LOW DIMENSIONAL EMBEDDING OF ENVIRONMENTAL VARIABLES

EA MAP581

16 mars 2018

Flore Martin and Lorraine Roulier

Table des matières

1 Introduction

Dimension reduction refers to the process of reducing the number of random variables under consideration. It is an efficient way to analyze a large dataset of high dimension. For instance, dimensionality reduction can be used for **visualizing** or exploring structure in data. It provides a **simple geometric interpretation** by mapping in 2D or 3D an original dataset of dimension d > 3. A 2D or 3D mapping is thus much easy to interpret for us human. In terms of performance, having data of high dimensionality is problematic because it can mean **high computational cost** to compute. Finally, dimensional reduction is often used for classification. Data points are being clustered, it is a way to establish clusters of gene and of population for example.

In practice, dimension reduction is widely used for denoising or compressing data, and embedding image processing like facial recognition.

2 Dataset presentation

Climate data amounts very quickly to a lot of unused data. In a day, we can collect temperature, pressure, wind data all over the world with satellites, even hourly. We used a set of datasets we found on the NASA website, that gathered various means on climate variables over 22 years at every given latitude and longitude. Thoses dataset can de bound here : https ://eosweb.larc.nasa.gov/cgi-bin/sse/global.cgi ?email=skip@larc.nasa.gov. AS each variable (temperature, radiation, pressure, wind speed and humidity) was stored in a separate dataset, we first had to merge all of them in a final table presented below :

Latitude	Longitude	Temperature $°C$	Pressure kPa	Relative Humidity $\%$	Wind Speed m/s	Radiation $kWh/m^2/day$

Figure 1 – First row of our dataset

The latitude parameter varies from -90 to 89 and the longitude parameter varies from -180 (south pole) to 179 (north pole). Depending on the running time of the method, we did not compute the dimension reduction with the 64800 lines, but with a subset. The subset is often a slice of longitudes containing all latitudes, as we assumed that the critical parameter to differentiate climate data was the latitude. We will detail the subset considered in every method.

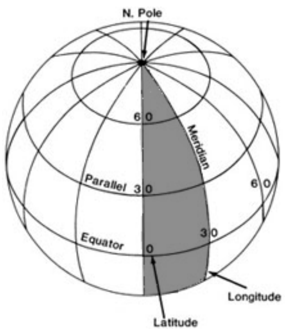


Figure 2 – An example of subset in grey

We classified the data according to the latitude, creating five classes listed in the table below :

	North	Temperate north	Equator	Temperate South	South
Latitudes	90 to 66	66 to 23	23 to -23	-23 to -66	-66 to -90

Figure 3 – First row of our dataset

3 Methods

Dimension reduction techniques can be divided in two classes, linear dimension reduction and non linear dimension reduction. However, in all methods, the main goal is to figure out a similarity function between vectors. Such a function will then enable to sort the dataset into classes of vectors with similar features, which would have been more intricate with the initial dataset.

Our project was two sided. First, we familiarized with various dimension reduction techniques, then we attempted to show that the geographical position of a point on the planet - e.g. it's latitude and longitude - were embedded in the climate data one could gather on it.

3.1 Principal Component Analysis - PCA

Principal Component Analysis detects tendencies in the data by maximizing the variance of the dataset matrix. There is an easy approach to understand PCA with a 2D dataset. On Figure ? ?, we have plotted a 2D dataset. If we implement 2D-PCA, we will find two new axes that reflects at best the variances of datapoints. Those ares the green axes. PCA just 'rotates' the figure to find a more explicit 'point of view'. The yielded axes are a linear combination of the original axes.

