DD2434/FDD3434 Machine Learning, Advanced Course

Assignment 2B

Abdessamad BADAOUI

1 Multidimensional Scaling (MDS) and Isomap

1.1 Question 2B.1

We know that the 'double centering' trick enables us to estimate the following expression for $s_{ij} = \frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$ (the reason we have to transform the matrix is to turn pairwise Euclidean distances into pairwise inner products of vectors). This is achieved by subtracting from each entry of matrix $\mathbf{D}$ the mean of the corresponding row and the mean of the corresponding column, and then adding back the mean of all entries. In fact, calculating the dot product of two vectors based on their distance changes when we translate the points while preserving the distance between the two points. Therefore, we require a reference point. The intuition behind this is that we aim to stabilize the position of the points by relocating the origin to the mean of the points. This way, we establish a fixed reference point to accurately calculate the similarity between the points.

Since the distance matrix is symmetric, performing double centering results in a symmetric matrix, because we subtract and add the same quantity to d_{ij} and d_{ji} to form s_{ij} and s_{ji}, respectively. Ultimately, we obtain a symmetric matrix suitable for eigen-decomposition, allowing us to apply Multidimensional Scaling (MDS) with the resulting matrix S.

1.2 Question 2B.2

The first-point trick now involves selecting the first point as our reference instead of the center of the data to calculate sii and sjj. It becomes evident that by relocating the origin to the first point, sii equals the squared distance between the first point and the ii-th point, denoted as d_{1i}^2. Substituting this into our formula $s_{ij} = \frac{1}{2}(d_{ij}^2 - s_{ii} - s_{jj})$, we obtain the expression for the first-point trick . It's important to note that even though we may obtain different solutions (as before when the origin was relocated to the mean of the data, and now when it's relocated to the first point), these solutions only differ in terms of translation. Both solutions are valid as the distances are preserved.

1.3 Question 2B.3

Let's consider the case where $\mathbf{Y}$ is known, and let's assume that it is centered (and if it's not, we can center it as follows : $\mathbf{Y} \leftarrow \mathbf{Y} - \dfrac{1}{n}\mathbf{Y}\mathbf{1_n}\mathbf{1_n}^T$).

Let's consider the Singular Value Decomposition of $\mathbf{Y}$:

$$\mathbf{Y}_{d\times n} = \mathbf{U}_{d\times d}\begin{pmatrix}\mathbf{\Sigma}\\ \mathbf{0}\end{pmatrix}\mathbf{V}_{n\times n}^T \tag{1}$$

PCA algorithm map data to k-dimensional space by :

$$\mathbf{X}_{k\times n}^{PCA} = \mathbf{U}_k^T\mathbf{Y}_{d\times n} \tag{2}$$

where $\mathbf{U}_k$ is of size $d \times k$ and it represents the k columns of $\mathbf{U}$ that correspond to the k largest singular values.

Let's have a look now on the MDS algorithm : We form the similarity matrix as :$\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$, and we can use the SVD decomposition to calculate it:

$$\begin{aligned}\mathbf{S} &= \mathbf{Y}^T\mathbf{Y}\\ &= \mathbf{V}\begin{pmatrix}\mathbf{\Sigma} & \mathbf{0}\end{pmatrix}\mathbf{U}^T\mathbf{U}\begin{pmatrix}\mathbf{\Sigma}\\ \mathbf{0}\end{pmatrix}\mathbf{V}^T \qquad \mathbf{U}^T\mathbf{U} = \mathbf{I}\\ &= \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T\end{aligned} \tag{3}$$

We get thus the eigen-decomposition of S, where the columns of $\mathbf{V}$ (right singular vectors of $\mathbf{Y}$) are the eigenvectors of $\mathbf{Y}^T\mathbf{Y}$. A k-dimesnional representation of the data is then obtained by :

$$\mathbf{X}^{MDS}_{k\times n} = \mathbf{I}_{k\times n}\mathbf{\Sigma}_{n\times n}\mathbf{V}^T_{n\times n} \tag{4}$$

Let's show that $\mathbf{X}^{PCA}_{k\times n} = \mathbf{X}^{MDS}_{k\times n}$:

$$\begin{aligned}\mathbf{X}^{PCA}_{k\times n} &= \mathbf{U}_k^T\mathbf{Y}_{d\times n}\\ &= \mathbf{U}_k^T\mathbf{U}_{d\times d}\begin{pmatrix}\mathbf{\Sigma}\\ \mathbf{0}\end{pmatrix}\mathbf{V}^T_{n\times n}\\ &= \mathbf{I}_{k\times d}\begin{pmatrix}\mathbf{\Sigma}\\ \mathbf{0}\end{pmatrix}\mathbf{V}^T_{n\times n}\\ &= \mathbf{I}_{k\times n}\mathbf{\Sigma}_{n\times n}\mathbf{V}^T_{n\times n}\\ &= \mathbf{X}^{MDS}_{k\times n}\end{aligned} \tag{5}$$

We can conclude that two methods, that is, classical MDS when $\mathbf{Y}$ is known and PCA on $\mathbf{Y}$ are equivalent.

The complexity of MDS is dominated by the computation of the eigen-decomposition, which is in the order of $O(n^3)$, where n is the number of data points. Meanwhile, the complexity of PCA using the SVD algorithm is given by $O(n \cdot d \cdot \min(d, n))$ according to [1] where d is the original dimensionality of the data points.

This implies that in the case of a small dataset with a large number of dimensions ($d > n$), the complexity of PCA is given by $O(d \cdot n^2)$. In such cases, classical MDS may be more efficient. However, for a large dataset with the number of dimensions smaller than n ($d < n$), the complexity of PCA becomes $O(n \cdot d^2)$, making it more efficient than MDS.

1.4 Question 2B.4

Let's argue that the process to obtain the neiborhood graph G in the Isomap method may yield a disconnected graph. The algorithm starts by computing distances in the original space. Then we construct the graph where vertices represent points, and each point is connected to its k nearest points according to the original space.

The choice of the parameter k, representing the number of nearest neighbors, has a significant impact on the Isomap algorithm. When k is small, the neighborhood graph formed may inadequately capture the local structure of the underlying manifold. In particular, points with insufficient neighbors may become isolated, leading to disconnected components in the graph. A small k means that each point forms connections with a limited set of neighbors, and regions with sparse data density may suffer from incomplete local connectivity. We can see an example of this in the following figures, where figure 1 shows data points in some high dimensional space, and figure 2 represents the resulting disconnected neighborhood graph when $k = 1$.

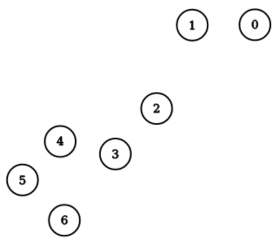


Figure 1: Data points in some high dimensional space

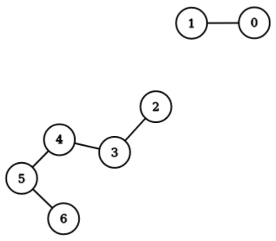


Figure 2: Example showing a disconnected graph when the number of nearest neighbors k=1

1.5 Question 2B.5

A heuristic addressing the issue of a disconnected graph has been suggested by Xiangyuan Li [2] in their paper "Density-based Multi-Manifold ISOMAP for Data Classification". In their approach, they calculate density peak points of each disconnected subgraph and establish connections between these points, assigning a weight greater than one to these inter-center edges. The intuition behind this concept is to leverage both density information and class label data inherent in the provided dataset to direct multi-manifold discriminant learning. Consequently, this heuristic aims to enhance the compactness of the low-dimensional dataset within each class while simultaneously increasing separation between different classes. This refinement is designed to improve the robustness of the classification task. This procedure transforms the disconnected graph into a connected one, enabling the subsequent application of Isomap. Figure 3 shows the new connected graph that we will get when applying this heuristic on the previous example (the red points represent density peak points, and the red edge is the new added edge linking the two density peak points from the two subgraphs) :

