Atelier : détection de communautés dans les réseaux sociaux

1 Introduction

Python, package ''scikit-learn'' – Application du positionnement multidimensionnel (MDS - Multidimensional Scaling) à la détection de communautés dans les réseaux sociaux.

La détection de communautés est une des applications phares de l'analyse des réseaux sociaux. Sous Python, l'objectif était de se familiariser d'une part avec les notions essentielles de l'analyse des communautés (voisinage, centralité, individus relais, …), d'autre part de découvrir les fonctionnalités du package ''igraph''. Les données se présentent sous la forme d'une matrice d'adjacence binaire (absence ou présence de lien entre les sociétaires du club).

Nous cherchons à aller plus loin avec une configuration où les connexions entre les individus sont valorisées par une valeur numérique positive ou nulle. Le tableau de départ de l'étude correspond à une matrice de dissimilarités.

2 Données

L'objectif est d'identifier les relations, les liens ou les connexions entre les personnes au sein d'un réseau social (les accointances). Analyse d'une sociomatrice. Chaque personne doit donner une note mesurant la fréquence de sa communication avec chacun de ses 25 camarades. Le système de notation est le suivant :

Vous communiquez avec (*untel*) …

1. Très souvent
2. Souvent
3. Rarement
4. Très rarement
5. Jamais

La matrice de données traitée est la suivante, ou δ_{ij} est la moyenne des deux notes :

$$\delta_{ij} = \tfrac{1}{2}\,(\text{Note donnée par } i \text{ à } j + \text{Note donnée par } j \text{ à } i)$$

Notons plusieurs éléments importants :

- Plus la note est faible, plus les étudiants sont proches.
- La proximité d'un individu avec lui-même est nulle.
- δ_{ij} est symétrique par construction.

L'information à traiter se présente par conséquent comme une matrice de dissimilarités.

IUT	BEN	BES	BOU	BRU	CAM	CHU	DUC	LAN	LEX	MAR	ROG	ROS	TOS	BAR	BEL	CAL	DIF	FIR	FRE	FUM	HAD	HEL	MZA	VER	VID
BEN	0	3	1	3	1.5	3	3	2	1	1.5	1.5	2	1	3	1.5	1.5	1.5	2.5	3	1.5	2.5	2.5	0	3	1.5
BES	3	0	3	1	1	0	0	4	2.5	3.5	4	0	2.5	3	2	2.5	1.5	1.5	3	1.5	1.5	1.5	2.5	3.5	2.5
BOU	1	3	0	2.5	2.5	2.5	3	2.5	2	0	1	2.5	2.5	3	1	2	2	2.5	2	0	1.5	2.5	0	3	2.5
BRU	3	1	2.5	0	0	0.5	1.5	0	1	2.5	2	0	1.5	1.5	1	2	1	0	0	1.5	2	0.5	2.5	0	3
CAM	1.5	1	2.5	0	0	1	1.5	1.5	1.5	3	3	0	2	2.5	1.5	1.5	1	0	0	2	2	2	2	1	2.5
CHU	3	0	2.5	0.5	1	0	0	1.5	2	3	2	1	1.5	3	2	2.5	2.5	1.5	1.5	2	1	2.5	2.5	3	2.5
DUC	3	0	3	1.5	1.5	0	0	3.5	2.5	4	4	0	3	3	2	2.5	1.5	1	2	1.5	1.5	1.5	3	3.5	2.5
LAN	2	4	2.5	0	1.5	1.5	3.5	0	1	3.5	3	3.5	4	2.5	2.5	4	1.5	0.5	0	2.5	3.5	2	2	0	3.5
LEX	1	2.5	2	1	1.5	2	2.5	1	0	2.5	1.5	3	1.5	2	1.5	2	1.5	2.5	2.5	2	2.5	2.5	0.5	2.5	2.5
MAR	1.5	3.5	0	2.5	3	3	4	3.5	2.5	0	1	3	4	4	1	2	2.5	3	1.5	0.5	1.5	3.5	0.5	4	3.5
ROG	1.5	4	1	2	3	2	4	3	1.5	1	0	3.5	2.5	3.5	3	1.5	1	3	1.5	2.5	2	3.5	1	2.5	3.5
ROS	2	0	2.5	0	0	1	0	3.5	3	3	3.5	0	2.5	3	2.5	4	2	1.5	2	2	1	1	1.5	2.5	3.5
TOS	1	2.5	2.5	1.5	2	1.5	3	4	1.5	4	2.5	2.5	0	3	0.5	0	2	1	2.5	1	1	3.5	2	4	0
BAR	3	3	3	1.5	2.5	3	3	2.5	2	4	3.5	3	3	0	3.5	3	0	0	0.5	3	3.5	0	2.5	1	3.5
BEL	1.5	2	1	1	1.5	2	2	2.5	1.5	1	3	2.5	0.5	3.5	0	1	3.5	1.5	1	0	0	2	1	2.5	1
CAL	1.5	2.5	2	2	1.5	2.5	2.5	4	2	2	1.5	4	0	3	1	0	2.5	1.5	1.5	1	0.5	2.5	2	4	0
DIF	1.5	1.5	2	1	1	2.5	1.5	1.5	1.5	2.5	1	2	2	0	3.5	2.5	0	0	0	3.5	3.5	0	2	0.5	4
FIR	2.5	1.5	2.5	0	0	1.5	1	0.5	2.5	3	3	1.5	1	0	1.5	1.5	0	0	0	1.5	1.5	0	2	0	1.5
FRE	3	3	2	0	0	1.5	2	0	2.5	1.5	1.5	2	2.5	0.5	1	1.5	0	0	0	1.5	2.5	1	1.5	0	2
FUM	1.5	1.5	0	1.5	2	2	1.5	2.5	2	0.5	2.5	2	1	3	0	1	3.5	1.5	1.5	0	0	3	0	2.5	0.5
HAD	2.5	1.5	1.5	2	2	1	1.5	3.5	2.5	1.5	2	1	1	3.5	0	0.5	3.5	1.5	2.5	0	0	2.5	0.5	3.5	0.5
HEL	2.5	1.5	2.5	0.5	2	2.5	1.5	2	2.5	3.5	3.5	1	3.5	0	2	2.5	0	0	1	3	2.5	0	2	1	2.5
MZA	0	2.5	0	2.5	2	2.5	3	2	0.5	0.5	1	1.5	2	2.5	1	2	2	2	1.5	0	0.5	2	0	3	1
VER	3	3.5	3	0	1	3	3.5	0	2.5	4	2.5	2.5	4	1	2.5	4	0.5	0	0	2.5	3.5	1	3	0	3.5
VID	1.5	2.5	2.5	3	2.5	2.5	2.5	3.5	2.5	3.5	3.5	3.5	0	3.5	1	0	4	1.5	2	0.5	0.5	2.5	1	3.5	0

