Université CADI AYYAD

FACULTE DES SCIENCES SEMLALIA MARRAKECH

RAPPORT DE PROJET DE FIN DE MODULE

Pour le module du titre :

PYTHON POUR SCIENCE DE DONNEES

Sujet :

ETUDE D'UN RESEAU COMPLEXE & IDENTIFIANT LES TOP-K NŒUDS & ET LES COMMUNAUTES ASSOCIES

REALISE PAR :

EDDAMIR AMINE

SOUIDI YASSINE

ENCADRE PAR:

PR. QAFFOU

Année Universitaire: 2021/2022

Résumé

Le présent rapport s'inscrit dans le cadre du projet de fin de module portant sur l'étude d'un réseau complexe.

Pour mener à bien notre projet, nous avons entamé notre travail par une étude théorique de chaque notion, ainsi que, nous nous sommes focalisés sur une étude technique et pratique pour maitriser les bibliothèques et les outils python utilisé.

Notre projet consiste à étudier un réseau complexe , basé sur une base des données facebook afin d'extraire les top-K nœuds plus influant ainsi que les communautés qui peuvent être associés à ces nœuds .

Abstract

This report is part of the end-of-module project on the study of a complex network.

To carry out our project, we began our work with a theoretical study of each concept, as well as, we focused on a technical and practical study to master the libraries and the python tools used.

Our project consists in studying a complex network, based on a facebook database in order to extract the most influential top-K nodes as well as the communities that can be associated with these nodes.

Tables Des Matières

CHAPITRE I : TOPSIS

- Définition
- Méthode
- Résultat
- Modèle SIR

Ce premier chapitre définit TOPSIS ainsi que les étapes à suivre dans cette méthode. Les résultat de notre application de cette méthode sur dataset , ainsi une comparaison du résultat TOPSIS et les différents critères en implémentant le modèle SIR .

1. TOPSIS

1. Définition

La **technique d'ordre de préférence par similitude avec la solution idéale** (**TOPSIS**) est une méthode d'analyse décisionnelle multicritère, qui a été développée à l'origine par Ching-Lai Hwang et Yoon en 1981 avec d'autres développements par Yoon en 1987 et Hwang, Lai et Liu en 1993.[] TOPSIS est basé sur le concept selon lequel l'alternative choisie doit avoir la distance géométrique la plus courte de la solution idéale positive (PIS)et la distance géométrique la plus longue de la solution idéale négative (NIS).

Il s'agit d'une méthode d'agrégation compensatoire qui compare un ensemble d'alternatives en identifiant des pondérations pour chaque critère, en normalisant les scores pour chaque critère et en calculant la distance géométrique entre chaque alternative et l'alternative idéale, qui est la meilleure note dans chaque critère. Une hypothèse de TOPSIS est que les critères augmentent ou diminuent de manière monotone. La normalisation est généralement nécessaire car les paramètres ou les critères sont souvent de dimensions incongrues dans les problèmes multicritères. Les méthodes compensatoires telles que TOPSIS permettent des compromis entre les critères, où un mauvais résultat dans un critère peut être annulé par un bon résultat dans un autre critère. Cela fournit une forme de modélisation plus réaliste que les méthodes non compensatoires, qui incluent ou excluent des solutions alternatives basées sur des seuils stricts.

TOPSIS est une méthode d'analyse multicritère. Dans ce projet, nous utilisons quatre critères :

Degré de Centralité : *La centralité du degré* est la mesure de centralité la plus simple à calculer. Rappelez-vous que le degré d'un nœud est simplement un décompte du nombre de connexions sociales (c'est-à-dire les arêtes) qu'il a. Le degré centralité d'un nœud est simplement son degré. Un nœud avec 10 connexions sociales aurait un degré de centralité de 10. Un nœud avec 1 arête aurait un degré de centralité de 1. Parfois, un programme SNA convertira ces nombres en une échelle de 0 à 1. Dans de tels cas, le nœud avec le degré le plus élevé dans le réseau aura une centralité de degré de 1, et la centralité de chaque autre nœud sera la fraction de son degré par rapport à ce nœud le plus populaire.

Par exemple, si le nœud de degré le plus élevé d'un réseau a 20 arêtes, un nœud avec 10 arêtes aurait une centralité de degré de 0,5 (10 ÷ 20). Un nœud avec un degré de 2 aurait un degré de centralité de 0,1 (2 ÷ 20). Pour la centralité de degré, **des valeurs plus élevées** signifient que le nœud est plus central. Comme mentionné ci-dessus, chaque mesure de centralité indique un type d'importance différent. La centralité du degré indique le nombre de connexions qu'une personne a. Ils peuvent être connectés à de nombreuses personnes au cœur du réseau, mais ils peuvent aussi être loin à la périphérie du réseau. Par exemple, dans la figure 21.5, les deux nœuds étiquetés « Bob » ont le même degré élevé (c.-à-d. beaucoup de liens sociaux, 9 dans ce cas), mais les deux rôles qu'ils jouent sont très différents. Celui de droite est très central et celui de gauche est périphérique. Ceux-ci montrent que si la centralité du degré nous indique avec précision qui a beaucoup de liens sociaux, elle ne montre pas nécessairement qui est au « milieu » du réseau.

Betweenness Centrality : En théorie des graphes, **Betweenness Centrality** est une mesure de la centralité dans un graphe basée sur les chemins les plus courts. Pour chaque paire de sommets dans un graphe connecté, il existe au moins un chemin le plus court entre les sommets de telle sorte que soit le nombre d'arêtes traversées par le chemin (pour les graphes non pondérés), soit la somme des poids des arêtes (pour les graphes pondérés) soit minimisé. Betweenness Centrality pour chaque sommet est le nombre de ces chemins les plus courts qui traversent le sommet. Betweenness Centrality a été conçue comme une mesure générale de la centralité elle s'applique à un large éventail de problèmes en théorie des réseaux, y compris les problèmes liés aux réseaux sociaux, à la biologie, aux transports et à la coopération scientifique. Bien que les auteurs précédents aient intuitivement décrit la centralité comme basée sur l'entre-deux, Freeman (1977) a donné la première définition formelle de la betweenness centrality. Betweenness centrality trouve une large application dans la théorie des réseaux; il représente le degré auquel les nœuds se tiennent entre eux. Par exemple, dans un réseau de télécommunications, un nœud avec une BC plus élevée aurait plus de contrôle sur le réseau, car plus d'informations passeront par ce nœud.

La centralité de proximité indique la proximité d'un nœud avec tous les autres nœuds du réseau. Il est calculé comme la moyenne de la longueur de chemin la plus courte du nœud à tous les autres nœuds du réseau. Dans le cas de la centralité de proximité, ou de la longueur moyenne du chemin le plus court, des valeurs plus faibles indiquent plus de nœuds centraux. Les avantages de la centralité de proximité sont qu'elle indique que les nœuds sont plus centraux s'ils sont plus proches de la plupart des nœuds du graphique. Cela correspond fortement à la centralité visuelle – un nœud qui apparaîtrait vers le centre d'un graphique lorsque nous le dessinons a généralement une centralité de proximité élevée.

La centralité du vecteur propre est une vision plus sophistiquée de la centralité : une personne avec peu de connexions pourrait avoir une centralité de vecteur propre très élevée si ces quelques connexions étaient à d'autres très bien connectées. La centralité du vecteur propre permet aux connexions d'avoir une valeur variable, de sorte que la connexion à certains sommets présente plus d'avantages que la connexion à d'autres. L'algorithme PageRank utilisé par le moteur de recherche de Google est une variante d'Eigenvector Centrality, principalement utilisé pour les réseaux dirigés. Le PageRank tient compte (1) du nombre de liens entrants (c.-à-d. les sites qui renvoient à votre site), (2) de la qualité des linkers (c.-à-d. le PageRank des sites qui renvoient à votre site) et (3) de la propension aux liens des linkers (c'est-à-dire le nombre de sites vers lesquels les linkers renvoient).

2. Méthode :

Le processus TOPSIS est effectué comme suit:

Étape 1 : Créer une matrice d'évaluation composée de m alternatives et de n critères , avec l'intersection de chaque alternative et des critères données suit : xij , nous avons donc une matrice (xij)m*n.