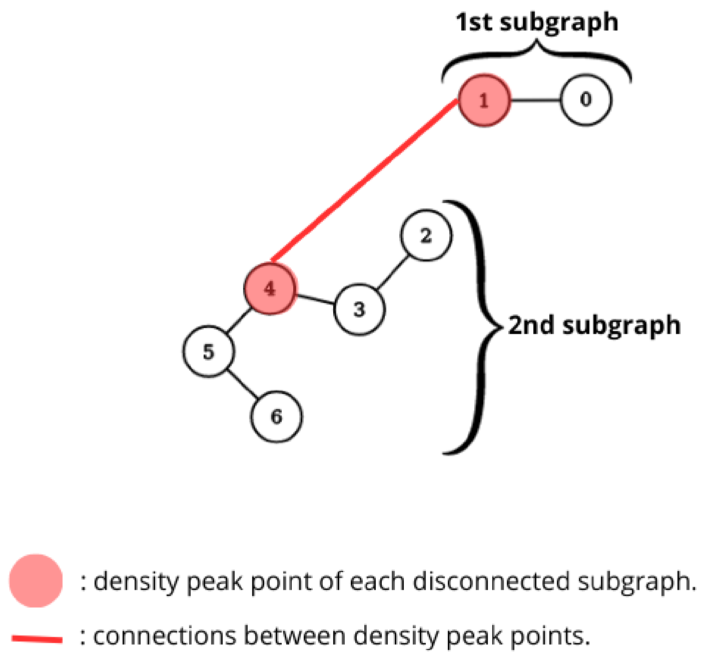

1st subgraph
1
0
2
4
3
5
6
2nd subgraph
: density peak point of each disconnected subgraph.
: connections between density peak points.

Figure 3: The resulting connected graph by applying the proposed heuristic. Red nodes represents density peak points : for the purpose of demonstration, we assumed node 1 to be the density peak point for the first subgraph and node 4 for the second subgraph.

2 Success probability in the Johnson-Lindenstrauss lemma

2.1 Question 2B.6:

Let's prove that $\mathcal{O}(n)$ independent trials are sufficient for the probability of success to be at least 95%:

The probability of success of a single trial $\mathbf{p}_s$ (meaning that (JL) holds for all pairs) is at least $\frac{1}{n}$, which means : $\frac{1}{n} \leq \mathbf{p}_s$

which means that the probability of failure of a single trail $\mathbf{p}_f$ is given by : $\mathbf{p}_f \leq 1 - \frac{1}{n}$

Now if we have $\mathcal{O}(n) = Mn$ independent trails, where M is a constant, the probability that all of them fail equlas to : $\mathbf{p}_f^{Mn} \leq \left(1 - \frac{1}{n}\right)^{Mn}$. Taking the complementary event, the probability of at least one of trials is successful is given by :

$$\text{Success probability} = 1 - \mathbf{p}_f^{Mn} \geq 1 - \left(1 - \frac{1}{n}\right)^{Mn} \geq 1 - e^{-M} \tag{6}$$

where we used the inequality $1 + x \leq e^x \quad \forall x$ by taking x to be $-\frac{1}{n}$.

Now, for M large enough, we can make the probability of success to be at least 95% :

$$\begin{aligned} 1 - e^{-M} &\geq 0.95 \\ e^{-M} &\leq 1 - 0.95 \\ M &\geq -\log(1 - 0.95) \simeq 1.3 \end{aligned} \tag{7}$$

Finally we conclude that $\sim 1.3n = \mathcal{O}(n)$ independent trials are sufficient for the probability of success to be at least 95%.

3 Node similarity for representation learning

3.1 Question 2B.7:

Let's try now to calculate the matrix $\mathbf{P}$:

$$\begin{aligned} \mathbf{P} &= \mathbf{D}^{-1}\mathbf{A} \\ \mathbf{P}_{ij} &= \left[\frac{A_{ij}}{D_{ii}}\right]_{1\leq i,j\leq n} = \left[\frac{A_{ij}}{\sum_{k=1}^{n} A_{ik}}\right]_{1\leq i,j\leq n} \end{aligned} \tag{8}$$

Note that $D_{ii} \neq 0 \quad \forall i$ because we have a connected graph, which means that at least one of A_{ij} in not zero for a fixed node i. So we actually we get a row-randomized adjacency matrix. We see that $P_{ij} = 0$ if $(i, j) \notin E$. If $(i, j) \in E$, then the magnitude of P_{ij} depends on how much other edges exist between i and other nodes. If node i has one neighbor node j, then P_{ij} will be maximal and equals to 1, which is somehow intuitive as i has only one neighbor and it will be considered to be similar to it. But when i have more neighbors, then the P_{ik}, where k representing the indices of its neighbors, will be less stronger. We can clearly notice that P is not symmetric, and because it's a row-normalized adjacency matrix, then it can be also interpreted also as the stochastic matrix of the markov chain. Which means that P^k_{ij} can be interpreted as the probability of getting from node i to node j in a k step random walk. So this similarity S measure captures the influence of neighboring nodes at different distances in the graph, with more distant nodes contributing less to the overall similarity, which is controlled by the discount factor α.

3.2 Question 2B.8:

We have :

$$\mathbf{S} = \sum_{k=0}^{\infty} \alpha^k \mathbf{P}^k \tag{9}$$

Let's show first that given a stochastic matrix $\mathbf{A}$, its largest eigenvalue is 1 :

Proof: To begin, consider the fact that if $\mathbf{A}$ is a right stochastic matrix, then the sum of each row of $\mathbf{A}$ is equal 1, which leads to $\mathbf{A1} = \mathbf{1}$ (where $\mathbf{1}$ is the column vector where all its values equal to 1). This establishes that 1 is indeed an eigenvalue of A.

Next, let's assume the existence of $\lambda > 1$ and a nonzero vector $\mathbf{x}$ such that $\mathbf{Ax} = \lambda\mathbf{x}$. Let x_i be the largest element of $\mathbf{A}$. Without loss of generality, we can assume that $x_i > 0$, as any scalar multiple of $\mathbf{x}$ will still satisfy the equation. Since the rows of $\mathbf{A}$ are positive and sum to 1, each entry in $\mathbf{x}$ is a convex combination of the elements of $\mathbf{x}$, which means that no entry in $\lambda\mathbf{x}$ can exceed x_i. However, given that $\lambda > 1$, it follows that $\lambda x_i > x_i$, leading to a contradiction. Therefore, the conclusion is that the largest eigenvalue of $\mathbf{A}$ is 1.

We know that $\mathbf{P}$ is a stochastic matrix, then all of its eigenvalues have absolute values less than or equal to 1, which leads to the following result : $|\lambda_i| \leq \alpha < 1$ for each λ_i eigenvalue of $\alpha\mathbf{P}$. Hence the geometric series generated by $\alpha\mathbf{P}$ converges, and it's given by :

$$\mathbf{S} = \sum_{k=0}^{\infty} \alpha^k \mathbf{P}^k = (\mathbf{I} - \alpha\mathbf{P})^{-1} \tag{10}$$

We conclude then that $\mathbf{S}$ can be computed efficiently using a matrix inversion operation $(\mathbf{I}-\alpha\mathbf{P})^{-1}$.

4 Spectral graph analysis

4.1 Question 2B.9

Let $G = (V, E)$ be an undirected d-regular graph, let A be the adjacency matrix of G, and let $L = I - \frac{1}{d}A$ be the normalized Laplacien of G. Let's prove that for any vector $x \in \mathbb{R}^{|V|}$ it is : $\mathbf{x}^T\mathbf{L}\mathbf{x} = \frac{1}{d}\sum_{(u,v)\in E}(x_u - x_v)^2$.

