Machine Learning, advanced course

Assignment 1

- *Author* -
Nadir Ait Lahmouch

December 1, 2020

Contents

1 Files

This PDF includes the answers to all the assignment problems.
The python scripts for the problem 7 are in my personal Kth Github account and here's its link:
`https://gits-15.sys.kth.se/nadiral/DD2434_advanced_ML/tree/master/Assignement%201`
The github contains:

1. PCA, MDS, Isomap scripts.
2. A complete notebook where I add comparison between the above methods and their equivalent in the python libraries + a comparison between the methods using code i implemented.
3. This report in pdf.

For a complete coherent code, the notebook is better to look into.

2 Problem 1

2.1 Question (i)

Let's prove that a real symmetric matrix has real eigenvalues.
Let λ be an eigenvalue of the matrix A and x an eigenvector, we then have $Ax = \lambda x$ and $\bar{A}\bar{x} = \bar{\lambda}$ with $x \neq 0$:

$$
\begin{aligned}
\bar{A}\bar{x} = \bar{\lambda}\bar{x} &\implies (A\bar{x})^T = \bar{\lambda}\bar{x}^T \qquad \text{(A is real)} \\
&\implies \bar{x}^T A^T = \bar{\lambda}\bar{x}^T \\
&\implies \bar{x}^T A = \bar{\lambda}\bar{x}^T \\
&\implies \bar{x}^T Ax = \bar{\lambda}\bar{x}^T x \\
&\implies \bar{x}^T \lambda x = \bar{\lambda}\bar{x}^T x \\
&\implies \lambda \bar{x}^T x = \bar{\lambda}\bar{x}^T x \qquad \text{and } \bar{x}^T x \neq 0 \\
&\implies \lambda = \bar{\lambda}
\end{aligned}
$$

Therefore we can say that $\lambda \in \mathbb{R}$.

2.2 Question (ii)

First suppose x,y are eigenvectors with distinct eigenvalues λ,ν for a real symmetric matrix A.
We have:

$$Ax = \lambda x \text{ and } Ay = \nu y$$
$$< Ax, y > \; = \; < x, A^T y >$$

$\lambda < x, y > = < \lambda x, y > = < Ax, y > = < x, A^T y > = < x, Ay > = < x, \nu y > = \nu < x, y >$
$\implies (\lambda - \nu) < x, y > = 0$

And we know that $\lambda \neq \nu$, therefore $< x, y > = 0$, i.e, x and y are orthogonal.

Now for the decomposition, we can convince ourselves that A is diagonalizable as it is real and symetric there is a matrixe Q and a diagonal one D such that:

$$A = Q^{-1}DQ \implies A^T = (Q^{-1}DQ)^T \quad (1)$$
$$\implies A^T = Q^T D^T (Q^{-1})^T \quad (2)$$
$$\implies A = Q^T D (Q^{-1})^T \text{ as A is symmetric and } D^T = D \quad (3)$$
$$\implies (Q^{-1})^T D Q^T = Q^T D (Q^{-1})^T \quad (4)$$

And we know that the decomposition is unique, we can therefore conclude that $Q^T = Q^{-1}$ and that $A = QDQ^T$.

2.3 Question (iii)

Let λ be an eigenvalue of a semidefinite matrix A associated to an eigenvector x, we therefore have:

$$Ax = \lambda x \implies x^T Ax = x^T \lambda x \quad (5)$$
$$\implies x^T Ax = x^T x \lambda \quad (6)$$
$$\implies x^T x \lambda \geq 0 \text{ as } x^T Ax \geq 0 \text{ (A is semidefinite)} \quad (7)$$
$$\implies \lambda \geq 0 \text{ as } x^T x \geq 0 \quad (8)$$

Therefore, a semidefinite matrix has non-negative eigenvalues.

2.4 Question (iv)

We have A is a PSD and we know from previous questions that $A = Q\Lambda Q^T$. We know that D is diagonal matrix with non-negative values so $\Lambda = \sqrt{\Lambda}\sqrt{\Lambda}$. Therefore :

$$A = Q\sqrt{\Lambda}\sqrt{\Lambda}Q^T \implies A = Q\sqrt{\Lambda}(Q\sqrt{\Lambda})^T \tag{9}$$
$$\implies A = VV^T \tag{10}$$

Let e_i be the vector where the i_{th} element is equal to 1. We have:

$$D_{ij} = (e_i - e_j)A(e_i - e_j)^T \implies D_{ij} = (e_i - e_j)VV^T(e_i - e_j)^T \tag{11}$$
$$\implies D_{ij} = (e_iV - e_jV)(e_iV - e_jV)^T \tag{12}$$
$$\implies D_{ij} = \| e_iV - e_jV\|_2^2 \tag{13}$$

Therefore, $v_i = e_iQ\sqrt{\Lambda}$.

3 Problem 2

Yes, a single SVD is enough to perform PCA on both rows and columns of a mtarix. Indeed, performing PCA on i.e the rows of $Y = U\Sigma V^T$ is done by choosing the W that maximizes:

$$tr(Y^TWW^TY) = tr(V\Sigma U^TWW^TU\Sigma V^T)$$

And the solution to that is $W = U_k$ (The k columns associated to the biggest k eigenvalues of Y). Performing PCA now on the columns would be, in the same logic, performing it on $Y^T = V\Sigma^{'}U^T$, i.e, choosing this time W that maximizes:

$$tr(YWW^TY^T) = tr(U\Sigma^{'}V^TWW^TV\Sigma^{'}U^T)$$

And this time, the solution would be $W = V_k$.
Therefore we would do $X = V_k^TY^T$ where we previously did $X = U_k^TY$.

