Student Name: Mervan Kaya
Email: mckaya@kth.se

Student Name: Erik Hamberg
Email: erikhamb@kth.se

DD2434 Machine Learning Advanced
Assignment 2A

2A/1 Principal Component Analysis

2A.1 The Importance of Data-Centering

The importance of data-centering in PCA can be shown by examining the reconstruction error, which is obtained by the mean square error of the higher dimensional data and the data generated by a latent variable in a lower dimensional space.

This answer will be based on the work by Miranda, Le Borgne and Bontempi. [7]

We can define the reconstruction error as:

$$\mathcal{J}_q(\tilde{x}_k) = \frac{1}{N}\sum_{k=1}^{N}\|\mathbf{x}_k - \tilde{\mathbf{x}}_k\|^2 \tag{1}$$

Our goal is to find orthonormal vectors that through a linear combination can produce a data point in a lower dimension that approximates the higher dimensional point with error shown in 1. Thus, the new data point $\tilde{\mathbf{x}}_k$ will take the following form:

$$\tilde{\mathbf{x}}_k = \sum_{i=1}^{q}(\mathbf{u}_i^T\mathbf{x}_k)\mathbf{u}_i \tag{2}$$

This approximation assumes that the origin of the orthonormal vector u_i is the same as the as that of the system it is defined in. However, we can reformulate 2 such that the new representation has some general origin $\boldsymbol{\theta}$. $\tilde{x}_k$ will take the following form:

$$\tilde{\mathbf{x}}_k = \boldsymbol{\theta} + \sum_{i=1}^{q}(\mathbf{u}_i^T(\mathbf{x}_k - \boldsymbol{\theta}))\mathbf{u}_i \tag{3}$$

By applying this new representation of the lower dimensional data to the error function in 1, we can define the problem as being a minimization of the error with $\tilde{x}_k$ being represented as 3. It is important to note that this minimization will be under the important constraint that $u_i^T u_i = 1$. This will be discussed in further detail in problem 2A.4.

Scaling this problem up to further dimensions, the lower dimensional approximation will instead be the combination between a vector $\mathbf{U}_K = \{\mathbf{u}_1, \cdots, \mathbf{u}_K\}$, and the displaced data point $\mathbf{x}_k - \boldsymbol{\theta}$ such that $\tilde{\mathbf{x}}_k$ becomes:

$$\tilde{\mathbf{x}}_k = \boldsymbol{\theta} + \mathbf{U}\mathbf{U}^T(\mathbf{x}_k - \boldsymbol{\theta}) \tag{4}$$

from 3. The displacement of $\tilde{\mathbf{x}}_k$ from $\mathbf{x}_k$ can be represented using a vector from $\tilde{\mathbf{x}}_k$ to $\mathbf{x}_k$. It can be found by taking the difference between the two data point. This will lead to the following displacement vector:

$$\mathbf{x}_k - \tilde{\mathbf{x}}_k = (\mathbf{x}_k - \boldsymbol{\theta}) - \mathbf{U}\mathbf{U}^T(\mathbf{x}_k - \boldsymbol{\theta}). \tag{5}$$

Using $\tilde{\mathbf{x}}_k = \mathbf{x}_k - \boldsymbol{\theta}$, the displacement vector can be described as following:

$$\mathbf{x}_k - \tilde{\mathbf{x}}_k = \mathbf{B}\tilde{\mathbf{x}}_k \tag{6}$$

where $\mathbf{B} = \mathbf{I} - \mathbf{U}\mathbf{U}^T$.

With this, we can start to try to minimize the error as we now have an expression for the displacement of the generated data and the true data, which can be seen in 1.

First, we need to use another form of the error which will allow us to perform the needed operations to find the minimum. This form is:

$$\mathcal{J}_q(\tilde{x}_k) = \frac{1}{N}\sum_{k=1}^{N}(\mathbf{x}_k - \tilde{\mathbf{x}}_k)^T(\mathbf{x}_k - \tilde{\mathbf{x}}_k) \tag{7}$$

By applying our definition of the displacement vector to our new form of the error, we gain the following expression:

$$\mathcal{J}_q(\mathbf{B}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N}\tilde{\mathbf{x}}_k^T\mathbf{B}^T\mathbf{B}\tilde{\mathbf{x}}_k \tag{8}$$

which can be reduced further by exploiting the definition of $\mathbf{B}$ and the that $\mathbf{B}$ is an orthogonal projector such that $\mathbf{B} = \mathbf{B}^2 = \mathbf{B}^T$. Using this reduces the error to:

$$\mathcal{J}_q(\mathbf{B}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N}\tilde{\mathbf{x}}_k^T\mathbf{B}\tilde{\mathbf{x}}_k \tag{9}$$

