كلية العلوم
السملالية - مراكش
FACULTÉ DES SCIENCES
SEMLALIA - MARRAKECH
MARRAKECH
جامعة القاضي عياض
UNIVERSITÉ CADI AYYAD

PROJET PYTHON

IDENTIFICATION DES NŒUDS LES PLUS INFLUENTS EN UTILISANT LA MÉTHODE TOPSIS

Réalisé par :
Maryame Laouina
Abderrahman Salhi

Supervisé par :
Pr.Issam Qaffou

Table des matières

Table des figures

Résumé :

L'identification des nœuds influents est parmi les questions importantes, des applications telles que l'accélération de la propagation des faits et le contrôle des rumeurs et des maladies, des nombreuses méthodes ont été proposées pour identifier les nœuds influents du réseau complexe, la mesure de la centralité des nœuds aux processus basés sur la propagation.

Dans ce projet, une nouvelle méthode est développée pour évaluer l'importance des nœuds dans les réseaux complexes Basé sur la technique de performance de la similarité à l'ordre idéal (TOPSIS) approche a été proposée. TOPSIS est d'abord appliqué pour identifier les nœuds influents du réseau complexe. Ce réseau à thème ouvert. Différentes façons de regarder plusieurs centralités différentes Mesurez plusieurs attributs de réseaux complexes dans les applications TOPSIS.

TOPSIS est utilisé pour agréger plusieurs attributs afin d'obtenir leurs scores, importance du nœud de chaque nœud. Cependant, vous n'êtes pas limité à une seule mesure de centralité. Considérez différentes mesures de centralité, puisque chaque mesure de centralité à sa propre mesure de centralité Inconvénients et limites. Utilisez ensuite le modèle sensible (SI) pour évaluer les performances.

Enfin, nous proposons d'utiliser l'algorithme K-Means pour trouver des clusters.

L'accent est mis sur les tops K influents détectés par la méthode TOPSIS.

Introduction :

La détection de nœuds influents dans les réseaux sociaux a gagné beaucoup d'attention dans la communauté de recherche. L'importance des nœuds est une mesure de base pour caractériser la structure et la dynamique de complexes réseaux. Par conséquent, l'identification des nœuds influents a été une question ouverte et une tâche de recherche critique dans les complexes réseaux. Diverses mesures de centralité ont été proposées au fil des ans pour classer les nœuds d'un graphique en fonction de leur importance topologique.

Dans les réseaux complexes, l'identification des nœuds influents est la partie la plus importante de la fiabilité analyse, qui a été un enjeu clé dans l'analyse de l'organisation structurelle d'un réseau.

Dans ce rapport, une nouvelle méthode d'évaluation de l'importance des nœuds dans les réseaux complexes basée sur technique pour la performance des commandes par similitude avec la solution idéale (TOPSIS) approche est proposée. TOPSIS en tant que technique de prise de décision à attributs multiples (MADM) est depuis lors une branche importante de la prise de décision. En outre, TOPSIS est d'abord appliqué pour identifier nœuds influents dans un réseau complexe. Dans différents types de réseaux en que l'information va par différents moyens, nous considérons plusieurs différentes mesures de centralité comme le multi-attribut de réseau complexe dans l'application TOPSIS.

TOPSIS est utilisé pour agréger le multi-attribut pour obtenir l'évaluation de l'importance de nœud de chaque nœud. Il ne se limite pas à une seule mesure de centralité, mais considère différentes mesures de centralité, parce que chaque mesure de centralité a son propre désavantage et limitation.

1-Des notions :

1-1 Les mesures de centralité :

De nombreuses mesures de centralité ont été proposées pour classer les nœuds dans les réseaux. Comme degré de centralité, à savoir, un nœud avec un plus grand degré est susceptible d'avoir une influence plus élevée (p. ex., en tant que nœud initialement infecté, il devrait se propager plus rapidement et plus largement) qu'un nœud avec un degré plus petit. Cependant, dans certains cas, cette méthode ne parvient pas à identifier nœuds influents car il considère seulement des informations très limitées. Par exemple, comme le montre Fig. 1, bien que le nœud 1 ait le degré le plus élevé parmi les 23 nœuds, la maladie, si son origine au nœud 1, ne peut pas se propager le plus rapide ou le plus large depuis tous les voisins de nœud 1 ont un degré très faible. En revanche, le nœud 23 peut être d'influence plus élevée bien qu'il a un degré inférieur par rapport au nœud 1. Un autre groupe de méthodes considérant l'information globale donne de meilleurs résultats de classement, tels que la centralité entre les proximités centralité, deux mesures de classement géodésiques bien en vue

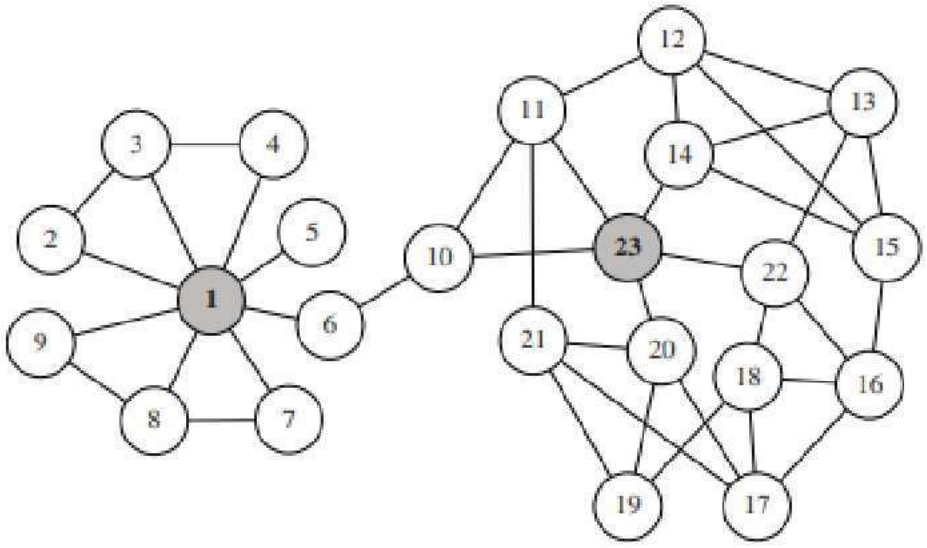

Figure 1 : Exemple de réseau composé de 23 nœuds et 40 arêtes

1-2 Betweenness Centrality:

La centralité intermédiaire est une mesure de la centralité dans un graphe basé sur les chemins les plus courts . Pour chaque paire de sommets dans un graphe connexe, il existe au moins un chemin le plus court entre les sommets tel 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) est minimisé. La centralité d'intermédiarité pour chaque sommet est le nombre de ces chemins les plus courts qui passent par le sommet.

La centralité de l'intermédiarité a été conçue comme une mesure générale de la centralité : elle s'applique à un large éventail de problèmes de la théorie des réseaux, y compris les problèmes liés aux réseaux sociaux, à la biologie, au transport et à la coopération scientifique. Bien que les auteurs précédents aient intuitivement décrit la centralité comme étant basée sur l'intermédiarité, a donné la première définition formelle de la centralité de l'intermédiarité.

La centralité intermédiaire 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 centralité d'intermédiarité plus élevée aurait plus de contrôle sur le réseau, car plus d'informations passeront par ce nœud.

La centralité intermédiaire d'un nœud est donnée par l'expression :

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}},$$

où σ_{st} est le nombre total de chemins les plus courts depuis le nœud s au nœud t et $\sigma_{st}(v)$ est le nombre de ces chemins qui passent par v (pas où v est un point final).

1-3 Closeness Centrality :

Dans un graphe connexe , **la centralité de proximité** d'un nœud est une mesure de la centralité dans un réseau , calculée comme l'inverse de la somme de la longueur des chemins les plus courts entre le nœud et tous les autres nœuds du graphe. Ainsi, plus un nœud est central, plus il est *proche* de tous les autres nœuds.

$$C_C(v) = \frac{1}{\sum_{t \in V \setminus v} d_G(v, t)},$$

Où dG (v, t) est la distance géodésique entre v et t. La proximité peut être considérée comme un mesure de combien de temps il diffusera des informations d'un nœud donné à d'autres joignables nœuds du réseau

1-4 Degree Centrality:

Le terme de *degré de centralité* est historiquement le premier et conceptuellement le plus simple. Il est défini comme le nombre de liens incidents à un nœud (c'est-à-dire le nombre de voisins que possède un nœud). Le degré peut être interprété en termes de l'aptitude d'un nœud de capter tout ce qui passe à proximité sur le réseau (comme un virus, ou une information). Dans le cas d'un réseau orienté (où les liens ont une direction), on définit habituellement deux mesures

distinctes de la centralité de degré, à savoir le *degré entrant* et le *degré sortant*. En conséquence, degré entrant compte le nombre de liens dirigés vers le nœud et degré sortant est le nombre de liens qui dirigent le nœud vers d'autres. Lorsque des liens sont associés à certains aspects positifs comme l'amitié ou la collaboration, le degré entrant est souvent interprété comme une forme de popularité, et le degré sortant comme mesure du caractère grégaire. La centralité de degré entrant est aussi appelée *centralité de prestige*.