Figure 1 - Matrice de fréquentation entre les personnes

3 MDS – Positionnement des individus dans le plan

3.1 Importations des données

Nous importons la première feuille (sheet_name = 0) du fichier "**etudiants_iut.xlsx**" avec la commande read_excel() de la librairie "pandas", en veillant à préciser que la première ligne correspond à l'étiquette des colonnes (header = 0), la première colonne à l'étiquette des lignes (index_col = 0).

```
#charger les données
import pandas
D = pandas.read_excel("Profils.xlsx",sheet_name=0,header=0,index_col=0)

#vérifications des colonnes
print(D.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 25 entries, BEN to VID
Data columns (total 25 columns):
BEN     25 non-null float64
BES   25  non-null  float64
BOU   25  non-null  float64
BRU   25  non-null  float64
CAM   25  non-null  float64
CHU   25  non-null  float64
DUC   25  non-null  float64
LAN   25  non-null  float64
LEX   25  non-null  float64
MAR   25  non-null  float64
ROG   25  non-null  float64
ROS   25  non-null  float64
TOS   25  non-null  float64
BAR   25  non-null  float64
BEL   25  non-null  float64
CAL   25  non-null  float64
DIF   25  non-null  float64
FIR   25  non-null  float64
FRE   25  non-null  float64
FUM   25  non-null  float64
HAD   25  non-null  float64
HEL   25  non-null  float64
MZA   25  non-null  float64
VER   25  non-null  float64
VID   25  non-null  float64
dtypes: float64(25)
memory usage: 5.1+ KB
```

```
#index des lignes
print(D.index)
```

```
Index(['BEN', 'BES', 'BOU', 'BRU', 'CAM', 'CHU', 'DUC', 'LAN', 'LEX', 'MAR',
       'ROG', 'ROS', 'TOS', 'BAR', 'BEL', 'CAL', 'DIF', 'FIR', 'FRE', 'FUM',
       'HAD', 'HEL', 'MZA', 'VER', 'VID'],
      dtype='object', name='IUT')
```