We know that $A_{ii} = 0 \quad \forall i$, and let $n = |V|$. We have thus the normalized Laplacien given by:

$$\begin{aligned}
\mathbf{L} &= \mathbf{I} - \frac{1}{d}\mathbf{A} \\
&= \begin{pmatrix} 1 & -\frac{A_{12}}{d} & \dots & -\frac{A_{1n}}{d} \\ -\frac{A_{21}}{d} & 1 & \dots & -\frac{A_{2n}}{d} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{A_{n1}}{d} & \dots & \dots & 1 \end{pmatrix} \\
&= \frac{1}{d}\begin{pmatrix} d & -A_{12} & \dots & -A_{1n} \\ -A_{21} & d & \dots & -A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -A_{n1} & \dots & \dots & d \end{pmatrix}
\end{aligned} \tag{11}$$

We multiply it now by the vector x :

$$\begin{aligned}
\mathbf{L}\mathbf{x} &= \frac{1}{d}\begin{pmatrix} d & -A_{12} & \dots & -A_{1n} \\ -A_{21} & d & \dots & -A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -A_{n1} & \dots & \dots & d \end{pmatrix}\begin{pmatrix} x_1 \\ \vdots \\ \vdots \\ x_n \end{pmatrix} \\
&= \frac{1}{d}\begin{pmatrix} dx_1 - \sum_{i=2}^{n} A_{1i}x_i \\ \vdots \\ dx_j - \sum_{i\neq j} A_{ji}x_i \\ \vdots \\ dx_n - \sum_{i=1}^{n-1} A_{ni}x_i \end{pmatrix}
\end{aligned} \tag{12}$$

We calculate now the scalar product between $\mathbf{x}$ and $\mathbf{L}\mathbf{x}$:

$$
\begin{aligned}
\mathbf{x}^T\mathbf{L}\mathbf{x} &= \frac{1}{d}\begin{pmatrix} x_1 & \dots & x_n \end{pmatrix} \begin{pmatrix} dx_1 - \sum_{i=2}^{n} A_{1i}x_i \\ \vdots \\ dx_j - \sum_{i\neq j} A_{ji}x_i \\ \vdots \\ dx_n - \sum_{i=1}^{n-1} A_{ni}x_i \end{pmatrix} \\
&= \sum_{i=1}^{n} x_i^2 - \frac{1}{d}\sum_{j=1}^{n}\sum_{i\neq j} A_{ji}x_ix_j \\
&= \sum_{i=1}^{n} x_i^2 - \frac{2}{d}\sum_{j=1}^{n}\sum_{i<j} A_{ji}x_ix_j \qquad \text{By symmetry of } \mathbf{A} \\
&= \sum_{i=1}^{n} x_i^2 - \frac{1}{d}\sum_{(u,v)\in E} 2x_ux_v \qquad A_{uv} = 0 \quad \forall (u,v) \notin E \\
&= \frac{1}{d}(d\sum_{v\in G} x_v^2 - \sum_{(u,v)\in E} 2x_ux_v)
\end{aligned} \tag{13}
$$

In fact, we have the following equality : $\sum_{(u,v)\in E} x_u^2 + x_v^2 = \sum_{v\in G} dx_v^2$. This is true because we have a d-regular graph. If we examine the left-hand side (L.H.S.) expression, we traverse all the edges of the graph. Since the degree of each node is d, every node is visited exactly d times, leading to the right-hand side (R.H.S.) expression. Consequently, we obtain:

$$
\begin{aligned}
\mathbf{x}^T\mathbf{L}\mathbf{x} &= \frac{1}{d}\sum_{(u,v)\in E} x_u^2 + x_v^2 - 2x_ux_v \\
&= \frac{1}{d}\sum_{(u,v)\in E} (x_u - x_v)^2
\end{aligned} \tag{14}
$$

4.2 Question 2B.10

Let's show that the normalized Laplacien is a positive semidefinite matrix.

First of all, we are going to show that $\mathbf{L}$ can be written as $\mathbf{M}^T\mathbf{M}$

We orient arbitrarily each edge $(u,v) \in E$ such that we have u is source and v is terminal. Let's take $\mathbf{M}_{m\times n}$, where m is the number of edges and n the number of nodes, defined by :

$$
M[e,v] = \begin{cases} \frac{1}{\sqrt{d}} & \text{if } v \text{ is source of e} \\ -\frac{1}{\sqrt{d}} & \text{if } v \text{ is terminal of e} \\ 0 & \text{otherwise.} \end{cases} \tag{15}
$$

We have $[\mathbf{M}^T\mathbf{M}]_{uv} = \sum_{e\in E} M[e,u]M[e,v]$

If $e' = (u',v') \in E$, $u' \neq v'$:

We will examine all the edges, and only one edge will remain, denoted as e'. The other terms become zero since u' and v' do not simultaneously serve as the source and terminal for any other edge, except for e' : $[\mathbf{M}^T\mathbf{M}]_{u'v'} = \sum_{e\in E} M[e,u']M[e,v'] = M[e',u']M[e',v'] = -\frac{1}{\sqrt{d}}\frac{1}{\sqrt{d}} = -\frac{1}{d}$

If $u' = v'$:

$[\mathbf{M}^T\mathbf{M}]_{u'u'} = \sum_{e\in E} M[e,u']^2 = \sum_{w\in V:(u',w)\in E}\left(\pm\frac{1}{\sqrt{d}}\right)^2 = d\times\frac{1}{d} = 1$
where in the last equality, we used the fact that the degree of the node u' is d.

If $(u',v')\notin E$: $[\mathbf{M}^T\mathbf{M}]_{u'v'} = 0$

We conclude that :

$$M^TM[u,v] = \begin{cases} \frac{-1}{d} & \text{if } (u,v)\in E \\ 1 & \text{if } u=v \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

and thus we have : $\mathbf{L} = \mathbf{M}^T\mathbf{M}$
Then it becomes easy to show that L is a positive semidefinite matrix. For all $\mathbf{x}\in\mathbb{R}^n$:

$$\begin{aligned} \mathbf{x}^T\mathbf{L}\mathbf{x} &= \mathbf{x}^T\mathbf{M}^T\mathbf{M}\mathbf{x} \\ &= ||\mathbf{M}\mathbf{x}||_2^2 \geq 0 \end{aligned} \tag{17}$$

we conclude then that the normalized Laplacian is a positive semidefinite matrix.

4.3 Question 2B.11

As we can clearly see, the expression x^TLx is positive and reaches its global minimum when x is a constant vector (all values of the vector are the same). So, a trivial solution refers to x^* being equal to a constant vector, with all nodes located at the some point c (i.e., $x_u = c$ for all $u\in V$). Thus, a non-trivial vector x^* means that the embedding occupies the vector space (different form the constant vector).

The vector x^* obtained can be considered as an embedding of the vertices of the graph into the real line. Each vertex is represented by a corresponding entry in x^*. The values of x^* provide a set of coordinates in a one-dimensional space that reflects the graph's structure.

Minimizing equation (1) is equivalent to minimizing the squared distances between nodes that are connected, which is a natural approach to graph embedding. This approach aims to place connected nodes close to each other in the real line, justifying the fact that we will obtain a meaningful embedding.

5 Programming Task

5.1 Question 2B.12

The code for the two methods is in the Appendix.