1-5 Eigenvector Centrality:

La centralité propre est une mesure de l'influence d'un nœud dans un réseau. Elle attribue des scores relatifs à tous les nœuds du réseau en fonction du concept selon lequel les nœuds de notation contribuent d'avantage au score du nœud en question qu'à des connexions égales aux nœuds à faible notation.

Qu'A soit une matrice de similarité n*n. La centralité du vecteur propre xi du nœud i est définie comme l'ième entrée dans le secteur propre normalisé appartenant À la plus grande valeur propre d'A. λ est La plus grande valeur propre de A et n est le nombre de sommets.

$$Ax = \lambda x, \qquad x_i = \mu \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, n$$

Avec μ facteur de proportionnalité $\mu = 1/\lambda$ de sorte que xi est proportionnel à la somme des scores de similarité de tous les nœuds qui lui sont connectés

2. Introduction à topsis et ses étapes:

Le principe de la méthode TOPSIS qui semble être simple une fois comprise, peut être résumé dans le schéma suivant :

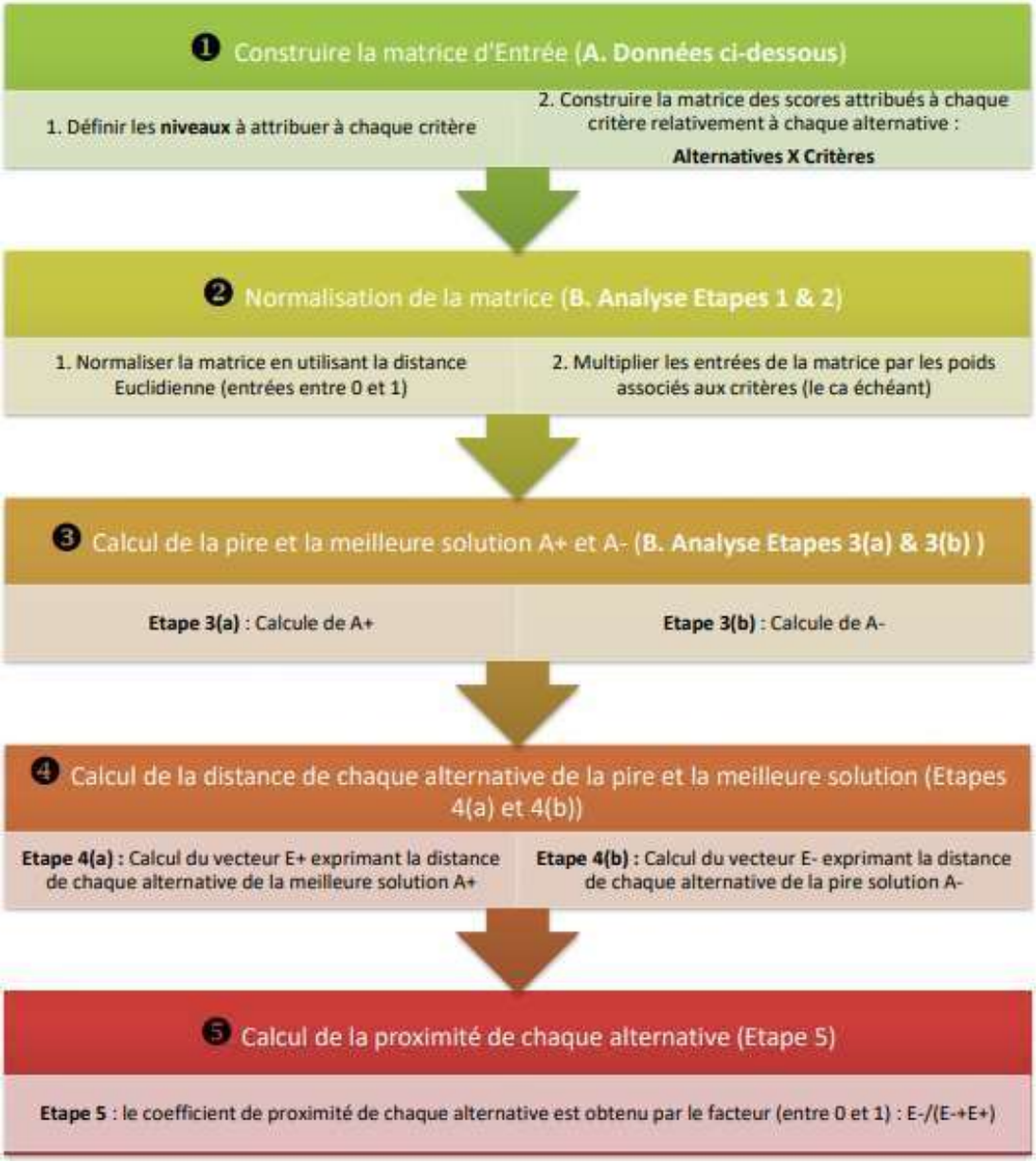

Figure 2 : Le principe de la méthode topsis

En se ramenant à 2 dimensions pour simplifier (au lieu de la dimension # des alternatives X # des critères), on peut grossir le principe précédent comme suit :

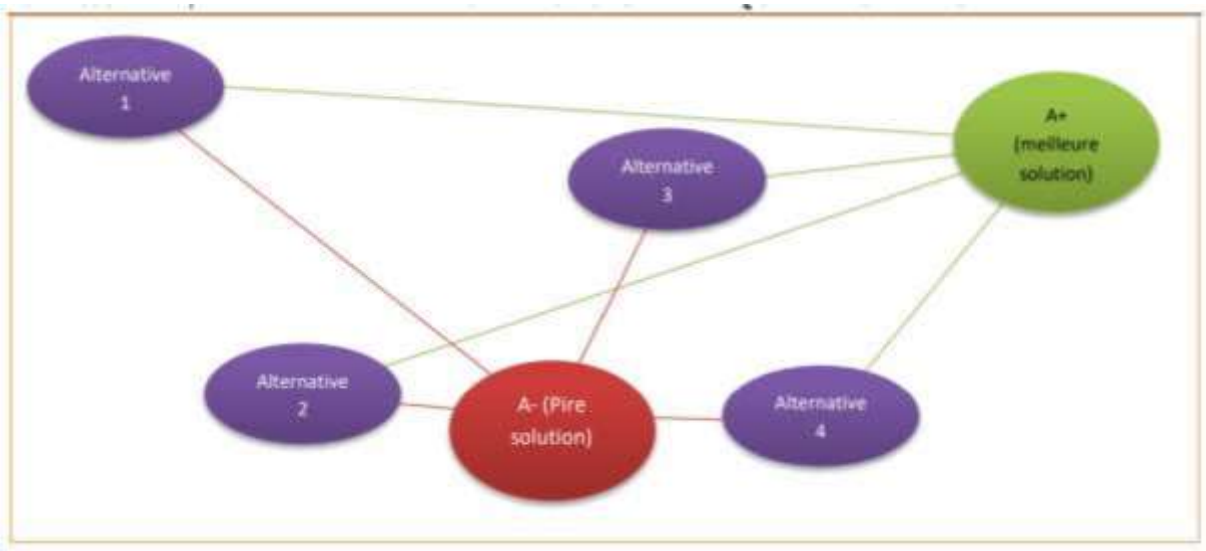

Figure 3 la méthode expliquée pas à pas

La meilleure alternative est celle qui est la plus lointaine de la solution A- (dite pire solution ou solution idéale négative) et la plus proche de la solution A+ (dite meilleure solution ou solution idéale positive). Je vous laisse le soin de trouver géométriquement à partir du schéma d'illustration ci-dessous, la meilleure alternative. Les solutions (meilleure et pire) ne sont pas des alternatives, mais uniquement deux repères qui nous permettent de déterminer la meilleure alternative.

2-1 Introduction topsis :

TOPSIS (Technique pour la préférence d'ordre par similarité à la solution idéale) méthode est appelée la solution idéale. C'est une méthode efficace de prise de décision à attributs multiples. Cette méthode est de construire les solutions idéales et négatives aux problèmes de Multiples attributs et utilise les deux repères d'être proche des solutions idéales et être loin des solutions négatives comme critères d'évaluation de la faisabilité projets. "Solution idéale" et " solution négative" sont les deux concepts de base de la méthode TOPSIS. La solution dite idéale (notée x+) est la solution optimal, toute sa valeur attributaire atteint la meilleure valeur de chaque projet alternatif, mais la solution négative (noté comme x-) est la pire solution (projet) dans l'hypothèse. La règle de classement des projets est de comparer chaque projet alternatif avec x+ et x-. Si l'un des projets sont proches de x+ et loin de x- en même temps, alors le projet est le meilleur projet des autres projets. Il devrait tenir compte de l'évaluation de la capacité d'attaque et de défense des joueurs étrangers, ce qui comprend le pourcentage de lancer tir à deux points,

tir à trois points et lancer franc, aide, rebond, faute, et ainsi de suite. Chaque index est un attribut. Donc, c'est une question typique de prise de décision de multiples attributs pour trier joueurs étrangers. Et puis la méthode TOPSIS peut être appliquée pour trier les joueurs étrangers.

2-2 Les étapes de base de la méthode TOPSIS :

Première étape : élaborer une matrice de décision normalisée A. Pour les questions d'évaluation avec n unités d'évaluation et m indices d'évaluation.

La matrice de décision A est :

$$A = \begin{array}{c} \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & \cdots & f_m \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \end{array}$$

Dans la formule, aij=fj (xi), qui indique le jème indice d'évaluation de la ième unité d'évaluation i=1,..., n; j=1,2,..., m.