Applying that $\tilde{\mathbf{x}}_k = \mathbf{x}_k - \boldsymbol{\theta}$ and expanding $\mathcal{J}_q$ results in:

$$\mathcal{J}_q(\mathbf{B}, \boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N}\left(\mathbf{x}_k^T\mathbf{B}\mathbf{x}_k - 2\boldsymbol{\theta}^T\mathbf{B}\mathbf{x}_k + \boldsymbol{\theta}^T\mathbf{B}\boldsymbol{\theta}\right) \tag{10}$$

Since we want to find the $\boldsymbol{\theta}$ which minimizes the error, we need to take the derivate of the error w.r.t. $\boldsymbol{\theta}$ and set it to zero. This will result in:

$$\frac{\partial \mathcal{J}_q}{\partial \boldsymbol{\theta}} = -2\mathbf{B}\left(\frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k - \boldsymbol{\theta}\right) \tag{11}$$

Setting this to zero will result in $\boldsymbol{\theta}$ becoming the following:

$$\boldsymbol{\theta} = \frac{1}{N}\sum_{k=1}^{N}\mathbf{x}_k = \boldsymbol{\mu} \tag{12}$$

From all of this, we can now conclude that the displacement which minimizes the error is simply the subtraction of the mean from the data point. Thus, we have shown the importance of data-centering in PCA, which is the process of subtracting the mean from the data.

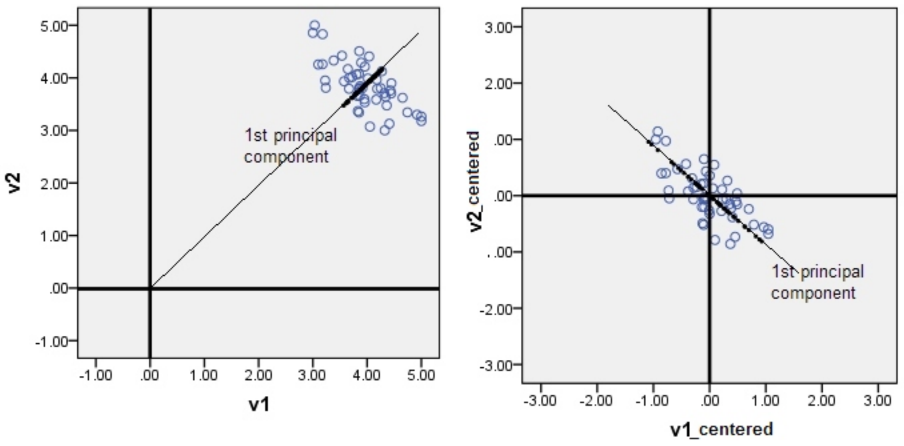

Figure 1: First principal components of centered and non-centered data [2]

Effects of not Centering Data One of the key effects that centering the data does, is that it places the data around the origin. This is important, in order to guarantee that the first principal component will pierce the data along the main direction of the origin. This can be shown by the plots in 1 where the non-centered and centered cases are shown. It is clear from the plots that although the principal component pierces the clouds of data, the non-centered data principal component does not pierce it along the main direction of variance. If this is the case when constructing a PCA model, the results will be statistically incorrect.

2A.2 Is a Single SVD Operation Sufficient?

Finding the principal components of the variables (i.e., the columns of the data matrix) by using PCA on the rows of the data matrix is equal to locating the right-singular vectors of V. The squares of the singular values in are used to get the appropriate eigenvalues for these vectors, which are the eigenvectors of the covariance matrix of the variables and the corresponding eigenvalues are given by the squares of the singular values in Σ.

Finding the principal components of observations (i.e., the rows of the data matrix) by doing PCA on the columns of the data matrix is equal to locating the left-singular vectors of U. These are the eigenvectors of the observations' covariance matrix, and the squares of the singular values in Σ provide the corresponding eigenvalues.

We can show this with an example shown in the SVD tutorial by Prof. Widge of MIT. [8]

In the example, consider a rectangular matrix $\mathbf{A}$ of form $n \times p$ where the n rows represents the genes and the p columns represent the experimental conditions. SVD states that:

$$\mathbf{A}_{n\times p} = \mathbf{U}_{n\times n}\mathbf{S}_{n\times p}\mathbf{V}^T_{p\times p} \tag{13}$$

where the $\mathbf{U}$ vector represents the left-singular values corresponding to the gene coefficients, and the $\mathbf{V}^T$ vector represents the right-singular values corresponding to the experimental conditions. The calculation of SVD involves determining the eigenvalues and eigenvectors of both $\mathbf{AA}^T$ and $\mathbf{A}^T\mathbf{A}$, where the eigenvectors of $\mathbf{A}^T\mathbf{A}$ form the columns of matrix $\mathbf{V}$, and the eigenvectors of $\mathbf{AA}^T$ make up the columns of matrix $\mathbf{U}$.

Consider the data:

$$\mathbf{A} = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Finding the left-singular values for $\mathbf{U}$ is done as follows:

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & 0 & 0 \\ 1 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 20 & 14 & 0 & 0 \\ 14 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{14}$$

and the vector $\mathbf{U}$ with the eigenvalues would be:

$$\mathbf{U} = \begin{bmatrix} 0.82 & -0.58 & 0 & 0 \\ 0.58 & 0.82 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{15}$$

We can now perform the same procedure with for the right-singular values using $\mathbf{A}^T\mathbf{A}$ instead.