Étape 2 : La matrice (xij)m*n est ensuite normalisé pour former la matrice R = (rij)m*n , à l'aide de la méthode de normalisation

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{k=1}^{m} x_{kj}^2}}, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n$$

Étape 3 : Calculer la matrice de décision normalisée pondérée

$$t_{ij} = r_{ij} \cdot w_j, \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n$$

where $w_j = W_j \Big/ \sum_{k=1}^{n} W_k, j = 1, 2, \ldots, n$ so that $\sum_{i=1}^{n} w_i = 1$, and W_j is the original weight given to the indicator $v_j, \quad j = 1, 2, \ldots, n.$

Étape 4 : Déterminer la pire alternative (Aw) et la meilleure alternative (Ab)

$$A_w = \{\langle \max(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_- \rangle, \langle \min(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_+ \rangle\} \equiv \{t_{wj} \mid j = 1, 2, \ldots, n\},$$
$$A_b = \{\langle \min(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_- \rangle, \langle \max(t_{ij} \mid i = 1, 2, \ldots, m) \mid j \in J_+ \rangle\} \equiv \{t_{bj} \mid j = 1, 2, \ldots, n\},$$

where,

$J_+ = \{j = 1, 2, \ldots, n \mid j\}$ associated with the criteria having a positive impact, and

$J_- = \{j = 1, 2, \ldots, n \mid j\}$ associated with the criteria having a negative impact.

Étape 5 : Calculer la distance entre l'alternative cible i et la pire condition Aw

Calculate the L^2-distance between the target alternative i and the worst condition A_w

$$d_{iw} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{wj})^2}, \quad i = 1, 2, \ldots, m,$$

and the distance between the alternative i and the best condition A_b

$$d_{ib} = \sqrt{\sum_{j=1}^{n} (t_{ij} - t_{bj})^2}, \quad i = 1, 2, \ldots, m$$

where d_{iw} and d_{ib} are L^2-norm distances from the target alternative i to the worst and best conditions, respectively.

Étape 6 : Calculez la similitude avec la pire condition:

$$s_{iw} = d_{iw}/(d_{iw} + d_{ib}), \quad 0 \leq s_{iw} \leq 1, \quad i = 1, 2, \ldots, m.$$

$s_{iw} = 1$ if and only if the alternative solution has the best condition; and

$s_{iw} = 0$ if and only if the alternative solution has the worst condition.

Étape 7 : Classez les alternatives en fonction de siw

3. Application & Résultat :

```python
from matplotlib.pyplot import prism
import networkx as nx
import pandas as pd
import numpy as np
g = nx.read_edgelist("facebook_combined.txt",create_using=nx.Graph(), nodetype = int)
# Creates pandas DataFrame.
list_of_nodes=np.array(list(g.nodes()))
Dic = {
  'BC':nx.betweenness_centrality(g),
  'CC':nx.closeness_centrality(g),
  'DC':nx.degree_centrality(g),
  'EC':nx.eigenvector_centrality(g, max_iter=5000)}

df=pd.DataFrame(data=Dic,index=list_of_nodes)
print(df.head())
```

La normalisation :

```python
# Normalize
N=[]
for j in range(len(df.columns)):
        S=0
        for i in range(len(df)):
                S+=np.square(df.iloc[i,j])
        N.append(np.sqrt(S))
        df.iloc[:,j]/=N[j]

print(df.head())
```

```python
#weight
df['BC']*=0.3
df['CC']*=0.3
df['DC']*=0.2
df['EC']*=0.2
```

Calcul du V+ et V- :

```
# Calcul v+ et v-
Max=df.max()
Min=df.min()
VP=np.array(Max)
VN=np.array(Min)
```

Calcul du S :

```
#Calcul S
SP=[]
SN=[]
C=[]
for i in range(len(df)):
        S_Pos=0
        S_Neg=0
        for j in range(len(df.columns)):
                S_Pos +=np.square(VP[j] - df.iloc[i,j])
                S_Neg +=np.square(VN[j] - df.iloc[i,j])
        SP.append(np.sqrt(S_Pos))
        SN.append(np.sqrt(S_Neg))
        Aux=SN[i]/(SN[i]+SP[i])
        C.append(Aux)
df['S+']=SP
df['S-']=SN
df['C']=C
print(df.head())
df.to_csv('TOPSIS.R2.csv')
```

Résultat :

```
df2 = pd.DataFrame({'nodes' : list_of_nodes,'C' : C})
df2.sort_values(by=['C'],ascending=False, inplace=True)
K = int(input("Entrer le nombre K de noeuds plus influant que vous voulez voir : "))
for i in range(0,K,1):
    print(df2.iloc[i,0])
```

```
Entrer le nombre K de noeuds plus influant que vous voulez voir : 10
107
1684
1912
3437
0
1085
698
567
58
428
```

4. Modèle SI & SIR :

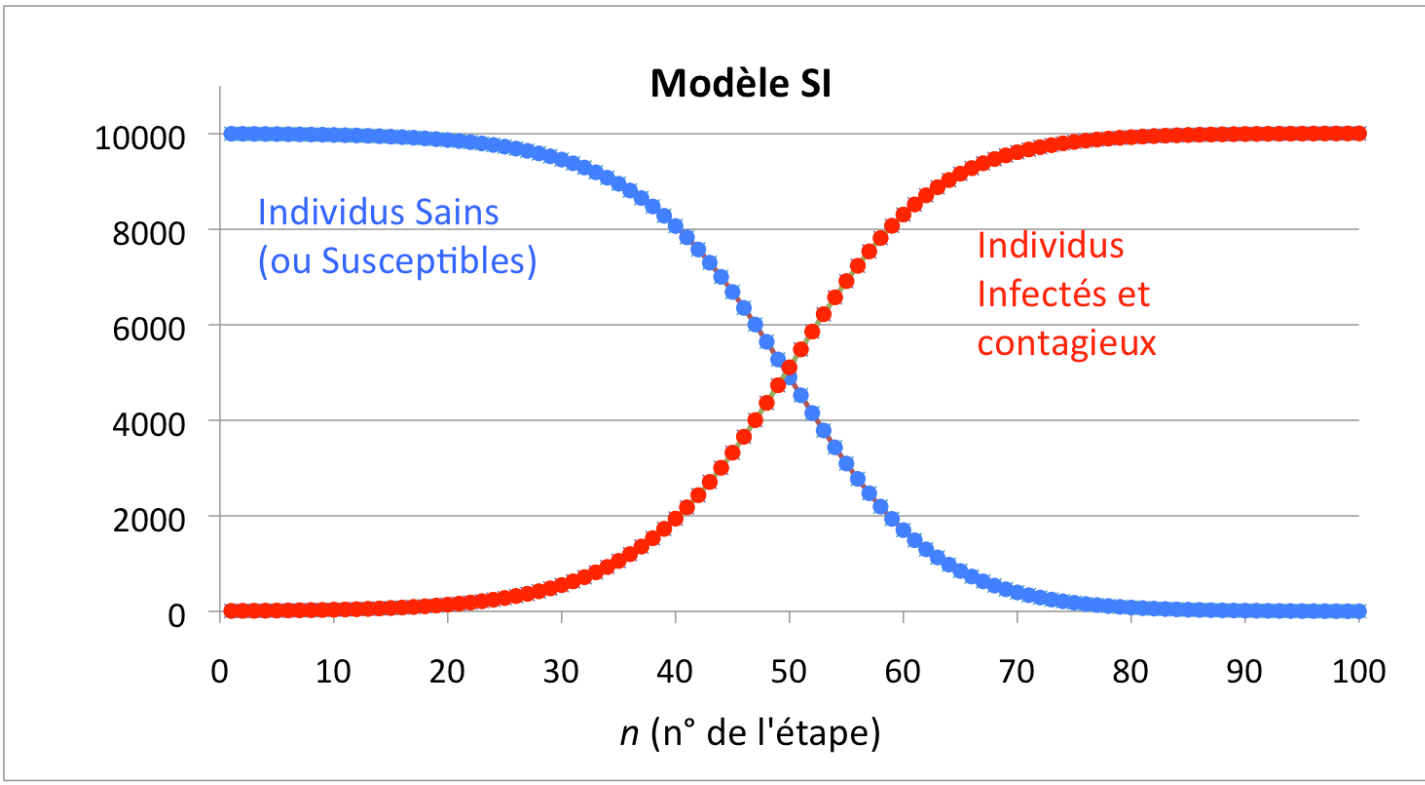
Le modèle SI : Soit une population qui à un instant donné contient *S* personnes saines *Susceptibles* d'être infectées , et *I* personnes *Infectées* contagieuses. Peut-on décrire simplement la vitesse à laquelle les effectifs de ces deux groupes vont évoluer ? Pour cela nous allons supposer que tout se passe comme si les rencontres entre personnes étaient aléatoires. Plus la proportion de personnes infectées est élevée, plus la probabilité qu'une rencontre se fasse avec une telle personne est élevée. Ainsi on peut supposer que la probabilité qu'une personne rencontrée par une Susceptible soit Infectée, est proportionnelle au nombre *I* de personnes infectées. De plus le nombre total de telles rencontres se produisant dans un laps de temps donné est proportionnel au nombre total de personnes susceptibles *S* : il est donc proportionnel au produit *IS*. Le nombre de nouvelles contaminations qui se produisent pendant le laps de temps peut donc s'écrire BIS où B est un coefficient

de proportionnalité. Ainsi pendant l'intervalle de temps considéré, le nombre des Susceptibles a diminué de BIS et le nombre d'Infectés a augmenté de la même quantité. Si le même processus se répète à chaque intervalle de temps, l'évolution des nombres *S* et *I* lors du passage d'une étape « *n* » à l'étape « *n+1* » est décrite par les relations :