4 Problem 3

We have :

$$tr(Y^T W W^T Y) = tr(W W^T Y Y^T) \tag{14}$$
$$= tr(W^T Y Y^T W) \tag{15}$$
$$= tr(W^T U \Sigma V^T V \Sigma U^T W) \tag{16}$$
$$= tr(W^T U \Sigma^2 U^T W) \tag{17}$$

Let $M = U^T W$, therefore we have:

$$tr(Y^T W W^T Y) = tr(M^T \Sigma^2 M) \tag{18}$$

We have Σ^2 is a diagonal matrix with eigenvalues $\sigma_1^2 \geq \sigma_2^2 \geq ... \geq \sigma_d^2$ associated to the corresponding orthonormal eigenvectors $v_1, v_2, v_3, ..., v_d$ therefore we can write $\Sigma^2 = \sum_{j=1}^{d} \sigma_j^2 v_j v_j^T$.
We have $M \in \mathbb{R}^{dxk}$ and let be $m_1, m_2, m_3, ..., m_k$ the columns of this matrix, therefore:

$$tr(M^T \Sigma^2 M) = \sum_{i=1}^{k} m_i \Sigma^2 m_i^T \tag{19}$$
$$= \sum_{i=1}^{k} m_i (\sum_{j=1}^{d} \sigma_j^2 v_j v_j^T) m_i^T \tag{20}$$
$$= \sum_{j=1}^{d} \sigma_j^2 \sum_{i=1}^{k} \langle v_j, m_i \rangle^2 \tag{21}$$
$$= \sum_{j=1}^{d} \sigma_j^2 a_j \tag{22}$$

We have:

$$a_j = \sum_{i=1}^{k} \langle v_j, m_i \rangle^2 \leq \sum_{i=1}^{d} \langle v_j, m_i \rangle^2 = 1$$

(we extended $m_1, m_2, m_3, ..., m_k$ to an othnormal basis $m_1, m_2, m_3, ..., m_d$)

Finally, since $v_1, v_2, \ldots, v_d$ is an orthonormal basis for $\mathbb{R}^d$

$$\sum_{j=1}^{d} a_j = \sum_{j=1}^{d} \sum_{i=1}^{k} \langle v_j, m_i \rangle^2 = \sum_{i=1}^{k} \sum_{j=1}^{d} \langle v_j, m_i \rangle^2 = \sum_{i=1}^{k} \|m_i\|_2^2 = k$$

Now, we know that $\sigma_1^2 \geq \sigma_2^2 \geq \sigma_3^2 \geq ... \geq \sigma_d^2$ and that $0 \leq a_j \leq 1$ and that $\sum_{j=1}^{d} a_j = k$, therefore the maximum value of $\sum_{j=1}^{d} \sigma_j^2 a_j$ is achieved when $\sum_{j=1}^{d} \sigma_j^2 a_j = \sum_{j=1}^{k} \sigma_j^2$ (when $a_1, ..., a_k = 1$ and $a_{k+1}, ..., a_d = 0$).
Then:

$$tr(Y^T W W^T Y) \leq \sum_{j=1}^{k} \sigma_j^2$$

And this maximum is achieved when $W = U_k$. Indeed, we have :

$$tr(Y^T W W^T Y) = tr(M^T \Sigma^2 M) \quad \text{with} \quad M = U^T W \tag{23}$$

$$= \sum_{j=1}^{d} \sigma_j^2 \sum_{i=1}^{k} \langle u_j, w_i \rangle^2 \tag{24}$$

Let's assume $W = U_k$, therefore:

$$tr(Y^T W W^T Y) = \sum_{j=1}^{d} \sigma_j^2 \sum_{i=1}^{k} \langle u_j, u_i \rangle^2 \tag{25}$$

$$= \sum_{j=1}^{k} \sigma_j^2 \langle u_j, u_j \rangle^2 \tag{26}$$

$$= \sum_{j=1}^{k} \sigma_j^2 \tag{27}$$

And with this we just proved both claims (i) and (ii).

5 Problem 4

We have $s_{ij} = -\frac{1}{2}\left(d_{ij}^2 - d_{1i}^2 - d_{1j}^2\right)$ with $d_{ij} = \| y_i - y_j \|_2^2$, therefore:

$$s_{ij} = -\frac{1}{2}\left(\| y_i - y_j \|_2^2 - \| y_1 - y_i \|_2^2 - \| y_1 - y_j \|_2^2\right) \tag{28}$$

$$= -\frac{1}{2}\sum_{k=1}^{d}(y_{ik} - y_{jk})^2 - (y_{1k} - y_{ik})^2 - (y_{1k} - y_{jk})^2 \tag{29}$$

$$= \sum_{k=1}^{d} y_{ik}y_{jk} - y_{1k}y_{ik} - y_{1k}y_{jk} + y_{1k}^2 \tag{30}$$

$$= \sum_{k=1}^{d}(y_{ik} - y_{1k})(y_{jk} - y_{1k}) \tag{31}$$

$$s_{ij} = \langle y_i - y_1, y_j - y_1 \rangle \tag{32}$$

Let $Z = Y - Y_1$ where Y_1 is the matrix where all the columns are the first column of Y. In other words, Z is a geometric transformation of Y (translation by the vector y_1. Therefore, we have :

$$s_{ij} = \langle y_i - y_1, y_j - y_1 \rangle \tag{33}$$

$$\implies S = Z^T Z \tag{34}$$

Also, we know that if Y has 0 reconstruction error, so does Z, because the geometric transformation does not change distances. Then, the similarity matrix $S = Z^T Z$ is also a correct estimation for the Gram matrix because it can be used to find Z, which is a valid solution.

