DD2343 Assignment 1

jacmalm@kth.se

November 21, 2021

1 Principal Component Analysis

1.1 Explain why data-centering is required before performing PCA. What might happen if we perform PCA on non-centered data?

PCA attempts to find new axes that are linear combinations of the old axes, such that these new axes, or principal components, minimize the mean squared distance between the original data points and their projection onto the principal axes [2]. One key thing to note is that these principal components are limited by the need to cross through the origin. This limitation means that data that is shifted cannot be approximated as well. A simple example is to consider the points $\{-1, 1\}$, $\{0, 0, \}$, $\{1, -1\}$. These points all lie perfectly along a 1 dimensional manifold, the line with slope -1 going through the origin. The mean of this dataset is 0. Thus, we expect the first principle component to consist of this line, and we expect all of the variance to be explained by the first principle component, as the PCA model is perfectly respected.When we run the PCA code (with the lines responsible for centering the data commented out in the SKLearn library file) given in appendix 1, this is exactly what we see. Great! PCA works.

However, if we shift the location of this manifold, while keeping its shape intact, this will change. If we now consider the points $\{49, 51\}$, $\{50, 50\}$, $\{51, 49\}$, it is easy to see that they also perfectly lie along a line, indeed a line with the same slope as the previous example. However, this line does not pass through the origin, and will thus not be found by PCA. Instead, what we expect to happen, is that the first principle component will be a line that goes through the origin and minimizes the distance from the data points and their projection onto the first principle axis. As the data is symmetric around their mean, we expect this to be a line going through the mean point, $\{50, 50\}$, and the origin. This is a line with slope 1. In fact, this line is perpendicular to the line that the data points lie along. When looking at the explained variance, we can also see that this line does not explain all of the variance, necessitating us to use both principal components to perfectly be able to get back all of our data, meaning that PCA fails to detect the inherent 1-dimensionality of the dataset. Again, these conclusions are verifiable with provided code in Appendix 1, and visually demonstrated in figure 1 and figure 2.

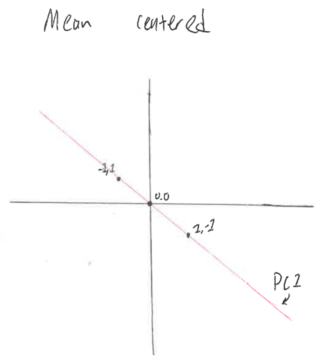


Figure 1.

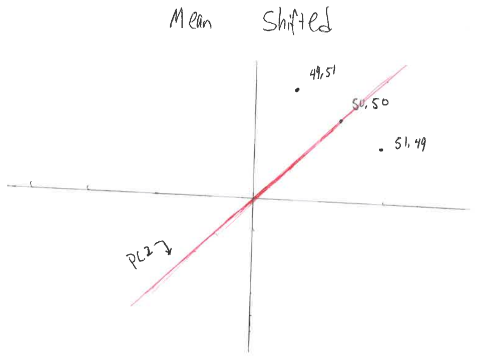


Figure 2.

1.2 Is a single SVD operation sufficient to performing PCA on both the rows and columns of a data matrix

The first step we do when we perform PCA is to mean center the data. In 1.1 we showed why this was necessary. However, when we mean center, we only mean center along one of the two axes. If columns represent a feature and rows represent observations, we can view each column as containing observations of a distinct random variable. Thus in this case, mean centering along each column is interpreted as shifting the mean of the distribution responsible for generating this column to zero. This keeps the structure of the data intact.

When we wish to perform PCA on the transpose of our data matrix, the interpretation of the rows and columns switch. Furthermore, our data will not be mean centered after transposing the matrix, as in the previous PCA operation we only center along the columns.