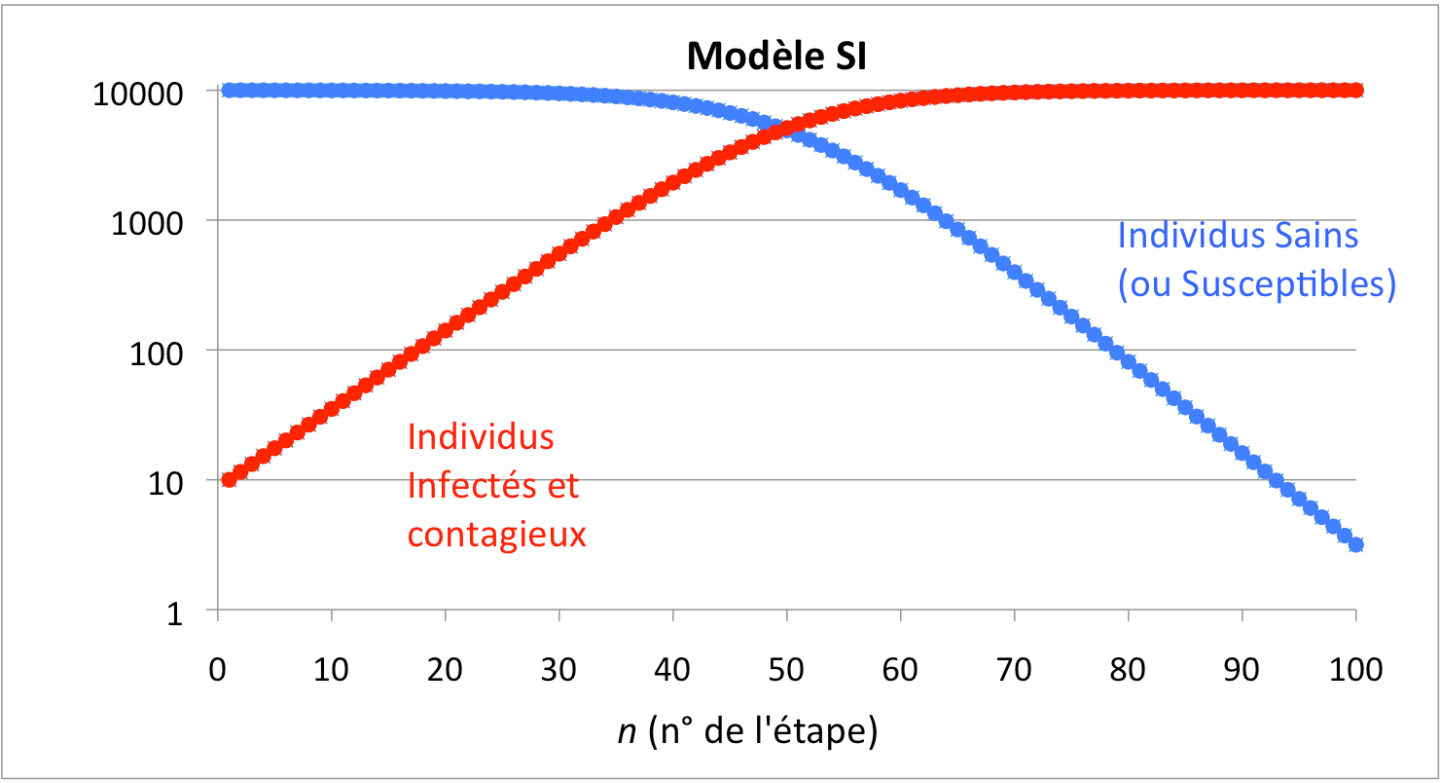
$$\begin{cases} S(n+1) = S(n) - \beta I(n) S(n) \\ I(n+1) = I(n) + \beta I(n) S(n) \end{cases}$$

Le coefficient B est un nombre d'autant plus élevé que les rencontres qui ont lieu pendant un intervalle de temps sont fréquentes et contaminantes. Voilà, nous avons déjà un modèle mathématique de propagation d'une épidémie ! Appelons-le le modèle SI. Pour l'implémenter il suffit de choisir des valeurs initiales *I(1)* et *S(1)* et une valeur de B . On peut ensuite calculer de proche en proche toutes les valeurs de *S*(*n*) et *I*(*n*), par exemple en utilisant un tableur. On obtient typiquement ce genre d'évolution pour les effectifs des deux groupes.

SI LINEAIRE :



SI LOG :



Comme dans ce modèle minimaliste tous les infectés restent contagieux (ne guérissent ni ne décèdent) tout le monde fini par être atteint. Le tracé en échelles semi-logarithmiques à droite montre que jusque vers la 40ième étape la croissance du nombre d'infectés est exponentielle (une exponentielle donne une droite en échelles semi-log). C'est une conséquence des hypothèses du modèle. En effet les fonctions exponentielles sont celles dont l'accroissement d'une étape *n* à l'étape *n+1* est proportionnel à leur valeur à l'étape *n* (si chaque étape correspond à un laps de temps très court pendant lequel les nombres varient peu). Or c'est ce que traduit la relation $I(n+1) = I(n) + \beta I(n) S(n)$

écrite plus haut : tant que l'épidémie n'a pas trop progressé, le nombre *S* n'a pas beaucoup varié en valeur relative, tout se passe à peu près comme s'il était resté égal à *S(1)* et l'on a $I(n+1) - I(n) \approx \beta S(1) I(n)$

L'accroissement de *I* est bien approximativement proportionnel à *I*. Remarquez qu'on peut faire un raisonnement analogue pour la fin de la décroissance de *S* qui est une décroissance exponentielle.

Le modèle SIR : Nous allons maintenant voir qu'il est assez simple de compléter le modèle. Par exemple nous allons introduire le fait qu'après un certain temps, les personnes infectées contagieuses guérissent et sont immunisées : on les dénomme alors par la lettre *R* (pour « Recovered » en anglais). On fait l'hypothèse (encore une !) qu'à chaque étape le nombre de personnes Infectées qui guérissent est proportionnel à leur effectif *I*, c'est à dire que les guérisons diminuent *I* d'une quantité de la forme γI . Autrement dit la relation précédente :

$$I(n+1) = I(n) + \beta I(n)S(n)$$

Devient :