6 Problem 5

One way to prove that is by proving that the solution given by MDS is maximizing the trace we want to maximize in PCA.
By MDS, we obtain $S = R\Lambda R^T$ with Λ having ordered $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_n$ eigenvalues of S. And we have:

$$S = Y^T Y = X^T X \implies X = I_{k,n}\Lambda^{\frac{1}{2}} R^T \tag{35}$$

Therefore, if we replace in the trace we try to maximize in PCA:

$$\begin{aligned} tr(Y^T WW^T Y) &= tr((YW)^T W^T Y) && (36)\\ &= tr(X^T X) && (37)\\ &= tr((I_{k,n}\Lambda^{\frac{1}{2}} R^T)^T I_{k,n}\Lambda^{\frac{1}{2}} R^T) && (38)\\ &= tr(R\Lambda^{\frac{1}{2}} I_{n,k} I_{k,n}\Lambda^{\frac{1}{2}} R^T) && (39)\\ &= tr(I_{k,n}\Lambda^{\frac{1}{2}} R^T R\Lambda^{\frac{1}{2}} I_{n,k}) && (40)\\ &= tr(I_{k,n}\Lambda I_{n,k}) && (41)\\ &= \sum_{j=1}^{k} \lambda_j && (42) \end{aligned}$$

From another side, we know the decomposition of Y from PCA:

$$Y = U\Sigma V^T \text{ where } \Sigma \text{ is a diagonal matrix with the ordered values}$$
$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$$
$$\implies S = Y^T Y = V^T \Sigma^2 V$$

Let's now prove that $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$ are eigenvalues for the matrix S. For that, let v_i the ith column of the matrix V. We have:

$$\begin{aligned} Sv_i &= V^T \Sigma^2 V v_i && (43)\\ &= V^T \Sigma^2 e_i \text{ where } e_i \text{ has 1 in the ith element and zeros in the rest} && (44)\\ &= V^T s_i \text{ where } s_i \text{ has } \sigma_i^2 \text{ in the ith element and zeros in the rest} && (45)\\ Sv_i &= \sigma_i^2 v_i && (46) \end{aligned}$$

We just proved that $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$ are ordered eigenvalues of S, and $\lambda_1, \lambda_2, ..., \lambda_n$ are oredered eigenvalues too, and we know that eigenvalues are unique therefore they're equal and:

$$\sum_{j=1}^{k} \lambda_j = \sum_{j=1}^{k} \sigma_j^2 \text{ (maximum value of our trace based on Problem 3)}$$

The solution of MDS is indeed giving the maximum value of the trace we try to maximize in PCA, therefore performing MDS is same as performing PCA.

Computational complexity
Given n data points, each represented with m features, the complexity of PCA is $O(nm.min(m,n) + nkm)$ because the complexity of svd o a matrix

nxm is $O(nm.min(m,n))$ (look 9) and the complexity of the multiplication is $O(nkm)$ with k is the dimension of the embedding. And for classical MDS, it's $O(n^3)$ with n being the number of data points, since it requires finding all the eigenpairs of a matrix the same size as the distance matrix.
Therefore, it's better to use PCA because :

$$O(nm.min(m,n) + nkm) = O(nm.min(m,n)) < O(n^3)$$

7 Problem 6

Let D a set of $2p + 2$ data points clustered into two neighborhoods like the image below shows.

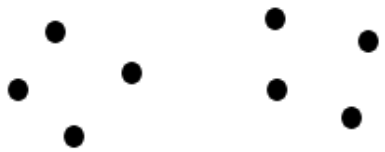


Figure 1: Our set of datapoints p = 3

The Isomap method constructs a graph G where each data point is connected to it's nearest k points, let $k = p$ here. In the graph, each point will be connected to its p nearest neighbors, i.e, to all the points in its cluster. Therefore, the graph in our case will be as follow:

We can see that the graph is disconnected. We can conclude that the Isomap method may yield a disconnected graph.