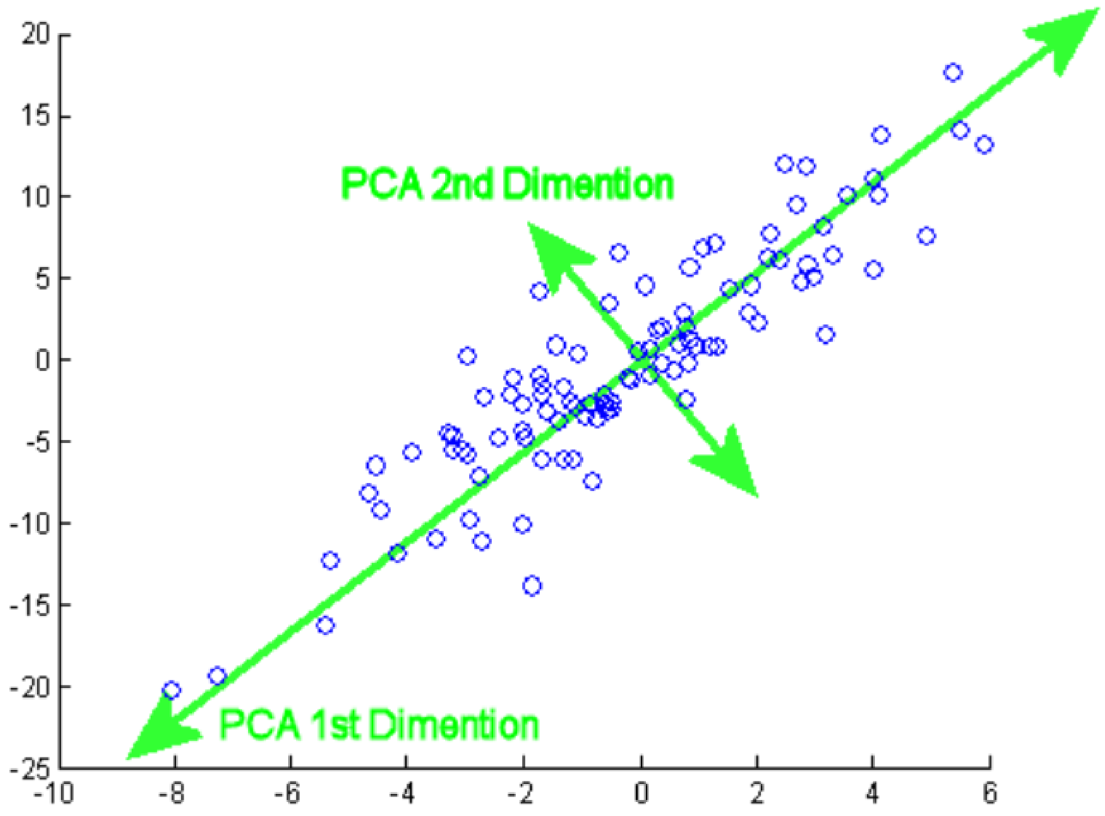


Figure 4 – role of PCA - interpretation

From a mathematical point of view, PCA yields an orthonormal matrix that can be diagonalized. The largest eigenvalues point to the eigenvectors that contain the most information about the dataset.

Let $X \in \mathbb{R}^{d\times n}$ be our dataset, PCA maximizes the following equation :

$$\|X - MM^T X\|^2$$

subject to $M \in \mathcal{O}^{d\times r}$ where $r < d$.

The r greatest eigenvalues of $XX^T = PDP^T$ yield the maximum $M = P_r$ where the columns are the eigenvectors associated to the eigenvalues mentioned above. The new dataset is thus $Y = P_r^T X$ of dimensions $r \times n$.

3.2 Kernel PCA

Kernel Principal Component analysis is a non linear method for dimension reduction. Instead of directly maximizing the variance of X, we map it into a larger dimension space called the feature space, using a non linear function ϕ.

Instead of directly mapping X into the feature space, we use a kernel, that represents a similarity function between pairs. Let K be this kernel, then

$$K(x_i, x_j) = \phi(x_{\hat{\imath}})\phi(x_j)^T$$

where x_i and x_j are vectors of the dataset X

This kernel will enable us to bypass the higher dimension calculation of the variance in the feature space. We only need to compute the pairwise values for ϕ, but there is no need to explicitly define ϕ.

The downside of this method is that since we don't compute directly the eigenvectors and eigenvalues of ϕ, the result is the projection of our dataset onto the eigenvectors. We thus don't have access to a simple interpretation of the principal components.

A classic kernel that is often used is the RBF function that is defined below :

$$K(x_i, x_j) = \mathrm{e} - \gamma\|x_i - x_j\|$$

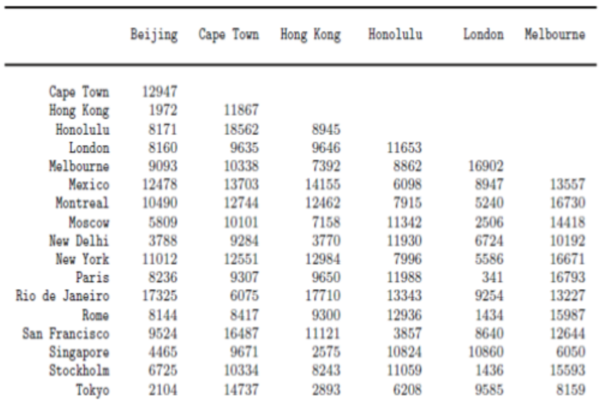
3.3 Multidimensional scaling

Multidimensional scaling is a non-linear approach to reduce dimension of a dataset. The principle is quite different from PCA : given a matrix of distance or "‘dissimilarity"’, MDS aims at reconstructing a map preserving at best the distances between datas. In our example, the MDS algorithm aims to place each object in 2-dimensional space such that the between-object distances are preserved as well as possible. Given a distance matrix D of dimension n*p (p=5 for us) , MDS attempts to find the datapoints y_1,... y_t in dimension d<p (d=2 for us) that minimizes :

$$\sum_{i=1}^{t}[\sum_{j=1}^{t} d_{ij}^{(X)} - d_{ij}^{(Y)}]$$

with $d_{ij}^{(X)}$ the euclidean distance between pairwise i and j in the original $n \times p$ matrix $D^{(X)}$ and $d_{ij}^{(Y)}$ the euclidean distance between pairwise i and j in the original $n \times p$ matrix $D^{(X)}$ and in the computed n*d matrix $D^{(Y)}$.

A basic approach to understand MDS is to use the example of cities. The input is the distance matrix between cities on the left on figure ? ?. MDS enabales us to find coordonates of each city based on this distance matrix.



	Beijing	Cape Town	Hong Kong	Honolulu	London	Melbourne
Cape Town	12947					
Hong Kong	1972	11867				
Honolulu	8171	18562	8945			
London	8160	9635	9646	11653		
Melbourne	9093	10338	7392	8862	16902	
Mexico	12478	13703	14155	6098	8947	13557
Montreal	10490	12744	12462	7915	5240	16730
Moscow	5809	10101	7158	11342	2506	14418
New Delhi	3788	9284	3770	11930	6724	10192
New York	11012	12551	12984	7996	5586	16671
Paris	8236	9307	9650	11988	341	16793
Rio de Janeiro	17325	6075	17710	13343	9254	13227
Rome	8144	8417	9300	12936	1434	15987
San Francisco	9524	16487	11121	3857	8640	12644
Singapore	4465	9671	2575	10824	10860	6050
Stockholm	6725	10334	8243	11059	1436	15593
Tokyo	2104	14737	2893	6208	9585	8159

