't required to be distances. For example Trevor and Micheal Cox [7] described how an integer score between 0 (identical) and 10 (totally different) comparing different whiskeys is a perfectly fine dissimilarity metric and Non-metric MDS (NMDS) can be applied to this type of data.

The theory in this section builds upon the work of Cox and Cox [7] and Kruskal [6]. Section 4.1 will introduce the most popular form of MDS called Classical MDS and examine its relationship to PCA. In section 4.2 we move onto a more general topic of Metric MDS. Section 4.3 will explain why in some cases, Non-metric MDS is required and how it differs from Metric MDS. Finally, in section 4.4 we will apply a form of MDS to a real data set and attempt to understand the spatial map created.

4.1 Classical MDS

Classical MDS (CMDS) was originally developed by Young and Householder (1938) [8]. They claimed that if we can obtain the euclidean distance matrix, Δ, from a set of n, p-dimensional points then it is possible to find a set of coordinates for the points in $(p-1)$-dimensional space such that the distances are retained perfectly. In other words, it is possible to, using only the euclidean distance matrix, find a data matrix, X, which contains a set of coordinates for the objects which perfectly retains the known distances.

4.1.1 Torgerson's technique for recovering the coordinate matrix

The most popular technique for recovering these coordinates was developed by Torgerson [9]. This technique is also described well by Cox and Cox [7]. To start, let X be the centered data matrix of coordinates we wish to acquire. Define Δ^2 to be the element-wise squared euclidean distance matrix Δ. Δ^2 has elements of the form,

$$\delta_{ij}^2 = (\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j) = {\mathbf{x}_i}^T \mathbf{x}_i + {\mathbf{x}_j}^T \mathbf{x}_j - 2{\mathbf{x}_i}^T \mathbf{x}_j, \tag{4.1}$$

where $\mathbf{x}_i$ is the i^{th} row and observation of X. Now let B be the inner product matrix of X,

$$B = XX^T = (b_{ij}), \qquad\qquad b_{ij} = {\mathbf{x}_i}^T \mathbf{x}_j.$$

Torgerson [9] stated that it is possible to calculate B from Δ^2 then we can use B to find X. Consider equation (4.1), the term ${\mathbf{x}_i}^T \mathbf{x}_i$ doesn't contain index j so when converted into matrix form, the columns are all identical. It is off the form,

$$\begin{bmatrix} {\mathbf{x}_1}^T \mathbf{x}_1 & {\mathbf{x}_1}^T \mathbf{x}_1 & {\mathbf{x}_1}^T \mathbf{x}_1 & \cdots \\ {\mathbf{x}_2}^T \mathbf{x}_2 & {\mathbf{x}_2}^T \mathbf{x}_2 & {\mathbf{x}_2}^T \mathbf{x}_2 & \cdots \\ {\mathbf{x}_3}^T \mathbf{x}_3 & {\mathbf{x}_3}^T \mathbf{x}_3 & {\mathbf{x}_3}^T \mathbf{x}_3 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

So, if we center the rows of this matrix it will be equal to 0. A similar argument can be made for term ${\mathbf{x}_j}^T \mathbf{x}_j$ in equation (4.1) however, in this case we must center the columns. So, if we multiply Δ^2 by the centering matrix $H = I_n - \frac{1}{n} J_n$ (where I_n is the identity matrix and J_n is the $(n \times n)$ matrix of all 1s) on both sides we are left with simply,

$$H\Delta^2 H = -2B.$$

Cox and Cox [7] removed the constant -2 by defining a new matrix $A = (a_{ij})$ where $a_{ij} = -\frac{1}{2}\delta_{ij}^2$. So,

$$B = HAH = XX^T \qquad [7].$$

we know that the product of a matrix and its transpose is always symmetric and therefore p.s.d. So, B is p.s.d. and has spectral decomposition,

$$B = \Gamma\Lambda\Gamma^T = XX^T \implies X = \Gamma\Lambda^{\frac{1}{2}}.$$

Finally, we have retrieved a set of n coordinates in p-dimensional space which preserve the euclidean distances given in Δ. Since we found these coordinates using just the distances rotating, reflecting or translating them will give an equally valid solution.

The steps we have used are,

1. Find the squared euclidean distances between the objects δ_{ij}^2.
2. Construct the matrix $A = (a_{ij})$ where, $a_{ij} = -\frac{1}{2}\delta_{ij}^2$.
3. Find the inner product matrix B by centering both the rows and columns of A. This can be done by multiplying A by the centering matrix H on both sides. $B = HAH$.
4. Using the spectral decomposition of B construct the coordinate matrix $X = \Gamma\Lambda^{\frac{1}{2}}$.

4.1.2 CMDS for dimensionality reduction

In order to reduce the dimensionality of the configuration obtained, we could carry out PCA on this data matrix X to rotate the coordinates to be in terms of its PCs. However if we look at the covariance matrix of X,

$$\begin{aligned}\Sigma_X = \frac{1}{n}X^TX &= \frac{1}{n}\Lambda^{\frac{1}{2}}\Gamma^T\Gamma\Lambda^{\frac{1}{2}} \\ &= \frac{1}{n}\Lambda^{\frac{1}{2}}\Lambda^{\frac{1}{2}} \\ &= \frac{1}{n}\Lambda.\end{aligned}$$

Λ contains the eigen values of B. So, Λ is diagonal which implies that the set of coordinates X is already in terms of its PCs.

This symmetry between CMDS and PCA was first discovered by Gower (1966) [10] where he named it 'The Dual of Principal Component Analyis'. For this reason CMDS is often refered to as Principal Coordinate Analysis (PCO), since the data matrix obtained contains the principal coordinates of the objects.

Using this, if we wish to reduce the dimensionality of the spatial map to k, we can take the first k columns of the obtained data matrix X. These if ordered in terms of decreasing corresponding eigenvalues contain the desired coordinates for a k-dimensional spatial map which best preserves the distances in Δ.