Heuristic :
One can say that a solution to the disconnectivity problem would be gradually increasing k and test each time until we reach a disconnected graph. But that can be yield to unconclusive results as there's no apparent bind between k and the good performance for a an embedding. Then, creating a function that

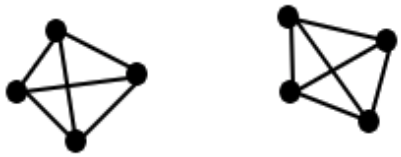


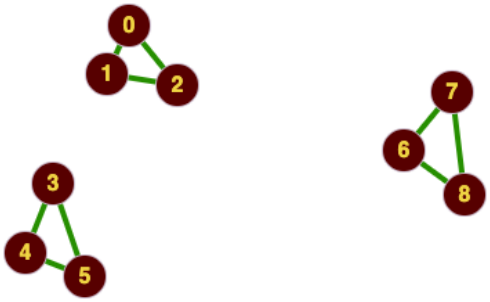
Figure 2: Graph G

connects the disconnected graphs seems more reasonable.
So this was my idea to do that:

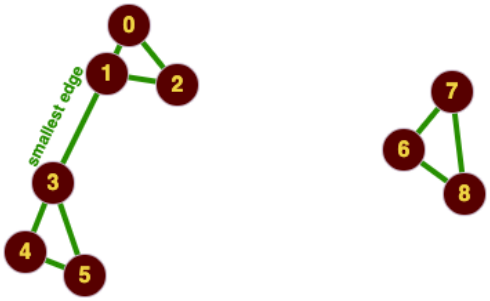
1. Store all the disconnected components of a graph in a vector.
2. Choose a random component.
3. Connect it to closest component from the vector : we would do that by finding the smallest edge that connects it with each of the components and take the component that yields to the minimum edge distance wise. We compute this edge by an iteration on all the possible couples of vertices from two components and taking their minimum.
4. Restart the operation **3** but this time by choosing the new connected component.

In the end, we will get a connected graph. I believe this might be a good solution, because it yields to less complex graph than adding the number of neighbors. Also, it minimizes the number of edges that we should add to connect the graph.

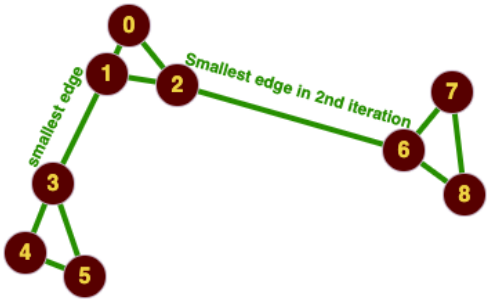
Illustration :
Let's suppose we are starting from the graph below:



We start from the component with the vertices 0, 1, 2 and look for the closest component and connect them with the algotihm described above.



Then we redo the same operation but this time starting from the newly connected graph having as vertices 0, 1, 2, 3, 4, 5. And here's the result.



8 Problem 7

8.1 Data processing

Before starting coding the dimension reduction methods, i had to deal with the data. What i did is transforming the numerical feature **'legs'** into 6 binary categorical features using **pandas** function **get_dummies()**.

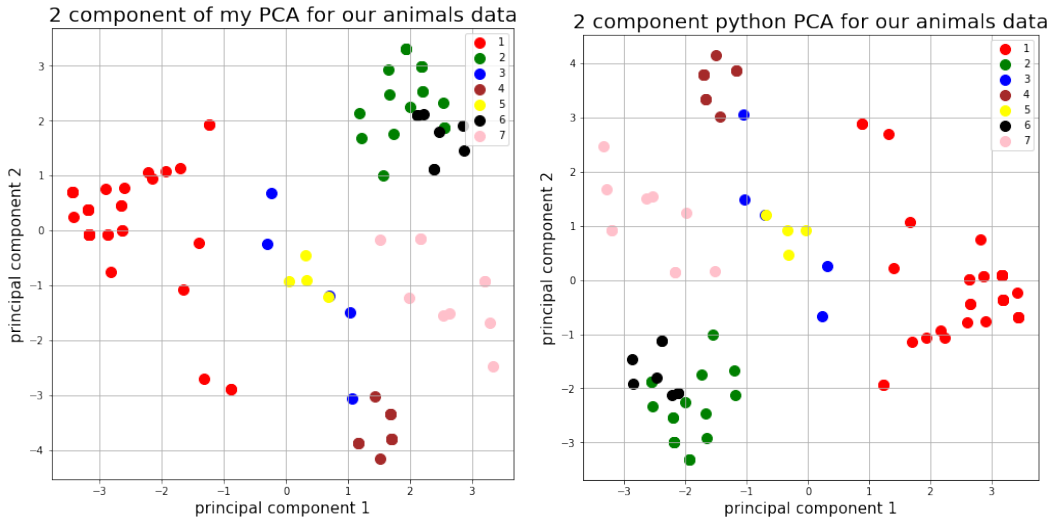
8.2 PCA

I implemented the PCA algorithm using the svd and compared it to the PCA function present in sklearn.decomposition.
Here's my code :

```
u, s, vt = np.linalg.svd(x, full_matrices=True)
v_2 = np.transpose(vt)[:,:2]
pc = np.dot(x, v_2)
```

Notice that here, we should use vt and not u like in the slides because the data is transposed.
Before computing the two first principal components, i centered the data using StandardScaler().