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 2 & 4 & 0 & 0 \\ 1 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 \\ 1 & 3 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{16}$$

Finding the eigenvalues for this will result in the following matrix:

$$\mathbf{V} = \begin{bmatrix} 0.40 & -0.91 \\ 0.91 & 0.40 \end{bmatrix} \tag{17}$$

and S is the square roots of the eigenvalues from $\mathbf{A}\mathbf{A}^T$ or $\mathbf{A}^T\mathbf{A}$.

$$\mathbf{S} = \begin{bmatrix} 5.47 & 0 \\ 0 & 0.37 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{18}$$

We have shown that a single SVD operation is sufficient to perform PCA on both the rows and the columns, as the left-singular vector $\mathbf{U}$ represents the rows and the right-singular vector $\mathbf{V}$ represents the columns.

2A.3 Why is the Pseudo-Inverse a Good Choice to Obtain the Inverse Mapping of a Linear Map

A matrix is not invertible if the matrix is fulfills one or more of the following:

1. Lower rank than size. E.g. A 3×3 matrix with < 3 rank is non-invertible.
2. Non-Square.
3. Square matrix with linearly dependent rows or columns, or is not full rank.

The Moore-Penrose, or commonly known as the Pseudo-Inverse, allows for generalization of the matrix inverse for cases when the matrix is invertible.

The connection between the Pseudo-Inverse and PCA is that in PCA, we wish to create a mapping between a high dimensional space $\mathbb{R}^d$ and a lower dimensional space $\mathbb{R}^k$ where $k << d$. In PCA, this mapping is done using a linear mapping where each data point $\mathbf{y} = (y_1, y_2, \cdots, y_d)^T \in \mathbb{R}^d$ is mapped to a latent vector $\mathbf{x} = (x_1, x_2, \cdots, x_k)^T \in \mathbb{R}^k$ by using a linear transformation $\mathbf{U} \in \mathbb{R}^{d\times k}$ such that $\mathbf{y} = \mathbf{U}\mathbf{x}$. The linear transformation $\mathbf{U}$ allows for decoding (dec) from $\mathbb{R}^k$ to $\mathbb{R}^d$. Inversely, for the mapping (cod) between $\mathbb{R}^d$ to, $\mathbb{R}^k$ the Pseudo-Inverse is used.

The Pseudo-Inverse is a good choice, as it displays some nice properties we can use together with some other properties of the orthonormal linear transformation matrix $\mathbf{U}$. As $\mathbf{U}$ is an orthonormal matrix, it displays (not necessarily) the property that $\mathbf{U}\mathbf{U}^T = \mathbf{I}$. Together with the Pseudo-Inverse property that $\mathbf{U}^+ = (\mathbf{U}\mathbf{U}^T)^{-1}\mathbf{U}^T$, we can identify that the property will simplify into just the transpose as the terms in the parentheses will become the identity matrix. Therefore, the Pseudo-Inverse of the linear transformation matrix is simply the transpose of the matrix. This, together with the property that the Pseudo-Inverse allows for matrix inverse even when the matrix is not square, makes the Pseudo-Inverse a good for this problem.

2A.4 Derivation of PCA Using Variance Maximization Criterion in K dimensions

The derivation of PCA using maximum variance is explained in Bishop section 12.1.1 for a projection onto a 1-dimensional space. Exercise 12.1 extends the 1-dimensional case further into K-dimensions [4]. This answer will be based on the section in Bishop and the exercise. Furthermore, Prof. Erik Bekkers has made a brilliant YouTube video demonstrating this problem and explaining it very well. Parts of this solution will also be based on his work in the video [3].

The 1-Dimensional Case

We begin with the 1-dimensional case. We wish to project our data onto a lower dimensional space whilst maximizing the variance of the data. We can define a direction of the low dimensional space using a D-dimensional vector, $\mathbf{u}_1$ for which the variance of the data is maximized. Consider the data in 2, we can identify that the direction of the longer vector shown will be the direction of the highest variance, whilst the direction of the smaller vector will have less. We therefore wish to choose the direction of highest variance for our $\mathbf{u}_1$ vector. Projection onto the direction of the highest variance vector will result in what can be seen in 3.

We constrain $\mathbf{u}_1$ to preserve generality. A vector of any size may run into that the solution will be found where the size of the vector $\mathbf{u}_1$ is infinitely large, as we will see further on.