4.2 Metric MDS

Metric MDS (MMDS) is a generalisation of CMDS. It looseness the assumption that the input dissimilarity data are euclidean distances to the assumption that the it can be any metric distance. A metric distance is a dissimilarity that follows these four rules:

1. Identity: $d_{ii} = 0$,
2. Non-negativity: $d_{ij} \geq 0$,
3. Symmetry: $d_{ij} = d_{ji}$,
4. Triangular: $d_{ik} \leq d_{ij} + d_{jk}$.

In MMDS the dissimilarities should correspond as closely as possible to the euclidean distances between points in the lower dimensional spatial map [11], and CMDS is a special case when the input dissimilarities are euclidean distances. There are in fact many other metric distances that are allowed in MMDS. For example Manhattan distance and Minkwoski distance. Since the input data for MMDS can be from a range of distance metrics in order to find the best choice, we are required to optimise a function that Kruskal [6] coined the 'Stress function'.

4.2.1 Stress Functions

The stress function stems from the goal of MDS, $\delta_{ij} \simeq d_{ij}$, we want to minimise the difference between the input dissimilarities and the distances between points in the spatial map. This is similar to least squares regression which explains why it is sometimes referred to as Metric least squares scaling. The raw stress function is,

$$\varepsilon_r = \sum_{i=1}^{n} \sum_{j<i}^{n} \left(\delta_{ij} - d_{ij}\right)^2. \qquad [11] \tag{4.2}$$

We use the inequality in the second summation to take advantage of the symmetry of distances. However, the raw stress equation (4.2) is not invariant to coordinate scaling [11]. Therefore in Kruskal's [6] Stress 1 equation (4.3), he normalises equation (4.2) like so.

$$\varepsilon_1 = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j<i}^{n} \left(\delta_{ij} - d_{ij}\right)^2}{\sum_{i=1}^{n} \sum_{j<i}^{n} d_{ij}^2}}. \qquad [11] \tag{4.3}$$

There are actually many more types of stress function, with different types chosen for different goals. For example, Cox and Cox [7] mention how Sammon's stress function,

$$\varepsilon_S = \frac{\sum_{i=1}^{n} \sum_{j<i}^{n} \frac{\left(\delta_{ij} - d_{ij}\right)^2}{\delta_{ij}}}{\sum_{i=1}^{n} \sum_{j<i}^{n} \delta_{ij}}, \qquad [7] \tag{4.4}$$

gives larger weights to smaller dissimilarities. Hence, better preserving smaller distances in the resulting spatial map. Stress functions act as a way of checking 'goodness of fit', high stress indicates a bad fit and low stress indicates a good fit, therefore minimsing this function gives the optimal fit.

Although optimising the stress function gives the optimal configuration, there is no guarantee this is a good fit. Kruskal [12] claims that any stress over 10% is unsatisfactory, and any under 5% is an impressive fit.

4.2.2 Optimising the stress function

Kruskal [12] claims that, 'The Method of Steepest Descent' was a good numerical way to optimise the stress function. He explains that if we start with an arbitrary configuration of points in the spatial map, then we can progressively improve this configuration with very small iterative movements [12]. To do this find the change which reduces the stress most quickly, then moving our configuration by a very small increment in that direction. After repeating this process we will eventually find a location where no further improvement can be made this is our minimum.

A flaw of this algorithm is that there is no way to determine whether the minimum found is a global minimum or just a local minimum [12]. Kruskal explains in [12] how repeating 'The Method of Steepest Descent' from multiple initial configurations will highlight if this problem occurs.

4.3 Non-Metric MDS

Non-metric MDS (NMDS) was first discovered by Kruskal (1964) [13]. It differs from MMDS in the fact that the input dissimilarities, δ_{ij} are not required to be metric distances. That is in particular they fail the triangle inequality. For example,

$$\delta_{ik} > \delta_{ij} + \delta_{jk}.$$

This makes it likely impossible for these dissimilarities to be well represented by euclidean distances in a spatial map. Therefore applying MMDS will result in a high stress configuration.

4.3.1 Kruskal's Approach

Kruskal's [13] approach of overcoming this was to focus on only preserving the rank order of the dissimilarities in the spatial map. That is if,

$$\delta_{12} < \delta_{23} < \delta_{13} \implies d_{12} < d_{23} < d_{13}.$$

Kruskal [13] proposed that for each dissimilarity, δ_{ij}, we can define a corresponding disparity, $\hat{d}_{ij}$, like so,

$$f(\delta_{ij}) = \hat{d}_{ij} \simeq d_{ij}.$$

Where f is some continuous, parametric and monotonic function therefore, preserving the rank order of the dissimilarities. NMDS consists of finding this function, f, then minimizing the following stress function in equation (4.5) using the disparities generated by f.

$$S = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j<i}^{n}\left(\hat{d}_{ij} - d_{ij}\right)^2}{\sum_{i=1}^{n}\sum_{j<i}^{n} d_{ij}^2}}. \qquad \text{[7].} \tag{4.5}$$

This is the same as the stress 1 function (4.3) however, we use the disparities rather than the dissimilarities. Kruskal [13] used a method called 'isotonic regression' other wise known as 'monotonic least squares regression' to find the best fucntion f.