And here's the plot :



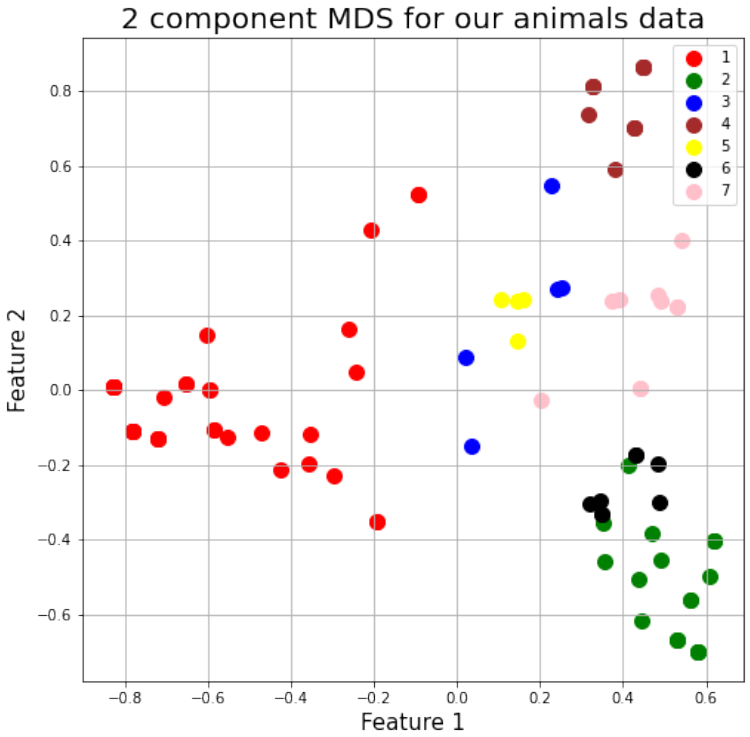
We can see that the embeddings are equivalents, the new data is however rotated but we know that geometric transformations are not a problem.

8.3 MDS

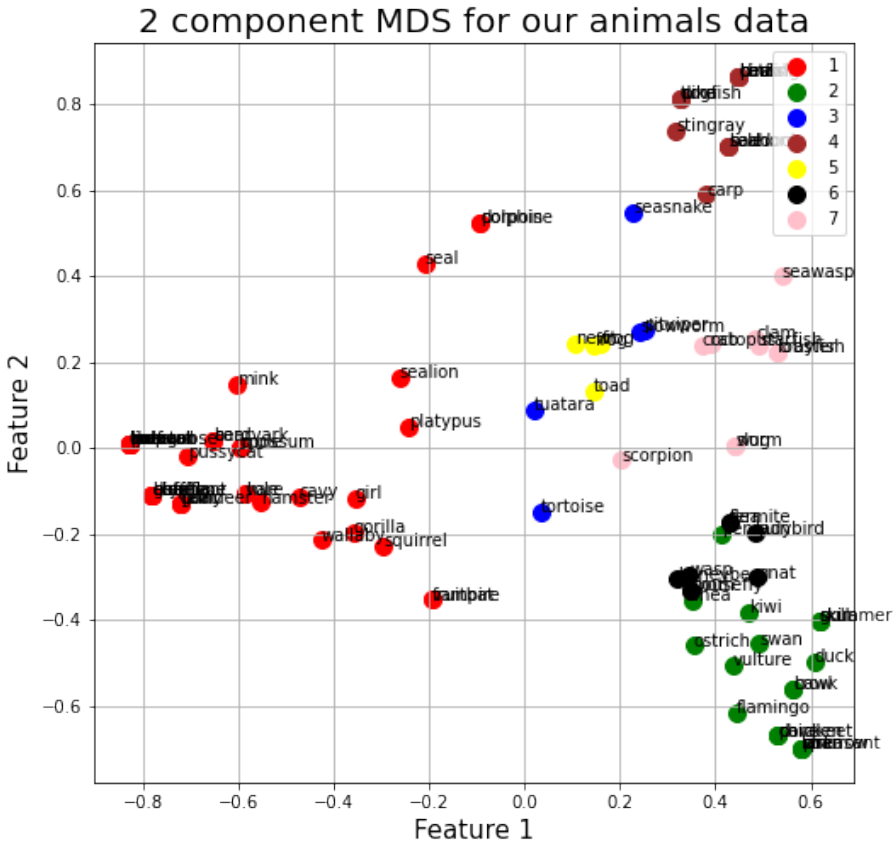
I implemented the classical MDS using a distance matrix (i computed the distance matrix using the function **pdist** with the **euclidian** function). We first compute the similarity matrix **S** using the the distance matrix **D** and the double centering trick. Then, we use its decomposition to compute to transformed data **X**. Here's my code for the MDS function :

```
def MDS(D, k):
  n = len(D)
  ones = np.ones((n, n))

  S = - (1/2) * (D - (1/n) * np.dot(D, ones) - (1/n) * np.dot(ones, D) + (1/n**2)
  eigenvalues, eigenvectors = lg.eig(S)
  Lambda = np.diag(np.abs(np.real(eigenvalues)))
  U = np.real(eigenvectors)

  I = np.eye(k, n)
  Lambda_sqrt = np.sqrt(Lambda)
  U_transpose = np.transpose(U)

  return np.transpose(np.dot(I, np.dot(Lambda_sqrt, U_transpose)))
```

Plotting now the results :



I choose not to use annotation so the data can be more readable, but here's what it looks like with annotations.