Normalisez la matrice A en tant que matrice R :

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ r_{21} & r_{22} & \cdots & r_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ r_{n1} & r_{n2} & \cdots & r_{nm} \end{bmatrix}$$

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

Deuxième étape : construire la matrice de décision pondérée et standardisée V, vecteur de pondération W= (w1, w2,..., wn).

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ v_{n1} & v_{n2} & \cdots & v_{nm} \end{bmatrix}$$

$$v_{ij}=w_j.r_{ij} \ ,i=1,2,\ldots,n, j=1,2,\ldots,m.$$

Troisième étape : déterminer la solution idéale x+ et moins la solution idéale x- :

$$x^+ = \{\max v_{ij} \ j\epsilon J), (\min v_{ij} \mid j\epsilon J') \mid i=1, 2,\ldots,n\}=\{x_1^+,x_2^+,\ldots,x_m^+\}$$

$$x^- = \{\min v_{ij} \ j\epsilon J), (\max v_{ij} \mid j\epsilon J') \mid i=1, 2,\ldots,n\}=\{x_1^-,x_2^-,\ldots,x_m^-\}$$

Quatrième étape : calculer la distance, la distance de chaque projet à la solution idéale x+ est

$$S_i^+ = \sqrt{\sum_{j=1}^{m} \left(v_{ij} - x_j^+\right)^2}$$

La distance de chaque projet à la solution idéale négative x- est :

$$S_i^- = \sqrt{\sum_{j=1}^{m} (v_{ij} - x_j^-)^2}$$

Cinquième étape : calculer l'indice de proximité relative de chaque projet à la solution idéale :

$$c_i = \frac{S_i^-}{S_j^+ + S_i^-}$$

Évidemment, $0 \leq ci \leq 1$.

3-La méthode wtopsis :

- La sélection d'une méthode appropriée de prise de décision par attributs multiples (MADM) pour un problème MADM donné est toujours une tâche difficile. La nécessité d'une comparaison comparative des méthodes lors de la sélection a été soulignée dans des études. Dans le domaine MADM, la technique de préférence d'ordre par similarité à la solution idéale (TOPSIS est la méthode MADM très appréciée, appliquée et adoptée en raison de sa simplicité et du concept sous-jacent selon lequel la meilleure solution est celle qui se rapproche le plus de la solution idéale positive et la plus éloignée de la solution idéale négative. Une variante de TOPSIS nommée WTOPSIS a été développée avec le processus d'élicitation du poids objectif basé sur l'entropie avec l'argument que le poids subjectif n'est pas toujours réalisable et que le poids de l'attribut est appliqué différemment du WTOPSIS lors de la résolution des problèmes MADM. Le TOPSIS et le WTOPSIS ont été appliqués pour la résolution de problèmes MADM par divers chercheurs et praticiens.
- Le WTOPSIS a été utilisé pour des études comparatives dans les estimations de poids d'attribut et le développement d'indices composites objectifs . L'étude est également appliquée dans la gestion des ressources, la sélection de logiciels , l'évaluation environnementale, l'évaluation de la durabilité, la sélection des matériaux, la sélection des machines, l'évaluation de la technologie, et développement d'autres méthodes.

- TOPSIS est l'une des méthodes fondamentales dans le domaine MADM et a été extrêmement populaire dans les applications et comme base pour le développement de nombreuses méthodes. Le TOPSIS modifié qui est également basé sur la méthode TOPSIS, a gagné en popularité en raison de sa nouvelle utilisation du processus objectif d'élicitation du poids basé sur celui de Shannon. théorie de l'entropie. Le TOPSIS et le TOPSIS modifié nécessitent des valeurs de pondération d'attribut discrètes établies avant que les calculs ne soient effectués. Dans TOPSIS, le poids reflète la préférence relative des attributs du décideur. D'autre part, dans TOPSIS modifié, les poids des attributs sont obtenus en fonction de leur signification basée sur l'entropie.

3-1 Les étapes de Wtopsis

- Etape 1 :

Considérons une matrice, la première n-1 les colonnes représentent la valeur des différentes mesures de centralité. La dernière colonne représente les résultats du modèle SIR ():

$$D\left(x_{mn}\right)=\begin{pmatrix} DC_1 & BC_1 & \cdots & F_1\left(t\right) \\ DC_2 & BC_2 & & F_2\left(t\right) \\ \vdots & \vdots & & \vdots \\ DC_n & BC_n & \cdots & F_n\left(t\right) \end{pmatrix}_{m\times n}.$$

- Etape 2 :

Construire une sous- matrice M= (ymk) de D=(Xmn) (k=n−1), La sous-matrice M est composé du premier n−1 colonnes dans la matrice D:

$$M\left(y_{mk}\right)=\begin{pmatrix} DC_1 & BC_1 & \cdots & CC_1 \\ DC_2 & BC_2 & & CC_2 \\ \vdots & \vdots & & \vdots \\ DC_m & BC_m & \cdots & CC_m \end{pmatrix}_{m\times k}.$$

- Etape 3 :

Normaliser la matrice D=(Xmn):

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

- Etape 4 :

Faites correspondre l'attribut à F(t) comme suit:

$$v_{ij} = \frac{1}{|r_{ij} - r_{in}|}, \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n$$

Où rin représente F(t).

- Etape 5 :

Calculez le poids de chaque attribut :

$$e_j = \sum_{i=1}^{m} v_{ij}, \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n$$

$$w_j = \frac{e_j}{\sum_{j=1}^{n} e_j}, \quad j = 1, 2, \ldots, n.$$

- Etape 6 :

Normaliser la matrice M= (ymk)

$$h_{ij} = \frac{y_{ij}}{\sqrt{\sum_{i}^{m} y^2{}_{ij}}}, \quad i = 1, \ldots, m;\ j = 1, \ldots, k.$$

- Etape 7 :

Multiplier les colonnes de la matrice normalisée par les poids associés pour obtenir la matrice de décision pondérée B= (bmk)

$$b_{ij} = w_j \times h_{ij}, \quad i = 1 \ldots, m; j = 1 \ldots, k$$

Où wj est le poids pour j critère.

- Etape 8 :

Confirmez la solution idéale positive et la solution idéale négative. La solution idéale positive est notée A+, et la solution idéale négative est notée A−. Ils sont définis comme suit : Où Kb est l'ensemble des critères d'avantages et Kc est l'ensemble des critères de coût.

$$A^+ = \{b_1{}^+, b_2{}^+, \ldots, b_k{}^+\} = \{(\max_i b_{ij} | j \in K_b)\,(\min_i b_{ij} | j \in K_c)\}$$
$$A^- = \{b_1{}^-, b_2{}^-, \ldots, b_k{}^-\} = \{(\min_i b_{ij} | j \in K_b)\,(\max_i b_{ij} | j \in K_c)\}$$

- Etape 9 :

  Obtenir les mesures de séparation des alternatives existantes des solutions idéales positives et négatives. Les mesures de séparation basées sur la distance euclidienne, Si+ et Si−, sont respectivement dérivées des suivants :

$$S_i^+ = \sqrt{\sum_{j=1}^{n} \left(b_j^+ - b_{ij}\right)^2}, \quad i = 1, \ldots, m; j = 1, \ldots, k$$
$$S_i^- = \sqrt{\sum_{j=1}^{n} \left(b_j^- - b_{ij}\right)^2}, \quad i = 1, \ldots, m; j = 1, \ldots, k.$$

- Etape 10 :

Calculez la proximité relative à la solution idéale :

$$C_i = \frac{S_i^-}{S_i^- + S_i^+}, \quad i = 1, \ldots, m.$$

Plus haut Ci représente une plus grande influence. Ainsi, la capacité de propagation du nœud dépend de la valeur Ci.

4- Model SI :

Tout d'abord, nous en tirons une équation qui représenterait comment une infection serait théoriquement générer une équation théorique pour un SI temporel réseau, nous regardons la relation entre S et I, qui est S + I = 1. Ensuite, nous calculons la probabilité pour chaque combinaison de bords, qui pourrait être SI, SS ou II.

Après avoir trouvé les probabilités de chaque combinaison possible, la valeur attendue pour le nombre de nœuds infectés à l'étape de temps suivante est déterminée. Depuis le SS et II combinaisons de nœuds ne sera pas augmenter dans la fraction des infectés, les combinaisons sont classé comme 0, parce que 0 nœuds deviennent infectés. La dernière combinaison, SI, entraînera dans un nœud de plus qui devient infecté, de sorte que la proportion de nœuds infectés augmente avec 1/n. Le taux de changement du nombre de nœuds infectés est alors écrit comme:

$$\frac{dI}{dt} = \begin{cases} \frac{0}{n} & \text{with probability } (1-I)^2 + I^2 \\ \frac{1}{n} & \text{with probability } 2I(1-I) \end{cases}$$

À partir de là, nous écrivons la fonction de valeur attendue pour le nombre de nœuds infectés, qui vient de l'idée que la valeur attendue est équivalente à la somme des possibles

$$\frac{dI}{dt} = \frac{2}{n} I(1-I).$$

Les résultats multipliés par leurs probabilités. Le changement attendu des nœuds infectés simplifie:

Nous passons ensuite par une dérivation similaire pour activer deux bords à la fois, puis trois, et concevoir une théorie générale du champ moyen basée sur le nombre de bords activés à une fois (w) :

$$\frac{dI}{dt} = \frac{2w}{n} I(1-I).$$

Dans cette dérivation, les bords qui partagent un nœud commun sont ignorés. Nous simulons ensuite la propagation de l'infection sur un réseau temporel selon les mêmes hypothèses que le modèle et la comparer à notre équation théorique.