3.2 Positionnement multidimensionnel

Le module manifold du fameux package ''scikit-learn'' propose l'outil MDS qui implémente un MDS métrique "Multidimensional Scaling" (une option permet de réaliser un MDS non métrique, "Analyse en Composantes Principales Non Linéaire" ou "Échelle Multidimensionnelle". Il s'agit d'une technique d'analyse de données utilisée en statistiques et en apprentissage automatique pour visualiser et réduire la dimensionnalité de données.). le MDS cherche à représenter des données multidimensionnelles dans un espace de dimension plus faible tout en préservant autant que possible les distances entre les points de données. Cela permet de visualiser les relations et les similitudes entre les données de manière plus compréhensible dans un espace en deux ou trois dimensions. Nous indiquons le nombre de composantes (n_components = 2), le type de matrice en entrée (dissimilarity = 'precomputed' (cela signifie que la matrice d'entrée est une matrice de dissimilarité précalculée plutôt qu'une matrice de données brutes); nous présentons à l'algorithme une matrice de distances ou de dissimilarités). (random_state = 1) (obtiendrez les mêmes résultats chaque fois que vous exécuterez le code). Après avoir instancié l'objet, nous lançons les calculs à l'aide de la commande fit().

```
#librairie pour MDS
from sklearn import manifold

#MDS
mds = manifold.MDS(n_components=2,random_state=1,dissimilarity="precomputed")

#apprentissage
mds.fit(D)
```

Les coordonnées dans le nouvel espace de représentation font partie des propriétés de l'objet :

```
#coordonnées des points dans le plan puisque (n_components = 2)
points = mds.embedding_
print(points)
```

```
[[ 0.60523707  1.1468044 ]
 [-1.49693267 -0.83407788]
 [ 0.63323177  1.39263585]
 [ 0.0414655  -0.80101129]
 [-0.22663826 -0.67679576]
 [-1.18574994 -0.55823329]
 [-1.45642689 -0.94930949]
...
 [ 0.52752546 -0.72365751]
 [-0.34610638  0.97662    ]
 [-0.89926011  0.84508419]
 [ 0.21037635 -1.48529244]
 [ 0.42104059  0.99613963]
 [ 1.20188218 -1.43807234]
 [-1.21400764  1.35968378]]
```

Nous disposons de n = 25 observations (profils) et p = n_components = 2 variables.

Un graphique nuage de points permet de situer les positions relatives des individus. Nous nous appuyons sur la très populaire librairie ''matplotlib''.

```
#librairie graphique
import matplotlib.pyplot as plt

#définir la taille du graphique
ax = plt.axes([0,0,2,2])

#ajuster le ratio abscisse - ordonnée
ax.set_aspect(aspect='equal')

#placer les points dans le plan (abscisse, ordonnée)
plt.scatter(points[:,0],points[:,1],color='silver',s=150)

#ajouter les étiquettes dans le graphique
for i in range(D.shape[0]):
    ax.annotate(D.index[i],(points[i,0],points[i,1]),color='black')

#afficher le graphique
plt.show()
```

L'ajustement du ratio entre l'abscisse et ordonnées à (aspect = 'equal') est primordial. En effet, le pouvoir de représentation des axes dépend de la dispersion des points. Pour rendre compte correctement des proximités, il faut absolument que les échelles en abscisse et en ordonnées soient identiques.Nous obtenons le graphique suivant :