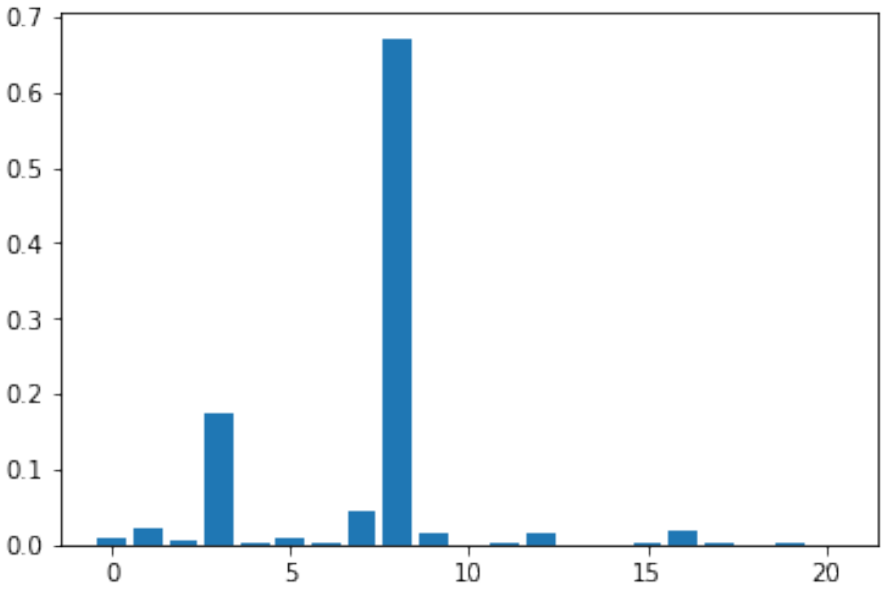
8.3.1 Attribute importance

One way to facilitate the embedding is to remove some unimportant features prior to the MDS. One way to do that is to compute the features importance and then remove the least important ones. I computed this importance vector using **RandomForest**, this is the code for it:

```
from sklearn.ensemble import RandomForestRegressor
from matplotlib import pyplot

model = RandomForestRegressor()
model.fit(data, types)
importance = model.feature_importances_

pyplot.bar([x for x in range(len(importance))], importance)
```

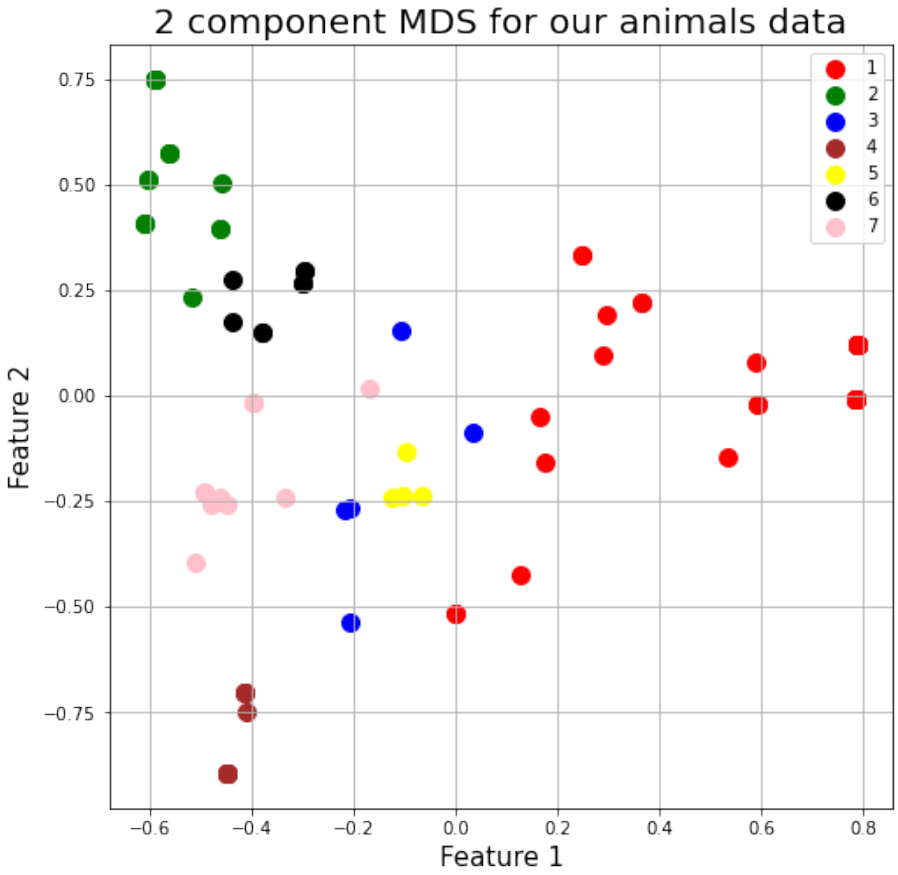
```
pyplot.show()
```

And here's the plot illustrating this vector :



Attributes importance Histogram

We notice that the 13th, 14th and 18th features have an importance of 0, so i decide to remove them from the data before redoing the MDS.
And here's what it looks like after that manipulation:



We can see a small improvement. Indeed, we now can clearly see a separation between the data points belonging to the second type (color green) and sixth type (color black).

There's must be other ways to do this task, for example, we can use Linear Regression or Decision trees to compute the importance vector, or maybe merge them and computing the mean from these three methods.
Also, instead of for example removing the feature with less importance, we can weight the data points with the importance vector.

8.4 Isomap

The first step to implement Isomap is create a graph of the datapoints and their neighbors. To do that, I simply used the library function ***kneighbors_graph*** from **sklearn.neighbors**.
If you state the parameter *mode* as *distance*, in the function above, it will return the distances between neighbors according to the given metric. Also, you have to indicate the number of neighbors of each vertice in the graph.

