Assignment 1, DD2434

Isac Lorentz

November 2021

Intro

Within the scope of what is allowed as specified in the assignment instructions, I have discussed the assignment questions with Stefan Christiansson, Wai-Hong Anton Fu and Parth Magdum.

1.1 Principal Component Analysis

Question 1.1.1

The data-centering step is required while performing PCA in order to make sure that the first principal component is describing the orientation of the maximum variance of the input data [1].

Performing PCA on non-centered data can cause the first principal component to point towards the mean of the dataset and also cause the PC1 to capture extra variance from the sum of squares from non centered data [2], [1].

Question 1.1.2

Only one SVD operation is required to perform PCA both on the rows and the columns of a data matrix. As shown in the Question 1.1.2 formulation since $\sum'$ differs from $\sum$ only in terms of size, this means that the singular values of $\boldsymbol{Y}$ is preserved when transposing the SVD. Since all that is required to obtain the $\boldsymbol{Y'}$ SVD can be obtained by using doing the SVD on $\boldsymbol{Y}$, these can just be calculated by transposing the matrices obtained by the $\boldsymbol{Y}$ SVD and then replace $\boldsymbol{U} \sum \boldsymbol{V^T}$ with $\boldsymbol{V} \sum' \boldsymbol{U^T}$ at the SVD step in the PCA in order to perform PCA on $\boldsymbol{Y^T}$ instead of $\boldsymbol{Y}$.

Question 1.1.3

The Moore-Penrose inverse is a good choice to obtain the inverse mapping of the linear map since the inverse since it gives a unique solution and works for both square and rectangular matrices [3, p.4]. Regular matrix inversion does

not work for non-square matrices. Other types of matrix inversion might for an $\boldsymbol{m} \times \boldsymbol{n}$ matrix give multiple solutions for wide matrices ($\boldsymbol{m} < \boldsymbol{n}$) and no solutions for tall matrices ($\boldsymbol{m} > \boldsymbol{n}$). For $\boldsymbol{m} < \boldsymbol{n}$, the Moore-Penrose inverse is the solution to $\boldsymbol{x} = \boldsymbol{W}^{+}\boldsymbol{y}$ that minimizes $\|\boldsymbol{x}\|_{\boldsymbol{2}}$. For $m > n$ the inverse is the solution that minimizes $\|\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y}\|_{\boldsymbol{2}}$ [4, p.44].

This relates to PCA since in the PCA algorithm one aims to produce the decoding that minimizes the L^2-norm between the the original datapoint $\boldsymbol{y}$ and its decoding $\boldsymbol{W}\boldsymbol{x}$. This is minimized by minimizing $\|\boldsymbol{W}\boldsymbol{x} - \boldsymbol{y}\|_{\boldsymbol{2}}$ [4, p.47], which is also what happens when using the Moore-Penrose inverse for the case where the dimension of $\boldsymbol{y}$ is larger than the dimension of $\boldsymbol{x}$ ($k < d$) .

1.2 Multidimensional Scaling (MDS) and Isomap

Question 1.2.5

Double centering is useful since the $\boldsymbol{Y}$ coordinates are often unknown and this needs to be calculated implicitly by using the double centering trick [5, p.76]. Using the $\boldsymbol{D}$ matrix and double centering, we can estimate each s_{ij} element [6].

When using $\boldsymbol{D}$ to perform the MDS, a origin point is required in order to calculate the scalar products. By performing the double centering trick, the centroid of the MDS configuration becomes the selected origin point [7, p.106].

Question 1.2.6

The choice of picking the centroid of the MDS configuration as the origin is what necessitates the double centering trick [7, p.106]. By performing the double centering we will get a zero mean embedding [8]. Any other point in $\boldsymbol{Y}$ could also have been selected as the origin and this would preserve the distances. One point as the origin is required to be able to compute the scalar products. Since the first point is in the dataset, this is a valid point for the origin and this will lead to a correct solution using the first point trick [7, p.106]. Since the first point is not necessarily located at the identical location as the centroid of the MDS configuration, this is what causes the solutions to be different between the first point trick and the double centering trick.