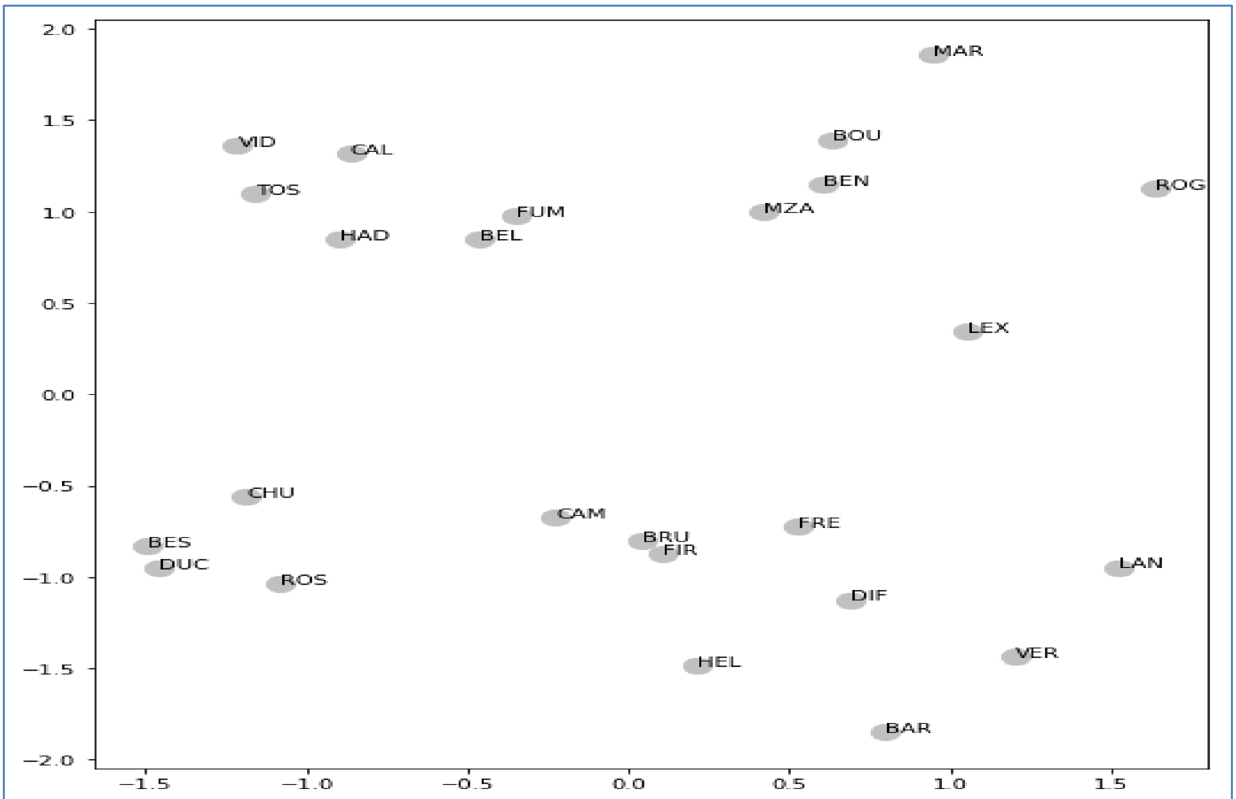

Figure 2 - Positionnement des étudiants dans le plan

Des individus avec une distance nulle ne sont pas superposés (ex. BRU et FIR, BES et DUC, etc.) parce qu'ils doivent composer avec leurs distances aux autres points. Mais ils sont néanmoins proches. Il faut mesurer la qualité de la représentation pour jauger la fidélité du nuage de points.

3.3 Qualité de la représentation – STRESS

L'outil MDS fournit une valeur de STRESS censée quantifier la qualité de la représentation.

```
#qualité --- valeur du stress MDS
print(mds.stress_)
```

```
107.87240219831337
```

l'indicateur opposait simplement les distances initiales aux distances mesurées dans le plan factoriel, sans normalisation.

$$stress = \sum_{j>i}(\hat{\delta}_{ij} - \delta_{ij})^2 = \frac{1}{2}\sum_{j\neq i}(\hat{\delta}_{ij} - \delta_{ij})^2$$

Où δ_{ij} correspond à la distance observée entre paires d'individus (i, j) fournie en entrée de l'algorithme de positionnement multidimensionnel. Et $\hat{\delta}_{ij}$ est la distance reconstituée dans l'espace de représentation euclidienne des données.

Sous Python, nous calculons les distances euclidiennes (distances estimées) entre paires d'individus à partir de leurs coordonnées.

```
#calculer les distances entre paires d'individus dans le plan
from sklearn.metrics import euclidean_distances
DE = euclidean_distances(points)
print(DE.shape) # matrice (25, 25)

#distances estimées entre paires d'individus -- 5 premières lignes et colonnes
print(DE[0:5,0:5])
```

```
[[0.         2.88842729 0.2474203  2.02776339 2.00437874]
 [2.88842729 0.         3.08153445 1.53875349 1.27999435]
 [0.2474203  3.08153445 0.         2.27206406 2.2409649 ]
 [2.02776339 1.53875349 2.27206406 0.         0.29548117]
 [2.00437874 1.27999435 2.2409649  0.29548117 0.        ]]
```

Nous pouvons obtenir le stress en opposant distances estimées et observées.

```
#vérification du STRESS de MDS
stress = 0.5 * numpy.sum((DE - D.values)**2)
print(stress)
```

```
107.85853039440943
```

Aux erreurs de précision près, nous avons bien la valeur fournie directement par l'outil MDS.

Nous le normalisons pour obtenir le ''Kruskal stress'' (stress1) qui est un indicateur reconnu dans la littérature :

$$stress1 = \sqrt{\frac{\sum_{j>i}\left(\hat{\delta}_{ij} - \delta_{ij}\right)^2}{\sum_{j>i}\left(\delta_{ij}\right)^2}}$$

Pour nos données, nous obtenons :

```
#kruskal stress (ou stress - formula 1)
stress1 = numpy.sqrt(stress/(0.5*numpy.sum(D.values**2)))
print(stress1)
```

```
0.27194058819275735
```

Stress1	Qualité
0.2	Poor
0.1	Fair
0.05	Good
0.025	Excellent
0.0	Perfect

0.2719 n'est pas une très bonne valeur semble-t-il.

Mais nous passons outre en espérant que la représentation dans le plan des individus soit suffisamment fidèle pour que l'algorithme de *clustering* que nous utiliserons par la suite permette de constituer des groupes (identifier des communautés) qui ont une certaine consistance.