4.3.2 Simple example of isotonic regression

Suppose we are provided with dissimilarity matrix Δ, and a random configuration of points representing these dissimilarities,

$$\Delta = \begin{bmatrix} 0 & 3 & 5 & 6 \\ 3 & 0 & 4 & 1 \\ 5 & 4 & 0 & 2 \\ 6 & 1 & 2 & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} 0 & 1 & 4 & 3 \\ 1 & 0 & 8 & 2 \\ 4 & 8 & 0 & 3 \\ 3 & 2 & 3 & 0 \end{bmatrix}$$

Clearly $4 = \delta_{32} > \delta_{34} + \delta_{42} = 3$ so, the triangle inequality doesn't hold for these dissimilarities. We can re-index these dissimilarities in increasing order,

$$\begin{array}{ccccccccccc} \delta_1 & < & \delta_2 & < & \delta_3 & < & \delta_4 & < & \delta_5 & < & \delta_6 \\ \| & & \| & & \| & & \| & & \| & & \| \\ \delta_{24} & < & \delta_{34} & < & \delta_{12} & < & \delta_{23} & < & \delta_{13} & < & \delta_{14} \\ \| & & \| & & \| & & \| & & \| & & \| \\ 1 & < & 2 & < & 3 & < & 4 & < & 5 & < & 6 \end{array}$$

Now we can re-index the corresponding distances of D in the same way,

$$\begin{array}{cccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \\ \| & \| & \| & \| & \| & \| \\ d_{24} & d_{34} & d_{12} & d_{23} & d_{13} & d_{14} \\ \| & \| & \| & \| & \| & \| \\ 2 & 3 & 1 & 8 & 4 & 3 \end{array}$$

Now lets plot the cumulative sum of our new d_i,

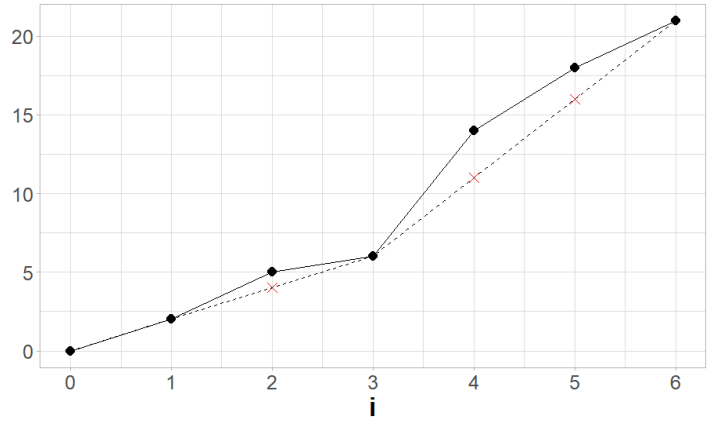


Figure 6: Plot showing the cumulative sum of d_i (solid line). Plot also contains the greatest convex minorant (dashed line)

Since this is a cumulative plot, the gradient of the line between connecting the black points is equal to the respective d_i. We can now take the greatest convex minorant (dashed line), Cox and Cox [7] described this well. Imagine holding a string, under the solid line, tight against the first and final point, that line is the greatest convex minorant [7]. Now, we can define the gradient between points on the dashed line as the disparities, $\hat{d}_i$. Therefore the disparities are,

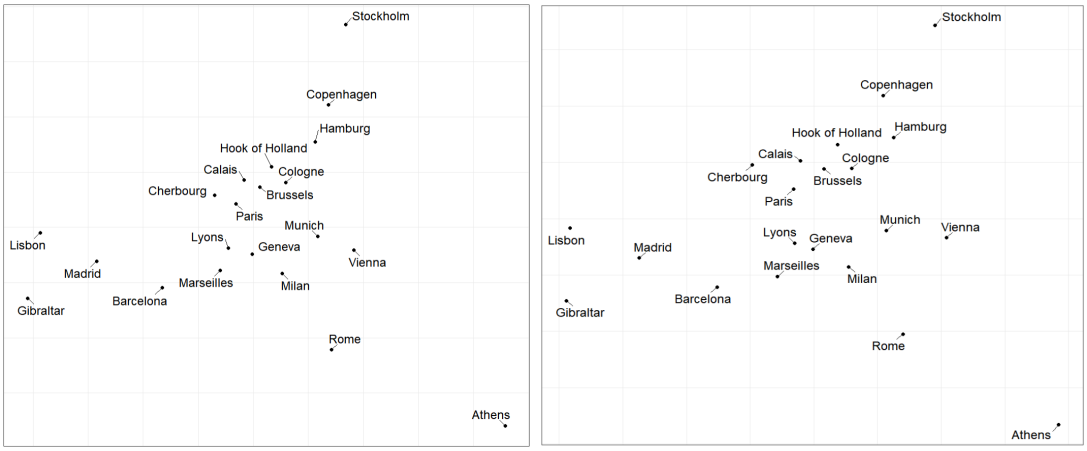
$$\begin{array}{cccccc} \hat{d}_1 & \hat{d}_2 & \hat{d}_3 & \hat{d}_4 & \hat{d}_5 & \hat{d}_6 \\ \| & \| & \| & \| & \| & \| \\ \hat{d}_{24} & \hat{d}_{34} & \hat{d}_{12} & \hat{d}_{23} & \hat{d}_{13} & \hat{d}_{14} \\ \| & \| & \| & \| & \| & \| \\ 2 & 2 & 2 & 5 & 5 & 5 \end{array}$$

Now inputing this into the stress equation (4.5) we get a value of 0.3941. Indicating that the configuration D is an extremely poor fit for the dissimilarities. However, using the disparities we found we can minimise this stress function by following Kruskal's [12] technique mentioned in the MMDS section. Repeating this process for many different initial configuration will find the minimal stress configuration. This is the best possible representation of the dissimilarity data, Δ.

4.4 Applying MDS to a dissimilarity data set

In this section we will apply two MMDS methods to a real data set. The data set I have chosen to use is the '`eurodist`' data set which is in built into '`R`'. This data set contains the road distances (in km) between 21 cities in Europe.

We will be using two different stress functions to find optimal configurations for the cities. In the first case we will use Kruskal's Stress 1 equation (4.3) and in the second case we will use Sammon's Stress function (4.4).



(a) Kruskal's Stress 1 equation (4.3) (b) Sammon's Stress function (4.4)

Figure 7: Two figures showing the optimal configurations of 21 European cities from the '`eurodist`' data set using two different stress functions.

Immediately we can see that both configurations in Figure 7 are very similar and both identify the outer cities like Stockholm, Athens and Lisbon very accurately. However, they differ slightly in the configuration of the cities in Central Europe. This is because, as we have previously mentioned, Sammon's stress function (4.4) gives larger weights to the smaller distances. In this case that is the distance between the Central European cities. Therefore, we can expect the configuration in Figure 7b contains a better representation of these smaller distances.