Question 1.2.8

Isomap works by generating a graph for the dataset and compare distances between points by using graph distance in the neighbourhood graph instead of euclidean distances. However, not all types of datasets can correctly be converted to a neighbourhood graph. Generally, only datasets that represents a single, connected manifold can be perfectly translated using Isomap [9], [5, p.104].

The example I provide is a broken Swiss roll dataset with two components

(Figure 1). For this dataset, it will not be possible to perform Isomap on the whole dataset. Generally datasets that is not possible to perform Isomap on include datasets lying on multiple manifolds or consisting of multiple clusters [9].

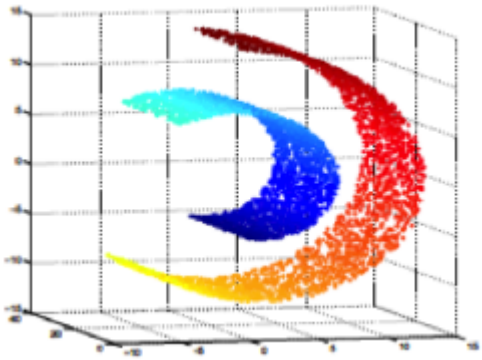

Figure 1: two-part broken swiss roll dataset

The problem of performing Isomap on datasets with multiple separate components such as a broken Swiss roll has technically to do with the fact that the neighbourhood graph will have no distance between points in different components [9]. This is problematic since neighbourhood graph based reduction techniques such as Isomap will not be able to recognize both global and local properties of such datasets [9]. It would be possible to do Isomap on each component individually, but this would not capture global information [9].

1.3 PCA vs. Johnson-Lindenstrauss random projections

Question 1.3.10

(i) Projection Error

The projection error of PCA is given by the reconstruction error. This is calculated as the difference of each datapoint and its predicted value reprojected back into the original dimension [10]. The projection error for random projections is bound by the Johnson-Lindenstrauss lemma. For PCA, there is no equally recognized formula that provides a bound for the reconstruction error as is in the case for random projections.

(ii) Computational Efficiency

This is a advantage for random projections, that it is faster. With d dimension of input data, k-dimensional subspace projection and N data instances, random

projections has $\mathcal{O}(dkN)$ time complexity and $\mathcal{O}(dcN)$ in the case of a spare data matrix with c non-zero entries per column. PCA time complexity is $\mathcal{O}(d^2N) + \mathcal{O}(d^3)$ [11].

(iii) Intended use case

Due to the time complexity for both methods depending on the size of the dataset, the less time consuming random projection is more suitable for larger datasets when runtime is a limiting factor [12]. PCA perform better in preprocessing for downstream classification for several supervised learning algorithms such as KNN, SVM and decision trees on several different datasets [12]. When known bounds on the error is required, this can be an advantage in favor of random projections, however PCA has better accuracy in many scenarios [12].

1.4 Programming task — MDS

Question 1.4.11

A Kaggle Dataset containing national capitals from all the worlds nations and some territories was used [13]. This is a good choice in my opinion since it covers all five continents. Some data cleaning was needed since some entries had missing data. This was done by just removing these values in MS Excel.

The Python library GeoPy was used to calculate the geodesic distance between the cities using latitudes and longitudes from the dataset [14]. See code in Appendix A.1.