5.1.1 PCA

Let's first try using PCA to explore whether we can obtain more interesting results. When applied to the datasets used in assignment 2A : `EP6_RCVs_2022_06_13.pkl`, the results are depicted in Figure 4 with respect to EPG:

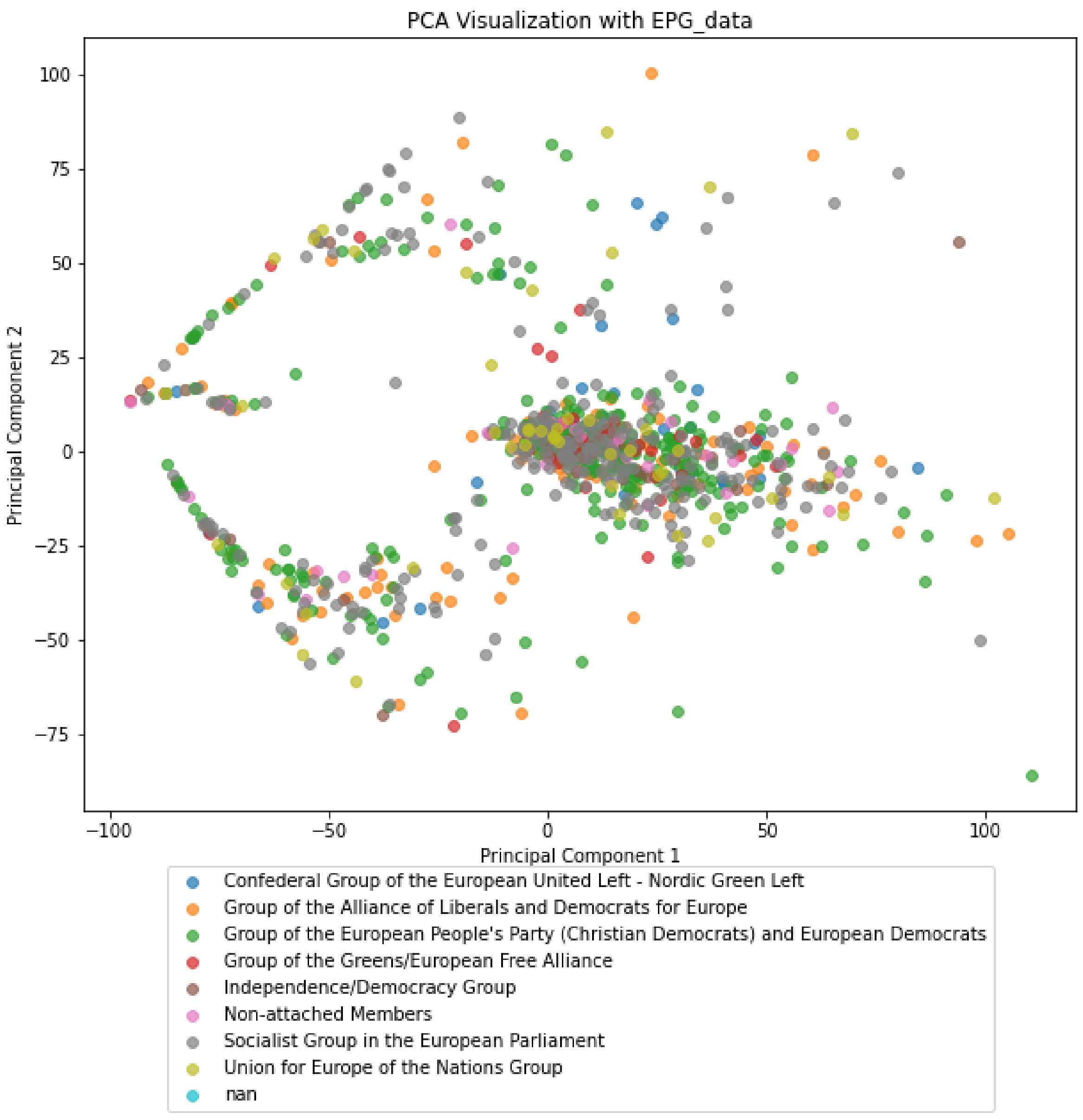


Figure 4: PCA applied on the dataset in 2A/3

These results are interesting as we observe a similarity to the outcomes obtained in MDS (see Figure 5). This similarity arises from the fact that the two methods, classical MDS when data points Y are known and PCA on Y, are equivalent.

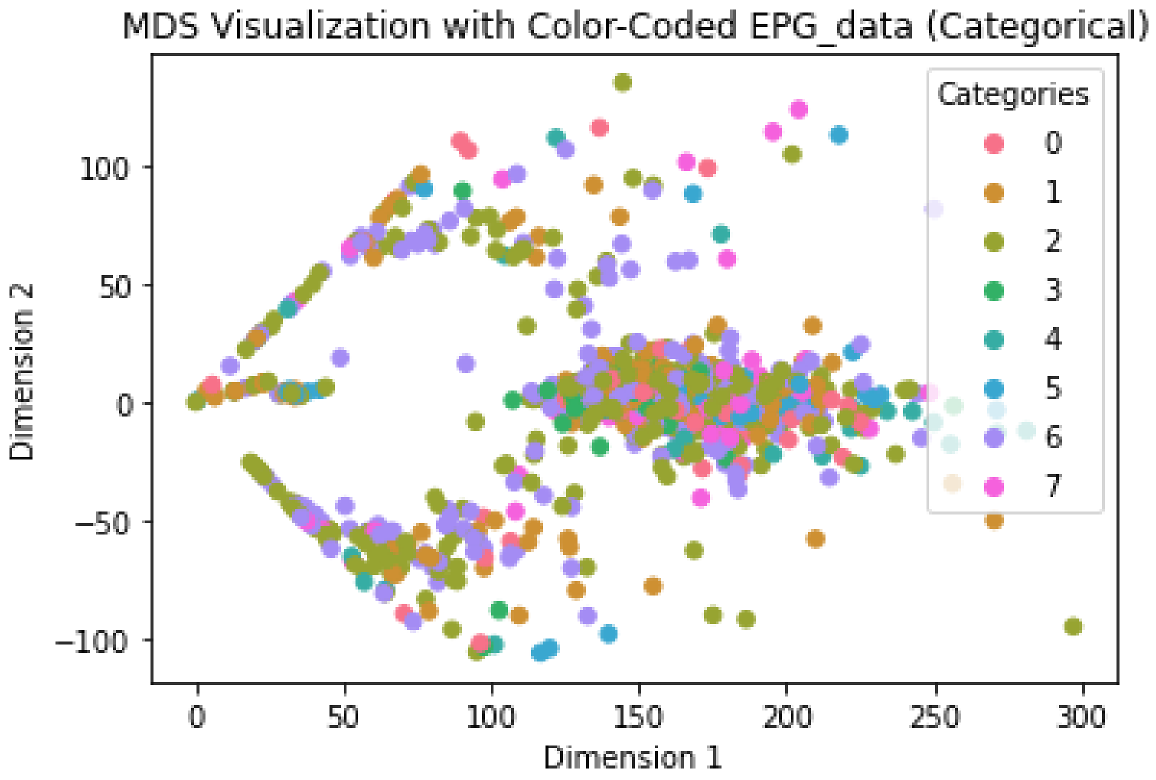


Figure 5: MDS applied on the dataset in 2A/3

5.1.2 Self-Organizing Maps

We will now employ an alternative embedding method in the pursuit of achieving improved results. For this purpose, we will utilize Self-Organizing Maps (SOMs). SOMs are particularly useful for visualizing high-dimensional data in a lower-dimensional space while preserving the topological relationships between data points.

The main idea behind SOMs is to map the input space, which can be high-dimensional, onto a grid of nodes (neurons) in a lower-dimensional space. During the training process, the SOM adjusts its weights to capture the underlying structure of the input data. Each node in the SOM represents a region in the input space, and neighboring nodes in the grid correspond to similar regions in the input space. In the context of embedding, SOMs can be used to embed high-dimensional data into a lower-dimensional grid where each grid cell represents a region of similar data. This can be especially useful for visualizing complex data structures and discovering patterns.

Here's a simplified overview of the SOM training process:

Initialization: Initialize the weights of the neurons with small random values.

Input Presentation: Randomly select an input vector from the dataset.

Competition: Find the neuron whose weights are most similar to the input vector (in terms of Euclidian distance in the input space). This neuron is called the Best Matching Unit (BMU).

Cooperation: Update the weights of the BMU and its neighboring neurons to move closer to the input vector. Neighboring nodes (in the output space) are determined based on a topological neighborhood function.

Iteration: Repeat steps 2-4 for a specified number of iterations or until convergence.

We implement this method and try it on the dataset of 2A/3. For this problem, we utilized a 10x10 2D grid as the output space. We ran a total of 100 epochs with a learning rate starting from 0.1 and decreasing exponentially with the time(epochs). The number of neighbors also decreases exponentially in this case, and we employ the Manhattan distance in the output space to determine which nodes will be updated along with the best matching units. Once the model is trained, we present all input data to it, incorporating additional information such as EPG or Country, encoded as colors. Points that fall within the same or neighboring nodes in the 2D grid are considered similar to each other, which means that they share features in the input space.