We are not able to mean center along rows and columns before performing both PCA operations, as this will alter the distribution of our data set. Imagine a case where we are given a data matrix with n observations and two normally distributed features, x, y. Each one of these observations can be plotted on a 2d plane, where its position on the x-axis is given by its value of feature x and its y-value is given by value of feature y. If we presume that these features are independent, our data points will fill in a circle as $n \to \infty$. Centering along each column will shift the center of this circle to the origin. Centering along the rows means that we need each observation to fulfill the following equation

$$0 = \frac{1}{2}(x_i + y_i,)i \in \{1...n\}$$

We can easily see that this will force all points to lie on the line $y = -x$, drastically changing the distribution of our observations.

Thus, we cannot perform PCA on both the rows and columns with one SVD operation as we need to mean center the matrix before the SVD operation and after the transposition.

1.3 Explain why the use of the pseudo-inverse is a good choice to obtain the inverse mapping of the linear map

The pseudo-inverse is a good choice to obtain the inverse mapping because when the PCA model is fully respected, that is our data points $\mathbf{y} \in \mathbb{R}^d$ are fully contained within a subspace $\mathbf{U} \in \mathbb{R}^k$ where $k < d$, and $\mathbf{U}$ is the column space of $\mathbf{W}$, $\mathbf{W}^+$ is the inverse mapping of $\mathbf{W}$, even when $\mathbf{W}$ is non-square.

However, when the PCA model is not fully respected and there are points in $\mathbf{y}$ that lie outside of $\mathbf{U}$, the pseudo-inverse computes the inverse mapping from the points projected onto $\mathbf{U}$, which is the approximation that minimizes the euclidean distance between computed and actual points [3].

Thus, using the pseudo-inverse of the linear transformation that is hypothesized to have generated $\mathbf{y}$ from $\mathbf{x}$, allows us to compute the projection onto the PCA axes even for points that do not completely lie within the k dimensional column space, which often occurs due to noisy measurements or an underlying model not quite aligned with the PCA assumptions.

1.4 Derive PCA using the criterion of variance maximization and show that you get the same results as with the criterion of minimizing the reconstruction error

We begin by assuming that our data set are observations from a set of d jointly distributed gaussian variables, $\mathbf{Y}$, that have undergone a linear transformation $\mathbf{W}$ applied to a set of p latent variables $\mathbf{X}$, where $p < d$. This linear transformation is restricted to be an axis change, meaning that $\mathbf{W}$ is orthogonal.

We assume that the latent variables $\mathbf{x}$ are uncorrelated. This means that the covariances are zero, and that the covariance matrix C_x is diagonal.

The covariance of $\mathbf{x}$ is given by

$$C_x = \mathbb{E}[xx^T]$$

Similarly, the covariance of y is given by

$$C_y = \mathbb{E}[yy^T]$$

Rewriting this equation using the PCA assumptions ($y = Wx$)

$$C_y = \mathbb{E}[Wxx^TW^T]$$

Using the fact that expectation is a linear operator

$$C_y = W\mathbb{E}[xx^T]W^T$$

$$C_y = WC_xW^T$$

Since **W** is orthonormal

$$W^TC_y = W^TWC_xW^T$$

$$W^TC_yW = W^TWC_xW^TW$$

$$W^TC_yW = IC_xI$$

$$W^TC_yW = C_x$$

Since C_y is symmetrical we can use its spectral decomposition to rewrite it as

$$C_y = Q\Lambda Q^T$$

And thus

$$C_x = W^TQ\Lambda Q^TW$$

Remembering our assumption that the variables in **x** are uncorrelated, C_x is a diagonal matrix. Since Λ is diagonal and Q is orthogonal,

$$WQ^T = I$$

$$WQ^TQ = IQ$$

$$W = Q$$

and

$$C_x = \Lambda$$

However, since C_X is a $p \times p$ matrix only the first p eigenvalues of Λ are non-zero (assuming the PCA model is followed). Thus the above equality only holds when choosing the p first eigenvectors when sorting by corresponding eigenvalue size, and restricting Λ to its non-zero diagonal values.

This suggests that the eigenvalues of the spectral decomposition of C_y represent the amount of variance captured in the direction of its corresponding eigenvector, with our eigenvectors spanning the linear subspace that contains $\mathbf{y}$. Thus our principal axes are given by Q in order of importance, when Λ is sorted in descending order [1].