4-1 Évaluation avec modèle SI :

Pour évaluer le rendement de notre méthode de classement, nous utilisons le modèle SI pour examiner la diffusion de l'influence des nœuds les mieux classés. Il a été largement utilisé pour la dynamique épidémique sur les réseaux.

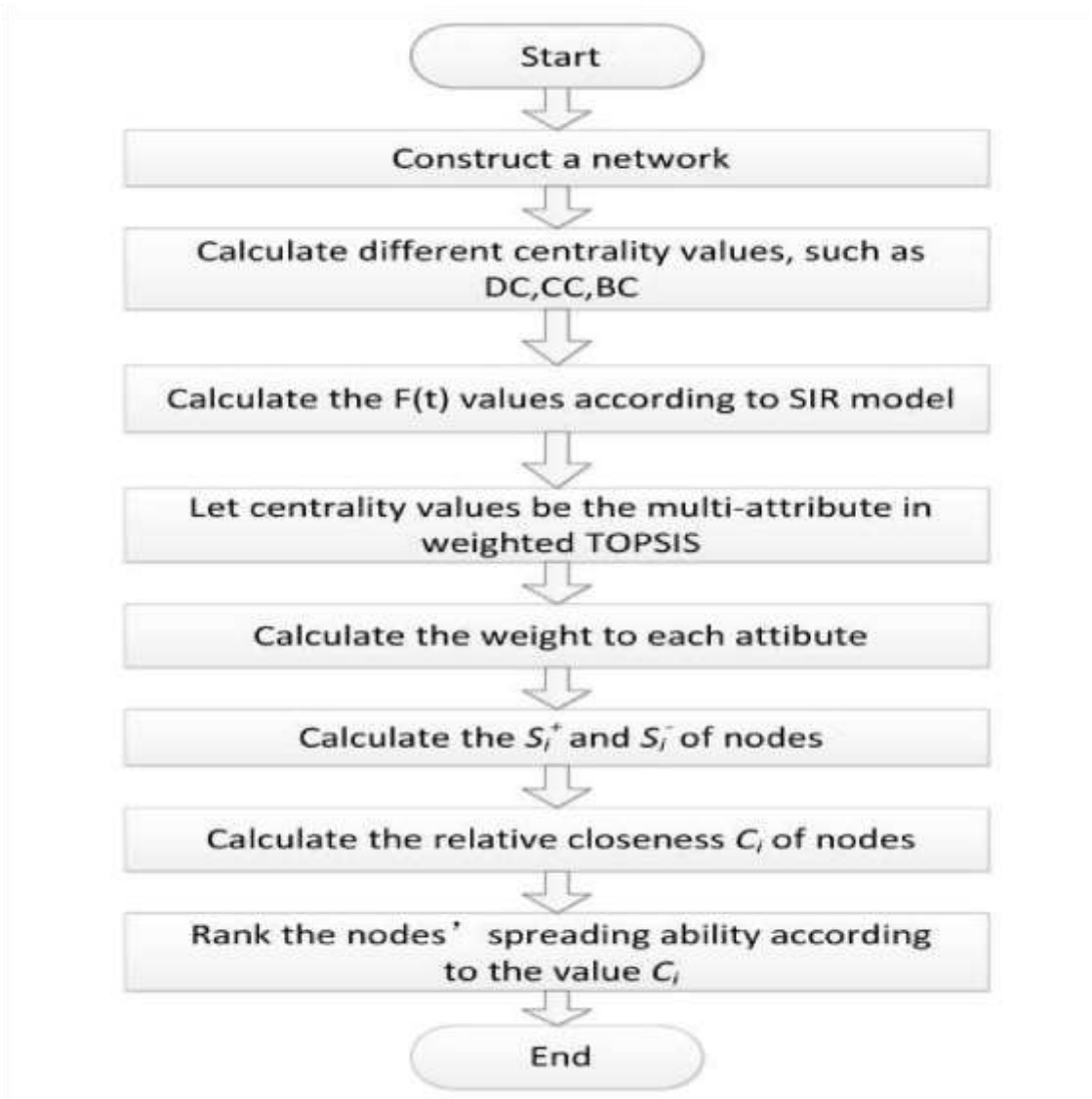

Figure 4 : L'organigramme de la méthode proposée.

Dans si modèle, chaque nœud comporte deux états distincts : (i) Sensible S(t) représente le nombre d'individus sensibles (pas encore infectés) à la maladie; et (ii) Infecté I(t) désigne le nombre d'individus qui ont été infectés et qui sont en mesure de transmettre la maladie aux individus

sensibles. À chaque étape, pour chaque nœud infecté, un voisin susceptible au hasard obtient infecté par la probabilité λ.

Pour la propagation épidémique sur les réseaux, λ détermine la portée ou l'échelle sur laquelle un nœud peut exercer une influence. La propagation épidémique sur les réseaux est un bon exemple pour illustrer le concept multi-échelle : un nœud infecté peut propager la maladie non seulement à ses voisins immédiats, mais aussi à ses voisins d'ordre supérieur à travers les intermédiaires.

Dans ce mode, le nombre de nœuds infectés au moment t est indiqué par F (t). À l'état stable

(État final), le nombre de nœuds infectés est identique en utilisant différents nœuds initialement infectés, et il devrait être égal au nombre total de nœuds dans les réseaux. Moins le réseau atteint l'état stable, plus le nœud initialement infecté est influent. Cela signifie que l'indicateur pour évaluer l'influence du nœud initialement infecté est le nombre moyen de nœuds infectés à chaque étape ou le taux de propagation.

5-Dynamic K-Means:

K-means (k-moyennes) est un algorithme non supervisé de clustering, populaire en Machine Learning.

K-means est un algorithme non supervisé de clustering non hiérarchique. Il permet de regrouper en K clusters distincts les observations du data set. Ainsi les données similaires se retrouveront dans un même cluster. Par ailleurs, une observation ne peut se retrouver que dans un cluster à la fois (exclusivité d'appartenance). Une même observation, ne pourra donc, appartenir à deux clusters différents. Pour pouvoir regrouper un jeu de données en K cluster distincts, l'algorithme K-Means a besoin d'un moyen de comparer le degré de similarité entre les différentes observations. Ainsi, deux données qui se ressemblent, auront une distance de dissimilarité réduite, alors que deux objets différents auront une distance de séparation plus grande. Les littératures mathématiques et statistiques regorgent de définitions de distance, les plus connues pour les cas de clustering sont :

- **La distance Euclidienne** : C'est la distance géométrique qu'on apprend au collège. Soit une matrice X à n variables quantitatives. Dans l'espace vectoriel Eespo(n). La distance euclidienne d entre deux observations X1 et X2 se calcule comme suit :

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} (x_{1n} - x_{2n})^2}$$

- **La distance de Manhattan (taxi-distance)** : est la distance entre deux points parcourue par un taxi lorsqu'il se déplace dans une ville où les rues sont agencées selon un réseau ou un quadrillage. Un taxi-chemin est le trajet fait par un taxi lorsqu'il se déplace d'un nœud du réseau à un autre en utilisant les déplacements horizontaux et verticaux du réseau.

5-1 Choisir K : le nombre de clusters

Choisir un nombre de cluster K n'est pas forcément intuitif. Spécialement quand le jeu de données est grand et qu'on n'ait pas un a priori ou des hypothèses sur les données. Un nombre K grand peut conduire à un partitionnement trop fragmenté des données. Ce qui empêchera de découvrir des patterns intéressants dans les données. Par contre, un nombre de clusters trop petit, conduira à avoir, potentiellement, des clusters trop généralistes contenant beaucoup de données. Dans ce cas, on n'aura pas de patterns "fins" à découvrir.

Pour un même jeu de données, il n'existe pas un unique clustering possible. La difficulté résidera donc à choisir un nombre de cluster K qui permettra de mettre en lumière des patterns intéressants entre les données. Malheureusement il n'existe pas de procédé automatisé pour trouver le bon nombre de clusters.