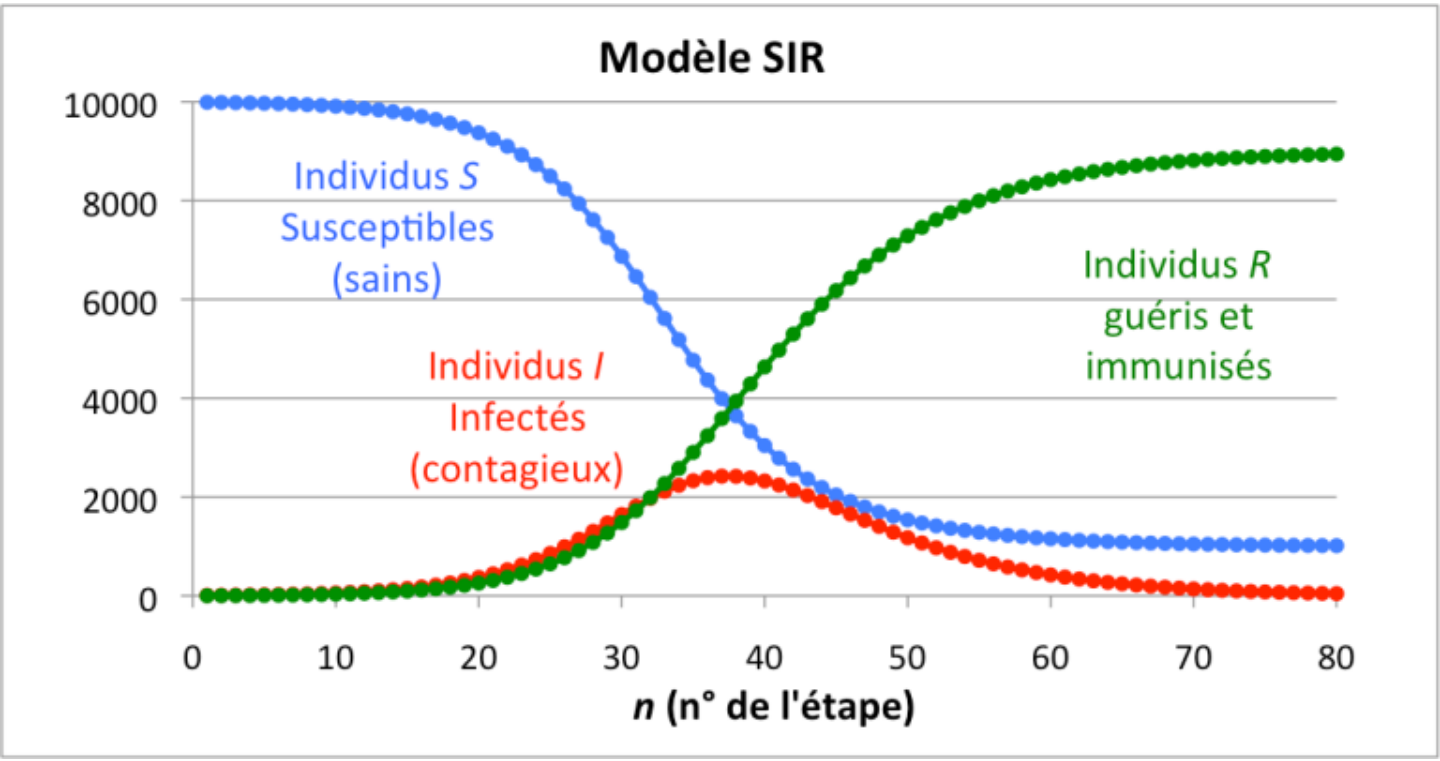
$$I(n+1) = I(n) + \beta I(n)S(n) - \gamma I(n)$$

D'autre part l'effectif *R* du nouveau groupe des malades guéris croit selon $R(n+1) = R(n) + \gamma I(n)$. Le modèle est donc maintenant constitué des trois relations :

$$\begin{cases} S(n+1) = S(n) - \beta I(n)S(n) \\ I(n+1) = I(n) + \beta I(n)S(n) - \gamma I(n) \\ R(n+1) = R(n) + \gamma I(n) \end{cases}$$

Comme précédemment on se donne un jeu de valeurs initiales *S(1)*, *I(1)* et *R(1)*, ainsi que des valeurs pour β et γ , et l'on calcule les *S(n)*, I*(n)* et *R(n)* pas à pas à l'aide des trois relations de récurrence. On obtient ce genre d'évolutions :

Modéle SIR courbe :



L'état contagieux est maintenant un état transitoire, dont l'effectif décrit un pic. Dans notre exemple l'état final est majoritairement constitué de personnes guéries, l'effectif qui n'a pas été touché (1000 personnes) est d'autant plus élevé que le paramètre γ est élevé c'est à dire que la guérison est rapide et « empêche » les personnes Infectées d'en contaminer d'autres.

Code Python où on a appliquer le modèle SIR dans notre cas en utilisant la bibliothèque python EoN :

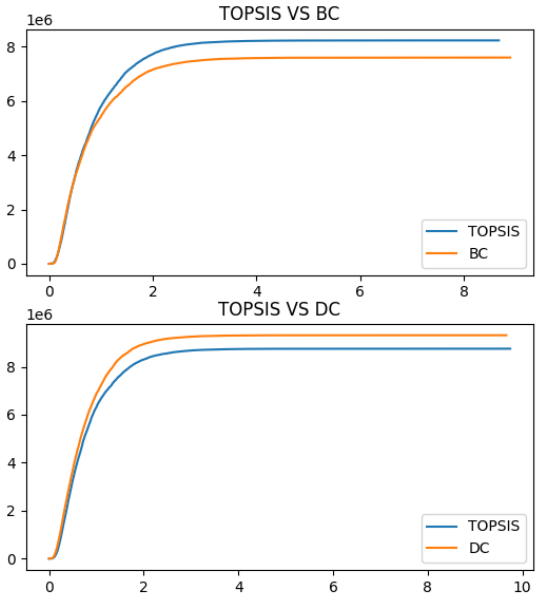
Préparer les listes pour chaque critères :

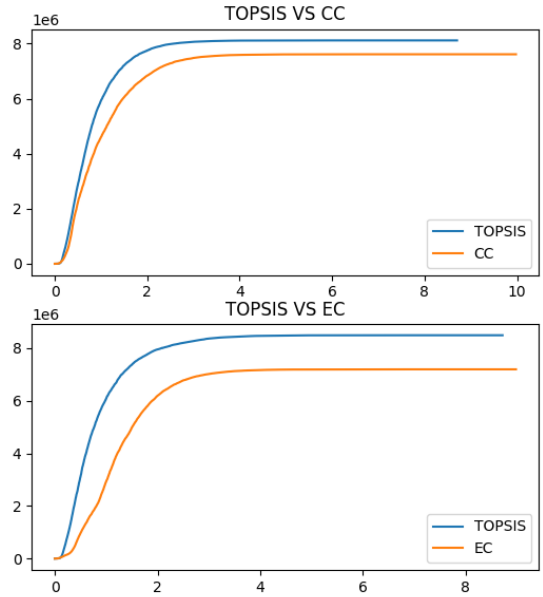
```
#sorted by C
list=[]
for i in range(0,K,1):
    list.append(df.iloc[i,0])
print(list)

#sorted by BC
df.sort_values(by=['BC'],ascending=False, inplace=True)
list_BC=[]
for i in range(0,K,1):
    list_BC.append(df.iloc[i,0])

#sorted by CC
df.sort_values(by=['CC'],ascending=False, inplace=True)
list_CC=[]
for i in range(0,K,1):
    list_CC.append(df.iloc[i,0])

#sorted by DC
df.sort_values(by=['DC'],ascending=False, inplace=True)
list_DC=[]
for i in range(0,K,1):
    list_DC.append(df.iloc[i,0])

#sorted by EC
df.sort_values(by=['EC'],ascending=False, inplace=True)
list_EC=[]
for i in range(0,K,1):
    list_EC.append(df.iloc[i,0])
```

Appliquant le modèle SIR et dessiner un graphe pour visualiser la résultat :

```
gamma = 1.
tau = 0.3
fig, axs = plt.subplots(2, 2)

#TOPSIS VS BC
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list)
axs[0,0].plot(t,np.cumsum(I),  label='TOPSIS')
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list_BC)
axs[0,0].plot(t,np.cumsum(I), label='BC')
axs[0,0].legend()
axs[0,0].set_title("TOPSIS VS BC")

#TOPSIS VS CC
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list)
axs[0,1].plot(t,np.cumsum(I),  label='TOPSIS')
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list_CC)
axs[0,1].plot(t,np.cumsum(I), label='CC')
axs[0,1].legend()
axs[0,1].set_title("TOPSIS VS CC")

#TOPSIS VS DC
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list)
axs[1,0].plot(t,np.cumsum(I),  label='TOPSIS')
t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list_DC)
axs[1,0].plot(t,np.cumsum(I), label='DC')
axs[1,0].legend()
axs[1,0].set_title("TOPSIS VS DC")

 #TOPSIS VS EC
 t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list)
 axs[1,1].plot(t,np.cumsum(I),  label='TOPSIS')
 t, S, I, R = EoN.fast_SIR(G, tau, gamma,tmax=10,initial_infecteds = list_EC)
 axs[1,1].plot(t,np.cumsum(I), label='EC')
 axs[1,1].legend()
 axs[1,1].set_title("TOPSIS VS EC")
 plt.show()

```

La visualisation du résultat :
TOPSIS VS BC , CC, DC, EC





CHAPITRE 2: K-Means

- Définition
- Les Etapes
- Application & Résultat
- Autre algorithme de détection des communautés

Ce chapitre présente l'algorithme K-means ainsi que la résultat obtenu en appliquant ce algorithme et une comparaison avec un autre algorithme de détection de communautés .