Similarly, when deriving PCA from a minimization of squared error perspective, our principal components are given by $\mathbf{U}$, with $X = U\Sigma V^T$ (SVD).

If we rewrite X into its SVD before computing the covariance matrix we can see that $U = Q$, and Σ^2 is proportional to Λ. Thus, due to the monotonicity of the square function, (when restricted to positive values) the principal axes will appear in the same order, and have the same values. As our principal axes determine our PCA projection, we can conclude that the two derivations yield equivalent results [1].

2 Multidimensional Scaling and Isomap

2.1 Explain intuitively why the double centering trick is needed to be able to solve for S given D

Given a squared distance matrix **D** we wish to find Gram matrix **S**. As mentioned, the equation for this conversion is given by

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$$

The issue here is that we do not know the coordinates of our data points and as a result s_{ii}, s_{jj} are unknown, making this definition circular. When we assume that our data is mean centered (which we are allowed to do since distance is invariant to location), it can be shown that

$$-(s_{ii} + s_{jj}) = \mu_{ij} - \mu_i - \mu_j$$

These μ's refer to the means of the distance matrix we are given. Thus, by assuming that our data is centered, we can rewrite the unknowns in our conversion equation in terms of information contained in the distance matrix that we are given. This makes the conversion possible [1].

The double centering trick simply performs

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 + (\mu_{ij} - \mu_i - \mu_j))$$

for all matrix elements in **D**, converting our given distance matrix into the desired Gram matrix.

2.2 Use the same reasoning as in the previous question to argue that s_{ij} can be computed as $s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$

Comparing the original equation

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$$

with

$$s_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{1i}^2 - d_{1j}^2)$$

We can deduce that

$$s_{ii} + s_{jj} = d_{1i}^2 + d_{1j}^2$$

must hold.

From Lee and Verleysen [1] equation 4.42 we know that we can rewrite this equation accordingly

$$s_{ii} + s_{jj} = (\langle \mathbf{y_1} \cdot \mathbf{y_1} \rangle - 2\langle \mathbf{y_1} \cdot \mathbf{y_i} \rangle + \langle \mathbf{y_i} \cdot \mathbf{y_i} \rangle) + (\langle \mathbf{y_1} \cdot \mathbf{y_1} \rangle - 2\langle \mathbf{y_1} \cdot \mathbf{y_j} \rangle + \langle \mathbf{y_j} \cdot \mathbf{y_j} \rangle)$$

Where $\mathbf{y_1}$ represents the vector holding the coordinates of the first point. Now, if instead of assuming that our data points are mean centered and assume that the first data point is located on the origin, which we are allowed to do since this does not affect the distance between any of the points, the above equation reduces to

$$s_{ii} + s_{jj} = \langle \mathbf{y_j} \cdot \mathbf{y_j} \rangle + \langle \mathbf{y_i} \cdot \mathbf{y_i} \rangle$$

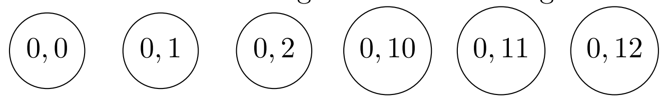
$$s_{ii} + s_{jj} = s_{ii} + s_{jj}$$

Thus we can see that we are justified in using this alternate equation to calculate our Gram matrix. It will not result in the same mapping as if we used $s_{ij} = -\frac{1}{2}(d_{ij}^2 + (\mu_{ij} - \mu_i - \mu_j))$, which will be mean centered, whereas our equation will fix the location of the first data point to the origin.

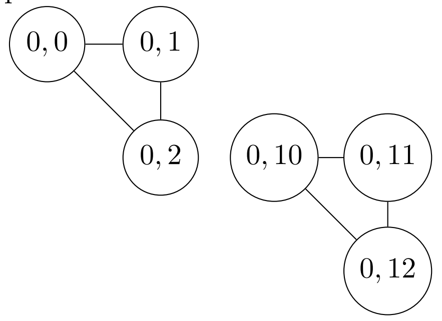
2.3 Argue that the process to obtain the neighbourhood graph G in the Isomap method may yield a disconnected graph. Provide an example. Explain why this is problematic.