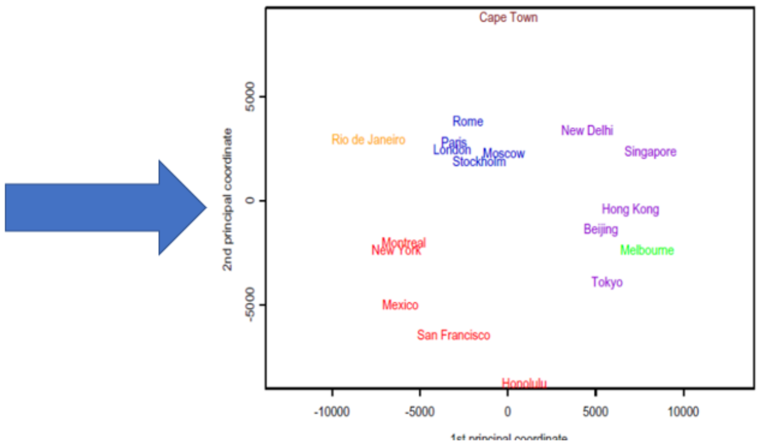


Figure 5 – role of MDS - interpretation

3.4 Isomap

Isomap is a low dimensional embedding method similar to MDS. The difference is, distance is not computed with euclidean norm but with geodesic distance. As the earth shape is round, isomap seems more relevant than MDS. ON figure ? ?, we can see the difference between the geodisic distance (in red) used in isomap, and the euclean distance used in MDS (in blue).

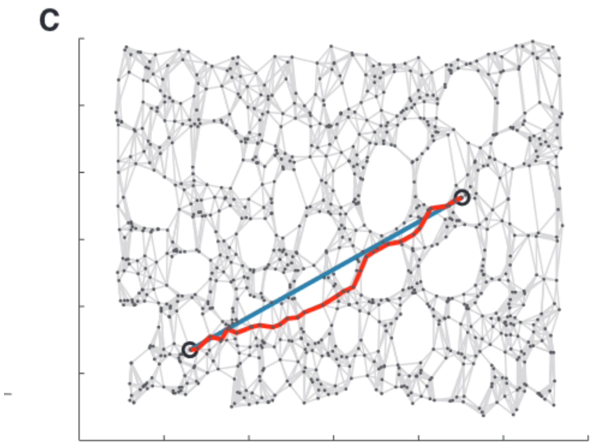


Figure 6 – geodisic and euclidean distance

4 Results

4.1 graphic interpretation

4.1.1 PCA

We first implemented PCA and ran it with only two principal components, which yielded the following graph for the whole dataset. The associated eigenvectors were

$$y_1 = \begin{bmatrix} -0.94220902 \\ -0.31329122 \\ 0.10889051 \\ -0.04573547 \\ 0.01191187 \end{bmatrix}$$

and

$$y_2 = \begin{bmatrix} 0.02637512 \\ -0.40428947 \\ -0.9117768 \\ 0.04144017 \\ -0.05291649 \end{bmatrix}$$

This enables us to understand the meaning of these vectors. y_1 is mostly related to a decreasing temperature and pressure, and y_2 represents decreasing humidity and pressure.

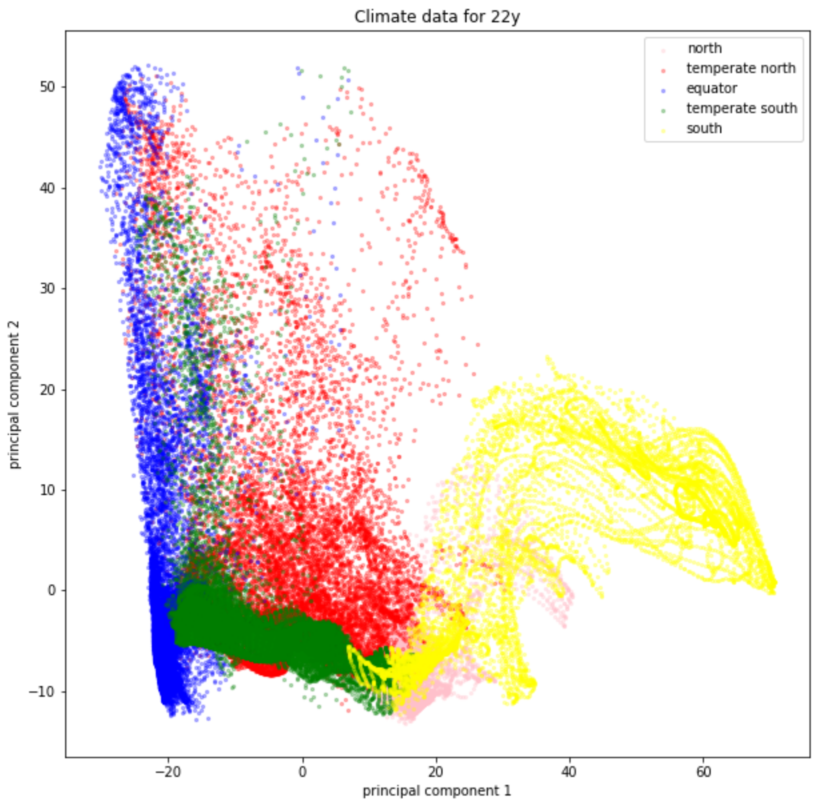


Figure 7 – PCA with two components

We can see on the graph that even if there is a strong dispersion, equator values are located at higher temperatures. On the contrary, north and south pole values are located at lower temperatures. Green and red classes overlap as these to categories have similar climate conditions.

We attempted to run the PCA method with three principal components.