Before deriving the projection, it is to be noted that the mean of the data and the covariance of the data are the following respectively:

1. $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$
2. $\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$

The projection will give the scalar $\mathbf{u}_1^T\mathbf{x}_n$ and the mean projection will be $\mathbf{u}_1^T\bar{\mathbf{x}}$. This will result in the variance of the projection to be:

$$\frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\bar{\mathbf{x}})^2 \tag{19}$$

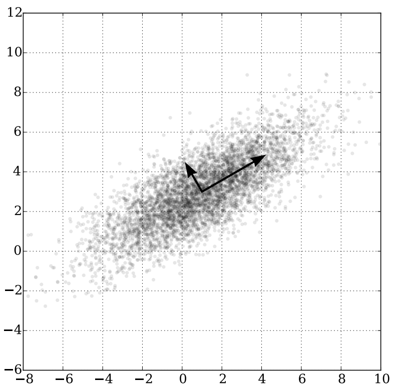

Figure 2: Multivariate Gaussian distribution centered at (1,3) [1]

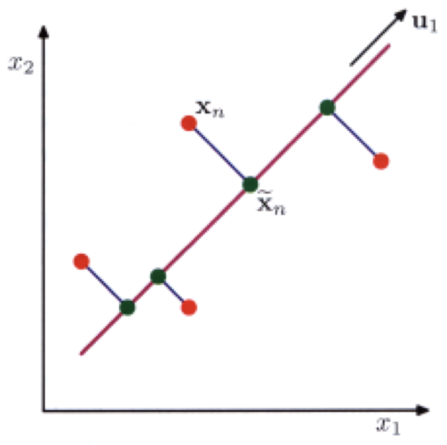

Figure 3: Projection onto a 1-Dimensional line. Figure 12.2 Bishop [4]

Grouping together the terms w.r.t $\mathbf{u}_1$ and expanding the square results in the following:

$$= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}}))^2 \tag{20}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_1^T(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T\mathbf{u}_1 \tag{21}$$

Since the $\mathbf{u}_1^T$ and $\mathbf{u}_1$ does not depend on n, they can be moved out of the sum. This will result in the following:

$$= \mathbf{u}_1^T\left(\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T\right)\mathbf{u}_1 \tag{22}$$

We can recognize that the sum in the parentheses is simply the covariance of the data, and thus the variance of the projected data becomes:

$$= \mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 \tag{23}$$

Now that we have a term for our variance of the projected data, we can begin to maximize it w.r.t the constraint that $\mathbf{u}_1^T\mathbf{u}_1 = 1$. To maximize the variance using the constraint, we introduce a Lagrange multiplier λ_1 and then construct the following unconstrained maximization (Bishop Eq. 12.4):

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 + \lambda_1(1 - \mathbf{u}_1^T\mathbf{u}_1) \tag{24}$$

As we want to maximize this, we take the derivative w.r.t $\mathbf{u}_1$ and set it equal to zero. This will result in the following:

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1 \tag{25}$$

This tells us that $\mathbf{u}_1$ must be an eigenvector of $\mathbf{S}$. Further, if we multiply each side with $\mathbf{u}_1^T$ and make use of the property that $\mathbf{u}_1^T\mathbf{u}_1 = 1$, we get the following:

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1 \tag{26}$$

From this, we can conclude that the largest variance will be found when we set the vector $\mathbf{u}_1$ to be equal to the eigenvector with the largest eigenvalue λ_1. This eigenvector is called the first *principal component.*

The K-Dimensional Case

We can easily extend the 1-dimensional case and consider projections onto K-dimensions.

For the K-dimensional case, we will instead have an K-dimensional orthogonal vector, $\mathbf{U}_K$ containing the eigenvectors with the largest eigenvalues, and is defined as following:

$$\mathbf{U}_K = \{\mathbf{u}_1, \cdots, \mathbf{u}_K\} \tag{27}$$

where $\mathbf{U}_K \in \mathbb{R}^{D\times K}$. The projections will take the form of the following:

$$\mathbf{z} = \mathbf{U}_K^T(\mathbf{x} - \bar{\mathbf{x}}) \tag{28}$$

Where the mean is subtracted to center the data. The necessity for this step is explained in question 2A.1 in the report.

The projected vector $\mathbf{z}$ will take the following form:

$$\mathbf{z} = \begin{bmatrix} \mathbf{u}_1^T \mathbf{x}_1 \\ \mathbf{u}_2^T \mathbf{x}_2 \\ \vdots \\ \mathbf{u}_K^T \mathbf{x}_K \end{bmatrix}$$

From the eigenvectors, we will also have K amount of eigenvalues arranged in size such that $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_K$. The total variance of the projected data will be the following:

$$tr[Cov(\mathbf{z})] = \sum_{j=1}^{K} \lambda_j \tag{29}$$

The result can be derived in further detail as follows:

Since $\mathbf{S}$ is a symmetric, positive, semi-definite matrix, it can be decomposed using eigenvalue-decomposition (SVD is equivalent for these types of matrices) into the following:

$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{30}$$

where $\mathbf{\Lambda}$ is a diagonal matrix containing all the λ_1 to λ_K eigenvalues. What this decomposition does is that since the eigenvectors are orthonormal, the data is being rotated as there is a change of basis. This basis change is done by multiplying with the $\mathbf{U}^T$ vector which will change the data so that the data is defined by the eigenvectors instead. The variance of this changed basis form data will now instead be represented by the diagonal eigenvalue vector. This is inline with the result we got for the one dimensional case in Eq. 8, where the variance was defined by the eigenvalue of the projection. By multiplying with $\mathbf{U}$ and $\mathbf{U}^T$ on both sides, we will end up with the same expression as Eq. 8, just with matrices as the dimension is increased.