The Isomap algorithm works similarly to MDS in that it relies on computing a Gram matrix **S** from a distance matrix **D**. However, dissimilarly to MDS, this distance matrix is not made up of euclidean distances. Instead, the distances are approximations of the shortest path from point A to point B where the path is restricted to the theoretical manifold our data distribution makes up. Computing the shape of this manifold as well as the length of the path through the manifold from point A to B is infeasible. An approximation of the manifold can be created by drawing a neighbourhood graph G. This neighbourhood graph G is created one of two ways. Either by joining each point to its k nearest neighbours, where a point A is considered closer to point B than to C if the euclidean distance between A-B is smaller than A-C. Alternatively the neighbourhood graph can be constructed by choosing to join a point A to all of its neighbours that lie within a certain distance d.

To demonstrate how this process can lead to a disconnected graph, assume we have the following nodes located at given coordinates.



If we go by the k-nearest neighbour approach and set $k = 2$, or go by max euclidean distance between nodes and set d to smaller than 8, our resulting graph will be disconnected.



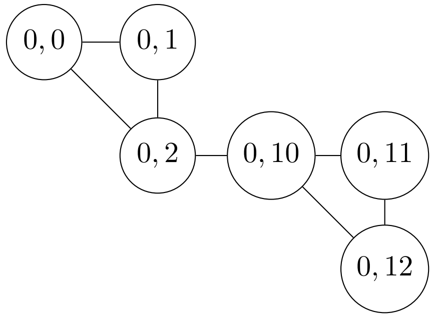
According to the Isomap algorithm, when we compute the distance matrix, the entry between point A and point B will be the summation of the distances between all nodes that lie along the shortest path between A and B through the neighbourhood graph G constructed in the previous step. If this is a disconnected graph, the distance between any point A to any point B where A and B are members of different disjoint subsets in G will be infinite, or undefined. In either of these cases the spectral decomposition of the Gram matrix corresponding to the distance matrix will be undefined and the Isomap algorithm will fail.

2.4 Propose a heuristic to patch the problem arising in the case of a disconnected neighborhood graph. Explain the intuition and why it is expected to work well.

A reasonable heuristic is to first build up the neighborhood graph as per the algorithm given in the previous section. Then, if disconnected clusters appear, calculate the minimum distance between each cluster to every other cluster, and add an edge between the closest disjoint nodes. This step is then repeated until a complete graph is formed.

This heuristic intuitively works well because it still attempts to form a respect the local structure of the manifold and form a neighborhood graph, it just requires us to relax some of the conditions (increase k for certain choice nodes or increase the radius we look for neighbors within) iteratively until the graph is connected. Therefore it is only a slight modification to the Isomap algorithm. We are however assuming that our data points belong to a single manifold, so we know that there should be a path between all points to all other points. Perhaps an alteration could be to assess the minimum distance between the disconnected clusters, and if it is above a certain error tolerance we choose to treat the clusters as separate manifolds and project them separately.

In the given example, we would determine that the minimum distance between the two clusters are between the node located at $0, 2$ and $0, 10$, and that they are a distance of 8 apart. If this is deemed to be acceptable we would join these two nodes creating the following graph