4.1.2 Kernel PCA

Kernel PCA has a complexity of $O(n^3)$ so we decided to run it on a subset of our dataset. Here is the two dimension result for all latitudes between longitudes -15 and +15.

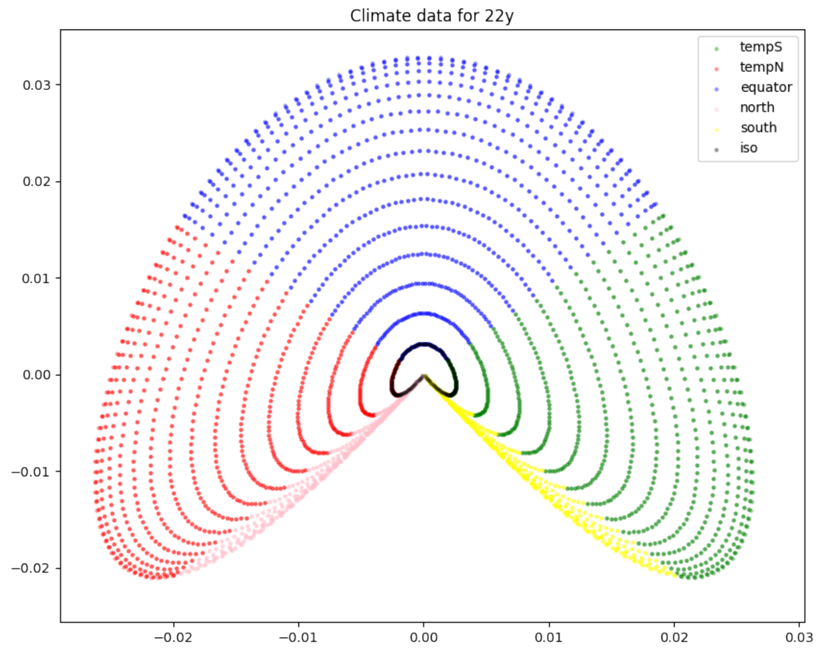


Figure 8 – Kernel PCA with two components and the RBF kernel

The black dots are all dots along one longitude, here 15. We see however that there is a degenerescence : -15 and +15 longitude are projected on the same positions. When changing the gamma

Another noticable flaw is the peak on the graph when we would expect the dots of same longitude to form a circle. The 3D visualisation of the graphic enables us to understand that peak : as we can see on the figure below, the lines that define one longitude form a sort of infinity symbol, a twisted circle, that can't render smoothly in two dimensions.

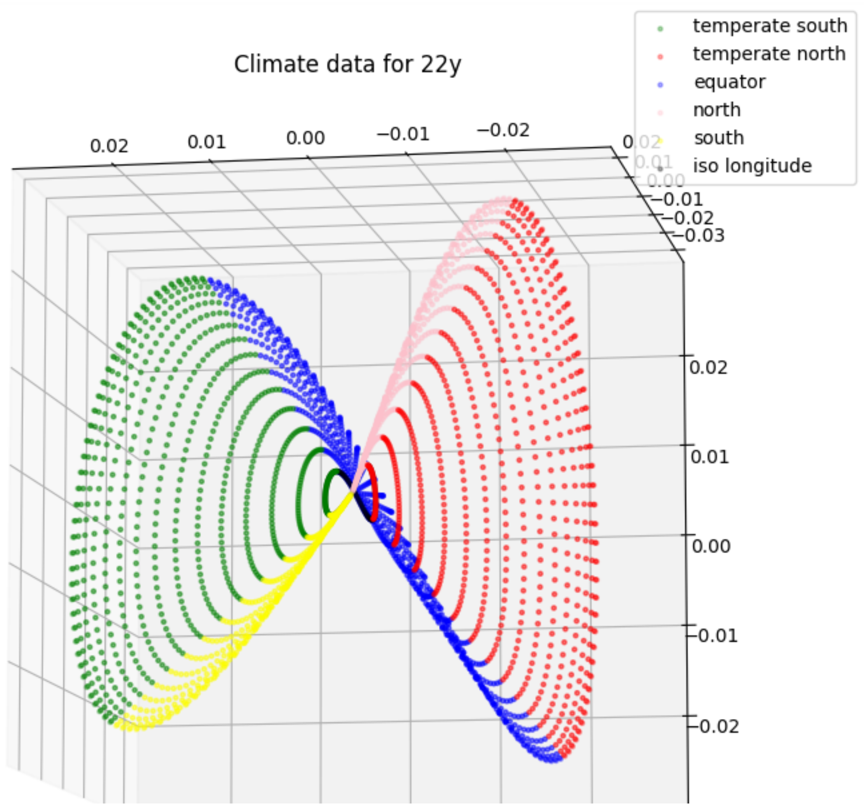


Figure 9 – 3D Kernel PCA

The dots representing the south and north poles are still closer together however, which might come from the smaller range of temperatures and climate conditions reached in these areas.

We tried to change the γ parameter, but it had no influence on the degenerescence. Changing the kernel function to a polynomial one however destroyed the degenerescence, but created a node on equatorial values as it can be observed just below. The purple and black lines represent the -15 and 15 longitudes : they are clearly different.