La méthode la plus usuelle pour choisir le nombre de clusters est de lancer K-Means avec différentes valeurs de K et de calculer la variance des différents clusters. La variance est la somme des distances entre chaque *centroid* d'un cluster et les différentes observations incluent dans le même cluster. Ainsi, on cherche à trouver un nombre de clusters K de telle sorte que les clusters retenus minimisent la distance entre leurs centres (centroids) et les observations dans le même cluster. On parle de minimisation de la distance intra-classe**.**

La variance des clusters se calcule comme suit :

$$V = \sum_j \sum_{x_i \to c_j} D(c_j, x_i)^2$$

Avec :

- c_j : Le centre du cluster (le centroïd)
- x_i : la ième observation dans le cluster ayant pour centroïd c_j
- $D(c_j, x_i)$: La distance (euclidienne ou autre) entre le centre du cluster et le point x_i

Généralement, en mettant dans un graphique les différents nombres de clusters K en fonction de la variance, on retrouve un graphique similaire à celui-ci :

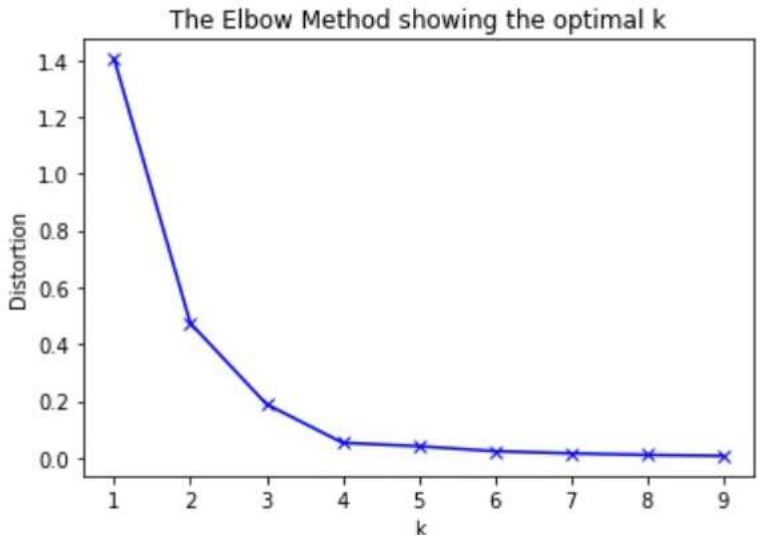

Figure 1: Elbow method implementation showing the optimale k

On remarque sur ce graphique, la forme d'un bras où le point le plus haut représente l'épaule et le point où K vaut 9 représente l'autre extrémité : la main. Le nombre optimal de clusters est le point représentant le coude**.** Ici le coude peut être représenté par K valant 2 ou 3. C'est le nombre optimal de clusters. Généralement, le point du coude est celui du nombre de clusters à partir duquel la variance ne se réduit plus significativement. En effet, la "chute" de la courbe de variance (distorsion) entre 1 et 3 clusters est significativement plus grande que celle entre 5 clusters et 9 clusters.

Le fait de chercher le point représentant le coude, a donné nom à cette méthode : La méthode Elbow (coude en anglais).

Finalement, le choix (au vu de ce graphique), entre 2 ou 3 clusters, reste un peu flou et à votre discrétion. Le choix se fera en fonction de votre jeu de données et ce que vous cherchez à accomplir. Finalement, Il n'y a pas de solution unique à un problème de clustering.

5-2 Cas d'utilisation K-means

K-Means en particulier et les algorithmes de clustering de façon générale ont tous un objectif commun : Regrouper des éléments similaires dans des clusters. Ces éléments peuvent être tous et n'importe quoi, du moment qu'ils sont encodés dans une matrice de données.

Les champs d'application de K-Means sont nombreux, il est notamment utilisé en :

- la segmentation de la clientèle en fonction d'un certain critère (démographique, habitude d'achat etc….)
- Utilisation du clustering en Data Mining lors de l'exploration de données pour déceler des individus similaires. Généralement, une fois ces populations détectées, d'autres techniques peuvent être employées en fonction du besoin.
- Clustering de documents (regroupement de documents en fonction de leurs contenus. Pensez à comment Google Actualités regroupe des documents par thématiques.)

5-3 Fonctionnement de l'algorithme K-Means :

k-means est un algorithme itératif qui minimise la somme des distances entre chaque individu et le centroïd. **Le** choix initial des centroïdes conditionne le résultat final**.** Admettant un nuage d'un ensemble de points, K-Means change les points de chaque cluster jusqu'à ce que la somme ne puisse plus diminuer. Le résultat est un ensemble de clusters compacts et clairement séparés, sous réserve de choisir la bonne valeur du nombre de clusters.

5-3-1 Principe algorithmique :

❖ Algorithme K-means

Entrée :

- K le nombre de cluster à former
- Le Training Set (matrice de données)

DEBUT

Choisir aléatoirement K points (une ligne de la matrice de données). Ces points sont les centres des clusters (nommé centroïd).

REPETER

Affecter chaque point (élément de la matrice de donnée) au groupe dont il est le plus proche au son centre, Recalculer le centre de chaque cluster et modifier le centroide.

JUSQU'A CONVERGENCE

OU (stabilisation de l'inertie totale de la population)

FIN ALGORITHME

Note 1: Lors de la définition de l'algorithme, quand je parle de "point", c'est un point au sens "donnée/data" qui se trouve dans un espace vectoriel de dimension n. Avec n : le nombre de colonnes de la matrice de données.

Note 2 : La convergence de l'algorithme K-Means peut être l'une des conditions suivantes :

- Un nombre d'itérations fixé à l'avance, dans ce cas, K-means effectuera les itérations et s'arrêtera peu importe la forme de clusters composés.
- Stabilisation des centres de clusters (les centroids ne bougent plus lors des itérations).

L'affectation d'un point µ à un cluster se fait en fonction de la distance de ce point par rapport aux différents K centroides. Par ailleurs, ce point µ se fera affecté à un cluster i s'il est plus proche de son centroid (distance minimale). Finalement, la distance entre deux points dans le cas de K Means se calcule par les méthodes évoquées dans le paragraphe "notion de similarité".

6-les bibliothèques de python utilisé:

Python :

Python est un langage de programmation le plus employé par les informaticiens. Ce langage est le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science. En tant que langage de programmation de haut niveau, Python permet aux programmeurs de se focaliser sur ce qu'ils font plutôt que sur la façon dont ils le font. Ainsi, développer du code avec Python est plus rapide qu'avec d'autre langage.

NetworkX :

NetworkX est un progiciel en langage Python pour la création, la manipulation et l'étude de la structure, de la dynamique et du fonctionnement de réseaux complexes. Il est utilisé pour étudier de grands réseaux complexes représentés sous forme de graphes avec des nœuds et des arêtes. En utilisant networkx, nous pouvons charger et stocker des réseaux complexes. Nous pouvons générer de nombreux types de réseaux aléatoires et classiques, analyser la structure du réseau, construire des modèles de réseau, concevoir de nouveaux algorithmes de réseau et dessiner des réseaux.

Figure 2: Networkx library logo

Matplotlib:

Matplotlib est une bibliothèque de traçage disponible pour le langage de programmation Python en tant que composant de NumPy, une ressource de traitement numérique du Big Data. Matplotlib utilise une API orientée objet pour intégrer des tracés dans les applications Python.

Figure 3: matplotlib library logo

Pandas :

Pandas est un package Python open source qui est le plus largement utilisé pour la science des données/l'analyse des données et les tâches d'apprentissage automatique. Il est construit au-dessus d'un autre package nommé Numpy , qui prend en charge les tableaux multidimensionnels. En tant que l'un des packages de gestion de données les plus populaires, Pandas fonctionne bien avec de nombreux autres modules de science des données au sein de l'écosystème Python et est généralement inclus dans chaque distribution Python, de celles

fournies avec votre système d'exploitation aux distributions de fournisseurs commerciaux comme ActivePython d'ActiveState .

Figure 4 :Pandas library logo

NumPy:

NumPy est le package fondamental pour le calcul scientifique en Python. Il s'agit d'une bibliothèque Python qui fournit un objet tableau multidimensionnel, divers objets dérivés (tels que des tableaux masqués et des matrices) et un assortiment de routines pour des opérations rapides sur des tableaux, y compris mathématiques, logiques, manipulation de forme, tri, sélection, E/S. transformées de Fourier discrètes, algèbre linéaire de base, opérations statistiques de base, simulation aléatoire et bien plus encore.

Au cœur du package NumPy, se trouve l'objet *ndarray*. Cela encapsule des tableaux à *n* dimensions de types de données homogènes, de nombreuses opérations étant effectuées dans du code compilé pour les performances. Il existe plusieurs différences importantes entre les tableaux NumPy et les séquences Python standard :

- Les tableaux NumPy ont une taille fixe à la création, contrairement aux listes Python (qui peuvent croître dynamiquement). Changer la taille d'un *ndarray* créera un nouveau tableau et supprimera l'original.
- Les éléments d'un tableau NumPy doivent tous être du même type de données et auront donc la même taille en mémoire. L'exception : on peut avoir des tableaux d'objets (Python, y compris NumPy), permettant ainsi des tableaux d'éléments de tailles différentes.

- Les tableaux NumPy facilitent les opérations mathématiques avancées et autres types d'opérations sur un grand nombre de données. En règle générale, ces opérations sont exécutées plus efficacement et avec moins de code qu'il n'est possible d'utiliser les séquences intégrées de Python.
- Une pléthore croissante de packages scientifiques et mathématiques basés sur Python utilisent des tableaux NumPy ; bien que ceux-ci prennent généralement en charge l'entrée de séquence Python, ils convertissent cette entrée en tableaux NumPy avant le traitement, et ils produisent souvent des tableaux NumPy. En d'autres termes, pour utiliser efficacement la plupart (peut-être même la plupart) des logiciels scientifiques/mathématiques basés sur Python d'aujourd'hui, il suffit de savoir comment utiliser les types de séquence intégrés de Python - il faut également savoir comment utiliser les tableaux NumPy.

Figure 5 :Numpy library logo

NDLIB :

NDLIB est un progiciel Python qui permet de décrire, simuler et étudier les processus de diffusion sur des réseaux complexes.