Question 1.4.12

Figure 2: Capitals of independent nations. *Google My Maps*

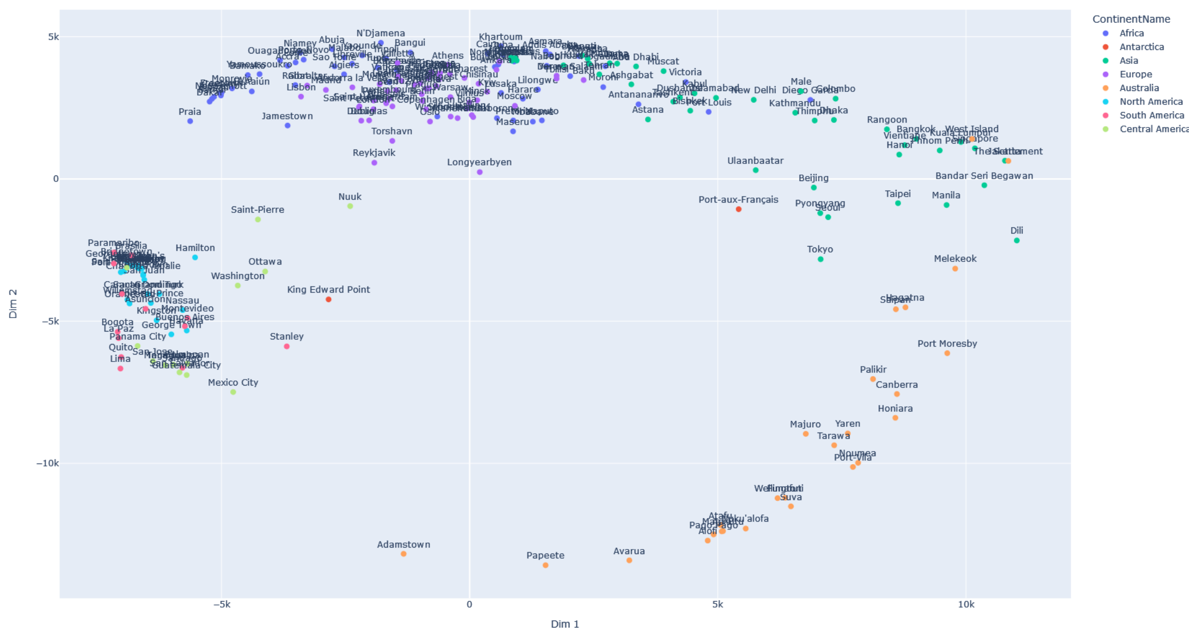

Figure 3: MDS output with city name as label

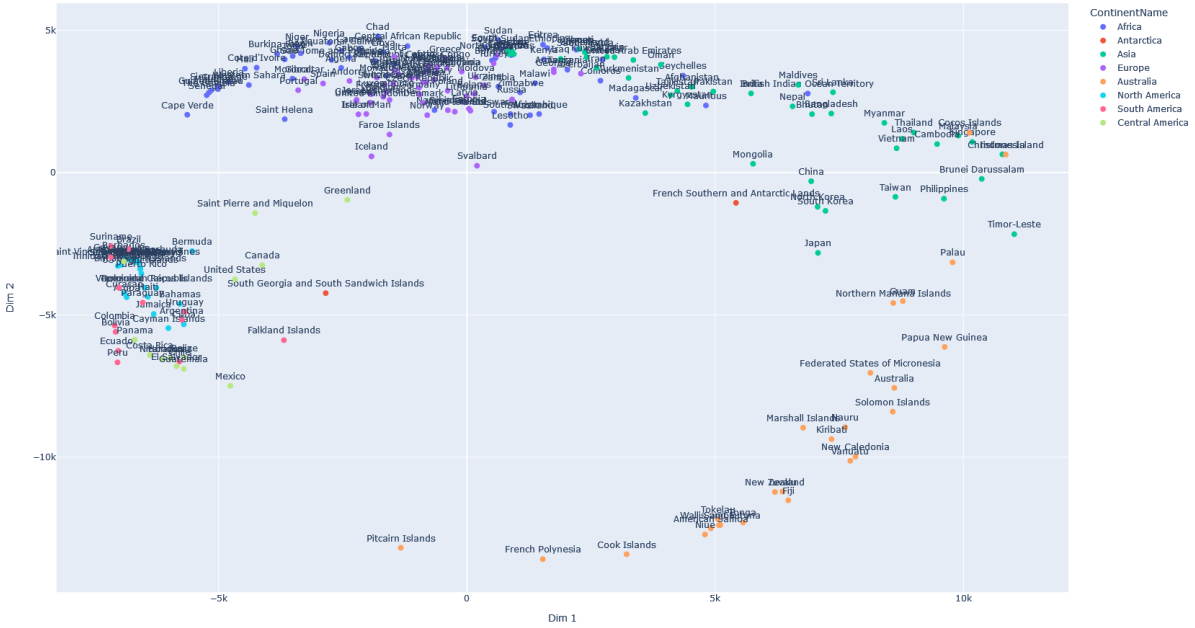

Figure 4: MDS output with country name as label

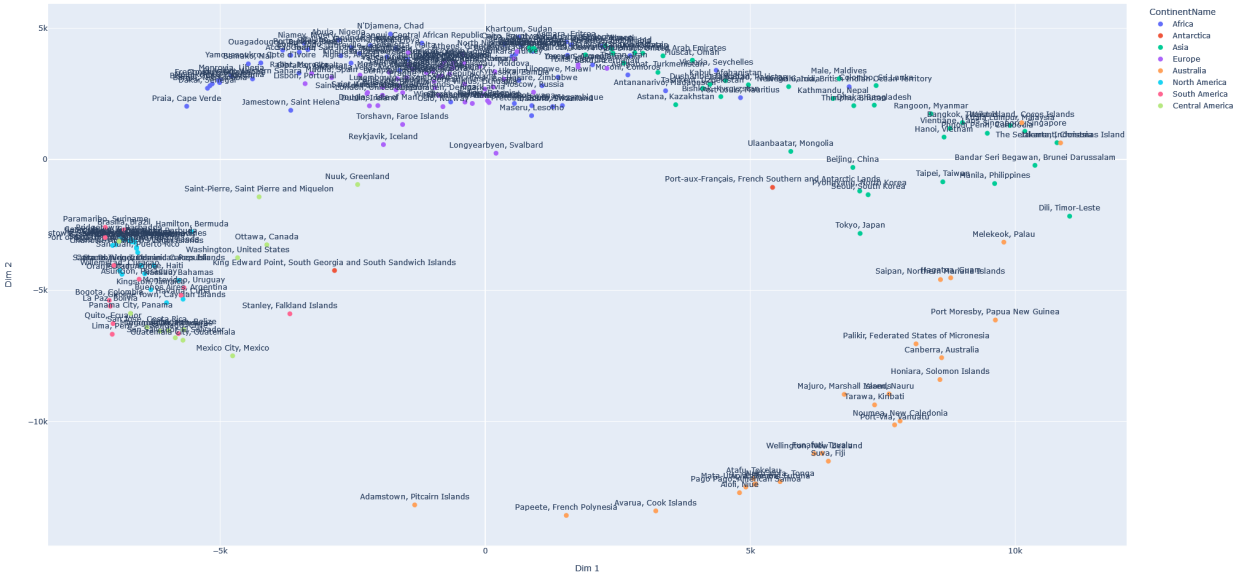

Figure 5: MDS output with both as label

Numpy Python package (numpy.linalg.eig) was used for eigenvalue decomposition [15]. When looking at the MDS output we can se that there is some logic to the reconstructed map. The countries are overall well grouped together by continent (Figure 3-5).

The West-east property of the cities are well captured along dimension 1, with Europe and Africa between The Americas at negative Dim 1 values and Asia and Australia at positive Dim 1 values (Figure 3-5).

The North-south properties of the cities are not captured as well along dimension 2 as the east-west properties along dimension 1. We see on upper left side of the figures that the Americas are all mixed together and there is no clear indication that Europe is further north than Africa. Looking at the right side of the pictures, the property that Australia is further south than Asia is overall preserved. We can also see that the two Antarctica cities become quite misplaced, both are at the center of the map (Figure 3-5).