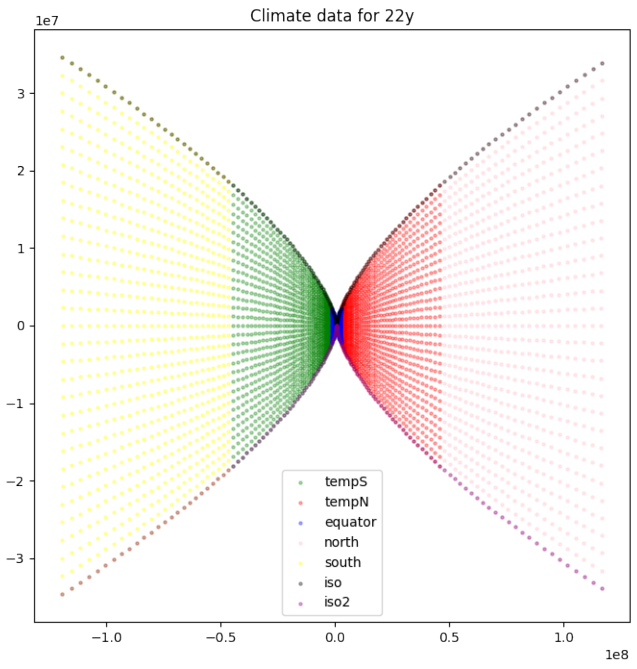


Figure 10 – Kernel PCA with polynomial kernel

4.1.3 Multidimensional Scaling - MDS

As MDS was extremeley low and seemed to use a lot of memory, we could not run the programm for more than 600 datapoints (ie 600 rows).Therefore, we used a new subset of longitudes between 0° and +5°, reducind the datasize to 546 datapoints. Figure 6 shows the results of MDS.

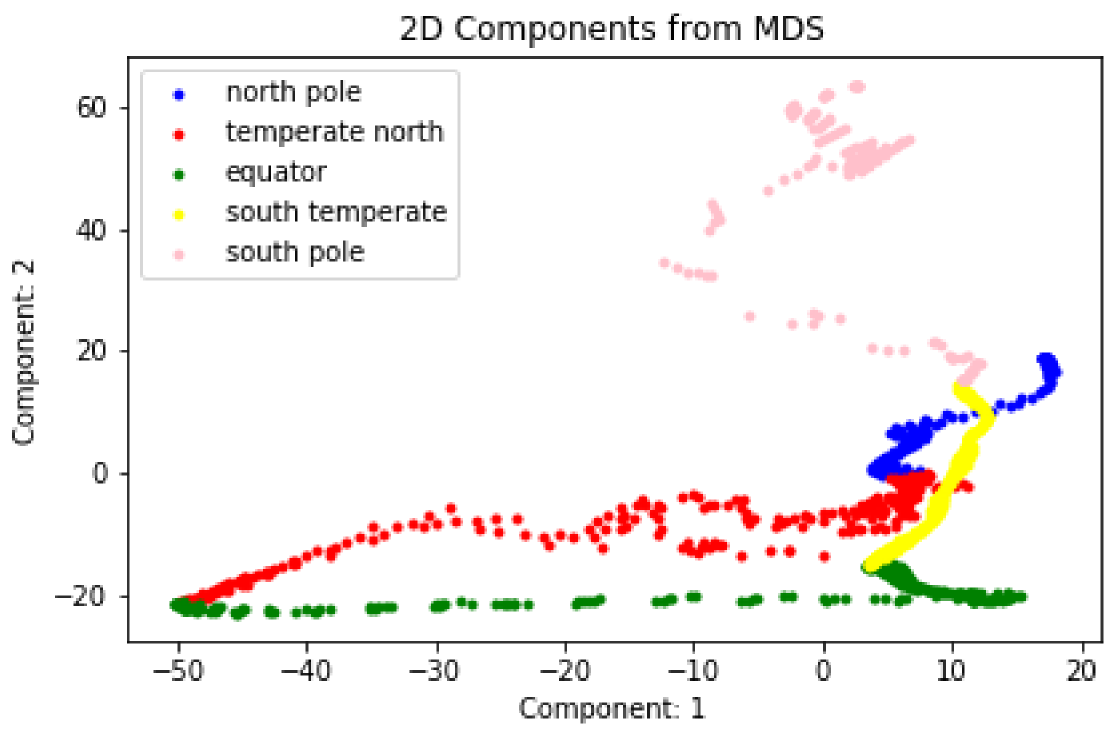


Figure 11 – results of MDS in 2D

4.1.4 isomap

As the main parameter of the method is the number of nearest neighbors taken into account, we present in figure ? ? the result of isomap for number of neighbour between 1 to and neighbors.