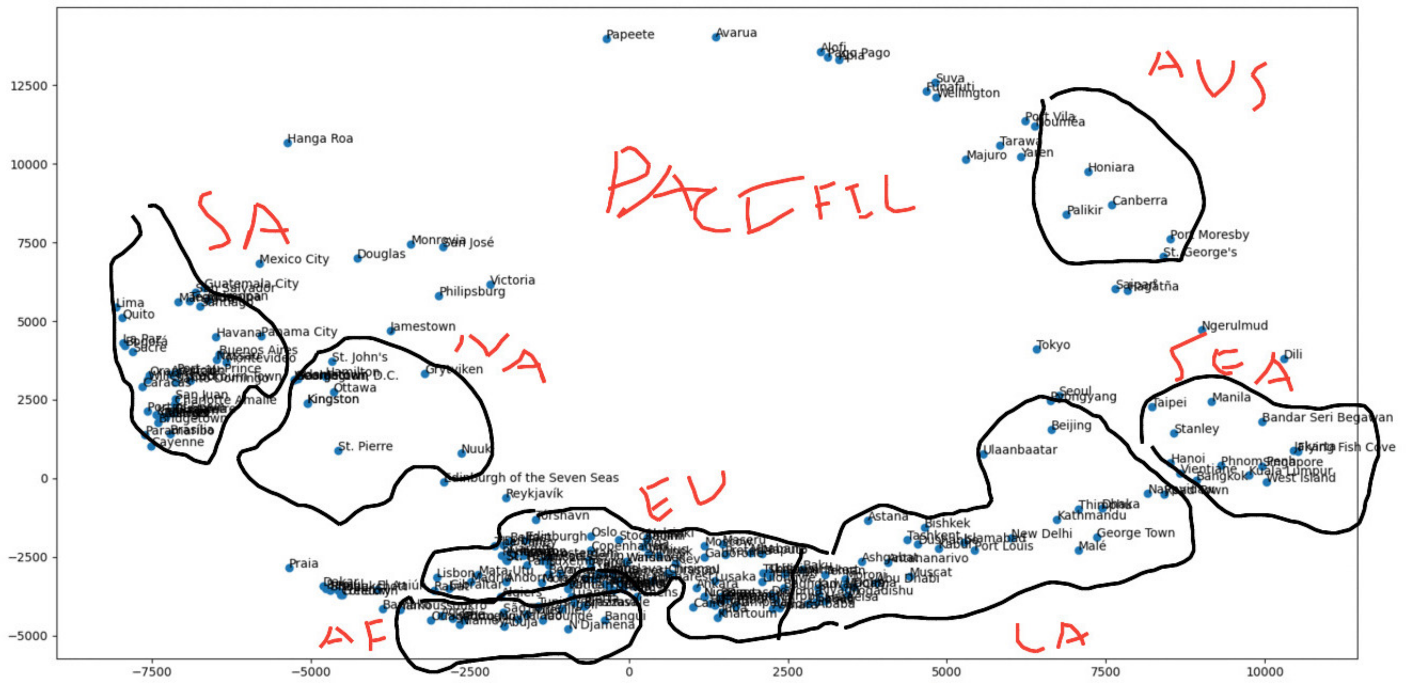
3 Programming task

3.1 Data collection

I collected a list of all the worlds capital cities from `https://github.com/icyrockcom/country-capitals/blob/master/data/country-list.csv`. The API i chose to use was provided by `https://se.avstand.org/api.xhtml`.
I filtered the list of cities by removing the cities that did not return a valid distance when pinging the API, a program for this can be found in Appendix 2. After this, I calculated the distance matrix for all the remaining 244 cities, and saved it as this operation was rather time consuming. The code for this task is given in Appendix 3.