Comparing to Figure 3-5 to figure 2, this doesn't look much similar to where the cities are placed on a map (I believe this map is using the Mercator projection). The Mercator projection preserve directions and shapes of landmasses but sizes of countries far away from the equator are presented as much larger than their actual size. It should be noted that no 2D map projection is able to perfectly preserve all properties of the spherical earth. It is not extremely easy to classify this reconstructed map created by classical MDS as good or bad since different types of maps aim to preserve different properties from the spherical earth.

See code in appendix A.2.

References

[1] Nikos Alexandris, S. Gupta, and Nikos Koutsias. "Remote sensing of burned areas via PCA, Part 1; centering, scaling and EVD vs SVD". In: *Open Geospatial Data, Software and Standards* 2 (2017), pp. 1–11.

[2] Neal Gallagher, Donal O'Sullivan, and Manuel Palacios. "The Effect of Data Centering on PCA Models". In: (June 2020).

[3] Adi Ben-Israel. *Generalized Inverses Theory and Applications*. eng. 2nd ed. 2003.. CMS Books in Mathematics, Ouvrages de mathématiques de la SMC. 2003. ISBN: 0-387-00293-6.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[5] Michael Verleysen Lee John A. (John Aldo). *Nonlinear dimensionality reduction*. eng. Information science and statistics. New York: Springer, 2007. ISBN: 978-0-387-39350-6.

[6] Aristides Gionis. *DD2434 Machine Learning, Advance Course Module 2: high-dimensional data and dimensionality reduction lecture slides*. Oct. 2021.

[7] I. Borg, P. J. F. Groenen, and P. Mair. *Applied Multidimensional Scaling and Unfolding*. 2nd ed. New York: Springer, 2018. ISBN: 978-3-319-73470-5.

[8] David W. Jacobs. *Multidimensional Scaling: More complete proof and some insights not mentioned in class*. URL: cs.umd.edu/~djacobs/CMSC828/MDSexplain.pdf.

[9] Jicong Fan et al. "Nonlinear Dimensionality Reduction for Data with Disconnected Neighborhood Graph". English. In: *Neural Processing Letters* 47.2 (Apr. 2018), pp. 697–716. ISSN: 1370-4621. DOI: 10.1007/s11063-017-9676-5.

[10] James A. Jablonski, Trevor J. Bihl, and Kenneth W. Bauer. "Principal Component Reconstruction Error for Hyperspectral Anomaly Detection". In: *IEEE Geoscience and Remote Sensing Letters* 12.8 (2015), pp. 1725–1729. DOI: 10.1109/LGRS.2015.2421813.

[11] Ella Bingham and Heikki Mannila. "Random Projection in Dimensionality Reduction: Applications to Image and Text Data". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '01. San Francisco, California: Association for Computing Machinery, 2001, pp. 245–250. ISBN: 158113391X. DOI: 10.1145/502512.502546. URL: https://doi.org/10.1145/502512.502546.

[12] M. Wojnowicz et al. "Projecting "Better Than Randomly": How to Reduce the Dimensionality of Very Large Datasets in a Way That Outperforms Random Projections". In: *2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2016, pp. 184–193. DOI: 10.1109/DSAA.2016.26. URL: https://doi.ieeecomputersociety.org/10.1109/DSAA.2016.26.

[13] Nikita Grecnik. *World capitals gps*. https://www.kaggle.com/nikitagrec/world-capitals-gps. Feb. 2019.

[14] *GeoPy*. https://www.kaggle.com/nikitagrec/world-capitals-gps.