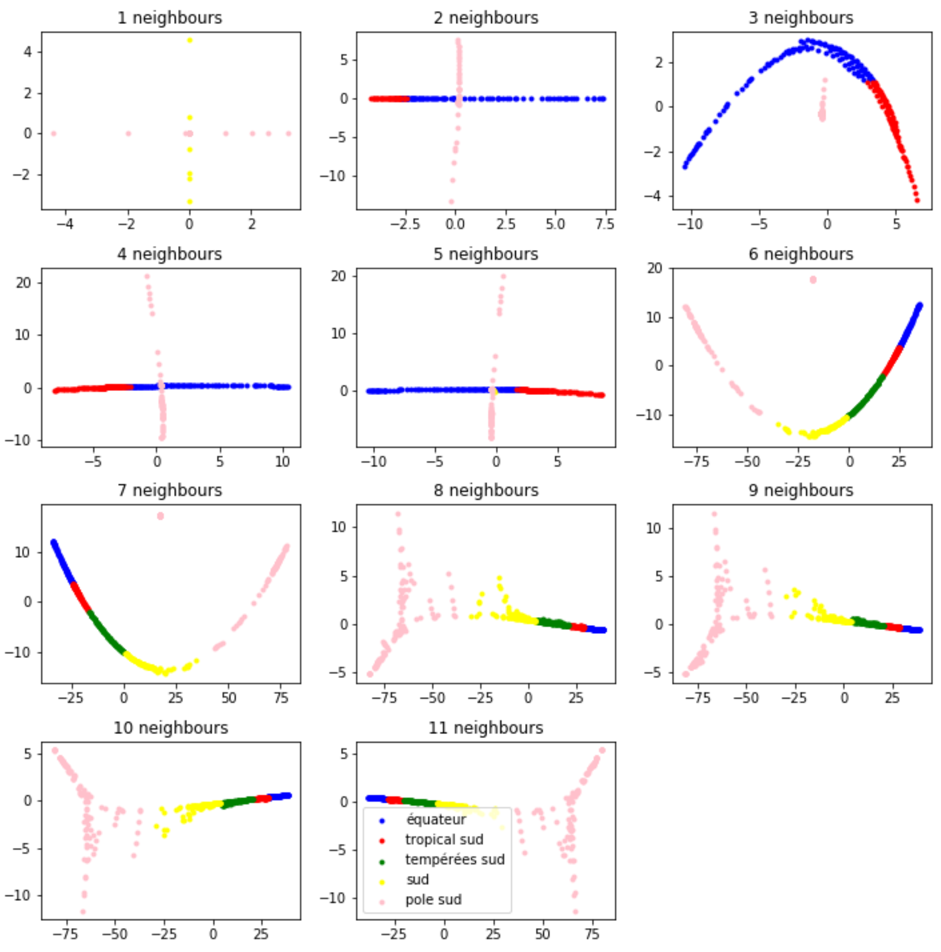


Figure 12 – results of isomap depending on the number of neighbors considered from 2 to 12

It is obvious that the number of neighbours considered strongly influence the 2D distribution of our dataset. To know which graph is the most accurate, we refer to the 'reconstruction error'. We computed this reconsruction error with respect to the number of neighbours considered. The results are presented in figure ? ?

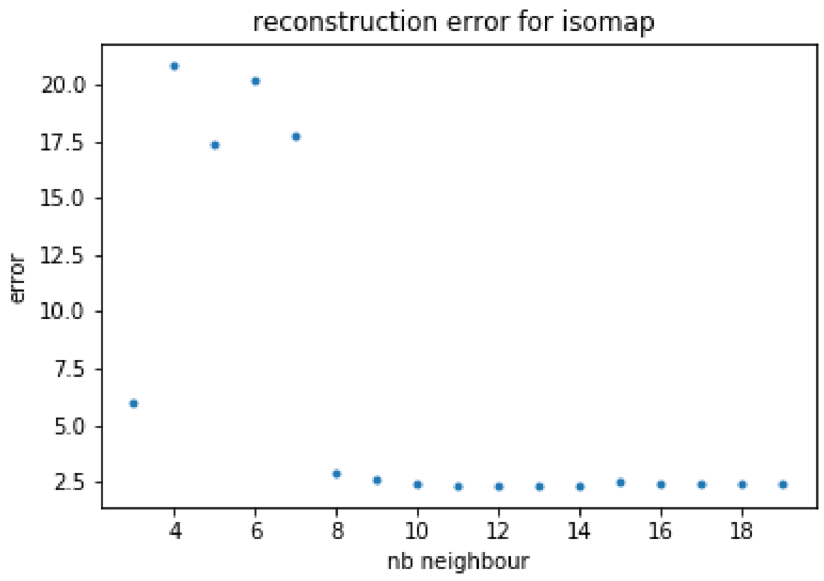


Figure 13 – reconstruction error as a function of number of neighbours considered

It is interesting to see that from eight neighbours, error seems to stabilize around 2,5. Indeed, we can see with graphs of figure 8 that from eight neighbours, graphs look quite all the same. Those are the most 'accurate' representations.

4.2 Time comparison

We implemented manually PCA, but for the other methods we used the built in fonction of the scikit package on python. PCA's running time varies linearly with the number of rows as we can see on the graphic below

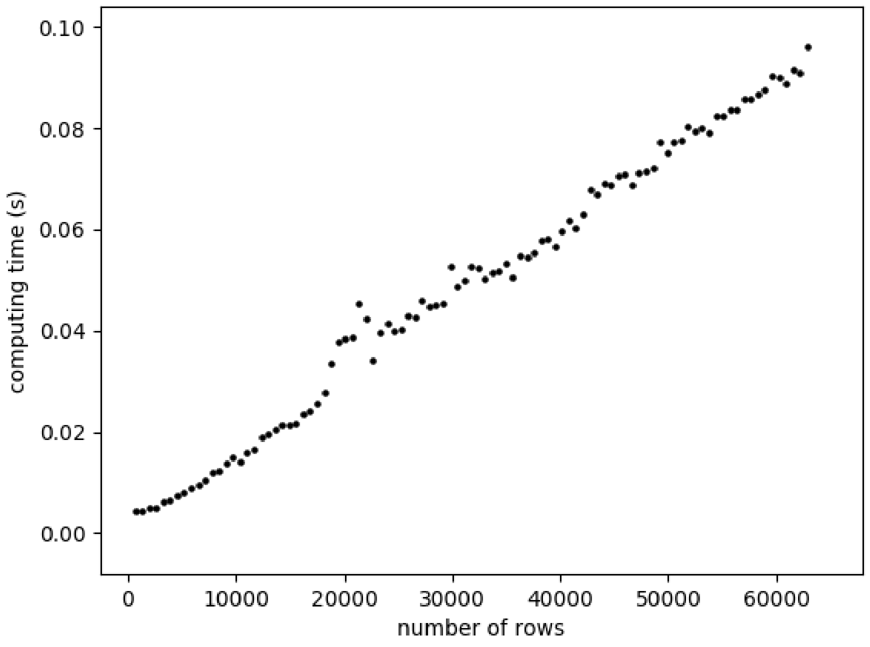


Figure 14 – running time for PCA

Kernel PCA was slower, we did not compute its running time for all the dataset, only until 12000 rows. It yields the following graph.