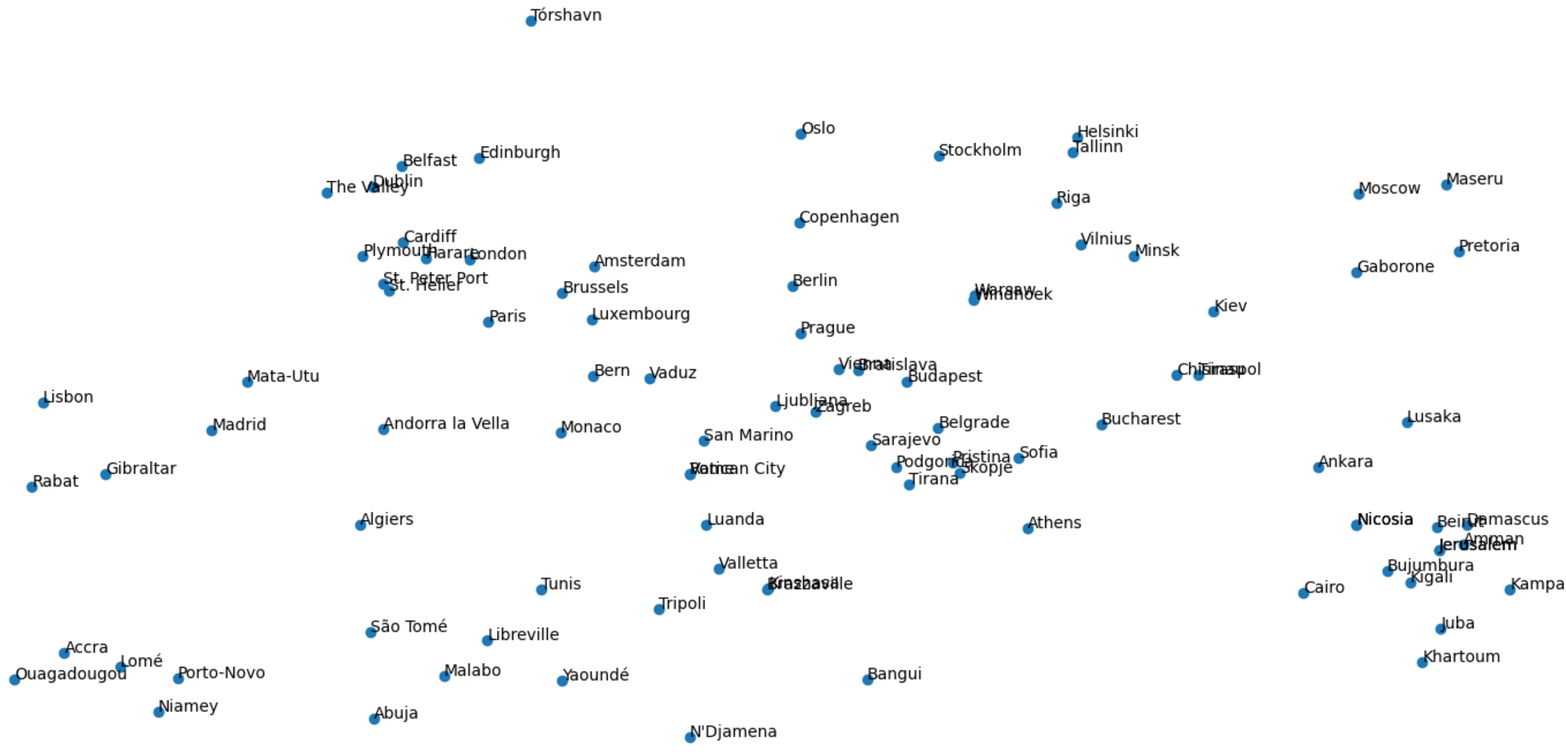
3.2 Classical MDS

The following is the world map that was generated by my classical MDS algorithm (Appendix 4)



The map is not oriented as we would normally view it, and it is slightly distorted in general. I believe this is due to the fact that I used all the worlds capital cities as my data points, thus when comparing an area of space like North America, which consists of two large countries and is thus represented by two data points, to an area like Europe, which consists of a lot of smaller countries, their relation is not entirely accurately represented. However, in general the geographical patterns as well as local relationships are retained. I have annotated the map with SA = South America, NA = North America, EU = Europe, AF = Africa, CA = Central Asia, SEA = South East Asia, AUS = Australia and friends.

Here is a zoomed in map of Europe and North Africa, as we can see the cities are positioned fairly accurately.