We can also compare the final stress values for these two configurations. A smaller stress indicates a better fit. The configuration in Figure 7a, has a stress of 7.506, therefore this fit is satisfactory however this can be improved. Sammon's stress function resulted in the configuration in Figure 7b, which has a stress of 0.00941. This is an extremely impressive fit and is close to a perfect representation of the original distances. In this case Sammon's stress function is a far superior choice of stress function.

5 Kernel PCA

Often, the purpose of PCA is to reveal the underlying insight about the relationships between data points. By creating the best possible lower dimensional representation of the data, it becomes

possible to apply many techniques that would be otherwise hindered by the 'Curse of Dimensionality'. For example both clustering and classification techniques, which help to identify groups of similar or dissimilar data points. However, as mentioned at the end of section 3, PCA makes the strong assumption about the linearity of relationships between variables. Therefore if the data exhibits non-linear relationships, PCA will reveal no insight into the true lower dimensional subspace that the data inhabits.

Kernel PCA (K-PCA) can be defined as an extension to PCA which allows the input data to exhibit non-linear relationships. K-PCA provides a technique that can separate input data which is not linearly separable. To achieve this, it makes use of kernel functions and a form of the popular trick called the 'kernel trick'.

In this chapter we will first discuss kernel methods introducing some important choices of kernel. Next, in section 5.2, we will explain how kernels can be used by defining the method of K-PCA. Finally we will apply K-PCA to a real data set and compare the results to the standard PCA method. The theory in this chapter builds upon work of Shawe-Taylor and Cristianini [14] and Marukatat [15].

5.1 Kernel Methods

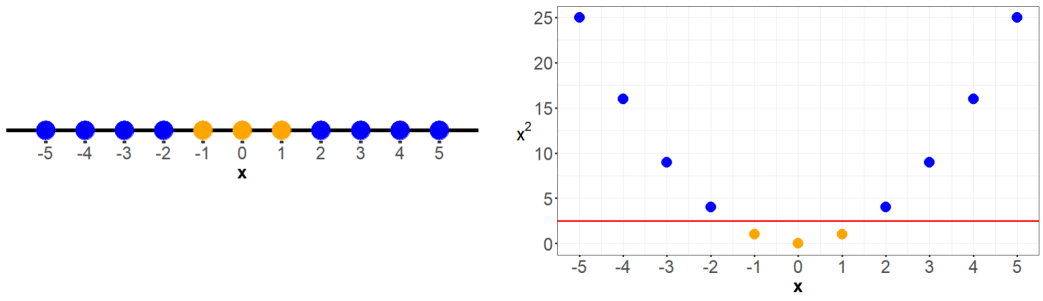
Kernel methods provide the ability to apply well known, familiar, linear techniques to data which contains non-linear relationships. Shawe-Taylor and Cristanini [14] explain how it is possible to map the data into a higher dimensional space such that linear techniques can be used. It may at first feel counterintuitive to increase the dimensionality of data set since throughout this report we have stressed the importance of doing the opposite. However, this increase in dimensionality is vital as it provides the ability to apply linear techniques. More specifically, after increasing the dimensionality we can apply PCA.

5.1.1 Feature spaces

We will refer to this higher dimensional space as the feature space, F. So for a given data set X we can define a map, Φ, such that each observation $\mathbf{x}_i$ is mapped to a higher dimensional point in a feature space,

$$\mathbf{x}_i \mapsto \Phi(\mathbf{x}_i), \tag{5.1}$$

where linear techniques can be applied to the mapped observations $\Phi(\mathbf{x}_i)$. The following figure 8 will be used to demonstrate a simple example of this idea.



(a) One dimensional space

(b) Two dimensional feature space

Figure 8: These plots are provided to show how a set of data points that are not linearly separable in 1-dimensional space can be mapped to a 2-dimensional feature space in which they are now linearly separable.

Consider Figure 8a, here we have data 11 data points on a one dimensional plane. 8 blue points and 3 orange points. These points are not linearly separable as there is no possible linear classification boundary which separates the blue points from the orange points. However, if we consider the map,

$$\Phi\colon \mathbb{R} \mapsto \mathbb{R}^2 \qquad \Phi(\mathbf{x}_i) = (\mathbf{x}_i, \mathbf{x}_i^2).$$

We can see from Figure 5.1 that in the higher dimensional feature space, the points are now linearly separable. This is an example of a linear technique which can only be used when data is mapped to a higher dimensional feature space.

This example uses a very simple map, Φ, and a 2-dimensional feature space, F, which is relatively low. In practice the dimensionality of F can be much greater than the dimension of the input data, F can perhaps even be of infinite dimension. Therefore it will be extremely hard, or in some cases impossible to compute the mapping explicitly. However, we can use a convenient way of bypassing this explicit calculation which is commonly referred to as 'The Kernel Trick'.

5.1.2 Kernel Functions

A kernel function, computes the inner product of two images under a given mapping, Φ,

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \,, \tag{5.2}$$

where $\langle \cdot, \cdot \rangle$ corresponds to the inner product of its two elements. Given a data set with n observations, the Gram matrix, G, is an $(n \times n)$ matrix with elements $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$. When using kernel functions, $G = K$ where K is the kernel matrix containing elements $k(\mathbf{x}_i, \mathbf{x}_j)$. In order for $k(\mathbf{x}, \mathbf{y})$ to be a valid kernel, K must be p.s.d. This guarantees that there exists a feature mapping Φ.