[15] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Appendix A: Source Code

A.1 Question 1.4.11 code

```
1 import math
2 from itertools import tee
3
```

```
import numpy as np
import pandas as pd
from geopy import distance

# Kaggle dataset from https://www.kaggle.com/nikitagrec/world-capitals-gps

df = pd.read_csv("worldCaps.csv")

names = df["CapitalName"].unique().tolist()

uniqueDf = df.copy()
uniqueDf = uniqueDf.drop_duplicates("CapitalName")
names = [str(elem) for elem in names]
names = [name for name in names if name != "nan"]

distMatrix = pd.DataFrame(index=names, columns=names)

for (i, row1) in uniqueDf.iterrows():
    if row1["CapitalName"] in names:

        LatOrigin = row1["CapitalLatitude"]
        LongOrigin = row1["CapitalLongitude"]
        origin = (LatOrigin, LongOrigin)

        for (j, row2) in uniqueDf.iterrows():
            if row2["CapitalName"] in names:
                if row1["CapitalName"] == row2["CapitalName"]:
                    distMatrix.at[row1["CapitalName"], row2["CapitalName"]] = 0
                else:
                    val = str(distMatrix.at[row1["CapitalName"], row2["CapitalName"]])
                    if val == "nan" or val == "NaN":

                        LatDestination = row2["CapitalLatitude"]
                        LongDestination = row2["CapitalLongitude"]

                        destination = (LatDestination, LongDestination)
                        geodesicDist = distance.distance(origin, destination).km
                        geodesicDist = math.pow(geodesicDist, 2)
                        geodesicDist = round(geodesicDist)

                        distMatrix.at[
                            row1["CapitalName"], row2["CapitalName"]
                        ] = geodesicDist
                        distMatrix.at[
                            row2["CapitalName"], row1["CapitalName"]
                        ] = geodesicDist
```

```
50
51  uniqueDf["cityCountry"] = uniqueDf["CapitalName"] + ", " + uniqueDf["CountryName"]
52
53
54  distMatrix.to_csv("distMatrixGeo.csv")
55  uniqueDf.to_csv("worldCapsInfo.csv")
56
57  print(distMatrix)
```

A.2 Question 1.4.12 code

```
1   import numpy as np
2   import pandas as pd
3   import plotly.express as px
4
5   np.set_printoptions(threshold=np.inf)
6   import math
7   from itertools import tee
8
9   from geopy import distance
10
11  # Distance matrix and the dataset are imported
12  df = pd.read_csv("distMatrixGeo.csv", index_col=0)
13  worldcapsDf = pd.read_csv("worldCapsInfo.csv", index_col=0)
14
15  D = df.to_numpy()
16
17  n = df.shape[0]
18
19  ones = np.ones((n, n))
20
21  s = (
22      D
23      - ((1 / n) * D @ ones)
24      - ((1 / n) * ones @ D)
25      + ((1 / pow(n, 2)) * ones @ D @ ones)
26  )
27
28  s = -0.5 * s
29
30  m = 2
31  w, v = np.linalg.eig(s)
32
33  lambdaM = (-w).argsort()[:m]
34  print(
35      lambdaM
36  )  # since 0 and 1 are the 2 largest eigenvalues these can be picked directly
37  print(w)
38
```

```
w = w[:2]
v = v.T[:2, :]

diagEigenV = np.diag(w)


Ut = v

I = np.eye(m, n)
X = np.sqrt(diagEigenV) @ Ut

worldcapsDf["Dim 1"] = X[0]
worldcapsDf["Dim 2"] = X[1]

fig = px.scatter(
    worldcapsDf, x="Dim 1", y="Dim 2", color="ContinentName", text="CapitalName"
)
fig.update_traces(marker_size=8, textposition="top center")
fig.show()

fig = px.scatter(
    worldcapsDf, x="Dim 1", y="Dim 2", color="ContinentName", text="CountryName"
)
fig.update_traces(marker_size=8, textposition="top center")
fig.show()

fig = px.scatter(
    worldcapsDf, x="Dim 1", y="Dim 2", color="ContinentName", text="cityCountry"
)
fig.update_traces(marker_size=8, textposition="top center")
fig.show()
```