Figure 6 shows the result when we map our data to this 2D grid with respect to EPG. Note that we are working with all features including time information.

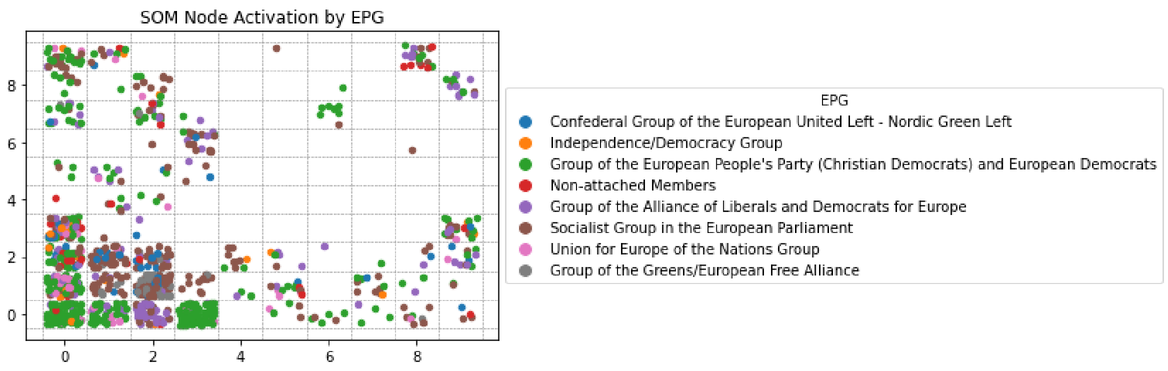


Figure 6: Self organizing map with respect to EPG

To enhance visibility, we assign each node a color based on the majority vote within that node. The resulting color-coded Self-Organizing Map (SOM) is depicted in Figure 7:

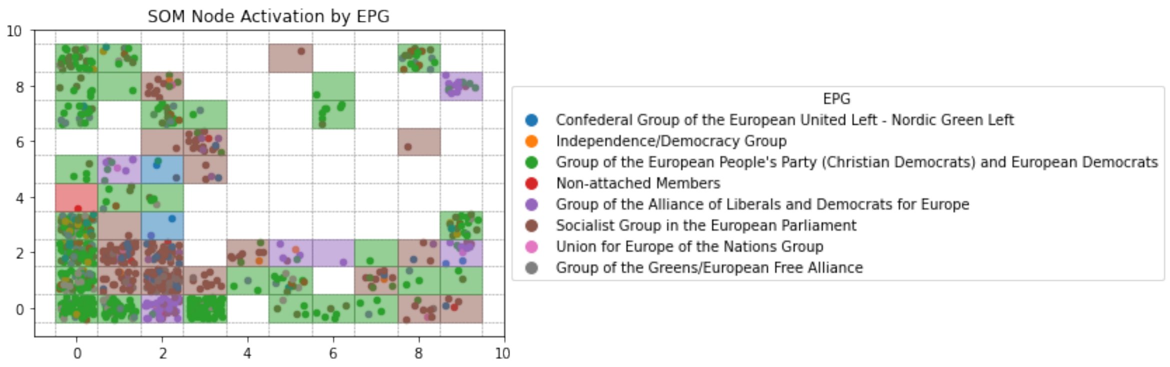


Figure 7: Self organizing map with respect to EPG. The colors of each node represent the most present EPG in that node

We can observe distinct clusters among parliament members based on their EPG, suggesting a notable influence of EPG affiliation on MPs' voting patterns. This clustering proves valuable for classification tasks, such as predicting the EPG of a given parliament member. Additionally, we can explore the SOM's behavior concerning countries, as depicted in Figure 8

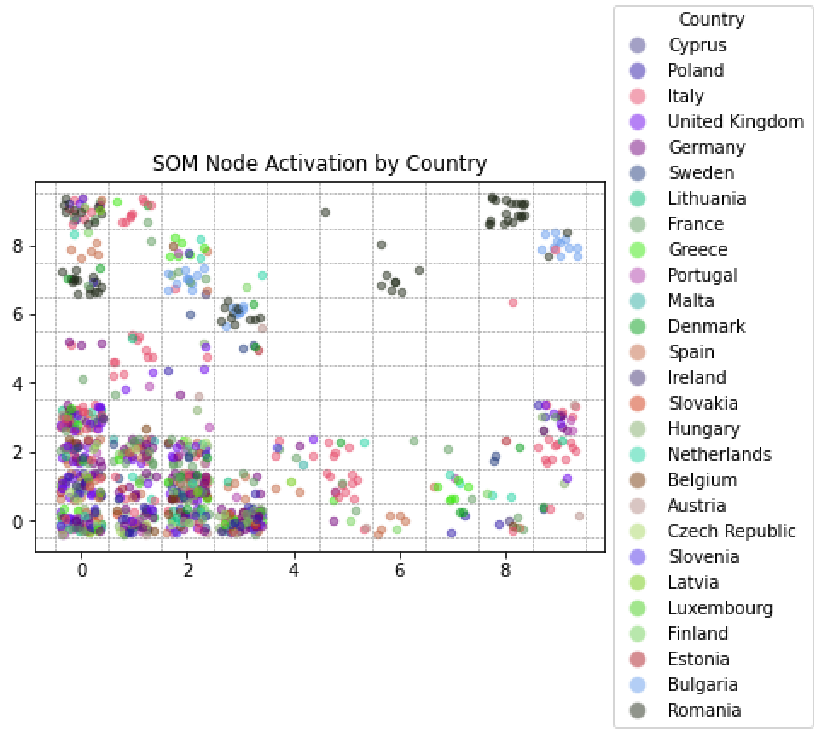


Figure 8: Self organizing map with respect to Country

Extracting meaningful patterns becomes more challenging with a larger number of countries. Nonetheless, certain nodes exhibit a concentration of points with the same color, implying that individuals from those countries tend to vote in a similar manner.

References

[1] Vinita Vasudevan and M Ramakrishna. A hierarchical singular value decomposition algorithm for low rank matrices. *arXiv preprint arXiv:1710.02812*, 2017.

[2] Jinrong He y Xiangyuan Li Cheng Cai. Density-based multi-manifold isomap for data classification. 2017.

Programming Task Code

```
[ ]: import pandas as pd
     import numpy as np
     from sklearn.metrics.pairwise import cosine_similarity
     import matplotlib.pyplot as plt
     import seaborn as sns
     from matplotlib.colors import ListedColormap
     from sklearn.preprocessing import LabelEncoder
```

```
[ ]: pickle_path = "EP6_RCVs_2022_06_13.pkl"
     df = pd.read_pickle(pickle_path)
     df = df[df.EPG.notna()]
     df = df[df.Country.notna()]
     df.head()
```

```
[3]: # Handling missing values
     df['Fname'] = df.apply(lambda row: row['FullName'].split()[0] if pd.
      ↪isnull(row['Fname']) else row['Fname'], axis=1)
     df['Lname'] = df.apply(lambda row: ' '.join(row['FullName'].split()[1:]) if pd.
      ↪isnull(row['Lname']) and len(row['FullName'].split()) > 1 else row['Lname'],␣
      ↪axis=1)
```

```
[4]: df['Party'].unique()
     df = df[df['Party'].str.contains('[a-zA-Z]')]
     # Convert 'Start' and 'End' columns to pandas datetime objects
     df['Start'] = pd.to_datetime(df['Start'])
     df['End'] = pd.to_datetime(df['End'])
     # Define the reference date as the oldest date in your data
     reference_date = df[['Start', 'End']].min().min()
     # Calculate the number of days with respect to the reference date
     df['Start'] = (df['Start'] - reference_date).dt.days
     df['End'] = (df['End'] - reference_date).dt.days
```

```
[5]: df['Activ']= df['Activ'].map({'yes': 1, 'no': 0})
     df.head()
```

```
[5]:    WebisteEpID        Fname      Lname              FullName  Activ  \
     1      28469.0       Adamos     ADAMOU        ADAMOU, Adamos      1
     2      28302.0        Filip     ADWENT         ADWENT, Filip      0
     3      28975.0     Vittorio  AGNOLETTO   AGNOLETTO, Vittorio      1
     4      28367.0     Gabriele  ALBERTINI   ALBERTINI, Gabriele      1
     5      28512.0   James Hugh   ALLISTER  ALLISTER, James Hugh      1

               Country                                             Party  \
     1          Cyprus  Anorthotiko Komma Ergazomenou Laou - Aristera ...
     2          Poland                               Liga Polskich Rodzin
     3           Italy  Partito della Rifondazione Comunista - Sinistr...
```

```
4           Italy                                           Forza Italia
5  United Kingdom                                   Traditional Unionist

                                                EPG  Start   End  ...  6191  \
1  Confederal Group of the European United Left -...      0  1990  ...   5.0
2                     Independence/Democracy Group      0   341  ...   0.0
3  Confederal Group of the European United Left -...      0  1990  ...   2.0
4  Group of the European People's Party (Christia...      0  1990  ...   1.0
5                             Non-attached Members      0  1990  ...   4.0

   6192  6193  6194  6195  6196  6197  6198  6199  6200
1   5.0   5.0   5.0   5.0   5.0   5.0   5.0   5.0   5.0
2   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
3   3.0   1.0   2.0   1.0   2.0   5.0   2.0   5.0   5.0
4   1.0   1.0   1.0   2.0   1.0   1.0   5.0   5.0   5.0
5   4.0   4.0   4.0   4.0   4.0   4.0   4.0   4.0   4.0

[5 rows x 6209 columns]
```