The linear kernel is the most simple kernel function it comes in the following form,

$$k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \,. \tag{5.3}$$

This is simply just the dot product of the two data points **x** and **y**. In this case the feature map is just the identity function, $\Phi(\mathbf{x}) = \mathbf{x}$. More useful kernels are those that are non-linear. An example of this is the polynomial kernel,

$$k(\mathbf{x}, \mathbf{y}) = \big(\langle \mathbf{x}, \mathbf{y} \rangle \big)^d = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \,. \qquad [14] \tag{5.4}$$

where d corresponds to the degree of polynomial, for example if $d = 2$, then this is also known as the quadratic kernel. The feature mapping, $\Phi(\mathbf{x})$, consists of every polynomial of polynomial of degree d using the entries of **x**. For example is $d = 2$ and $\mathbf{x} = (\mathrm{x}_1, \mathrm{x}_2)$ then,

$$\Phi(\mathbf{x}) = \big(\mathrm{x}_1^2, \sqrt{2}\mathrm{x}_1\mathrm{x}_2, \mathrm{x}_2^2\big). \tag{5.5}$$

The mapping in equation 5.5 contains every combination of elements of **x** that creates a polynomial of degree 2. The most commonly used kernel function is the Gaussian kernel or the Radial Basis Function kernel (RBF kernel).

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(- \frac{||\mathbf{x} - \mathbf{y}||^2}{2\sigma} \right) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \,. \qquad [14] \tag{5.6}$$

Here, σ is a parameter. The RBF kernel can be interpreted as a measure of similarity, with $k(\mathbf{x}, \mathbf{y})$ being larger when **x** and **y** are closer in the input space. The feature space of this kernel has infinite dimension so there is no explicit form of Φ. As mentioned, 'The Kernel Trick' can get around this issue.

5.1.3 'The Kernel Trick'

'The Kernel Trick' uses this and is described well by Marukatat [15]. The idea is to re-formulate a linear technique so that the input data only appears in the form of an inner product, in this case we can replace all the inner products with the corresponding kernel function. Therefore we are never required to explicitly calculate the coordinates of the mapped data as the necessary information is held within the inner products between data points.

5.2 Applying Kernels to PCA

To start K-PCA, we require a set input data, X, and a chosen kernel function, $k(\mathbf{x}, \mathbf{y})$. The idea is we will implicitly map the input data into a higher dimensional feature space using the the feature map, Φ, which corresponds to the chosen kernel function. Then in theory we would apply standard PCA to the mapped data in the feature space. However, using 'The Kernel Trick' we can do this implicitly using the kernel matrix, K.

5.2.1 Centering the data in the feature space

The first step for standard PCA is to center the data, however since Φ may not have an explicit form this can be difficult. We can define $\underline{\mu}$ to be the mean vector for the mapped data.

$$\underline{\mu} = \frac{1}{n}\sum_{i=1}^{n}\Phi(\mathbf{x}_i).$$

Therefore we can now define a new mapping of the input data using Φ which maps them to be centered in the feature space.

$$\widetilde{\Phi}(\mathbf{x}_i) = \Phi(\mathbf{x}_i) - \underline{\mu} = \Phi(\mathbf{x}_i) - \frac{1}{n}\sum_{i=1}^{n}\Phi(\mathbf{x}_i).$$

The corresponding kernel function for this feature mapping is [15],

$$\begin{aligned}
\widehat{k}(\mathbf{x}, \mathbf{y}) = \left\langle \widetilde{\Phi}(\mathbf{x}), \widetilde{\Phi}(\mathbf{y}) \right\rangle &= \left\langle \Phi(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n}\Phi(\mathbf{x}_i), \Phi(\mathbf{y}) - \frac{1}{n}\sum_{j=1}^{n}\Phi(\mathbf{y}_j) \right\rangle \\
&= \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle - \frac{1}{n}\sum_{j=1}^{n}\left\langle \Phi(\mathbf{x}), \Phi(\mathbf{y}_j) \right\rangle - \frac{1}{n}\sum_{i=1}^{n}\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{y}) \rangle \\
&\quad + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{y}_j) \right\rangle \\
&= k(\mathbf{x}, \mathbf{y}) - \frac{1}{n}\sum_{j=1}^{n}k(\mathbf{x}, \mathbf{y}_j) - \frac{1}{n}\sum_{i=1}^{n}k(\mathbf{x}_i, \mathbf{y}) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}k(\mathbf{x}_i, \mathbf{y}_j).
\end{aligned}$$

We will call $\widehat{k}(\mathbf{x}, \mathbf{y})$ the modified kernel function and we will use this instead. The kernel matrix, K, will also have elements corresponding to the modified kernel function.

5.2.2 Deriving the principal axis

Given an input data set, X, with rows corresponding to observation, $\mathbf{x}_i$, in the input space, let's define the matrix $\widetilde{X}$ which has rows consisting of the mapped observations, $\widetilde{\Phi}(\mathbf{x}_i)$, that are

centered in the feature space. Consider the kernel matrix, k,

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

where $\widehat{k}(\mathbf{x}_i, \mathbf{x}_j)$ is equal to the inner product between two mapped observations, $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$. Therefore K is the inner product matrix of these observations. Hence $K = \widetilde{X}\widetilde{X}^T$. We know that K is p.s.d. so has a spectral decomposition of the form,

$$\widetilde{X}\widetilde{X}^T = K = V\Lambda V^T \tag{5.7}$$

$$\implies KV = V\Lambda \tag{5.8}$$

We know from chapter 3 that the PCs of the mapped data set, $\widetilde{X}$, are in the directions of the eigenvectors of the covariance matrix, $\Sigma_{\widetilde{X}} = \widetilde{X}^T\widetilde{X}$. Consider,

$$\begin{aligned} \Sigma_{\widetilde{X}}\big(\widetilde{X}^T V\big) &= \widetilde{X}^T\widetilde{X}\widetilde{X}^T V \\ &= \widetilde{X}^T K V && \text{(using equation (5.7))} \\ &= \widetilde{X}^T \Lambda V && \text{(using equation (5.8))} \\ &= \big(\widetilde{X}^T V\big)\Lambda. && \text{(Since } \Lambda \text{ is diagonal)} \end{aligned}$$

So, $\widetilde{X}^T V$ is a matrix containing a set eigen vectors of $\Sigma_{\widetilde{X}}$. We must now normalise these vectors to find the orthonormal set of eigen vectors, U, which are our PCs.

$$||\widetilde{X}^T V|| = V^T\widetilde{X}\widetilde{X}^T V = V^T K V = \Lambda.$$

Therefore, $U = \Lambda^{-\frac{1}{2}}\widetilde{X}^T V$. Each principal component, $\mathbf{u}_j$, is of the form,

$$\mathbf{u}_j = \lambda_j^{-\frac{1}{2}} \sum_{i=1}^{n} (\mathbf{v}_j)_i \Phi(\mathbf{x}_i),$$

where $\mathbf{v}_j$ and λ_j are the eigenvectors and eigenvalues of K respectively.