2. K-Means

1. Définition

K means clustering est un autre algorithme simplifié dans l'apprentissage automatique. Il est classé dans l'apprentissage non supervisé parce qu'ici nous ne connaissons pas déjà le résultat (aucune idée de quel cluster sera formé). Cet algorithme est utilisé pour *la quantification vectorielle* des données et a été tiré de la méthodologie de traitement du signal. Ici, les données sont divisées en plusieurs groupes, les points de données de chaque groupe ont des caractéristiques similaires. Ces clusters sont décidés en calculant la distance entre les points de données. Cette distance est une mesure de la relation entre de nombreux points de données non réclamés.

K-means ne doivent pas être confondues avec **l'algorithme** KNN car les deux utilisent la même technique de mesure de distance. Il existe une différence fondamentale entre les deux algorithmes d'apprentissage automatique populaires. K signifie travaille sur les données et les divise en divers clusters / groupes tandis que KNN travaille sur de nouveaux points de données et les place dans les groupes en calculant la méthode du plus proche voisin. Le point de données se déplace vers un cluster ayant un nombre maximal de voisins.

2. Les étapes :

K-means : étapes de l'algorithme de clustering :

1- Choisissez un nombre aléatoire de centroïdes dans les données. c'est-à-dire k=3.(k-means statique)

2- Choisissez le même nombre de points aléatoires sur la zone de travail 2D que les centroïdes.

3- Calculez la distance de chaque point de données par rapport aux centroïdes.

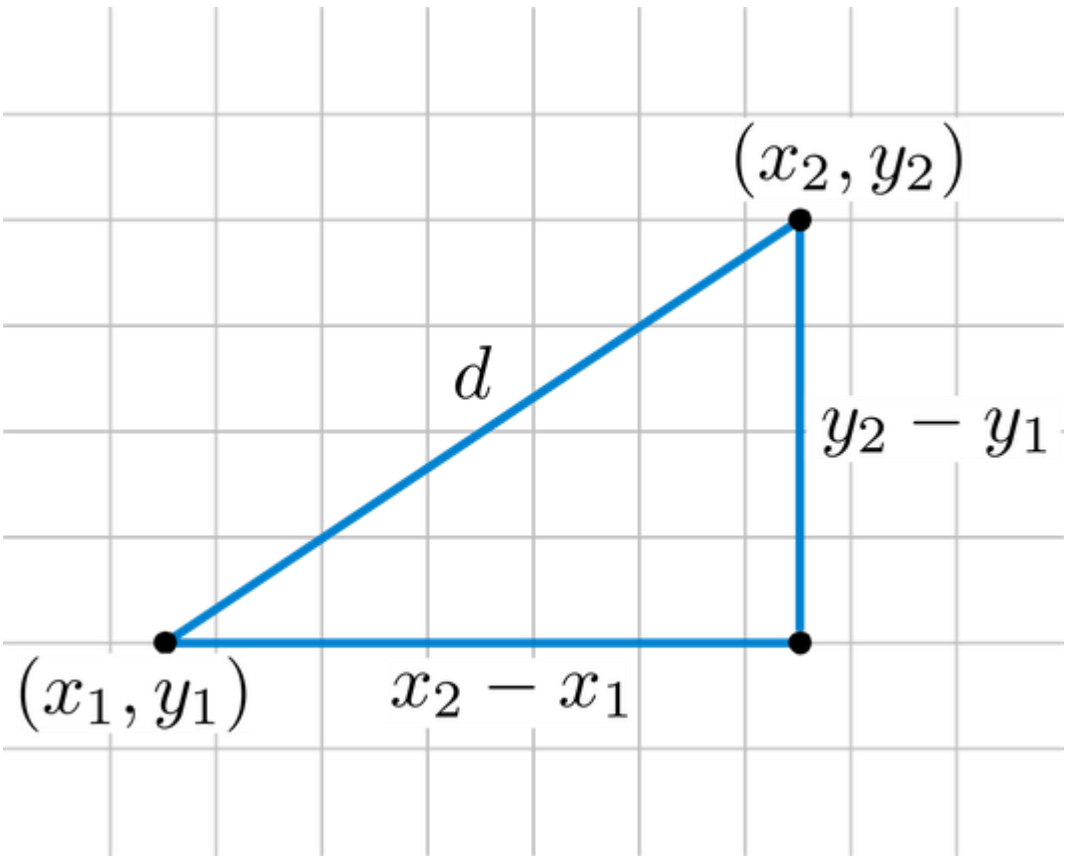
4- Allouez le point de données à un cluster où sa distance par rapport au centroïde est minimale.

5- Recalculez les nouveaux centroïdes.

6- Recalculez la distance entre chaque point de données et les nouveaux centroïdes.

7- Répétez les étapes à partir du point 3, jusqu'à ce qu'aucun point de données ne modifie son cluster.

K means divise les données en différents clusters et le nombre de clusters est égal à la valeur de k, c'est-à-dire que si k = 3 alors les données seront divisées en 3 clusters. chaque valeur de k est un centroïde autour duquel les points de données vont se rassembler.



Le calcul de la distance peut être effectué par l'une des quatre méthodes, à savoir Euclidean, Manhattan, Correlation et Eisen. Ici, nous utilisons la méthode euclidienne pour la mesure de distance, c'est-à-dire que la distance entre deux points (x1,y1) et (x2,y2) sera :

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Voici quelques avantages et inconvénients de l'utilisation de l'algorithme de clustering k means dans l'apprentissage automatique

Avantages:

1- Un algorithme relativement simple à appliquer

2- Flexible et fonctionne bien avec des données volumineuses

3- La convergence est garantie

Contre:

1- Nous devons définir manuellement le nombre de centroïdes

2- Pas à l'abri des valeurs aberrantes

3- Dépend des valeurs initiales du centroïde choisi

3. Application & Résultat :

Application :

```
#read data from facebook_combined.xlsx
df = pd.read_csv("TOPSIS.R2.csv")
print(df.tail())
G = nx.read_edgelist("facebook_combined.txt",create_using=nx.Graph(), nodetype = int)
list_of_nodes=np.array(list(G.nodes()))
df['nodes']=list_of_nodes
```

Preprocessing using min max scaler :

```
14  #Preprocessing using min max scaler
15  scaler = MinMaxScaler()
16  scaler.fit(df[['C']])
17  df['C'] = scaler.transform(df[['C']])
18
19  scaler.fit(df[['nodes']])
20  df['nodes'] = scaler.transform(df[['nodes']])
21
22
```

Visualiser nos données :

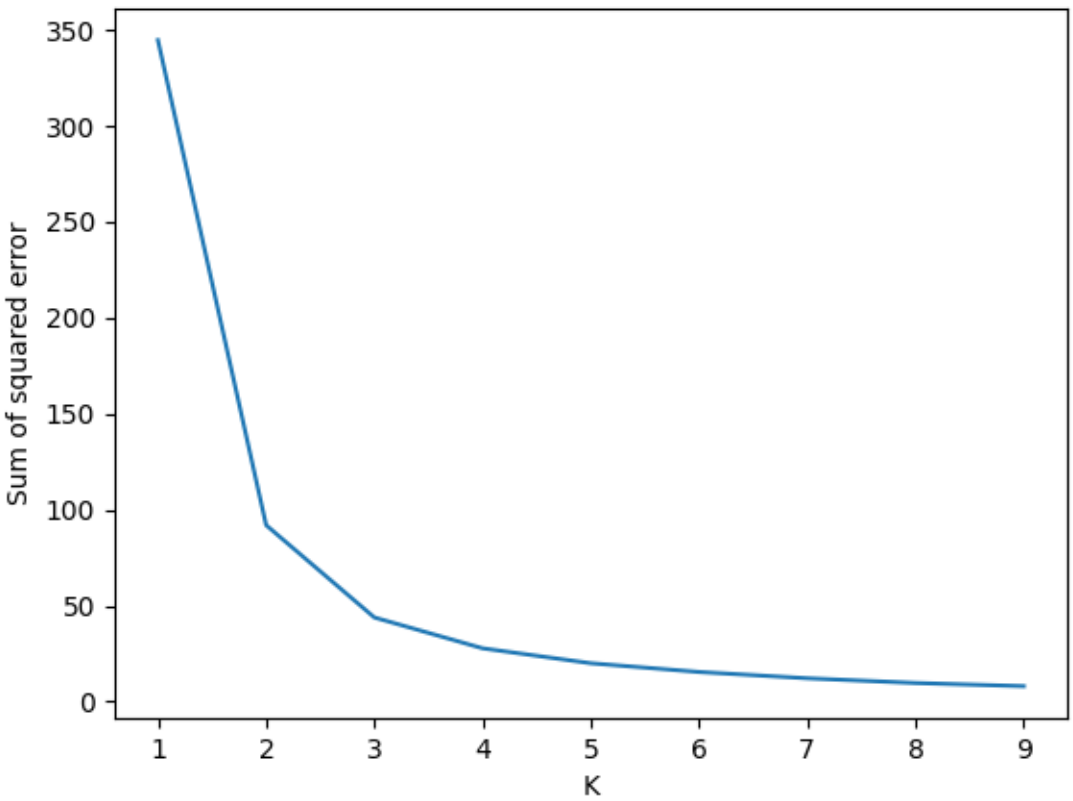
```
23  #draw a scatter for visualize our data
24  plt.scatter(df.nodes,df.C)
25  plt.xlabel('nodes')
26  plt.ylabel('C')
27  plt.show()
```