Figure 6:Ndlib library logo

Scikit-learn est une bibliothèque d'analyse de données open source et la référence en matière d'apprentissage automatique (ML) dans l'écosystème Python. Principaux concepts et caractéristiques : Méthodes de prise de décisions algorithmiques, y compris : Classification : identification et catégorisation des données en fonction des tendances

- Des outils simples et efficaces pour l'analyse prédictive des données
- Accessible à tous et réutilisable dans divers contextes
- Construit sur NumPy, SciPy et matplotlib
- Open source, utilisable commercialement - licence BSD

Figure 7:Scikit learn library logo

6-Code:

a- Les données:

Cet ensemble de données comprend des cercles (ou des listes d'amis) de Facebook. Les données de Facebook ont été recueillies auprès des participants à l'enquête. L'ensemble de données comprend les caractéristiques des nœuds (profils), des cercles et des réseaux d'ego.

Les données de Facebook ont été anonymisées en remplaçant les identifiants internes de Facebook pour chaque utilisateur par une nouvelle valeur. De plus, bien que des vecteurs de caractéristiques de cet ensemble de données aient été fournis, l'interprétation de ces caractéristiques a été obscurcie. Par exemple, lorsque l'ensemble de données original peut contenir une caractéristique "political=Democratic Party", les nouvelles données contiendraient simplement "political=anonymized feature 1". Ainsi, en utilisant les données anonymisées, il est possible de déterminer si deux utilisateurs ont les mêmes affiliations politiques, mais pas ce que leurs affiliations politiques individuelles représentent.

Informations sur nos données :

```
Graph = nx.read_edgelist('Data/facebookData.txt',create_using=nx.Graph(),nodetype=int)

print(nx.info(Graph))
```

```
Name:
Type: Graph
Number of nodes: 4039
Number of edges: 88234
Average degree:  43.6910
```

Figure 8:chargement des données de Facebook

Voici notre graphe de dataset Facebook

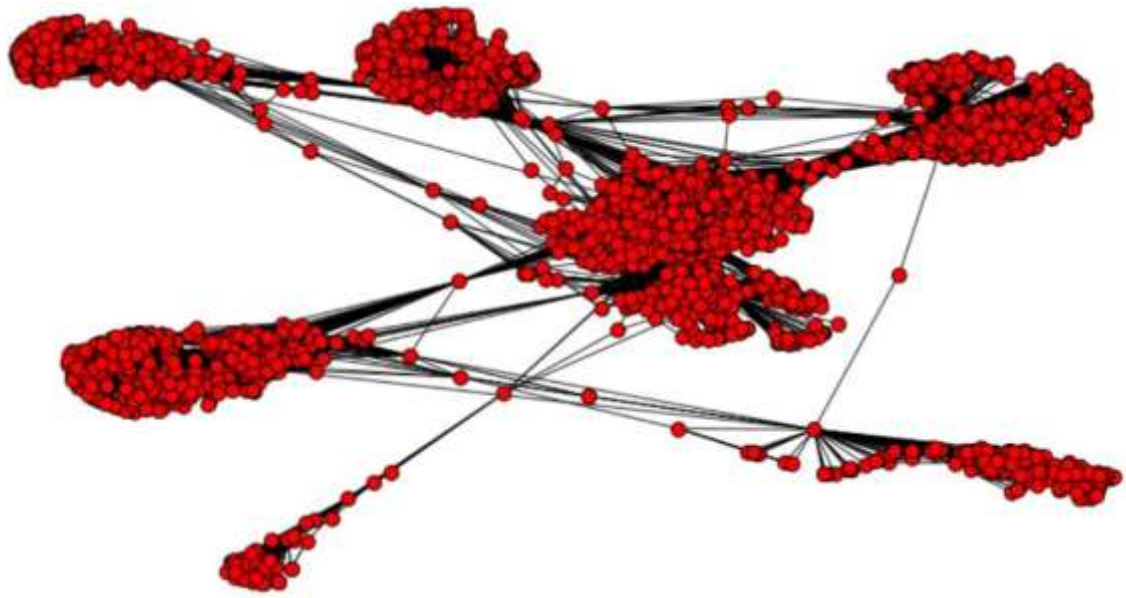

Figure 9:le graphe de datasets Facebook

L'ensemble de données utilisé ici se compose de « bords » Facebook (ou « listes d'amis »). Ces données ont été recueillies auprès des participants au sondage au moyen de cette application Facebook. L'ensemble de données comprend des caractéristiques de nœuds égocentriques (profils), de cercles et de réseaux. Ces données ont été anonymisées en remplaçant Facebook. -ID interne pour chaque utilisateur avec une nouvelle valeur. Chaque ego-réseau que nous avons cercle, bords, egofeat, feat, featnames fichier. Comme nous ne regrouperons les gens que par leur amitié, nous nous concentrons sur les bords du dossier.

b- les mesures de centralité :

```
print(nx.degree_centrality(Graph))
...
print(nx.betweenness_centrality(Graph))
...
print(nx.closeness_centrality(Graph))
...
print(nx.eigenvector_centrality(Graph))
```

Figure 10:les mesures de centralité

Les mesures de centralité nous permettent de repérer les nœuds les plus importants d'un graphique. Cela nous aide essentiellement à identifier :

- Nœuds influents dans un réseau social.
- Nœuds qui diffusent de l'information à de nombreux nœuds
- hubs dans un réseau de transport
- Nœuds qui empêchent le réseau de se briser

c- l'implémentation de topsis :

Première étape : Créer la matrice d'évaluation.

```
# Read a csv file (DC.csv) to a dataframe with custom delimiter
DC =  pd.read_csv('Data/DC.csv', sep=','  , engine='python',header=None)
DC_T = DC.transpose()

DC_T[['Node', 'DC']] = DC_T[0].str.split(':', 1, expand=True)
DC_T = DC_T.drop([0],axis=1)
DC_T
```

```
# Read a csv file (BC.csv) to a dataframe with custom delimiter
BC =  pd.read_csv('Data/BC.csv', sep=','  , engine='python',header=None)
BC_T = BC.transpose()

BC_T[['Node', 'BC']] = BC_T[0].str.split(':', 1, expand=True)
BC_T = BC_T.drop([0,"Node"],axis=1)
BC_T
```

```
# Read a csv file (CC.csv) to a dataframe with custom delimiter
CC =  pd.read_csv('Data/CC.csv', sep=','  , engine='python',header=None)
CC_T = CC.transpose()

CC_T[['Node', 'CC']] = CC_T[0].str.split(':', 1, expand=True)
CC_T = CC_T.drop([0,"Node"],axis=1)
CC_T
```

```
# Read a csv file (EC.csv) to a dataframe with custom delimiter
EC =  pd.read_csv('Data/EC.csv', sep=','  , engine='python',header=None)
EC_T = EC.transpose()

EC_T[['Node', 'EC']] = EC_T[0].str.split(':', 1, expand=True)
EC_T = EC_T.drop([0,"Node"],axis=1)
EC_T
```

```
#Decision Matrix
Matrix = pd.concat([DC_T, BC_T, CC_T, EC_T], axis=1)
Matrix
```

Figure 11: Building the decision matrix code

La matrice de décision (matrice) devrait être construite avec chaque ligne représentant un modèle de rechange, et chaque colonne représentant un critère comme les mesures DC, BC, CC et EC

	Node	DC	BC	CC	EC
0	0	0.08593363051015354	0.14630592147442917	0.35334266713335666	3.3917961722702005e-05
1	1	0.004210004952947003	2.7832744209034606e-06	0.2613761408505405	6.045346134948106e-07
2	2	0.0024764735017335313	7.595021178512074e-08	0.26125776397515527	2.2334609371911963e-07
3	3	0.004210004952947003	1.6850656559280464e-06	0.2613761408505405	6.63564808392105e-07
4	4	0.0024764735017335313	1.8403320547933104e-07	0.26125776397515527	2.2364157028893598e-07
...	...	...	...	...	...
4034	4034	0.0004952947003467063	0.0	0.18398870005012075	2.951270291832791e-10
4035	4035	0.0024764735017335313	0.0	0.1839803171131766	2.912901428470978e-10
4036	4036	0.0004952947003467063	0.0	0.18398870005012075	2.9312234293826837e-10
4037	4037	0.0009905894006934125	7.156846879751761e-08	0.18400546821599453	2.98923251382765e-10
4038	4038	0.002228826151560178	6.338921522065847e-07	0.18404740200546946	8.915174731918863e-10

4039 rows × 5 columns

Figure 12:matrice de décision

• Deuxième étape : Normalisation pour former la matrice.

```
def step_2(self):
    # normalized scores
    self.normalized_decision = np.copy(self.evaluation_matrix)
    sqrd_sum = np.zeros(self.column_size)
    for i in range(self.row_size):
        for j in range(self.column_size):
            sqrd_sum[j] += self.evaluation_matrix[i, j]**2
    for i in range(self.row_size):
        for j in range(self.column_size):
            self.normalized_decision[i,
                                     j] = self.evaluation_matrix[i, j]/(sqrd_sum[j]**0.5)
```

Figure 13:normalisation

• Troisième étape : Multiplier les colonnes de la matrice de décision normalisée par les pondérations associées pour obtenir la matrice de décision pondérée

```
def step_3(self):
    from pdb import set_trace
    self.weighted_normalized = np.copy(self.normalized_decision)
    for i in range(self.row_size):
        for j in range(self.column_size):
            self.weighted_normalized[i, j] *= self.weight_matrix[j]
```