5.2.3 Projecting a data point onto its principal axis

The projection of a given mapped observation, $\Phi(\mathbf{x})$, onto its j^{th} principal axis, $\mathbf{u}_j$ is simply just equal to the dot product $\langle \mathbf{u}_j, \Phi(\mathbf{x}) \rangle$.

$$\begin{aligned} \langle \mathbf{u}_j, \Phi(\mathbf{x}) \rangle = \left\langle \lambda_j^{-\frac{1}{2}} \sum_{i=1}^{n} (\mathbf{v}_j)_i \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \right\rangle &= \lambda_j^{-\frac{1}{2}} \sum_{i=1}^{n} (\mathbf{v}_j)_i \big\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}) \big\rangle \\ &= \lambda_j^{-\frac{1}{2}} \sum_{i=1}^{n} (\mathbf{v}_j)_i k(\mathbf{x}_i, \mathbf{x}). \end{aligned}$$

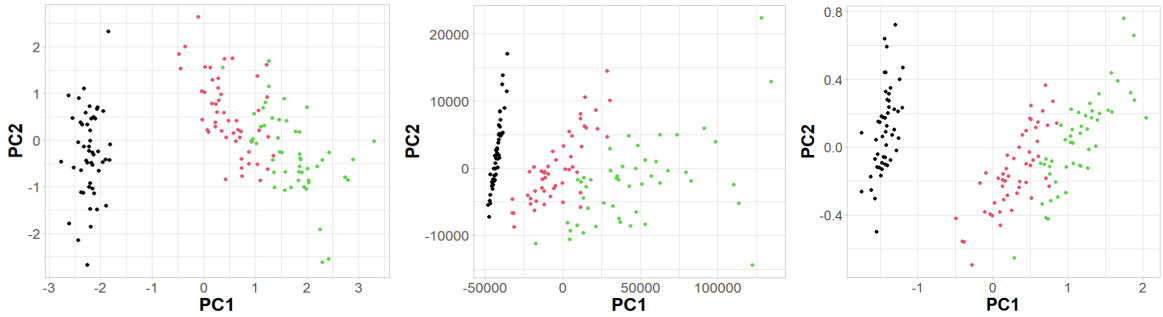
The final form that we have derived above is completely independent of the feature mapping, Φ. Therefore, we can use the formula above to find each observation $\mathbf{x}_i$ in terms of its PCs in the feature space, without ever explicitly mapping the observation to this feature space. We can create an m-dimensional K-PCA plot by taking only the projections onto the first m PCs.

5.3 An example of K-PCA

Now we will apply kernel PCA to a set of data points. I have chosen to use the in-built `R` data set called `iris`. The `iris` data set consists of a sample of 150 iris flowers. The four variables we have data about the flowers are all direct measurements.

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

There are 3 different species of iris flower, with 50 of each species in this data set. The goal is to use a dimension reduction technique to create a plot which separates these species into clusters. We will use the RBF kernel, the polynomial kernel with $d = 4$ and standard PCA.



(a) Standard PCA. (b) Polynomial Kernel (5.4). (c) RBF kernel (5.6).

Figure 9: Comparison of two choices of kernel function to the outcome of standard PCA when applied to the `iris` data set. Different colours of the points correspond to to the different species of iris flower, setosa (Black), versicolor (Red) and virginica (Green).

All three techniques in Figure 9 effectively show the difference between the species setosa from the both versicolor and virginica. However, in Figure 9a, when using standard PCA the overlap between the latter two species is broad. This can lead to misclassifications of future flower species if they are located in this region of overlap. Figure 9b and Figure 9c both reduce this region of overlap considerably helping to solve this potential issue.

6 Conclusion:

In this report we illustrated the profound importance of dimension reduction techniques when performing high dimensional data analysis. We introduced four very popular techniques namely, the singular value decomposition (SVD), principal component analysis (PCA), multidimensional scaling (MDS) and kernel principal component analysis (K-PCA). For each technique we provided a method for its implementation paired with a simple example of its application using real world data sets. Throughout the report we have clarified the relationships between these dimension reduction techniques and compared their effectiveness when dealing with data of varying forms. This has helped develop a deeper understanding of when each technique thrives and how they are limited.

Bibliography

[1] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[2] Dan Kalman. A singularly valuable decomposition: The svd of a matrix. *The College mathematics journal*, 27(1):2–23, 1996.

[3] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, second edition, 2022.

[4] I. T. Jolliffe. *Principal component analysis*. Springer series in statistics. Springer, New York, second edition, 2002.

[5] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[6] Joseph B Kruskal and Myron Wish. Basic concepts of multidimensional scaling. In *Multidimensional Scaling*, volume 11 of *Quantitative Applications in the Social Sciences*. SAGE Publications Inc, United States, 1978.

[7] Trevor Cox and Michael Cox. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2000.

[8] Gale Young and A S Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika.*, 3(1), 1938-03.

[9] Warren S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, 1952.

[10] J. C. Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966.

[11] Nasir Saeed, Haewoon Nam, Mian Imtiaz Ul Haq, and Dost Muhammad Saqib Bhatti. A survey on multidimensional scaling. *ACM computing surveys*, 51(3):1–25, 2019.

[12] J. B. Kruskal. Nonmetric multidimensional scaling: A numerical method. *Psychometrika*, 29(2):115–129, 1964.

[13] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[14] John. Shawe-Taylor and Nello. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, Cambridge, 2004.