Nous pouvons voir qu'en choisissant manuellement le nombre de centroïdes, nos amas ne sont pas très séparés. Un cluster dispose d'un ensemble similaire d'informations et notre objectif est de le rendre aussi unique que possible. Cela aide à extraire plus d'informations de notre ensemble de données donné. Ainsi, nous pouvons tracer une courbe de coude qui peut clairement représenter un compromis entre le nombre de centroïdes et le gain d'information.

```
30  #what the perfect number of scatter for our data
31  sse = []
32  k_rng = range(1,10)
33  for k in k_rng:
34      km = KMeans(n_clusters=k)
35      km.fit(df[['nodes','C']])
36      sse.append(km.inertia_)
37  plt.xlabel('K')
38  plt.ylabel('Sum of squared error')
39  plt.plot(k_rng,sse)
40  plt.show()
41
```

Vous pouvez voir qu'il y a **K-means++** comme méthode que les k-means conventionnels. La première méthode surmonte l'inconvénient d'une mauvaise sélection des centroïdes, ce qui se produit généralement en raison de la sélection manuelle. Parfois, les centroïdes choisis sont trop éloignés des points pour ne pas avoir de points de données dans leur cluster.

Le graphique de sortie peut nous aider à déterminer le nombre de centroïdes à choisir pour un meilleur regroupement.



Maintenant appliquant l'algorithme K-means dynamique selon le nombre de clusters proposés par la variable visualizer.elbow_value comme vous voyez dans le code ci-dessous:

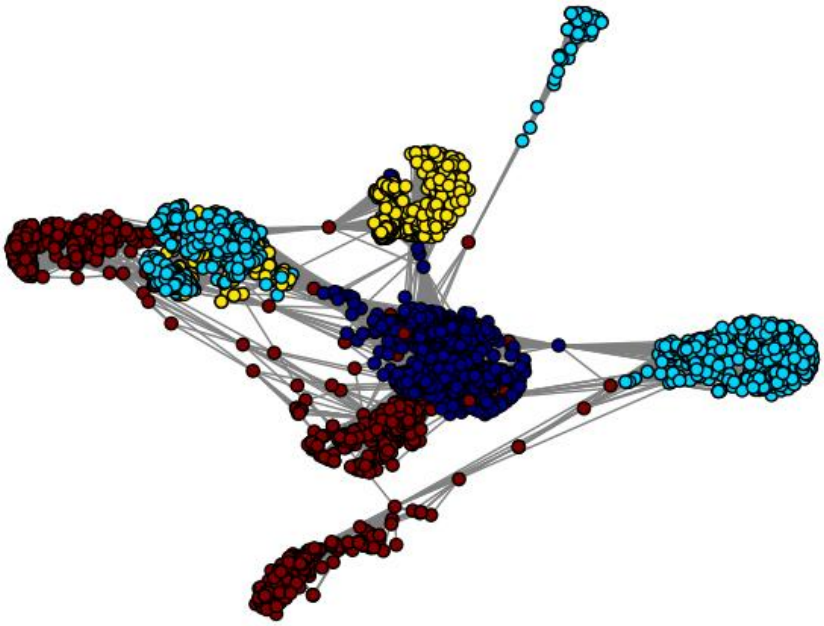
```
#prepare dataset
data=df[['nodes','C']]
# using Kmeans
model = KMeans()
# know number of clusters
visualizer = KElbowVisualizer(model, k=(1,12)).fit(data)
print(visualizer.elbow_value_)
visualizer.show()


#clustering our data and print the clusters id
km = KMeans(n_clusters=visualizer.elbow_value_)
y_predicted = km.fit_predict(df[['nodes','C']])
print(y_predicted)

#add a columns for clusters id and print te centroid coordinates
df['cluster']=y_predicted
print(df.tail())
print(km.cluster_centers_)
```

Visualiser nos résultats des différents clusters détectés :

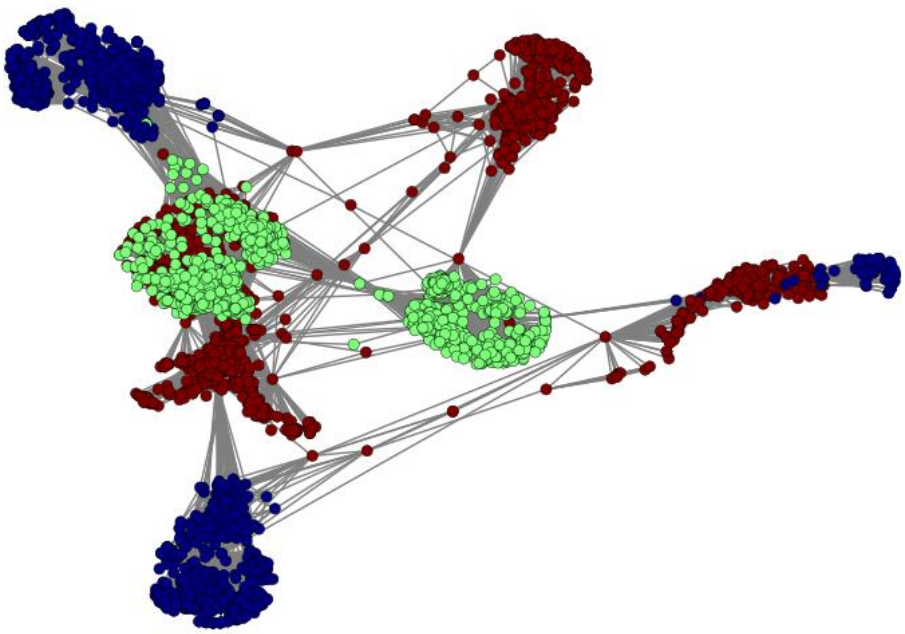
```
#draw a scatter plot who represent the different clusters with different colors and the centroid of each cl
for i in range(visualizer.elbow_value_):
    plt.scatter(df[df.cluster==i].nodes,df[df.cluster==i].C,label = i)
plt.scatter(km.cluster_centers_[:,0],km.cluster_centers_[:,1],color='purple',marker='*',label='centroid')
plt.xlabel('nodes')
plt.ylabel('C')
plt.legend()
plt.show()


options = {
        'cmap'       : plt.get_cmap('jet'),
        'node_color' : y_predicted,
        'node_size'  : 35,
        'edge_color' : 'tab:grey',
        'with_labels': False
    }
plt.figure()
pos = nx.spring_layout(G)
nx.draw(G,**options,pos=pos,edgecolors='black')
plt.show()
```

Résultat :

K-means statique (4 clusters) :



K-means dynamique (3 clusters) :



3. Autre algorithme de détection de communautés :

Pourquoi la détection de communautés ?

Lors de l'analyse de différents réseaux, il peut être important de découvrir des communautés en leur sein. Les techniques de détection de communautés sont utiles aux algorithmes de médias sociaux pour découvrir des personnes ayant des intérêts communs et les garder étroitement connectées.
La détection de communautés peut être utilisée dans l'apprentissage automatique pour détecter les groupes ayant des propriétés similaires et extraire des groupes pour diverses raisons. Par exemple, cette technique peut être utilisée pour découvrir des groupes manipulateurs dans un réseau social ou un marché boursier.

Détection de communautés vs. clustering

On peut dire que la détection de communautés est similaire au clustering. Le clustering est une technique d'apprentissage automatique dans laquelle des points de données similaires sont regroupés dans le même cluster en fonction de leurs attributs. Même si le clustering peut être appliqué aux réseaux, il s'agit d'un domaine plus large de l'apprentissage automatique non supervisé qui traite de plusieurs types d'attributs. En revanche, la détection des communautés est spécialement conçue pour l'analyse des réseaux, qui dépend d'un seul type d'attribut appelé "arêtes".
En outre, les algorithmes de regroupement ont tendance à séparer les nœuds périphériques uniques des communautés auxquelles ils devraient appartenir.
Cependant, les techniques de clustering et de détection de communauté peuvent être appliquées à de nombreux problèmes d'analyse de réseau et peuvent présenter des avantages et des inconvénients différents selon le domaine.