4 K-Means – Détection des communautés

La méthode des K-Means est un algorithme de *clustering* (classification automatique, en français). Il permet de circonscrire les groupes d'observations similaires (proches) à partir d'un tableau ''individus x variables''. Sa popularité repose en grande partie sur sa simplicité et sa souplesse.

4.1 K-Means en 4 classes

Nous avions montré la mise en œuvre des K-Means sous Python (package ''scikit-learn'') dans un précédent tutoriel (Mars 2016). Nous allons donc à l'essentiel dans ce document. Précision très importante, dans notre contexte, il ne faut surtout pas standardiser les variables issues du MDS avant de lancer l'algorithme de clustering, d'une part parce qu'elles sont centrées par nature, d'autre part parce que les dispersions rendent compte des proximités entre les points, annihiler cette information en réduisant les variables faussera les résultats.

Nous faisons appel à la classe KMeans du package ''scikit-learn''. Nous demandons (n_clusters = 4) classes à la lumière du graphique nuage de points ci-dessus (Figure 2).

Barycentres conditionnels. Nous affichons les barycentres conditionnels (centroïdes) des groupes.

```
#K-means
from sklearn.cluster import KMeans
km = KMeans(n_clusters=4,random_state=1)

#modélisation
km.fit(points)

#barycentres
print(km.cluster_centers_)
```

```
[[-0.82384387  1.07412063]
 [ 0.54107733 -1.102469  ]
 [ 0.88273353  1.14304518]
 [-1.30575849 -0.84519346]]
```

Les valeurs brutes comme cela sont un peu arides, mieux vaut les situer dans le plan factoriel.

```
#graphique avec les barycentres
Import matplotlib.pyplot as plt
ax = plt.axes([0,0,2,2])
ax.set_aspect(aspect='equal')
plt.scatter(points[:,0],points[:,1],color='silver',s=100)
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='blue',s=150)
```

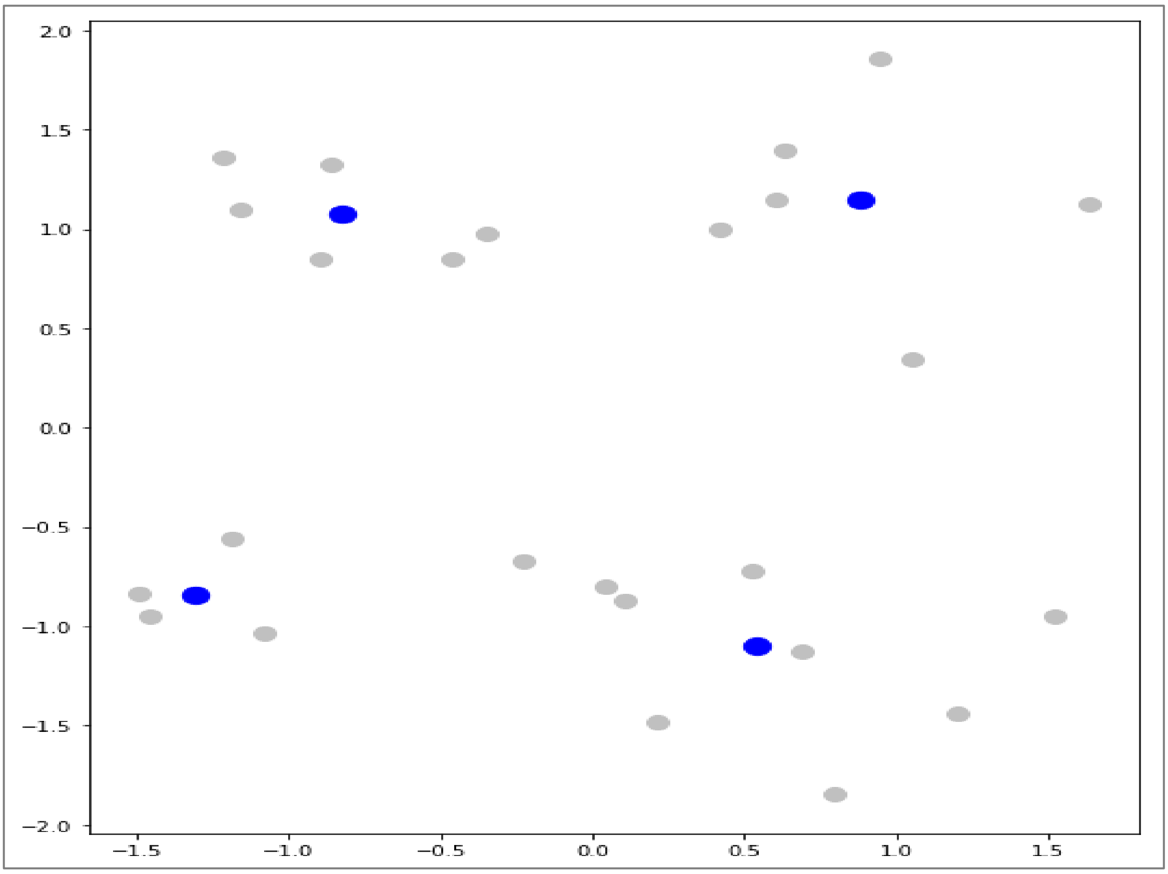

Figure 3 - Nuage de points avec les barycentres des 4 clusters (en bleu)