[15] Sanparith Marukatat. Tutorial on pca and approximate pca and approximate kernel pca. *The Artificial intelligence review*, 56(6):5445–5477, 2023.

A R Code for plots

```
# Greyscale SVD example
library(ggplot2)

grey1 <- matrix(c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1,0,
                  0,1,0,1,0,0,1,0,0,1,0,0,0,0,0,1,0,
                  0,1,0,1,0,0,1,0,0,1,1,1,0,0,1,1,0,
                  0,1,0,1,0,1,1,0,0,0,0,1,0,0,0,1,0,
                  0,1,1,1,0,0,1,0,0,1,1,1,0,1,1,1,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),nrow = 17,byrow = F)

image(grey1, col=grey(seq(0, 1, length = 256)))
svd <- svd(grey1)
svd_d = as.data.frame(cbind(1:7,svd$d))
sing_val_sum = sum(svd$d)

p = ggplot(svd_d, aes(V1,V2))
p = p + geom_point(shape = "x",size = 4) + geom_line() +
    xlab('Component') + ylab('Singular Value') + ggtitle('Scree plot')
p + scale_x_continuous(breaks = seq(1, 7, by = 1)) +
    scale_y_continuous(breaks = seq(0,6,1))

svd_1 = svd(grey1, nu = 1, nv = 1)
u_1 = svd_1$u
v_1 = svd_1$v
svd_rank1_approx = round(svd_1$d[1] * u_1 %*% t(v_1),2)
image(svd_rank1_approx, col=grey(seq(0.1, 0.8, length = 256)))


svd_2 = svd(grey1, nu = 2, nv = 2)
u_2 = svd_2$u
v_2 = svd_2$v
svd_rank2_approx = round(u_2 %*% diag(svd_2$d[1:2]) %*% t(v_2),2)
image(svd_rank2_approx, col=grey(seq(0, 1, length = 256)))


svd_3 = svd(grey1, nu = 3, nv = 3)
u_3 = svd_3$u
v_3 = svd_3$v
svd_rank3_approx = round(u_3 %*% diag(svd_3$d[1:3]) %*% t(v_3),2)
image(svd_rank3_approx, col=grey(seq(0, 1, length = 256)))


svd_5 = svd(grey1, nu = 5, nv = 5)
u_5 = svd_5$u
v_5 = svd_5$v
svd_rank5_approx = round(u_5 %*% diag(svd_5$d[1:5]) %*% t(v_5),2)
image(svd_rank5_approx, col=grey(seq(0, 1, length = 256)))
```

```r
# PCA general idea demonstration.

library(ggplot2)
library(lme4)
library(ggrepel)

x = c(1,2,4)
y = c(1,4,4)

x1 = c(1,2,4,0.5766,2.635,3.789)
y1 = c(1,4,4,1.4941,3.259,4.247)

x2 = c(1,2,4,1.6,1.2,2.8)
y2 = c(1,4,4,2.2,2.4,1.6)

lines = c(0,1,2,0,1,2)

df <- data.frame(x = x1,y = y1,lines = lines)
df2 <- data.frame(x = x2, y = y2, lines = lines)


lm.fit = lm(y ~ x)

ggplot(data = df, aes(x, y,
                      label = c('x','y','z','x*','y*','z*'))) +
    geom_abline(intercept = lm.fit$coefficients[[1]],
                slope = lm.fit$coefficients[[2]],
                col = 'black',size = 0.8) +
    geom_line(aes(group = lines), linetype = 2, col = 'red',size = 0.5) +
    geom_point(pch = 19, size = 4,
               col = c('black','black','black','red','red','red')) +
    geom_text_repel(box.padding = 1.5, max.overlaps = Inf,size = 4,
                    segment.linetype = 1) +
    ylim(0,5) + xlim(0,5) +
    theme_minimal()

ggplot(data = df2, aes(x, y,
                       label = c('x','y','z','x*','y*','z*'))) +
    geom_abline(intercept = 3, slope = -0.5, col = 'black',size = 0.8) +
    geom_line(aes(group = lines), linetype = 2, col = 'red',size = 0.5) +
    geom_point(pch = 19, size = 4,
               col = c('black','black','black','red','red','red')) +
    geom_text_repel(box.padding = 1.5, max.overlaps = Inf,size = 4,
                    segment.linetype = 1) +
    ylim(0,5) + xlim(0,5) +
    theme_minimal()


#PCA implementation
```

```
library(dplyr)
library(ggplot2)
library(ggfortify)

coffee_df = read.csv("C:/Users/ciang/OneDrive - University of Leeds/Year 3/Modules/MA

coffee_numeric <- select_if(coffee_df,is.numeric)
coffee_numeric <- coffee_numeric[-c(1,2,3,10,11,12,14,15,16,17,18,19)]

pca <- prcomp(coffee_numeric, scale. = T)
components <- round(pca$rotation,2)
eigen_values <- pca$sdev**2
total_var <- sum(eigen_values)
variance <- eigen_values/total_var


qplot(c(1:7), variance) +
    geom_point(col = 'blue',size = 4) +
    geom_line(col = 'blue',size = 0.75,linetype = 2) +
    xlab('Principal component') + ylab('Variance Explained') +
    ylim(0,1) + ggtitle('Scree plot') +
    scale_x_discrete(limits = c('1','2','3','4','5','6','7')) +
    theme_light() +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))


autoplot(pca) + theme_light() + ggtitle('PCA plot') +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))

# Isotonic regression example

library(ggplot2)

d2 = c(0,2,3,1,8,4,3)
d = c(0,2,3,1,8,4,3)
i = c(0,1,2,3,4,5,6)

dcumsum <- c(cumsum(d))
line2 <- c(0,2,4,6,11,16,21)

df <- data.frame(cbind(i,dcumsum, line2))

ggplot(df,aes(i,dcumsum)) +
    geom_point(pch = 16,aes(i,dcumsum),size = 3)  +
    geom_point(mapping = aes(i,line2),
               pch = c(NA,NA,4,NA,4,4,NA),
```

```
                col = 'red',size = 3) +
    geom_line(aes(i,dcumsum)) +
    geom_line(aes(i,line2),linetype = 'dashed') +
    xlab('i') + ylab('') +
    scale_x_continuous(breaks = c(0,1,2,3,4,5,6)) +
    theme_light() +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))


# Eurodist MMDS implementation

library(dplyr)
library(ggplot2)
library(ggrepel)
library(MASS)


dist <- as.matrix(eurodist)
names = colnames(dist)

mds2 <- isoMDS(dist, k=2)
mds3 <- sammon(dist, k=2)

x2 = mds2$points[,1]
y2 = mds2$points[,2]

x3 = mds3$points[,1]
y3 = mds3$points[,2]


data2 <- as.data.frame(cbind(x2,y2))