Figure 14 weight de normalisation

• Quatrième étape : Déterminer la pire et la meilleure solution

```
def step_4(self):
    self.worst_alternatives = np.zeros(self.column_size)
    self.best_alternatives = np.zeros(self.column_size)
    for i in range(self.column_size):
        if self.criteria[i]:
            self.worst_alternatives[i] = min(self.weighted_normalized[:, i])
            self.best_alternatives[i] = max(self.weighted_normalized[:, i])
        else:
            self.worst_alternatives[i] = max(self.weighted_normalized[:, i])
            self.best_alternatives[i] = min(self.weighted_normalized[:, i])
```

Figure 15:determiner worst et best alternative code

Cinquième étape : Calculez la distance entre l'alternative cible et l'idéal négatif.

```
def step_5(self):
    self.worst_distance = np.zeros(self.row_size)
    self.best_distance = np.zeros(self.row_size)

    self.worst_distance_mat = np.copy(self.weighted_normalized)
    self.best_distance_mat = np.copy(self.weighted_normalized)

    for i in range(self.row_size):
        for j in range(self.column_size):
            self.worst_distance_mat[i][j] = (self.weighted_normalized[i][j]-self.worst_alternatives[j])**2
            self.best_distance_mat[i][j] = (self.weighted_normalized[i][j]-self.best_alternatives[j])**2

            self.worst_distance[i] += self.worst_distance_mat[i][j]
            self.best_distance[i] += self.best_distance_mat[i][j]

    for i in range(self.row_size):
        self.worst_distance[i] = self.worst_distance[i]**0.5
        self.best_distance[i] = self.best_distance[i]**0.5
```

Figure 16:calcule de similarity

Sixième étape : Classer les solutions de rechange en fonction de la proximité de la solution idéale

```
def step_6(self):
    np.seterr(all='ignore')
    self.worst_similarity = np.zeros(self.row_size)
    self.best_similarity = np.zeros(self.row_size)

    for i in range(self.row_size):
        # calculate the similarity to the worst condition
        self.worst_similarity[i] = self.worst_distance[i] / \
            (self.worst_distance[i]+self.best_distance[i])
```

Figure 17:calcule closeness to the ideal solution code

```
#sort DataFrame by Closnness Column (Descending)
Out = Out.sort_values(by="C",ascending=False)
Out

# Save it into a csv file (out.csv)
Out.to_csv("Data/Out.csv",index=False)
```

```
#Function return the index of the grather value(Closness vector) based on a descending order
def ranking_closness(data):
    return [i for i in data.argsort()[-len(data):][::-1]]
```

```
#Call the function Ranking_closness
C = ranking_closness(Closnness)
C
```

```
#retrieve the nodes that correspond to the indices returned by the function above
Tab = []
for i in range(len(C)):
      Tab.append(Matrix['Node'].loc[C[i]])

Tab
```

```
#Convert Tab above to DataFrame
Tab_Data=pd.DataFrame(Tab,columns=["Node"])

Tab_Data["Node"].to_csv("Data/Topsis.csv",index=False)
```

Figure 18: trier dataframe par code de colonne de closeness(décroissant)

Classer les alternatives en fonction de la proximité relative de la solution idéale : les alternatives avec un Ci plus élevé sont considérées comme plus importantes et devraient être prioritaires

The nodes depending on the relative closeness to the ideal solution

```
Final_Tab = pd.concat([pd.read_csv("Data/Out.csv"),Tab_Data],axis=1)

Final_Tab.head(10)
```

Classer les alternatives en fonction de la proximité relative à la solution idéale : les alternatives avec un Ci plus élevé sont considérées comme plus importantes et devraient être données plus élevé priorité

	S+	S-	C	Node
0	0.019029	0.200396	0.913277	107
1	0.061948	0.141539	0.695566	1684
2	0.102563	0.100961	0.496063	1912
3	0.103349	0.098846	0.488865	3437
4	0.140349	0.061395	0.304321	0
5	0.142845	0.060458	0.297379	1085
6	0.155715	0.046803	0.231106	698
7	0.163087	0.039165	0.193646	567
8	0.168330	0.034346	0.169464	58
9	0.174851	0.026805	0.132923	428

Figure 19:The ranking depending on the relative closeness to the ideal solution

L'utilisateur peut saisir le nombre de nœuds influents désirés

User

```
K = int(input("Enter the number of the top influential nodes:"))

for i in range(K):
    print("The",i+1,"top influential node is ",Final_Tab["Node"][i])
```

```
Enter the number of the top influential nodes:10
The 1 top influential node is   107
The 2 top influential node is   1684
The 3 top influential node is   1912
The 4 top influential node is   3437
The 5 top influential node is  0
The 6 top influential node is   1085
The 7 top influential node is   698
The 8 top influential node is   567
The 9 top influential node is   58
The 10 top influential node is   428
```

Figure 20:Top k-nodes by the user

Pour évaluer la performance de notre méthode de classement, nous utilisons le modèle Susceptible-Infected (SI) pour examiner l'influence de propagation des nœuds les mieux classés. Il a été largement utilisé pour la dynamique épidémique sur les réseaux. Dans le modèle SI, chaque nœud a deux états distincts:

Sensible S(t) représente le nombre d'individus sensibles à la maladie (pas encore infectés); et (ii) Infecté I(t) indique le nombre d'individus qui ont été infectés et qui sont en mesure de transmettre la maladie aux individus sensibles. À chaque étape, pour chaque nœud infecté, un voisin susceptible au hasard est infecté avec la probabilité λ (pour l'uniformité, ici nous fixons λ = 0,3 initialement). Pour la propagation épidémique sur les réseaux, λ détermine la portée ou l'échelle sur laquelle un nœud peut exercer une influence. Concept d'échelle : un nœud infecté peut propager la maladie non seulement à ses voisins immédiats, mais aussi à ses voisins d'ordre supérieur à travers les intermédiaires. Dans ce mode, le nombre de nœuds infectés au moment t est indiqué par F (t). À l'état stable (état final), le nombre de nœuds infectés est identique en utilisant différents nœuds initialement infectés, et il devrait être égal au nombre total de nœuds dans les réseaux. Moins le réseau atteint l'état stable, plus le nœud initialement infecté est influent. Cela signifie que l'indicateur pour évaluer l'influence du nœud initialement infecté est le nombre moyen de nœuds infectés à chaque étape ou le taux de propagation.

DC, CC, BC et EC sont comparés à la méthode proposée (TOPSIS). Simultanément, dans Facebook ego Network dans lequel l'information circule au moyen de promenades, nous comparons la méthode proposée avec DC, CC, BC et EC. Si l'on compare la méthode proposée avec celle de DC,

on constate qu'il y a cinq mêmes membres dans les 10 premières listes. La méthode proposée et la C.-B. ont tous les mêmes membres dans les 10 premières listes. Les quatre premières listes

Les méthodes (à l'exception de l'EG) sont les mêmes, et elles ont le même classement : Nœud 107 autres nœuds, où « ‘’ » représente « meilleur que » ou « plus influent que », c'est-à-dire que le nœud 107 est le nœud le plus influent dans le réseau de l'ego de Facebook en utilisant ces quatre mesures.

```
#The top-10 ranked nodes by degree centrality (DC)
DCR = res[["Node", "DC"]]
DCR = DCR.sort_values(by='DC', ascending=False)
DCR.rename(columns = {'Node':'DCN'}, inplace=True)

DCR["DCN"].to_csv("Data/DCR.csv",index=False)
```

```
#The top-10 ranked nodes by betweenness centrality (BC)
BCR = res[["Node", "BC"]]
BCR = BCR.sort_values(by='BC', ascending=False)
BCR.rename(columns = {'Node':'BCN'}, inplace=True)

BCR["BCN"].to_csv("Data/BCR.csv",index=False)
```

```
#The top-10 ranked nodes by closeness centrality (CC)
CCR = res[["Node", "CC"]]
CCR = CCR.sort_values(by='CC', ascending=False)
CCR.rename(columns = {'Node':'CCN'}, inplace=True)

CCR["CCN"].to_csv("Data/CCR.csv",index=False)
```

```
#The top-10 ranked nodes by eigenvector centrality (EC)
ECR = res[["Node", "EC"]]
ECR = ECR.sort_values(by='EC', ascending=False)
ECR.rename(columns = {'Node':'ECN'}, inplace=True)

ECR["ECN"].to_csv("Data/ECR.csv",index=False)
```

```
fac = pd.concat([pd.read_csv("Data/DCR.csv"),pd.read_csv("Data/BCR.csv"), pd.read_csv("Data/CCR.csv"), pd.read_csv("Data/ECR.csv"
```

Figure 21: Sorting the nodes by TOPSIS and centrality measures code

	DCN	BCN	CCN	ECN	Topsis
0	107	107	107	1912	107
1	1684	1684	58	2266	1684
2	1912	3437	428	2206	1912
3	3437	1912	563	2233	3437
4	0	1085	1684	2464	0
5	2543	0	171	2142	1085
6	2347	698	348	2218	698
7	1888	567	483	2078	567
8	1800	58	414	2123	58
9	1663	428	376	1993	428