The total variance is given by:

$$tr(\mathbf{S}) = tr(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T) \tag{31}$$

$$= tr(\mathbf{\Lambda}) = \sum_{j=1}^{K} \lambda_j \tag{32}$$

as

$$tr(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T) = tr(\mathbf{U}\mathbf{U}^T\mathbf{\Lambda}) = tr(\mathbf{\Lambda}) \tag{33}$$

where $\mathbf{U}\mathbf{U}^T = \mathbf{I}$, as $\mathbf{U}$ and $\mathbf{U}^T$ are orthonormal vectors.

PCA Using Minimal Reconstruction Error

In question 2A.1, we used the reconstruction error to show that data-centering leads to the smallest error. We can use the same derivations to show that minimizing the reconstruction error leads to the same conclusion as maximizing variance, which is also shown in the report by Miranda, Le Borgne and Bontempi. [7]

We begin by using 9 and substituting back $\mathbf{B}$ for $\mathbf{I} - \mathbf{U}\mathbf{U}^T$. This will yield:

$$\mathcal{J}_q(\mathbf{U}) = \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{x}}_k^T(\mathbf{I} - \mathbf{U}\mathbf{U}^T)\tilde{\mathbf{x}}_k \tag{34}$$

multiplying in the $\tilde{\mathbf{x}}_k$ terms and separating the sums will yield:

$$\mathcal{J}_q(\mathbf{U}) = \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{x}}_k^T\tilde{\mathbf{x}}_k - \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{x}}_k^T\mathbf{U}\mathbf{U}^T\tilde{\mathbf{x}}_k \tag{35}$$

By identifying that the sample covariance conditioned on $\boldsymbol{\theta} = \boldsymbol{\mu}$ is given by:

$$\mathbf{S} = \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{x}}_k\tilde{\mathbf{x}}_k^T \tag{36}$$

and that the trace of the covariance matrix is given by:

$$\mathbf{S} = \frac{1}{N}\sum_{k=1}^{N} \tilde{\mathbf{x}}_k^T\tilde{\mathbf{x}}_k \tag{37}$$

allows us to rewrite the error in 35 as the following:

$$\mathcal{J}_q(\mathbf{U}) = tr(\mathbf{S}) - \sum_{i=1}^{q} \mathbf{u}_i^T\mathbf{S}\mathbf{u}_i \tag{38}$$

where we use the definition of $\mathbf{U}$ in the sum. We can now identify that minimizing the error w.r.t. orthonormal basis vectors $\mathbf{u}_i$ will be to maximize the sum in 38 as the trace of the covariance is not dependent on the orthonormal basis vectors. We can immediately recognize that this is the exact same case as we have in the variance maximization criterion method in Eq. 23 and on. We have thus shown that minimizing the reconstruction error converges into the same method as maximizing the variance, as both methods will be based on maximizing the sum $\sum_{i=1}^{q} \mathbf{u}_i^T\mathbf{S}\mathbf{u}_i$.

2A/2

2A.5 Qualitative comparison between PCA and the Johnsson-Lindenstrauss Random Projections

Principal Component Analysis (PCA) and the Johnson-Lindenstrauss random-projections methods are both linear dimensionality reduction techniques, but they differ in terms of their properties and intended use cases.

Projection error: PCA aims to minimize the projection error, which is the difference between the original high-dimensional data and the projected low-dimensional data. In other words, it tries to preserve as much of the original data's variance as possible in the lower-dimensional space.[6] On the other hand, the Johnson-Lindenstrauss random-projections method tries to preserve the pairwise distances between the points in the data set, and the projection error can be arbitrarily high. [5]

Computational efficiency: PCA is generally considered more computationally expensive than the Johnson-Lindenstrauss random-projections method. PCA involves computing the eigenvectors and eigenvalues of the data's covariance matrix, which can be computationally intensive for large data sets. The Johnson-Lindenstrauss random-projections method is computationally more efficient as it only requires computing a matrix-vector multiplication to project the data.[5]

Target use cases: PCA is useful in situations where multi-colinearity can exist between different dimensions. It can also be used in situations where the input dimensions are high and for denoising or compression. It can make for nice data visualisation, when reducing the dimensions to 2. [6] Random Projections can be used for high dimensional data as well, especially in cases where the amount of data is very large, as the computational efficiency is higher.[5]

In summary, PCA is a more powerful and accurate dimensionality reduction technique when dealing with the structure of the data while the Johnson-Lindenstrauss random-projections method is mainly used in high-dimensional data problems with computational efficiency in mind, and the projection error can be arbitrarily high.