The second step of the Isomap algorithm is to compute a distance matrix using a shortest path algorithm, here we will use the Floyd-Warshall algorithm.
And to do that i used the function ***graph_shortest_path*** from the library **sklearn.utils.graph_shortest_path**.
The last step is to use MDS (described in the previous section) to compute the embedding. Here's what the code looks like:

```
from sklearn.neighbors import kneighbors_graph
from sklearn.utils.graph_shortest_path import graph_shortest_path

G = kneighbors_graph(np.array(data), 16, mode='distance')
D_iso = graph_shortest_path(G, directed = False , method='FW')

X_iso = MDS(D_iso, 2)
```

8.4.1 Connectivity of the graph G

One problem we can encounter when we use Isomap is a graph disconnectivity (as proven in problem 6). Such a problem can yield to a bad embedding, so we have to make sure that our graph is connected before computing the distance matrix.
And to do that, I implemented a function **isConnected()** that takes a graph as an argument and return whether it's connected or not. This function uses the basic **DFS** algorithm to browse the graph and mark each visited vertice, in the end we test whether the number of visited vertices is equal to the number of data points. Here's what the code looks like:

```
def dfs(G , vertice, visited):
  visited[vertice] = True
  n = len(G)
  for v in range(n):
    if G[vertice, v] != 0  and (not visited[v]):
      dfs(G, v, visited)

def isConnected(G):
  visited = {}
  n = len(G)
  for i in range(n):
    visited[i] = False
  dfs(G, 0, visited)
  for i in range(n):
    if not visited[i]:
      return False

  return True
```

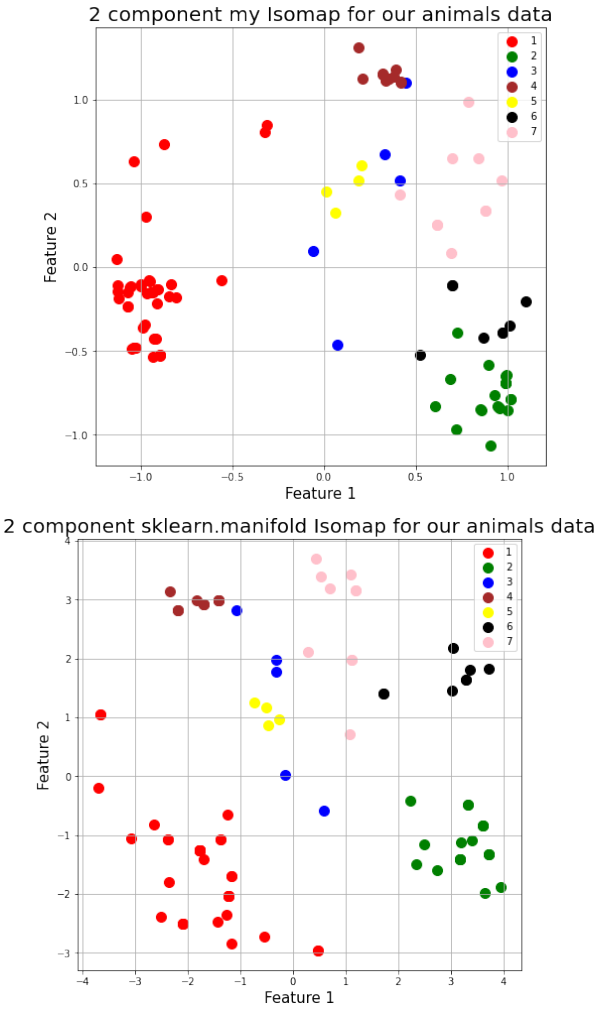
Using this function and our dataset, the least number of neighbors to have a connected graph is **16**.

8.4.2 Reflexion on duplicates

Another problem in building proper graphs is the high number of duplicated data points. Indeed, I noticed an important number of duplicates. I then asked myself whether it has a consequence on building the graph.
So i decided to sort of "delete" the duplicates point to see if the min number of neighbors before getting a connected graph would change. One way to smartly do that is to give D the distance matrix (using **short_path**) instead of the graph to my **isConnected** function. Indeed, in that matrix, the distance between duplicates is 0. Therefore, only non duplicates points will be counted as neighbors. The result of the least number of neighbors is **10** instead of **16**. But as I am not sure that what I did is a good manipulation, i will stick to 16 neighbors for the rest of the problem.