Techniques de détection des communautés :

Les méthodes de détection des communautés peuvent être classées en deux catégories : les méthodes agglomératives et les méthodes de division.
Dans les méthodes agglomératives, les arêtes sont ajoutées une par une à un graphe qui ne contient que des nœuds. Les arêtes sont ajoutées de l'arête la plus forte à l'arête la plus faible.
Les méthodes de division sont à l'opposé des méthodes agglomératives. Dans ce cas, les arêtes sont retirées une par une d'un graphe complet.
Il peut y avoir un nombre quelconque de communautés dans un réseau donné et elles peuvent être de tailles différentes. Ces caractéristiques rendent la procédure de détection des communautés très difficile.

Cependant, de nombreuses techniques différentes ont été proposées dans le domaine de la détection des communautés.
Deux algorithmes populaires de détection de communautés sont expliqués ci-dessous.

1. Détection de communauté de Louvain :

L'algorithme de détection de communauté de Louvain a été proposé à l'origine en 2008 comme une méthode rapide de dépliage de communauté pour les grands réseaux. Cette approche est basée sur la modularité, qui tente de maximiser la différence entre le nombre réel d'arêtes dans une communauté et le nombre attendu d'arêtes dans la communauté. Cependant, l'optimisation de la modularité dans un réseau est une tâche difficile, d'où la nécessité d'utiliser des heuristiques. L'algorithme de Louvain est divisé en deux phases qui se répètent de manière itérative .

- Déplacement local des nœuds
- Agrégation du réseau

L'algorithme commence avec un réseau pondéré de N nœuds.

Dans la première phase, l'algorithme attribue une communauté différente à chaque nœud du réseau. Ensuite, pour chaque nœud, il considère les voisins et évalue le gain de modularité en retirant le nœud particulier de la communauté actuelle et en le plaçant dans la communauté du voisin. Le nœud sera placé dans la communauté du voisin si le gain est positif et maximisé. Le nœud restera dans la même communauté s'il n'y a pas de gain positif. Ce processus est appliqué de manière répétée et pour tous les nœuds jusqu'à ce qu'il n'y ait plus d'amélioration. La première phase de l'algorithme de Louvain s'arrête lorsqu'un maximum local de modularité est obtenu.

Dans la deuxième phase, l'algorithme construit un nouveau réseau en considérant les communautés trouvées dans la première phase comme des nœuds. Une fois la deuxième phase terminée, l'algorithme réapplique la première phase au réseau résultant.

Ces étapes sont répétées jusqu'à ce qu'il n'y ait plus de changement dans le réseau et que la modularité maximale soit obtenue.
L'algorithme de détection de communautés de Louvain découvre des communautés de communautés au cours du processus. Il est très populaire en raison de la facilité de mise en œuvre et de la rapidité de l'algorithme.
Cependant, une limitation majeure de l'algorithme est l'utilisation du stockage du réseau dans la mémoire principale.
L'utilisation de l'algorithme de détection des communautés de Louvain à l'aide de la bibliothèque python **networkx.algorithms** est présentée ci-dessous.

```
import community as community_louvain
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import networkx as nx
# load  graph
G= nx.read_edgelist("facebook_combined.txt",create_using=nx.Graph(),
                    nodetype = int)
#first compute the best partition
partition = community_louvain.best_partition(G)
# draw the graph
pos = nx.spring_layout(G)
# color the nodes according to their partition
cmap = cm.get_cmap('viridis', max(partition.values()) + 1)
nx.draw_networkx_nodes(G, pos, partition.keys(), node_size=35,
                       cmap=cmap, node_color=list(partition.values()))
nx.draw_networkx_edges(G, pos, alpha=0.5)
plt.rcParams.update({'figure.figsize':(15,8)})
plt.show()
```

2. Greedy modularity communities :

Greedy modularity communities fonction de détection des communautés basées sur la modularité.
La modularité est une mesure de la qualité d'un partitionnement des nœuds d'un graphe, ou réseau, en communautés (ou classes). Elle est principalement utilisée en analyse des réseaux sociaux. Elle a été introduite par M. E. J. Newman en 2004.
C'est aussi une fonction d'optimisation pour certaines tâches de détection de communautés dans les graphes. Le principe est qu'un bon partitionnement d'un graphe implique un nombre d'arêtes intra-communautaires important et un nombre d'arêtes intra-communautaires faible.
La modularité est décrite comme la proportion des arêtes incidentes sur une classe donnée moins la valeur qu'aurait été cette même proportion si les arêtes étaient disposées au hasard entre les nœuds du graphe. La modularité prend ses valeurs entre -1/2 et 1 inclus.
On peut considérer qu'un graphe a une structure de communautés significative quand une partition obtient un score de modularité supérieur à 0,3.

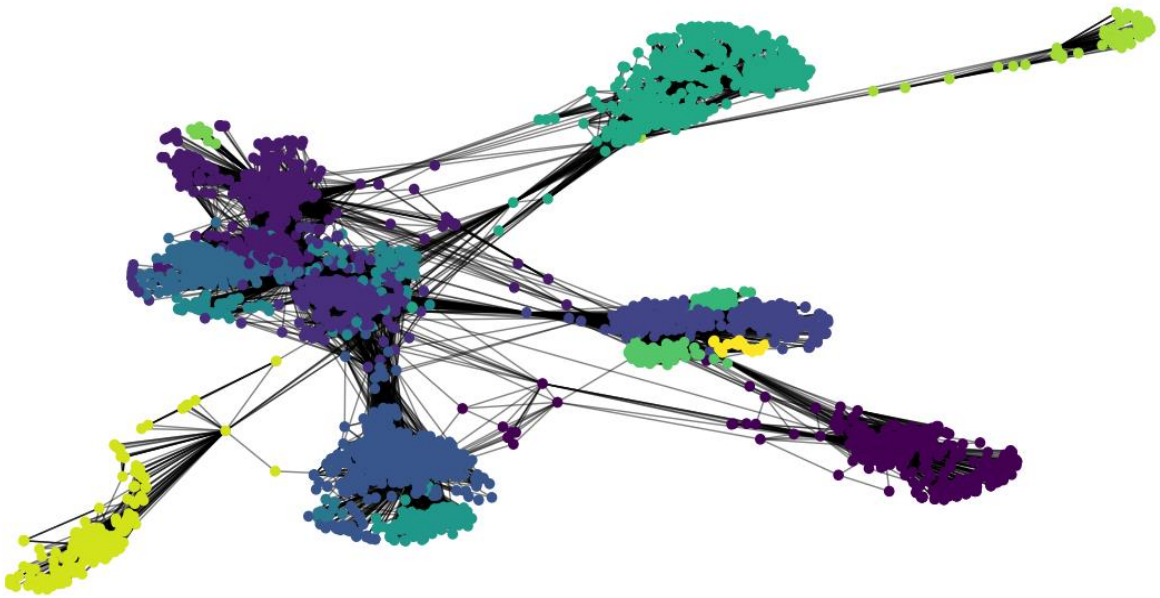
```
1   from networkx.algorithms import community
2   import networkx as nx
3   import matplotlib.pyplot as plt
4
5   G = nx.read_edgelist("facebook_combined.txt",create_using=nx.Graph(),
6                        nodetype = int)
7   partition = community.greedy_modularity_communities(G)
8
9   #Nombre de partitions trouvées
10  print(len(partition))
11
12  #Affichage de la composition des communautés
13  for i in range(len(partition)):
14          community = list(partition[i])
15          print("Communauté ",str(i+1))
16          print([ i for i in community ])
17
```

Visualiser nos résultats des différents communautés détectés :