Groupes d'appartenance. Une autre manière d'illustrer les résultats est d'identifier le groupe d'appartenance de chaque individu.

```
#groupe d'appartenance
print(km.labels_)
```

```
[2 3 2 1 1 3 3 1 2 2 2 3 0 1 0 0 1 1 1 0 0 1 2 1 0]
```

Le 1er individu BEN appartient au groupe n°2, BES appartient au n°3, BOU au n°2, etc.

Puisque nous travaillons dans le plan, un graphique où l'on attribuerait une couleur prédéfinie à chaque groupe permettra de mieux situer leurs situations respectives.

```
#couleurs pour groupes d'appartenance
import numpy
couleurs_groupes = numpy.array(['turquoise','violet','goldenrod','lightsalmon'])[km.labels_]

#graphique avec les couleurs pour les groupes
ax = plt.axes([0,0,2,2])
ax.set_aspect(aspect='equal')
plt.scatter(points[:,0],points[:,1],color=couleurs_groupes,s=150)
for i in range(D.shape[0]): ax.annotate(D.index[i],(points[i,0],points[i,1]),color='black')
plt.show()
```

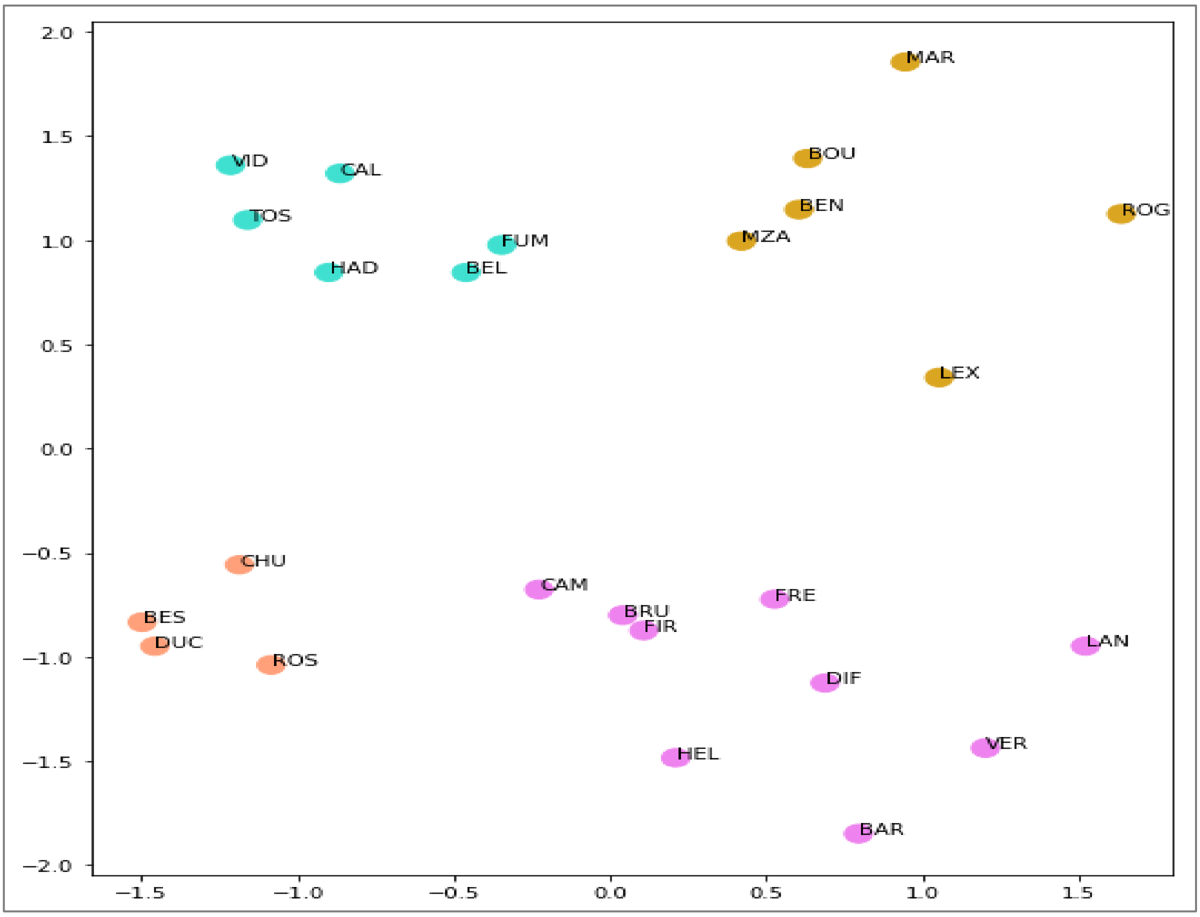

Figure 4 - Position des individus dans le plan et groupes d'appartenance

4.2 Détermination du nombre de classes

La détermination du nombre de clusters (K) est l'arlésienne de la classification automatique.. Une piste simple consiste à le faire varier et à surveiller l'évolution d'un indicateur numérique qui rendrait compte de la qualité de la partition. ''scikit-learn'' propose plusieurs critères, dont le coefficient silhouette qui présente l'incommensurable avantage d'être insensible au nombre de groupes. De fait, nous nous plaçons dans une situation d'optimisation simple : le bon nombre de clusters correspond à la configuration qui maximise le coefficient silhouette.

Pour ce faire, nous balayons les différentes solutions allant de K = 2 à K = 6 groupes.

```
#outil pour le calcul du coefficient silhouette
from sklearn.metrics import silhouette_score
#balayage des différentes valeurs de K
sil = numpy.zeros(5)
for K in range(2,7):
    kms = KMeans(n_clusters=K,random_state=1).fit(points)