2A/3 Programming task - MDS

2A.6 Data Collection

We use the Google Places API[1] to get the coordinates of different cities. Then we use the GeoPy Python package to calculate the geodesic distances between the cities.[2]

```python
def get_geodesic_distance(places: list):

    # Get the latitude and longitude of each place.
    positions = []
    for place in places:
        url = f"https://maps.googleapis.com/maps/api/place/details/json?place_id={place}&
            fields=geometry&key={API_KEY}"
        payload = {}
        headers = {}
        response = requests.request("GET", url, headers=headers, data=payload)
        res = response.json()
        location = res["result"]["geometry"]["location"]
        positions.append((location["lat"], location["lng"]))

    # Get the geodesic distance between the points in a distance matrix
    d_matrix = np.zeros((len(places), len(places)))
    n = len(places)

    for i, x in enumerate(positions):
        for j, y in enumerate(positions):
            if i > j:
                d_matrix[i, j] = d_matrix[n - i - 1, n - j - 1] = geodesic(x, y).
                    kilometers

    return d_matrix
```

We decided to include half of the worlds capitals, as well as a couple more cities in Australia and the United States of America, to have more points in those regions. The full list of cities is included in the appendix.

2A.7 Classical MDS

We implemented the steps of the MDS like this (steps taken from Wikipedia [9]):

1. Apply double centering: $B = -\frac{1}{2}CD^{(2)}C$ using the centering matrix $C = I - \frac{1}{n}J_n$, where n is the number of objects, I is the $n \times n$ identity matrix, and J_n is an $n \times n$ matrix of all ones.

```python
rows, cols = d_matrix.shape
J = np.eye(rows) - np.ones((rows, rows)) / rows
B = -J.dot(d_matrix ** 2).dot(J) / 2
```

2. Determine the m largest eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ and corresponding eigenvectors $e_1, e_2, ..., e_m$ of B (where m is the number of dimensions desired for the output).

```python
# eigenvalue decomposition
eigenvalues, eigenvectors = np.linalg.eigh(B)
```

[1]https://developers.google.com/maps/documentation/places/web-service/overview
[2]https://geopy.readthedocs.io/en/stable/index.html?highlight=geodesic#geopy.distance.geodesic

```

# Sort eigenvalues and eigenvectors in descending order
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Keep top m eigenvalues and eigenvectors
top_k_eigenvalues = eigenvalues[:m]
top_k_eigenvectors = eigenvectors[:, :m]
```

3. Now, $X = E_m \lambda_m^{1/2}$, where E_m is the matrix of m eigenvectors and λ_m is the diagonal matrix of m eigenvalues of B

```
# Get square root of top m eigenvalues
sqrt_eigenvalues = np.sqrt(np.diag(top_k_eigenvalues))

# Get final representation
X = top_k_eigenvectors.dot(sqrt_eigenvalues)
```

To plot the plane the different places are in, we use these lines of code:

```
# Plot the resulting points
plt.figure(figsize=(16, 12))
plt.scatter(-X.T[0], X.T[1])
for i, txt in enumerate(place_ids.keys()):
    if i % 3 == 0:
        plt.annotate(txt, (-X.T[0, i], X.T[1, i]))
plt.show()
```

This produces the following image:

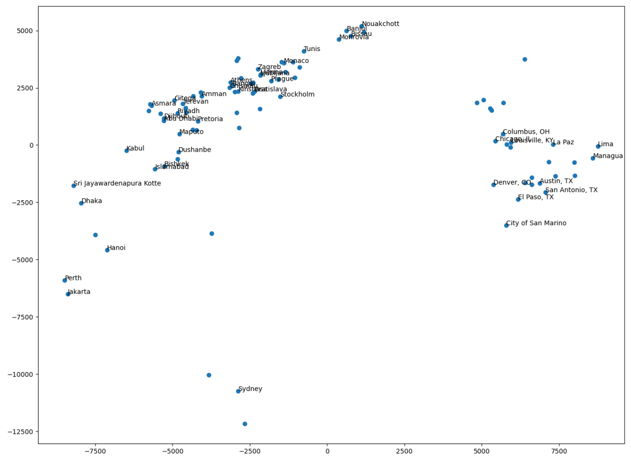

Figure 4: Plot of 92 cities as mapped by the Classical MDS reconstruction, with a half of the names added. Other versions of the plot available in the appendix.

Two different clusters are separated in the plot, kind of resembling the western and eastern hemispheres. Why this is is likely because this maintains a distance between the two different hemispheres well. There is no semblance of the structure of the continents as displayed on a Mercator map projection of the world. We believe this is because the usage of a Mercator map projection has the implicit choice to center the map on Europe, with distances distorted to make for a clean projection on paper. Many of the places are placed close to their physical neighbors, with some outliers, like San Marino City, being placed close to cities in Texas.

References

[1] N. . Multivariate gaussian distribution centered at (1,3), 02 2016.

[2] t. . How does centering the data get rid of the intercept in regression and pca?, 02 2012.