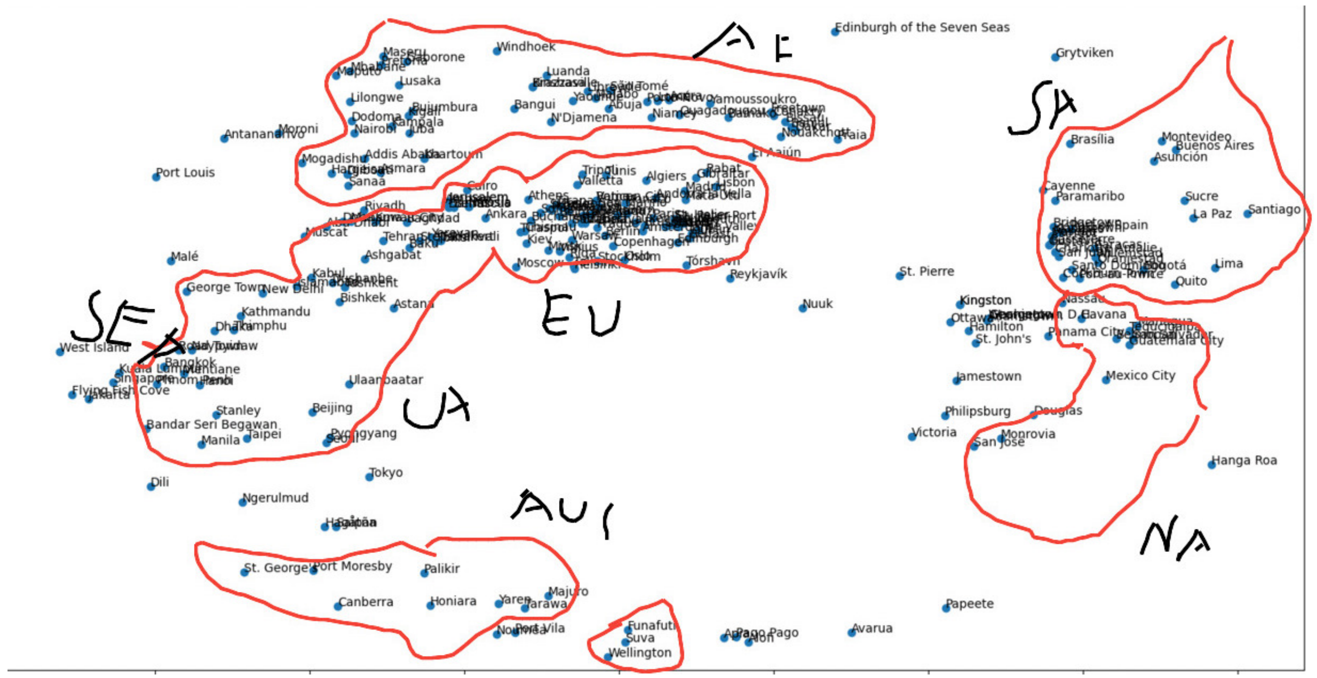


Pseudo-code and a helpful example was obtained from [4]

3.3 Metric MDS

In the metric MDS implementation I used (from the sklearn library) there were not so many parameters one could tune. I tried setting metric to False, this resulted in a map where all the cities where relatively evenly spaced, with the global structure of continents was not so well retained, but cities were well placed in relation to their near neighbours. This is to be expected as non-linear mds prioritizes keeping local relations intact over global structure. Increasing the tolerance at which the algorithm was declared converged changed gradually made the local relationships less reflective of the truth. Decreasing this tolerance had little to no effect, presumably because the algorithm already converged. All things considered, the default settings were fairly good.

Here is the map as produced by the metric mds library. It is fairly similar to the representation created by my classical mds implementation, apart from the fact that it is oriented slightly differently. Also I do have to say that Africa has better structural integrity in this version. On the whole, local relationships between cities are very similar between the two maps, cities form the same clusters in both implementations.



References

[1] *Nonlinear Dimensionality Reduction.* Springer, 2007.

[2] *Advanced Data Analysis from an Elementary Point of View.* Cambridge University Press, 2013.

[3] Guido Gerig. Least squares, pseudo-inverses, pca and svd. Technical report, University of Utah, 2012.

[4] Florian Wickelmaier. An introduction to mds. Technical report, Aalborg University, 2003.