    sil[K-2] = silhouette_score(points,kms.labels_,metric='euclidean')
#valeurs des silhouettes
print(sil)
```

```
[0.47240332 0.49295122 0.57465299 0.52895001 0.49251385]
```

Un graphique, c'est toujours mieux pour situer la valeur optimale K*.

```
#graphique - partition en 4 classes justifiée
plt.plot(range(2,7),sil,'--o',color='deepskyblue',linewidth=2,markersize=10)
```

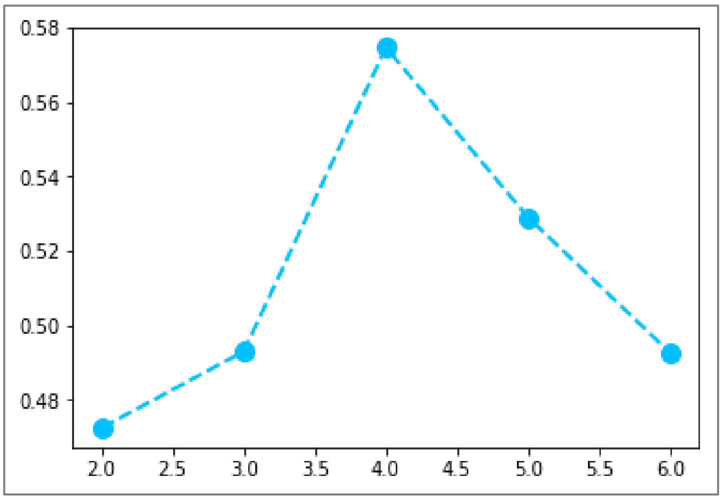

Figure 5 - Coefficient silhouette en fonction du nombre de classes K

K* = 4 semble être la solution la plus satisfaisante. L'intuition que nous avions eu en inspectant le graphique nuage de points est confirmée par le critère silhouette.

4.3 Lien entre amis proches et appartenance aux communautés

En analysant les groupes, les profils proches (avec une dissimilarité nulle) étaient situés dans des communautés (classes) différentes. Dans cette section, nous essayons de les identifier. S'ils sont trop nombreux, cela poserait la question de la qualité de la partition.

Nous procédons en plusieurs étapes. Tout d'abord, nous définissons une liste segments qui contient les paires de points qui doivent être reliés parce que leur distance est nulle. Pour ce faire, nous passons en revue les camarades de chaque profil.

```
#lien entre amis
segments = []

#travailler sur la partie traingulaire de la matrice de dissimilarités
for i in range(0,D.shape[0]-1): for
    j in range(i+1,D.shape[0]):
        #uniquement si la distance est nulle
        if (D.iloc[i,j] == 0):
            segments.append([points[i,:],points[j,:]])

#nombre de segments concernés
print(len(segments))
```

```
37
```

37 individus sont très proches ($\delta = 0$) dans notre matrice de données.

La première liaison par exemple…

```
#1er segment concerné
print(segments[0])
```

```
[array([0.60523707, 1.1468044 ]), array([0.42104059, 0.99613963])]
```

… concerne les individus de coordonnées (0.60523707, 1.1468044) [**BEN**] et (0.42104059, 0.99613963) [**MZA**], qui appartiennent à la même communauté (cluster) (Figure 4).