[3] E. Bekkers. 10.2 principal component analysis: Minimal reconstruction error (uva - machine learning 1 - 2020), 11 2020.

[4] C. Bishop. *Pattern recognition and machine learning.* Springer Verlag, 2006.

[5] L. Jianan. Random projection in python, Jan 2022.

[6] S. Loukas. Pca clearly explained - when, why, how to use it and feature importance: A guide in python, Oct 2021.

[7] A. A. Miranda, Y.-A. Le Borgne, and G. Bontempi. New routes from minimal approximation error to principal components. *Neural Processing Letters*, 27:197–207, 01 2008.

[8] A. S. Widge. Singular value decomposition (svd) tutorial, 09 2002.

[9] Wikipedia. Multidimensional scaling, 12 2022.

Appendix

Cities in A2/3

Sydney
Melbourne
Perth
Amsterdam
Athens
Berlin
Bratislava
Bucharest
Chișinău
Helsinki
Ljubljana
Luxembourg City
Monaco
Paris
Prague
Riga
City of San Marino
Skopje
Stockholm
Tirana
Vienna
Warsaw
Zagreb
New York, NY
Chicago, IL
Philadelphia, PA
San Antonio, TX
Dallas, TX
Austin, TX
Indianapolis, IN
Columbus, OH
Charlotte, NC
El Paso, TX
Boston, MA
Denver, CO
Nashville, TN
Louisville, KY
Oklahoma, OK
Kabul
Luanda
Yerevan
Baku
Dhaka
Porto Novo
La Paz
Brasilia
Gitega
Yaounde
Bangui
Santiago
Kinshasa
San Jose, Costa Rica
Djibouti
San Salvador
Asmara
Addis Ababa
Banjul
Guatemala City
Bissau
Tegucigalpa
Jakarta
Baghdad
Amman
Nairobi
Bishkek
Beirut
Monrovia
Lilongwe
Nouakchott
Ulaanbaatar
Maputo
Windhoek
Managua
Abuja
Islamabad
Port Moresby
Lima
Kigali
Riyadh
Freetown
Pretoria
Cape Town
Sri Jayawardenapura Kotte
Paramaribo
Dushanbe
Bangkok
Tunis
Ashgabat
Abu Dhabi
Tashkent
Hanoi
Lusaka

Appendix

Plots in A2/3

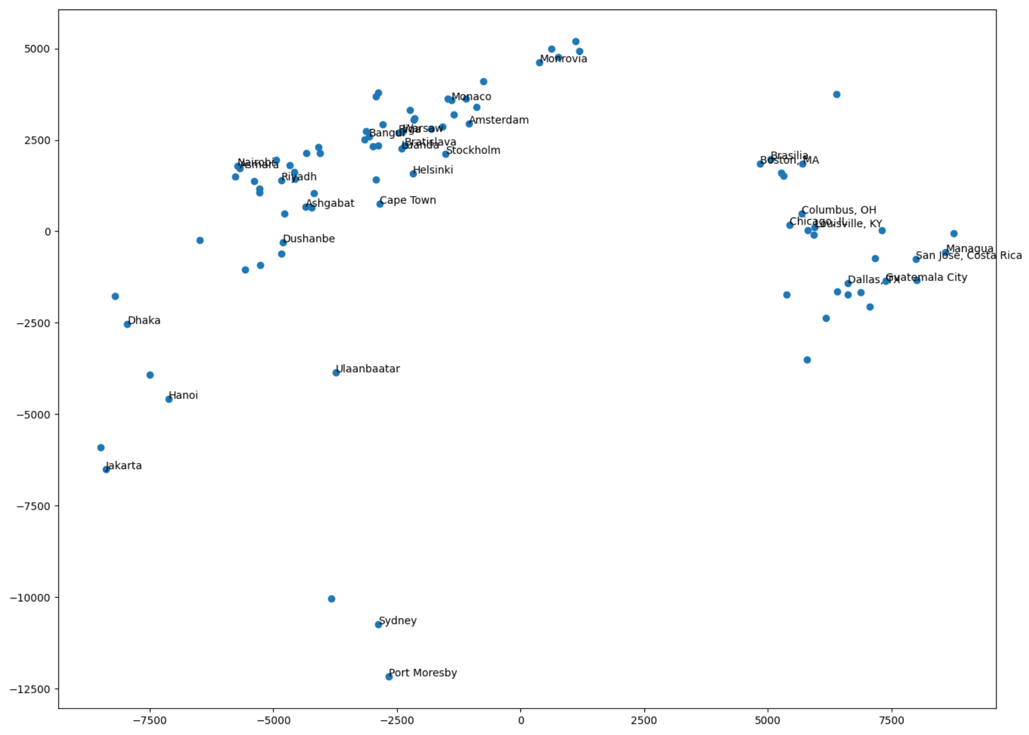

Figure 5: Plot of 92 cities as mapped by the Classical MDS reconstruction, with a third of the names added