8.4.3 Results of Isomap

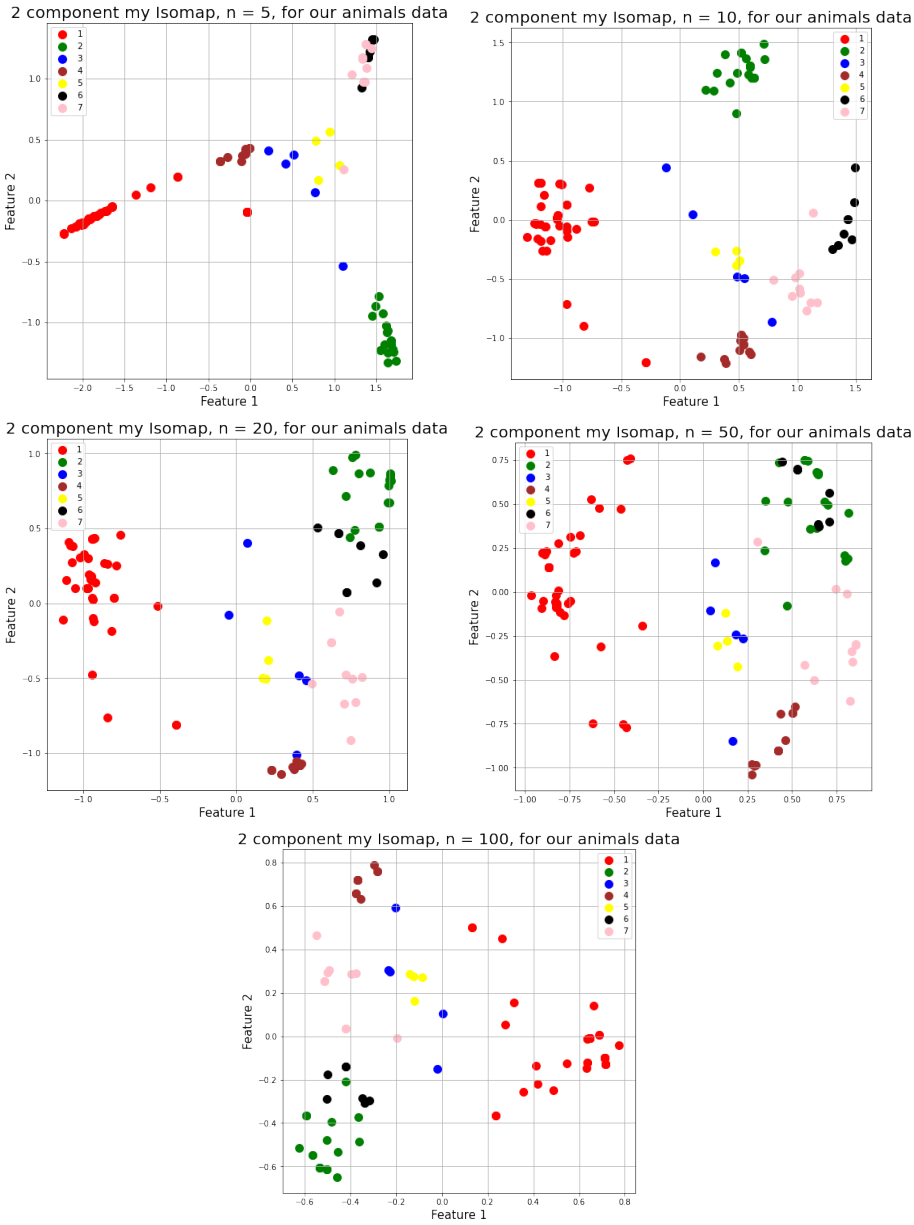
After choosing the number of neighbors, let's compute the embedding and plot the results. I will also plot the **sklearn.manifold** Isomap function as a comparison.



We can notice that clustering is pretty much the same with a rotation, with more accuracy in the **sklearn.manifold** Isomap, but that's probably due to more complex manipulations in the implementation.

8.4.4 Exploring other amount of neighbors

Here's the Isomap embedding using different amount of neighbors.



Conclusion: We notice that the embedding depends on the number of neighbors but we can't, at least from the observations above, state a rule that

binds these parameter with how good the embedding is. We should define and use a metric (error) to compute that.

8.4.5 Remark

Instead of increasing the amount of neighbors until we reach a connected graph, we can use a function to connect (heuristic in problem 6), and that's what probably does the already implemented Isomap functions in the python libraries.

8.5 Comparison of the methods

Let's now proceed to a comparison of the methods we used. But, as we showed in problem 5, PCA and MDS are equivalent, and with PCA having a lower computational complexity, using it is better.
So let's compare PCA and Isomap. One way to compare them is by computing the reconstruction error. For that, I used the method reconstruction_error() already present for the **sklearn.manifold** Isomap. For PCA, we can compute the inverse transform and then compute the mean of the difference between the data and the inverse transform.
Here's the code for both:

```
# Reconstruction error for PCA
X_projected = pca.inverse_transform(principalComponents)
pca_error = ((x - X_projected) ** 2).mean()

# Reconstruction error for Isomap
isomap_error = embedding.reconstruction_error()
```

And here's the results :

$$\text{pca\_error} = \mathbf{0.54}$$
$$\text{isomap\_error} = \mathbf{1.76}$$

Since the PCA objective places much more emphasis on preserving the large pairwise distances, PCA achieves lower reconstruction error.
We can create a metric to compute how well the local structures are preserved.

Indeed, reconstruction error may be an unfair metric, since Isomap aims to preserve local distances. One way to do it is to see how many of each point's k-nearest neighbors are in common between the data-space and the embedding (for both PCA and Isomap).
You can find the code I implemented to compute this metric in the **Methods comparison** section in the assignment notebook.
And here's the results :

Neighbors similarity between data and PCA embedding: **3.70**
Neighbors similarity between data and Isomap embedding: **2.27**

Here also, PCA surpasses Isomap even if it's not intuitive. Indeed, Isomap should perform better on local structures when it comes to complex manifolds. But because it's less performant than PCA, we can conclude that PCA is enough to have a good embedding for our data in the 2D space.

9 References

1. Vinita Vasudevan, M.Ramakrishna "A Hierarchical Singular Value Decomposition Algorithm for Low Rank Matrices" `https://arxiv.org/pdf/1710.02812.pdf`