Nous construisons ensuite une collection de ''lignes'' (LineCollection) qui vont matérialiser chaque segment dans le graphique que nous allons construire.

```
#collection de lignes
from matplotlib.collections import LineCollection
lc = LineCollection(segments=segments,color='lightgray')
```

Il ne reste plus qu'à afficher le graphique avec les lignes reliant les amis proches.

```
#graphique avec les couleurs pour les groupes et les segments des amis
ax = plt.axes([0,0,2,2])
ax.set_aspect(aspect='equal')
plt.scatter(points[:,0],points[:,1],color=couleurs_groupes,s=150)
for i in range(D.shape[0]): ax.annotate(D.index[i],(points[i,0],points[i,1]),color='black')
ax.add_collection(lc)
plt.show()
```

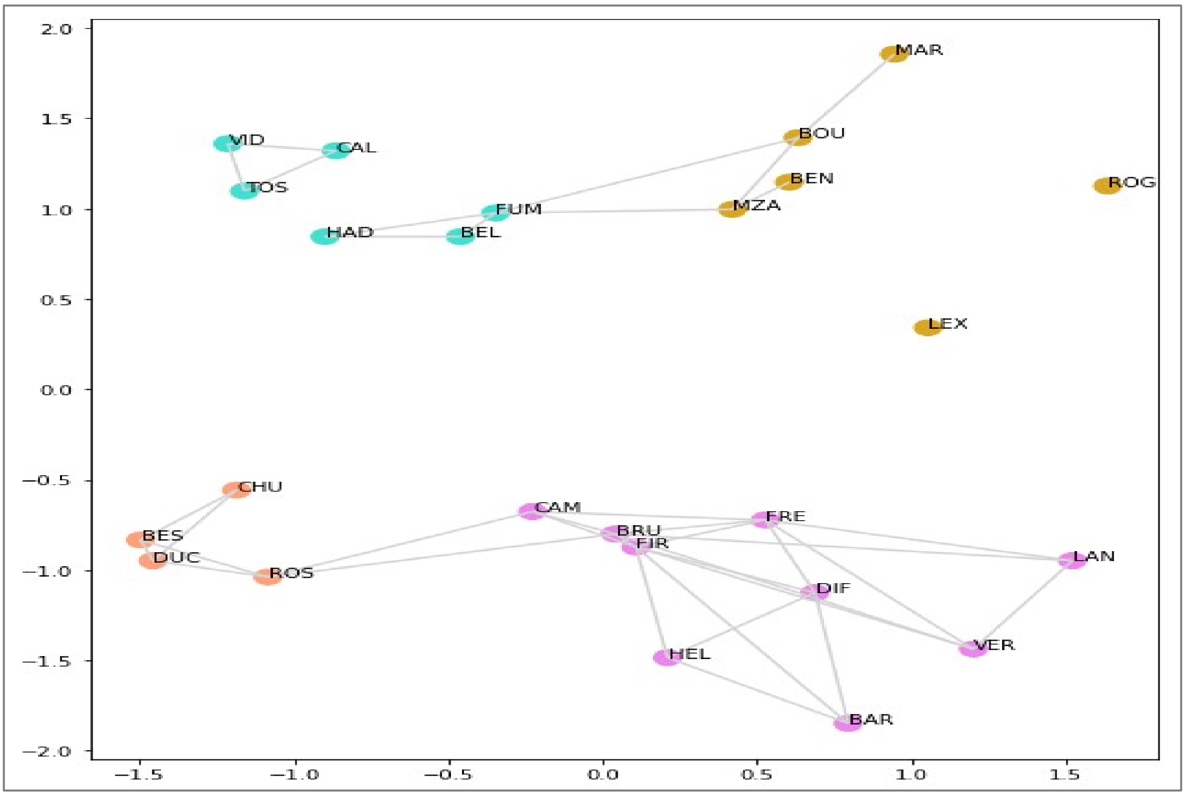

Figure 6 - Communautés et amitiés fortes

Les amitiés fortes sont avant tout intracommunautaires. C'est un signal très positif pour la fiabilité de nos résultats. Mais nous constatons quand-même qu'il existe des rares proximités intercommunautaires (FUM avec BOU et MZA ; ROS avec CAM et BRU).

5 Conclusion

L'objectif premier de cet atelier était de montrer, sous Python, l'utilisation du positionnement multidimensionnel (MDS) pour la détection des communautés dans les réseaux sociaux. La projection des individus dans un espace euclidien permet à la fois de représenter leurs positions respectives dans le plan, et de mettre en œuvre l'algorithme des K-Means pour identifier les groupes.