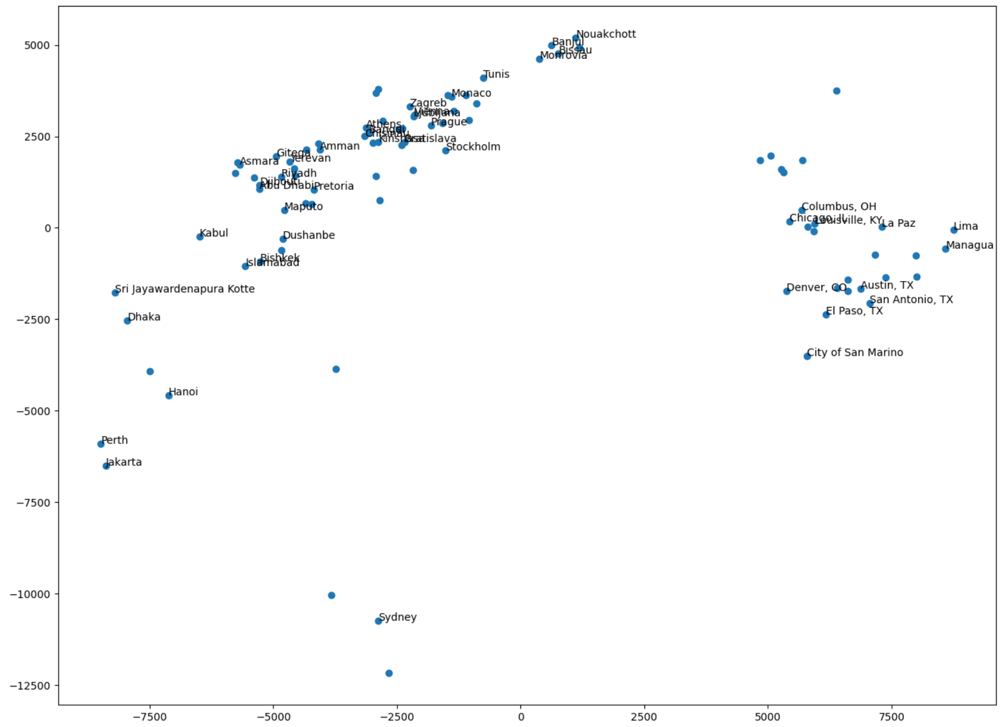

Figure 6: Plot of 92 cities as mapped by the Classical MDS reconstruction, with a half of the names added

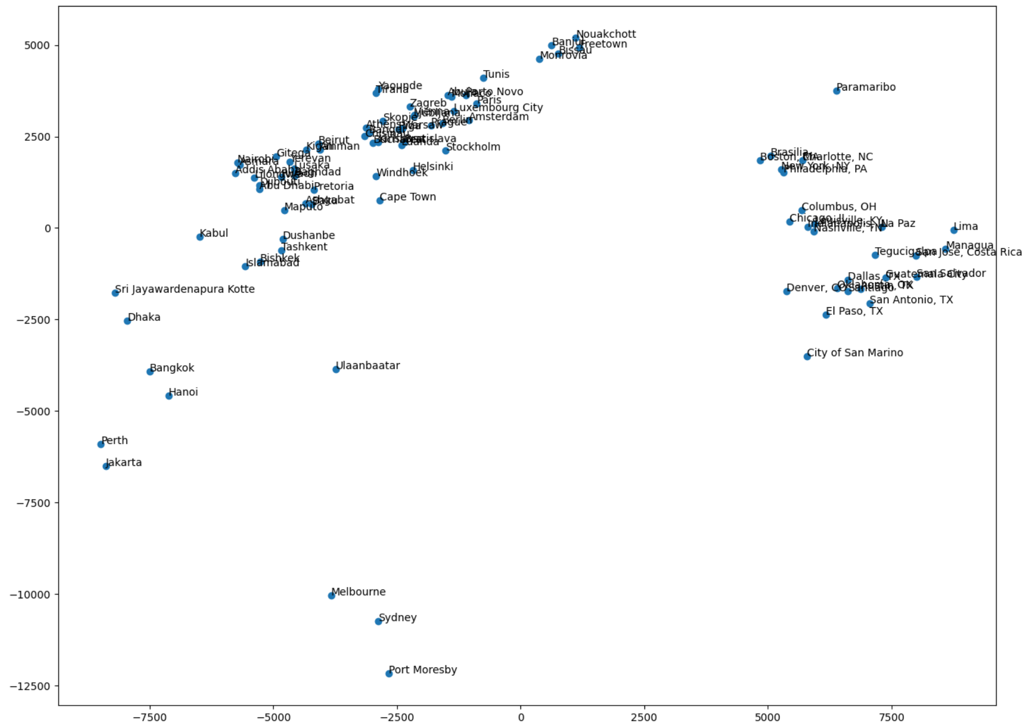

Figure 7: Plot of 92 cities as mapped by the Classical MDS reconstruction, with all of the names added