4 Appendices

4.1 Appendix 1

```
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[49, 51], [50, 50], [51, 49]])
X_mean_centered = np.array([[-1, 1], [0, 0], [1, -1]])
pca = PCA()
pca.fit(X)
print(pca.components_)
print(pca.singular_values_)
print(pca.explained_variance_ratio_)

pca.fit(X_mean_centered)
print(pca.components_)
print(pca.singular_values_)
print(pca.explained_variance_ratio_)
```

4.2 Appendix 2

```
import requests
import pandas
import numpy as np
import csv

base_url = "https://se.avstand.org/route.json?stops="

def main():
    cities = read_in_cities("cities.csv")
    with open('treated_cities.csv', 'w') as f:
        write = csv.writer(f)
        write.writerow(cities)
    print(cities)

def call_api(city1, city2):
    response = requests.request("GET", base_url + str(city1) + "|" + str(city2))
    data = response.json()
    try:
        if len(data["distances"]) is not 0:
            return 0
        if data["stops"][0]["type"] == "Invalid":
            return -1
    except:
        print(city1, city2)
    return 0

def read_in_cities(file):
    cities = pandas.read_csv(file)
    cities = cities['capital'].tolist()
    treated_cities = []
    for i in range(len(cities)):
        if i == len(cities) - 1:
            result = call_api(cities[i], cities[i - 1])
        else:
            result = call_api(cities[i], cities[i + 1])
        if result != -1:
            treated_cities.append(cities[i])
    return treated_cities


main()
```

4.3 Appendix 3

```
import requests
import numpy as np
import csv


base_url = "https://se.avstand.org/route.json?stops="
```

```
def get_distance_matrix(calculate_distance_matrix=False):
    cities = read_in_cities("treated_cities.csv")
    if calculate_distance_matrix is True:
        D = create_distance_matrix(cities)
    else:
        D = np.load("distance_matrix.npy")
    return cities, D


def get_distance(city1, city2):
    response = requests.request("GET", base_url + str(city1) + "|" + str(city2))
    data = response.json()
    distance = data["distances"][0]
    return distance

def create_distance_matrix(cities):
    N = len(cities)
    D = np.zeros((N, N))
    for i in range(N):
        for j in range(i, N):
            if i == j:
                continue
            else:
                distance = get_distance(cities[i], cities[j])
                D[i][j] = distance
                D[j][i] = distance
    np.save("distance_matrix", D)
    return D

def read_in_cities(file):
    with open(file, 'r') as f:
        reader = csv.reader(f)
        data = list(reader)
    return data[0]
```

4.4 Appendix 4

```
import numpy as np
import get_distance_matrix as d
import matplotlib.pyplot as plt
import sklearn.manifold as sk


def main():
    cities, D = d.get_distance_matrix()
    classical_embedding = classical_mds(D)
    metric_embedding = metric_mds(D)
    create_plot(cities, metric_embedding)


def create_plot(labels, data_points):
    x = list(data_points[:, 0])
    y = list(data_points[:, 1])
    fig, ax = plt.subplots()
    ax.scatter(data_points[:, 0], data_points[:, 1])
    for i, label in enumerate(labels):
        ax.annotate(label, (x[i], y[i]))
    plt.show()


def metric_mds(D):
    model = sk.MDS(n_components=2, dissimilarity='precomputed')
    embedding = model.fit_transform(D)
    return embedding


def get_gram_matrix(D):
    N = len(D)
    D_squared = np.square(D)
    # Double centering
    J = np.identity(N) - 1 / N * np.ones((N, N))
    S = -0.5 * np.matmul(np.matmul(J, D_squared), J)
    return S


def classical_mds(D):
    S = get_gram_matrix(D)
    eigenval, eigenvec = np.linalg.eigh(S)
    # flip values since eigh returns in ascending order
    eigenval = eigenval[::-1]
    eigenvec = np.fliplr(eigenvec)
    # choose two dimensions
    eigenval = eigenval[:2]
    eigenval = np.sqrt(eigenval)
    eigenvec = eigenvec[:, :2]
    # do matrix multiplication to get our points
    eigenval = np.diag(eigenval)
    X = np.matmul(eigenvec, eigenval)
    return X


main()
```