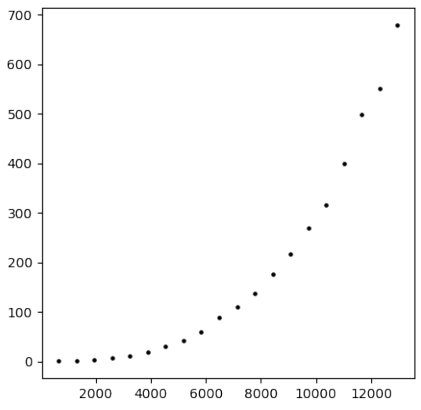


Figure 15 – running time for Kernel PCA

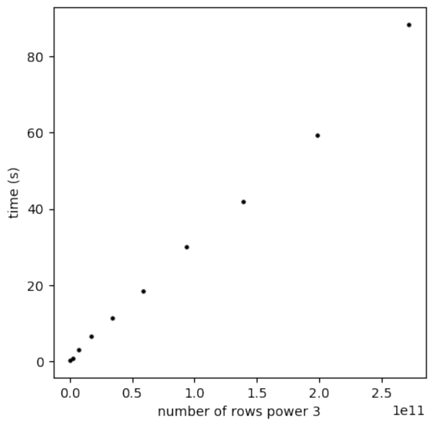


Figure 16 – Kernel PCA running time is a $O(n^3)$

In figure 10, we present computing time for different methods as a function of the number of rows. From this we can conclude that the quickest method is PCA. Then comes kernel PCA, isomap and MDS. It is quite surprising to see that isomap is quicker than MDS for the same number of rows. Indeed, isomap uses the same algorithm than MDS but also needs to compute distance with neighbors first. It is thus expected to be slower. We found an explanation on the "‘manifold guide"’ of skicit.

> In Scikit-Learn implementation, Isomap algorithm runs faster that Multi Dimensional Scaling on the S-Curve dataset [...]. In the third stage of algorithm, the implementation uses Partial Eigen Value decomposition instead of MDS which is the version proposed by the researchers

We did not use the ’S curve dataset’ as mentioned above, but we may think that due to the round shape of earth, our dataset is quite similar to the S-curve and thus Isomap runs faster than MDS. For a fixed dimension d, the following table present the complexity of every method as it appeared to us.

Method	PCA	Kernel PCA	MDS	Isomap
Complexity	$O(n)$	$O(n^3)$	$O(n^3)$	

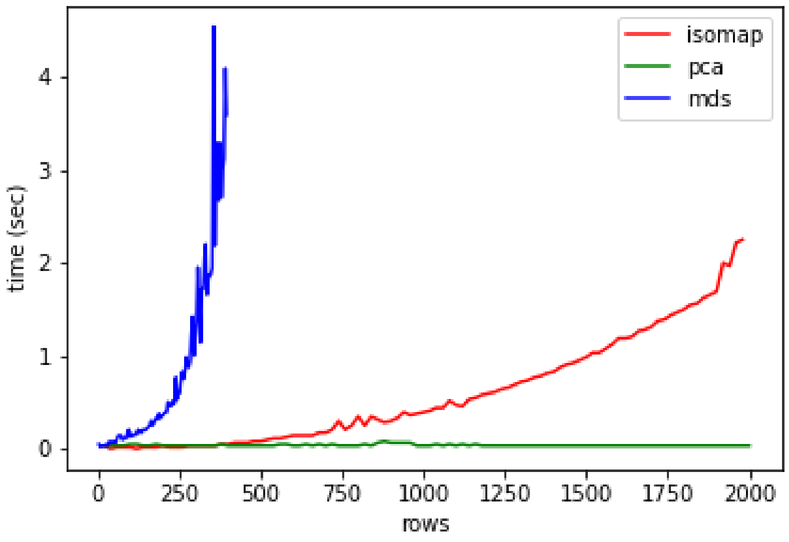


Figure 17 – computing time comparison

We also check how isomap computing time varies with the number of neighbor taken into account. Figure 11 shows that isomap efficiency time increases with the number of neighbors. So isomap efficiency of computing time may be limited by the number of neighbors.

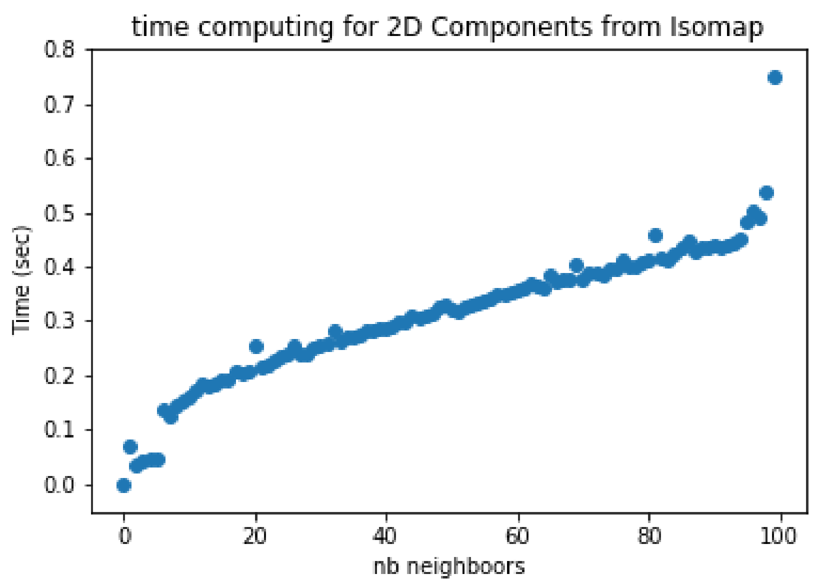


Figure 18 – computing time for isomap as a function of the number of neighbors considered

4.3 Error and preserved information

4.3.1 PCA

The main indicator of how much information has been preserved with PCA lies in eigenvalues. We plot the eigenvalues to see their relative importance in the dimension reduction.