[6]:
```
# df = pd.get_dummies(df, columns=['Party'], prefix='')
# df.head()
```

[7]:
```
# Similarity Functions :
def dot_product_similarity(vector1, vector2):
    return np.dot(vector1, vector2)
```

[8]:
```
# Output data :
EPG_data = df['EPG']
Country_data = df['Country']

# Input Data :
columns_to_drop = ['WebisteEpID','Fname','Lname','FullName','EPG','Country']
df.drop(columns=columns_to_drop,inplace=True)

dataset_matrix = df.values

print("Shape of input array :", dataset_matrix.shape)
```

```
Shape of input array : (925, 6202)
```

SOMs

[9]:
```
# Define the parameters
start_value = 50
end_value = 0.1
num_values = 200
rows = 100
columns = dataset_matrix.shape[1]
```

```
weights = np.random.rand(rows, columns)

# Create an array of exponentially spaced values from start to end
exponential_values = np.geomspace(start_value, end_value, num=num_values)

# Convert the array to a list of discrete values
radiuses = [int(x) for x in exponential_values]
```

[10]:
```
def BMU_2D(dataset,input_data):
  # Calculate the differences between the input and each row of the dataset
  distances = dataset - input_data
  differences = np.linalg.norm(dataset - input_data, axis=1)

  # Find the index of the row with the smallest difference
  min_index = np.argmin(differences)

      # Convert the 1D index to 2D coordinates
  x_bmu, y_bmu = min_index // 10, min_index % 10

  return min_index, x_bmu, y_bmu, distances
```

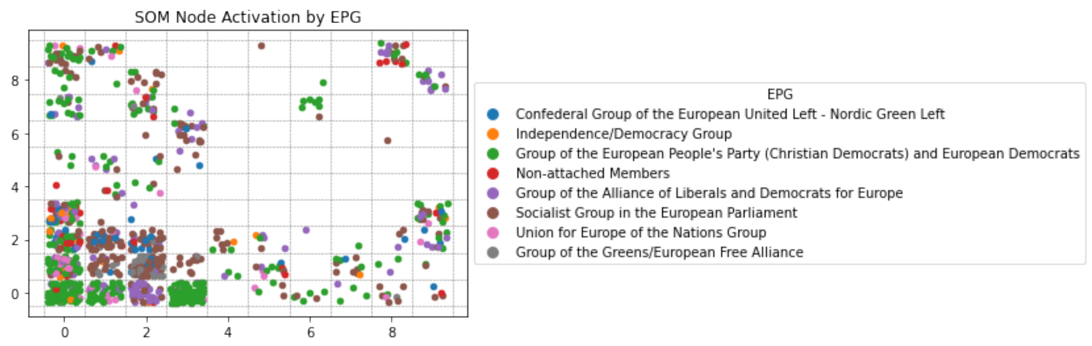
[ ]:
```
from scipy.spatial.distance import cityblock
import matplotlib.pyplot as plt
import random
num_samples = 100
num_epochs = 100

print(weights)
for epoch in range(num_epochs):
  random_indices = [random.randint(0, len(dataset_matrix) - 1) for _ in␣
 ↪range(num_samples)]
  for i in random_indices:
    input_sample = dataset_matrix[i]
    index_BMU, x_bmu, y_bmu, distances = BMU_2D(weights, input_sample)
    radius = radiuses[epoch]

    # Determine the range for neighbors within the 10x10 grid
    x_start = max(x_bmu - radius, 0)
    x_end = min(x_bmu + radius + 1, 10)
    y_start = max(y_bmu - radius, 0)
    y_end = min(y_bmu + radius + 1, 10)

    for x in range(x_start, x_end):
        for y in range(y_start, y_end):
            manhattan_distance = cityblock([x,y], [x_bmu,y_bmu])
            if manhattan_distance <= radius :
              index = x * 10 + y
```

```
                learning_rate = max(0.0001,0.1*(0.0001/0.1)**(epoch/num_epochs))
                weights[index] = weights[index] - learning_rate * distances[index]

print(weights)
```

[ ]:
```
from matplotlib.colors import to_rgba

party_names = EPG_data.unique()

# Create a 10x10 grid (assuming a 10x10 SOM)
grid_size = int(np.sqrt(len(weights)))
grid = np.zeros((grid_size, grid_size), dtype=int)


party_colors = ['#1f77b4',  # blue
                '#ff7f0e',  # orange
                '#2ca02c',  # green
                '#d62728',  # red
                '#9467bd',  # purple
                '#8c564b',  # brown
                '#e377c2',  # pink
                '#7f7f7f',  # gray
                '#FFFFFF']

cmap = plt.matplotlib.colors.ListedColormap(party_colors)

default_color_code = len(party_colors) - 1
grid = np.full((100, 100), default_color_code, dtype=int)

# Find the BMU for each input data point and assign the corresponding color
for i in range(len(dataset_matrix)):
    input_sample = dataset_matrix[i]
    bmu_index, bmu_x, bmu_y, _ = BMU_2D(weights, input_sample)
    party_group = EPG_data.iloc[i]
    color_code = np.where(party_names == party_group)[0][0]
    color = party_colors[color_code]  # Use the index as a color code
    x_offset = np.random.uniform(-0.4, 0.4)
    y_offset = np.random.uniform(-0.4, 0.4)
    y = bmu_x + x_offset
    x = bmu_y + y_offset
    plt.scatter(x, y, c=[color], cmap=cmap, s=20)
    grid[bmu_x, bmu_y] = color_code

# Generate a random color for each party dynamically
# party_colors = [(*np.random.rand(3), 0.5) for _ in party_names]
```

```
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', label=party,␣
↪markersize=10, markerfacecolor=party_colors[i]) for i, party in␣
↪enumerate(party_names)]
plt.legend(handles=legend_handles, title='EPG', loc='center left',␣
↪bbox_to_anchor=(1, 0.5))

for x in range(11):  # Marqueurs de grille verticale
    plt.axvline(x=x - 0.5, color='gray', linestyle='--', linewidth=0.5)
for y in range(11):  # Marqueurs de grille horizontale
    plt.axhline(y=y - 0.5, color='gray', linestyle='--', linewidth=0.5)


plt.title("SOM Node Activation by EPG")
plt.show()
```

```
/tmp/ipykernel_35913/221537682.py:36: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(x, y, c=[color], cmap=cmap, s=20)
```