data3 <- as.data.frame(cbind(x3,y3))


ggplot(data2,aes(x2,-y2,label = names))  + geom_point() +
    geom_text_repel(box.padding = 0.5,
                    max.overlaps = Inf, size = 5,
                    segment.linetype = 1) +
    theme_bw() +
    theme(legend.position="none",
          axis.title = element_blank(),
          axis.text = element_blank(),
          axis.ticks = element_blank(),)


# Demonstration of linearity of data in higher dimensions

x = c(-5,-4,-3,-2,-1,0,1,2,3,4,5)
```

```
y = c(25,16,9,4,1,0,1,4,9,16,25)
colour = c('blue','blue','blue','blue','orange','orange','orange',
           'blue','blue','blue','blue')

df <- data.frame(cbind(x,y))

ggplot(data.frame(x), aes(x=x, y=0)) +
  annotate("segment",x=-6,xend=6, y=0, yend=0, size=2) +
  geom_point(size = 10, col = colour)  +
  scale_x_continuous(breaks= x) +
  scale_y_continuous(limits = c(-1,1)) +
  theme(legend.position="none",
        rect = element_blank(),
        panel.grid = element_blank(),
        axis.title.y = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 20),
        axis.ticks.x = element_line(size = 2),
        axis.title.x=element_text(size=20,face="bold")
  ) +
  coord_fixed(ratio = 0.25)

ggplot(df,aes(x,y)) +
    geom_point(pch = 20, size = 8, col = colour) + theme_bw() +
    ylab(expression(x^2)) +
    theme(legend.position="none",
          axis.text.y = element_text(size = 20),
          axis.ticks.y = element_line(size = 1),
          axis.text.x = element_text(size = 20),
          axis.ticks.x = element_line(size = 1),
          axis.title.x=element_text(size=20,face="bold"),
          axis.title.y=element_text(angle = 0,size=20,face="bold",
                                    vjust = 0.5, hjust=1)) +
    scale_x_continuous(breaks= x) +
    scale_y_continuous(limits = c(0,25)) +
    geom_hline(yintercept = 2.5, col = 'red', pch = 'dashed', size = 1)

# KPCA implemenation on iris dataset

library(dplyr)
library(ggplot2)
library(ggfortify)

pca <- prcomp(iris[-5], scale. = T)
components <- round(pca$rotation,2)
eigen_values <- pca$sdev**2
total_var <- sum(eigen_values)
variance <- eigen_values/total_var
```

```
x = pca$x[,1]
y = pca$x[,2]

df <- data.frame(cbind(x,y,iris$Species))
ggplot(df,aes(x,y)) + geom_point(colour = as.integer(df$V3)) +
    theme_light()  +
    scale_x_continuous(breaks=c(-3,-2,-1,0,1,2,3)) +
    xlab('PC1') + ylab('PC2') +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))

# Polynomial kernel

kpc <- kpca(~.,data=iris[-5],kpar=list(degree=4),
            kernel="polydot",features=2)

x1 = kpc@rotated[,1]
y1 = kpc@rotated[,2]


df1<-data.frame(cbind(x1,y1,iris$Species))

ggplot(df1,aes(x1,y1)) + geom_point(colour = as.factor(df1$V3)) +
    theme_light()  +
    xlab('PC1') + ylab('PC2') +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))

# RBF kernel

kpc <- kpca(~.,data=iris[-5],kpar=list(sigma=0.001),
            kernel="rbfdot",features=2)

x1 = kpc@rotated[,1]
y1 = kpc@rotated[,2]

df1 <- data.frame(cbind(x1,y1,iris$Species))


ggplot(df1,aes(x1,y1)) +
    geom_point(colour = as.integer(df1$V3)) +
    theme_light()  +
    xlab('PC1') + ylab('PC2') +
    theme(plot.title = element_text(size = 30),
          axis.text=element_text(size=15),
          axis.title=element_text(size=20,face="bold"))
```

B Declarations of Academic Integrity

I am aware that the University defines plagiarism as presenting someone else's work, in whole or in part, as your own. Work means any intellectual output, and typically includes text, data, images, sound or performance.

I promise that in the attached submission I have not presented anyone else's work, in whole or in part, as my own and I have not colluded with others in the preparation of this work. Where I have taken advantage of the work of others, I have given full acknowledgement. I have not resubmitted my own work or part thereof without specific written permission to do so from the University staff concerned when any of this work has been or is being submitted for marks or credits even if in a different module or for a different qualification or completed prior to entry to the University. I have read and understood the University's published rules on plagiarism and also any more detailed rules specified at School or module level. I know that if I commit plagiarism I can be expelled from the University and that it is my responsibility to be aware of the University's regulations on plagiarism and their importance.

I re-confirm my consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to monitor breaches of regulations, to verify whether my work contains plagiarised material, and for quality assurance purposes.

I confirm that I have declared all mitigating circumstances that may be relevant to the assessment of this piece of work and that I wish to have taken into account. I am aware of the University's policy on mitigation and the School's procedures for the submission of statements and evidence of mitigation. I am aware of the penalties imposed for the late submission of coursework.

Signed: Cian Goodwin **Date:** 28/03/2025