Figure 19 – PCA with two components

We can see that with only two components, we obtain an explaned variance of ? ?, which is quite enough to interpret our dataset. PCA seems to be a relevant technic to analyse data, and in particular to establish classification.

4.3.2 MDS

One indicator to estimate of the reconstruction is accurate is the 'stress'. It is roughly the square difference between the final distance of two items in the MDS model and the true distance between them. The exact formula is :

$$s = \sqrt{\frac{\sum(d_{ij} - d(i,j))}{\sum d_{ij}}}$$

The aim of MDS is to minimize this stress. Figure 7 presents the value of stress depending on the number of lines. We can see it is relatively low (<0,002) with our dataset.

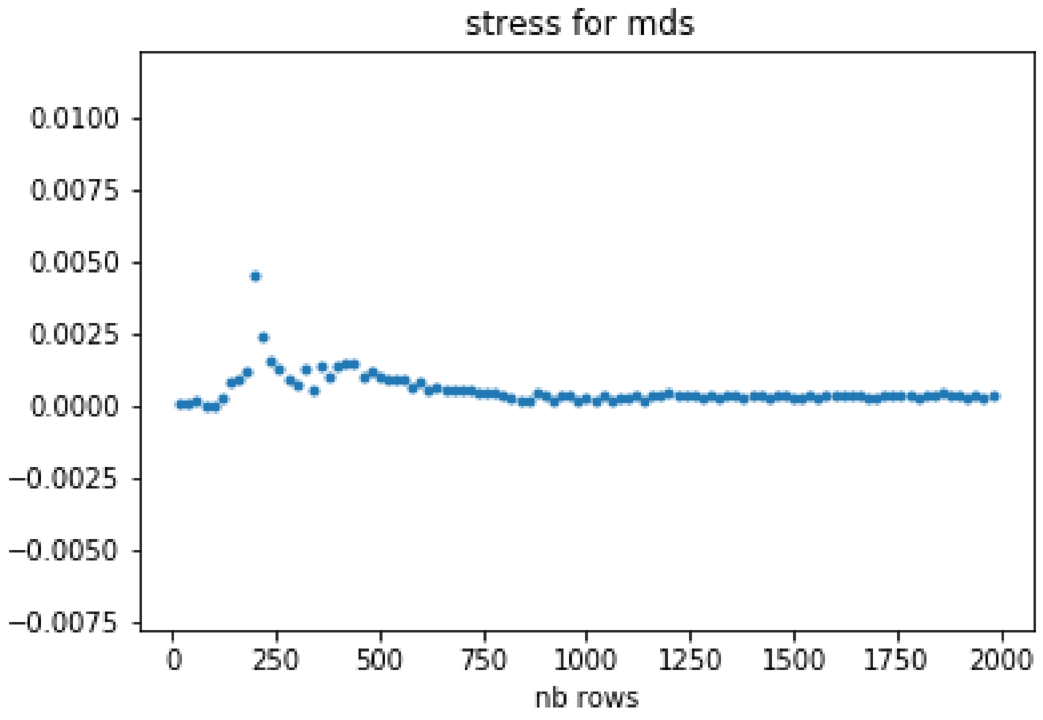


Figure 20 – stress as a function of sample size

4.3.3 isomap

We have already talked about error reconstruction of isomap, a function already embedded in scikit. We also computed the error reconstruction as a function of the number of rows in figure ? ?. This error keeps increasing with the dataset size, so isomap accuracy seems to be limited by the size of the dataset.

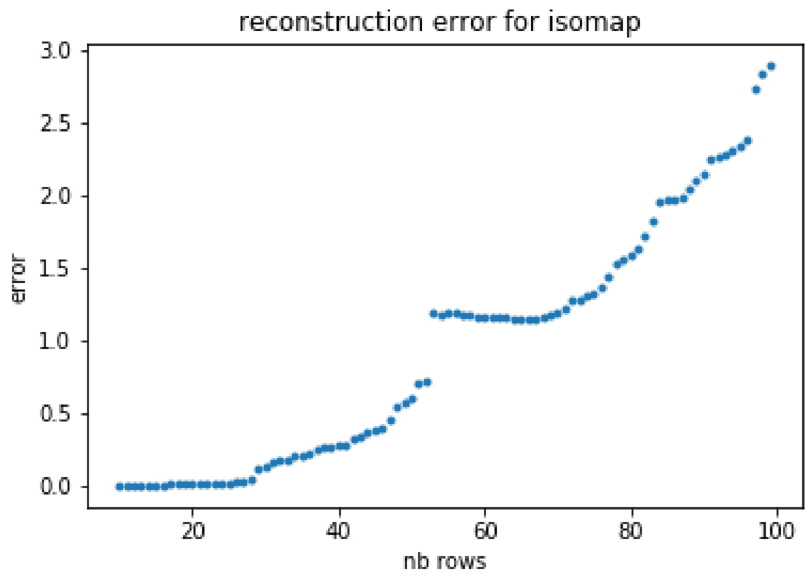


Figure 21 – reconstruction error as a function of the datasize

5 Conclusion

6 Bibliography