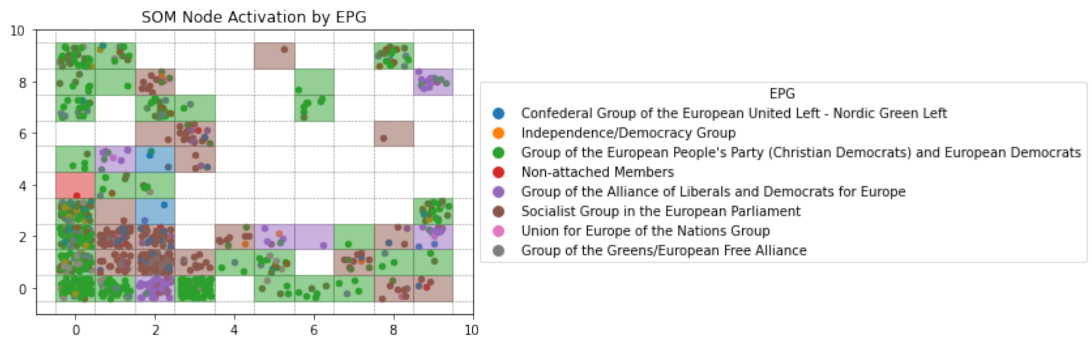


```
[ ]: from matplotlib.colors import to_rgba

     party_names = EPG_data.unique()

     # Create a 10x10 grid (assuming a 10x10 SOM)
     grid_size = int(np.sqrt(len(weights)))
     grid = np.zeros((grid_size, grid_size), dtype=int)


     party_colors = ['#1f77b4',  # blue
                     '#ff7f0e',  # orange
                     '#2ca02c',  # green
                     '#d62728',  # red
                     '#9467bd',  # purple
                     '#8c564b',  # brown
```

```
                '#e377c2',  # pink
                '#7f7f7f',  # gray
                '#FFFFFF']

# Initialize the grid with default_color_code
default_color_code = len(party_colors) - 1  # Index of white color
grid = np.full((grid_size, grid_size), default_color_code, dtype=int)

# Count the occurrences of each color in each square
color_counts = np.zeros((grid_size, grid_size, len(party_colors)), dtype=int)

for i in range(len(dataset_matrix)):
    input_sample = dataset_matrix[i]
    bmu_index, bmu_x, bmu_y, _ = BMU_2D(weights, input_sample)
    party_group = EPG_data.iloc[i]
    color_code = np.where(party_names == party_group)[0][0]

    color_counts[bmu_x, bmu_y, color_code] += 1

# Plot the points first
for i in range(len(dataset_matrix)):
    input_sample = dataset_matrix[i]
    bmu_index, bmu_x, bmu_y, _ = BMU_2D(weights, input_sample)
    party_group = EPG_data.iloc[i]
    color_code = np.where(party_names == party_group)[0][0]
    color = party_colors[color_code]
    x_offset = np.random.uniform(-0.4, 0.4)
    y_offset = np.random.uniform(-0.4, 0.4)
    y = bmu_x + x_offset
    x = bmu_y + y_offset
    plt.scatter(x, y, c=[color], cmap=cmap, s=20)

# Assign the color with the maximum count to each square
for i in range(grid_size):
    for j in range(grid_size):
        # Check if the square has at least one point
        if np.sum(color_counts[i, j]) > 0:
            majority_color_code = np.argmax(color_counts[i, j])
            grid[i, j] = majority_color_code
            plt.fill_between([j - 0.5, j+1 - 0.5], i - 0.5, i+1 - 0.5,␣
 ↪color=party_colors[majority_color_code], alpha=0.5)

# Overlay the grid on top of the points
cmap = ListedColormap(party_colors)
# plt.imshow(grid, cmap=cmap, interpolation='none', aspect='auto', alpha=0.5)
```

```
legend_handles = [plt.Line2D([0], [0], marker='o', color='w', label=party,␣
 ↪markersize=10, markerfacecolor=party_colors[i]) for i, party in␣
 ↪enumerate(party_names)]
plt.legend(handles=legend_handles, title='EPG', loc='center left',␣
 ↪bbox_to_anchor=(1, 0.5))

for x in range(11):  # Marqueurs de grille verticale
    plt.axvline(x=x - 0.5, color='gray', linestyle='--', linewidth=0.5)
for y in range(11):  # Marqueurs de grille horizontale
    plt.axhline(y=y - 0.5, color='gray', linestyle='--', linewidth=0.5)


plt.title("SOM Node Activation by EPG")
plt.show()
```

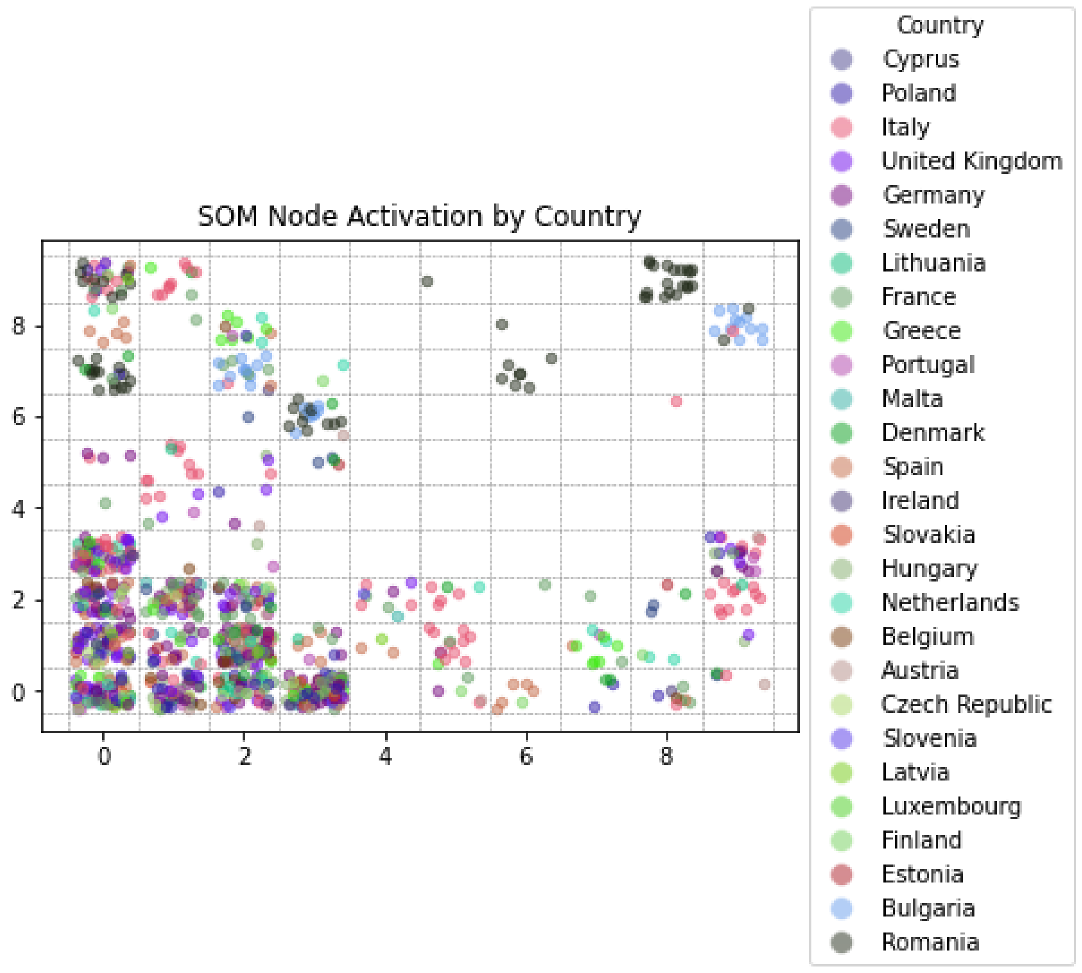
```
/tmp/ipykernel_35913/4166189035.py:46: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(x, y, c=[color], cmap=cmap, s=20)
```



```
[ ]: from matplotlib.colors import to_rgba

country_names = Country_data.unique()

# Create a 10x10 grid (assuming a 10x10 SOM)
grid_size = int(np.sqrt(len(weights)))
grid = np.zeros((grid_size, grid_size), dtype=int)

country_colors = [(*np.random.rand(3), 0.5) for _ in country_names]
country_colors.append((1,1,1))

default_color_code = len(country_colors) - 1
grid = np.full((100, 100), default_color_code, dtype=int)

# Find the BMU for each input data point and assign the corresponding color
```

```python
for i in range(len(dataset_matrix)):
    input_sample = dataset_matrix[i]
    bmu_index, bmu_x, bmu_y, _ = BMU_2D(weights, input_sample)
    country_group = Country_data.iloc[i]
    color_code = np.where(country_names == country_group)[0][0]
    color = country_colors[color_code]  # Use the index as a color code
    x_offset = np.random.uniform(-0.4, 0.4)
    y_offset = np.random.uniform(-0.4, 0.4)
    y = bmu_x + x_offset
    x = bmu_y + y_offset
    plt.scatter(x, y, c=[color], cmap=cmap, s=20)
    grid[bmu_x, bmu_y] = color_code

# Generate a random color for each party dynamically


legend_handles = [plt.Line2D([0], [0], marker='o', color='w', label=party,␣
 ↪markersize=10, markerfacecolor=country_colors[i]) for i, party in␣
 ↪enumerate(country_names)]
plt.legend(handles=legend_handles, title='Country', loc='center left',␣
 ↪bbox_to_anchor=(1, 0.5))

for x in range(11):  # Marqueurs de grille verticale
    plt.axvline(x=x - 0.5, color='gray', linestyle='--', linewidth=0.5)
for y in range(11):  # Marqueurs de grille horizontale
    plt.axhline(y=y - 0.5, color='gray', linestyle='--', linewidth=0.5)


plt.title("SOM Node Activation by Country")
plt.show()
```