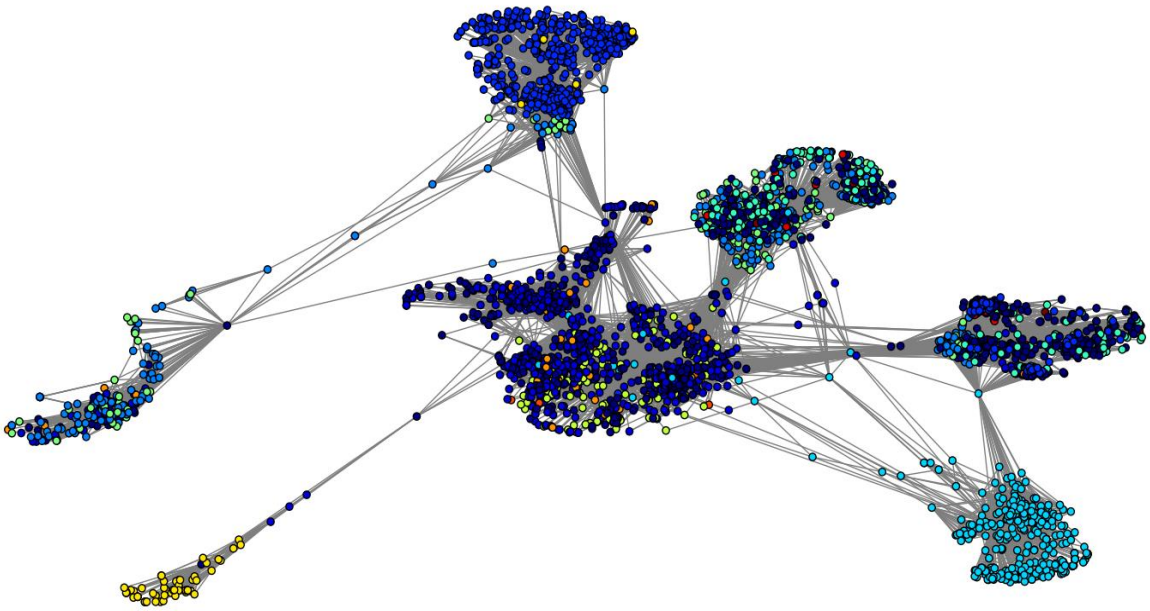
```
#Visualisation par contruction d'un vecteur de couleurs :
couleurs_num = [0] * G.number_of_nodes()
for i in range(len(partition)):
    for j in partition[i]:
        couleurs_num[j] = i

options = {
        'cmap'       : plt.get_cmap('jet'),
        'node_color' : couleurs_num,
        'node_size'  : 35,
        'edge_color' : 'tab:grey',
        'with_labels': False
    }
plt.figure()
pos = nx.spring_layout(G)
nx.draw(G,**options,pos=pos,edgecolors='black')
plt.show()
```

Résultat :

Détection de communauté de Louvain



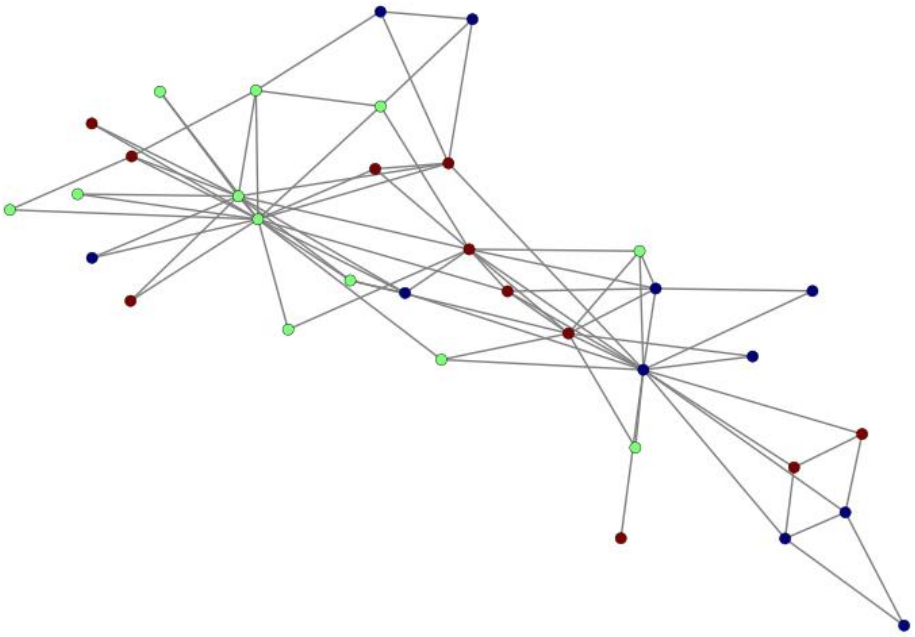
Greedy modularity communities



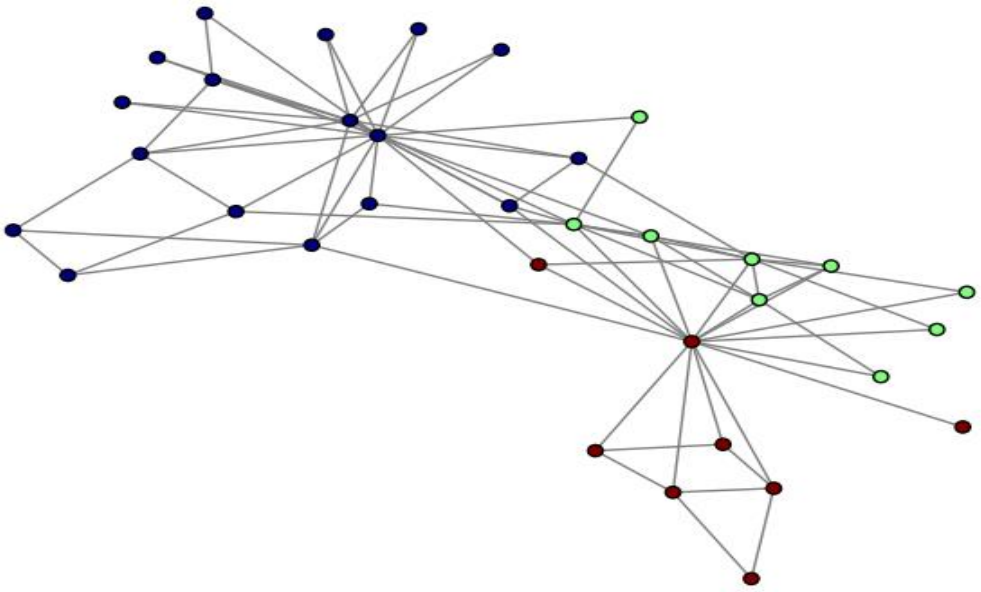
ZACHARY DATASET :

Ici c'est une autre application sur Zachary dataset :

K-Means :



Greedy modularity communities :



CHAPITRE 3: Conception et implémentation

❑ Ressources utilisée:

✓ Les ressources physiques exploitées
✓ Les ressources logicielles

nous présentons dans ce chapitre les outils utilisés pour le développement de notre système. Nous validons notre travail par la description de différentes expérimentations réalisées.

Ressources utilisées

Les ressources physiques exploitées :

- Processeur Intel(R) Core (TM) i5-5200U CPU @ 2,20 GHz 2,19 GHz
- Mémoire vive d'une capacité de 8,00 Go.

Et comme ressources logicielles, nous avons utilisé :

- ✓ Système d'exploitation : Windows10 pro.
- ✓ Langage de programmation : Python.
- ✓ L'EDI : Microsoft Visual Studio Code.
- ✓ Les bibliothèques : Pandas , NetworkX , matplotlib , EoN, Numpy , sklearn, yellowbrick

Pourquoi Visual Studio Code

Visual Studio Code est un éditeur de code extensible développé par Microsoft pour Windows, Linux et macOS.
Le code source de Visual Studio Code provient du projet logiciel libre et open source VSCode de Microsoft publié sous la licence MIT permissive, mais les binaires compilés constituent un freeware, c'est-à-dire un logiciel gratuit pour toute utilisation mais privateur.

Pourquoi NetworkX

NetworkX est un package Python pour la création, la manipulation et l'étude de la structure, de la dynamique et des fonctions des réseaux complexes.

EoN (Epidemics on Networks) est un paquetage Python pour la simulation d'épidémies sur des réseaux et la résolution de modèles ODE de propagation de maladies.

Pourquoi sklearn

Les données non étiquetées peuvent être regroupées à l'aide du module **sklearn.cluster**.
Il existe deux variantes de chaque algorithme de clustering :
une classe qui implémente la méthode fit pour apprendre les clusters sur les données d'apprentissage ; et une fonction qui, compte tenu des données d'apprentissage, renvoie un tableau d'étiquettes entières correspondant aux différents clusters. Pour cette classe, les étiquettes sur les données d'apprentissage se trouvent dans labels_ attribute.

Pourquoi matplotlib

Matplotlib : Visualisation avec Python
Matplotlib est une bibliothèque complète pour créer des visualisations statiques, animées et interactives en Python.
Matplotlib rend les choses faciles faciles et les choses difficiles possibles.

Pourquoi Pandas

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données.
Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.