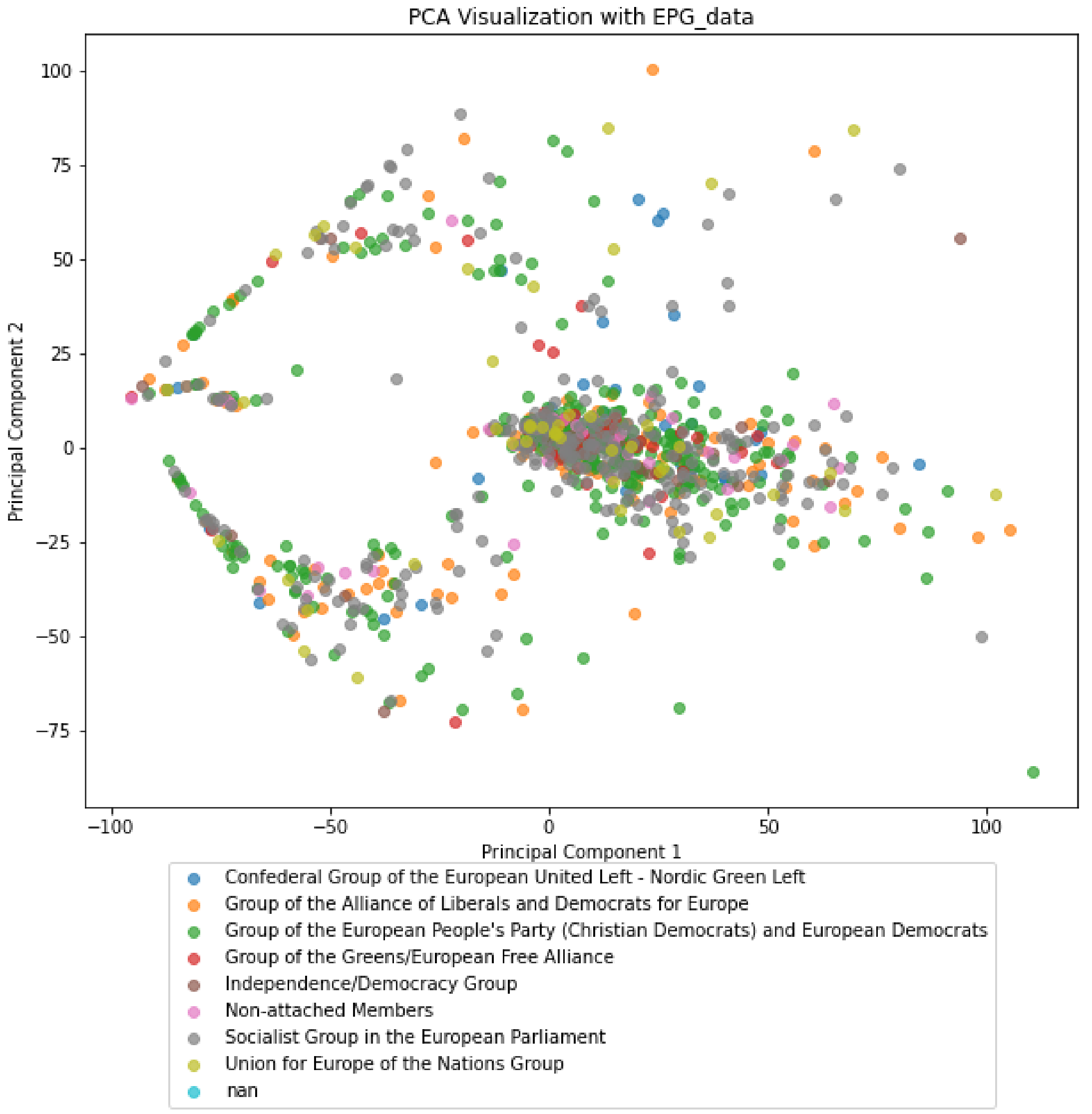
```
/tmp/ipykernel_35913/269642847.py:26: UserWarning: No data for colormapping
provided via 'c'. Parameters 'cmap' will be ignored
  plt.scatter(x, y, c=[color], cmap=cmap, s=20)
```



PCA

[43]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Assuming you have your dataset_matrix, EPG_data, and Country_data defined
# dataset_matrix: np.array with shape (925, 6202)
# EPG_data: DataFrame with a single column and 925 rows
# Country_data: DataFrame with a single column and 925 rows

# Step 1: Standardize the data
mean = np.mean(dataset_matrix, axis=0)
std_dev = np.std(dataset_matrix, axis=0)
```

```
standardized_data = (dataset_matrix - mean) / std_dev

# Step 2: Apply PCA
n_components = 2  # You can choose the number of components you want
pca = PCA(n_components=n_components)
pca_result = pca.fit_transform(standardized_data)

# Step 3: Combine the PCA result with EPG_data
pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
final_df = pd.concat([pca_df, EPG_data, Country_data], axis=1)

# Step 4: Convert categorical EPG_data to numerical values
le = LabelEncoder()
final_df['EPG_data_numeric'] = le.fit_transform(final_df['EPG'])

# Step 5: Define distinct colors for each category
category_colors = plt.cm.tab10(np.linspace(0, 1, len(le.classes_)))

# Step 6: Visualize the results with different colors for each category
plt.figure(figsize=(10, 8))

# Scatter plot with colors representing EPG_data
for category, color in zip(le.classes_, category_colors):
    subset = final_df[final_df['EPG'] == category]
    plt.scatter(subset['PC1'], subset['PC2'], label=category, color=color,␣
 ↪alpha=0.7)

plt.title('PCA Visualization with EPG_data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(loc='lower center',bbox_to_anchor=(0.5, -0.4))
plt.show()
```


PCA Visualization with EPG_data
Principal Component 1
Principal Component 2
100
75
50
25
0
−25
−50
−75
−100
−50
0
50
100
Confederal Group of the European United Left - Nordic Green Left
Group of the Alliance of Liberals and Democrats for Europe
Group of the European People's Party (Christian Democrats) and European Democrats
Group of the Greens/European Free Alliance
Independence/Democracy Group
Non-attached Members
Socialist Group in the European Parliament
Union for Europe of the Nations Group
nan