Figure 22: The top-10 ranked nodes by the TOPSIS and degree centrality (DCN), closeness centrality (CCN), betweenness centrality (BCN), Eigenvector centrality (ECN) in Facebook Ego network

Nous utilisons le modèle SI pour comparer la méthode proposée avec celle du degré, de la proximité, de la centralité entre l'appartenance ou le secteur propre. Dans chaque implémentation,

les 10 principaux nœuds sont sélectionnés pour être infectés, puis l'information se propage dans le réseau selon si model. Nous comparons l'influence des nœuds qui apparaissent dans la liste des 10 premiers par la méthode proposée ou d'autres quatre mesures de centralité. Notez que, sans considérer les effets des nœuds communs dans les deux listes de classement, les différences de ces méthodes peuvent être bien distinguées. Les simulations sur les nœuds infectés cumulés, à savoir F (t), en fonction du temps pour notre réseau. Le nombre cumulatif de nœuds infectés augmente avec le temps et atteint finalement la valeur stable

```
def SI(nodes):
    n = 4039
    Lst = []
    model = ep.SIModel(Graph)
    cfg = mc.Configuration()
    cfg.add_model_parameter('beta', 0.3)
    cfg.add_model_initial_configuration('Infected', nodes)
    model.set_initial_status(cfg)
    res = pd.DataFrame(columns=['iteration', 'Nb_Susceptible', 'Nb_infected'])
    for i in range(n):
        iteration = model.iteration()
        res.loc[len(res.index)] = [iteration['iteration'],iteration['node_count'][0], iteration['node_count'][1] ]
        if iteration['node_count'][1] == n:
            break

    Lst.append(res['iteration'].tolist())
    Lst.append(res['Nb_Susceptible'].tolist())
    Lst.append(res['Nb_infected'].tolist())

    return Lst
```

Figure 23:SI implementation using NDLIB library

```
Rank_DC = set(facebook['DCN'])
Rank_BC = set(facebook['BCN'])
Rank_CC = set(facebook['CCN'])
Rank_EC = set(facebook['ECN'])
Rank_Topsis = set(facebook['Topsis'])
```

```
DClst = []
BClst = []
CClst = []
EClst = []
TOPSISlst = []
for i in range(100000):
    DClst.append(SI(Rank_DC))
    BClst.append(SI(Rank_BC))
    CClst.append(SI(Rank_CC))
    EClst.append(SI(Rank_EC))
    TOPSISlst.append(SI(Rank_Topsis))
```

Figure 24: Applying SI model to each centrality measure and TOPSIS

Comme il est montré à la figure 29, Le modèle SI a pris 5 itérations pour affecter le réseau de trous pour le degré de centralité.

Rank_DC

	iteration	nb_Susceptible	Nb_infected
0	0	4029	10
1	1	576	3463
2	2	201	3838
3	3	55	3984
4	4	0	4039

Figure 25: SI results for DC measure

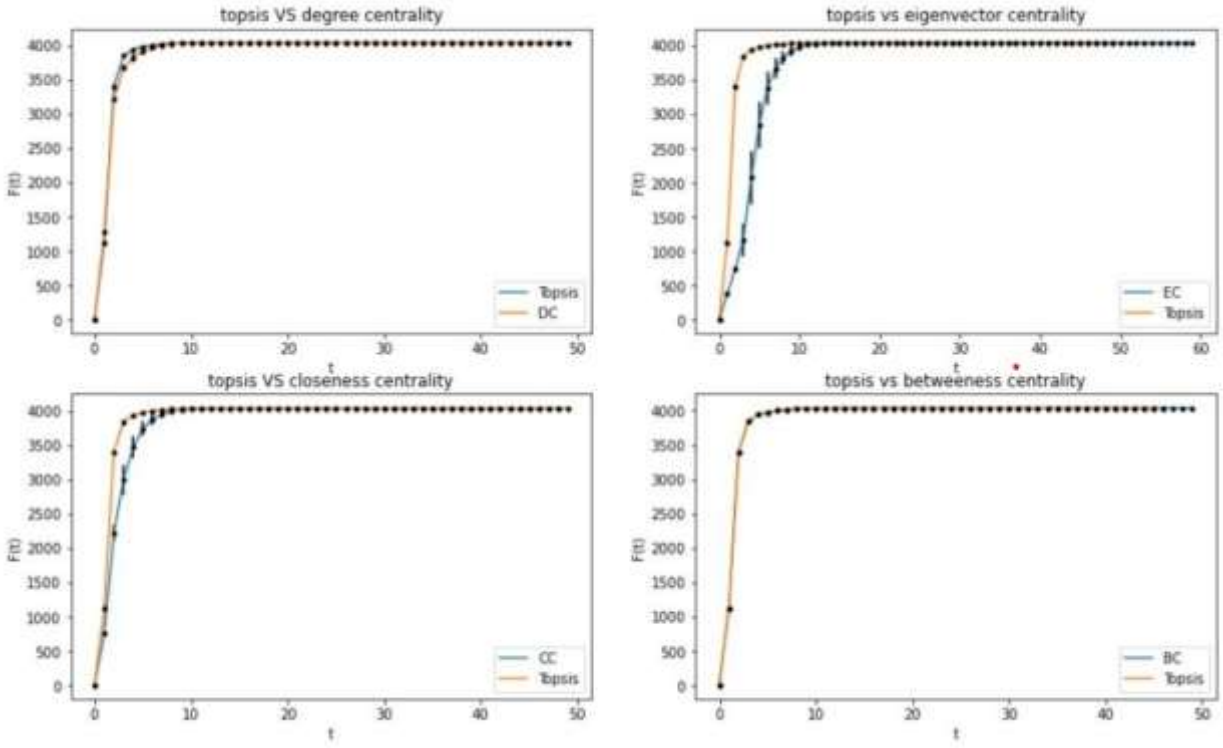

Figure 26:The cumulative number of infected nodes as a function of time, with the initially infected nodes being those that either appear in the top-10 list by the proposed method or DC, CC, and EC. Results are obtained by averaging over 100000 implements (λ = 0.3)

Lorsque nous avons considéré λ = 0,3, les résultats ont changé à chaque itération. Cela peut affecter le modèle même si nous avons augmenté le nombre d'itérations (100000 itérations ont pris 72 heures). Donc, si l'objectif est de comparer la méthode proposée avec les autres mesures de

centralité, nous avons enseigné qu'il serait une excellente idée de propager l'infection avec une plus grande probabilité.

Si nous supposons que $\lambda = 1$, les résultats de la figure 33 montrent comment le nombre cumulatif de nœuds infectés augmente avec le temps.

Par la méthode proposée et BC, le résultat est très similaire, parce que les lignes de la méthode pproposée et BC se chevauchent parfaitement

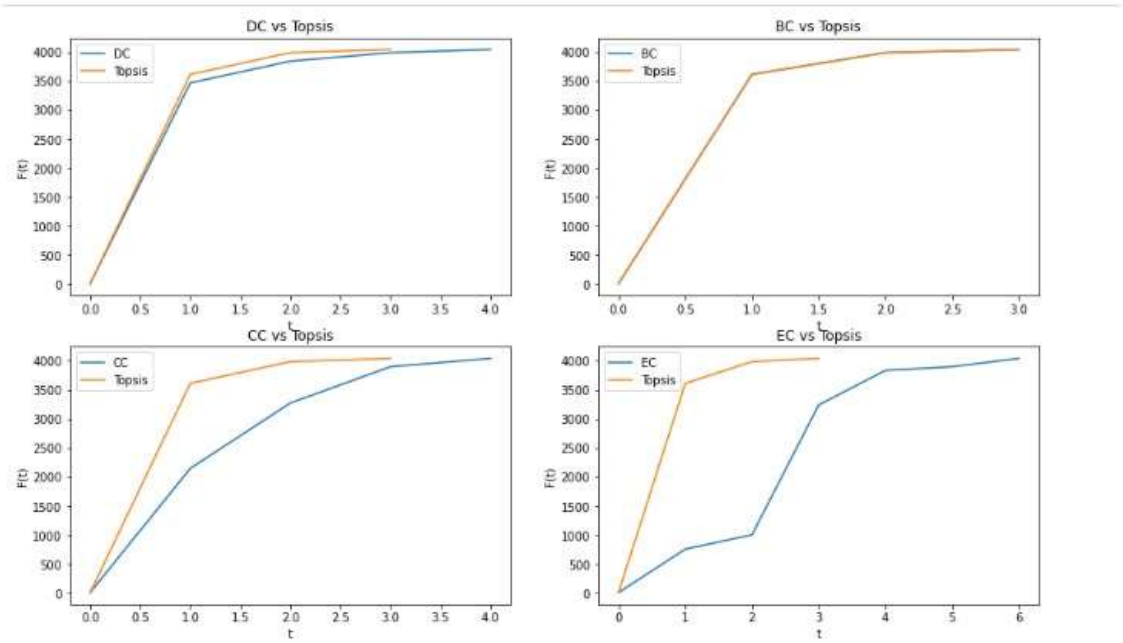

Figure 27: Le nombre cumulé de nœuds infectés en fonction du temps, les nœuds initialement infectés étant ceux qui soit apparaître dans la liste des 10 premiers par la méthode proposée ou DC, CC et EC. Les résultats sont obtenus en faisant la moyenne sur 100 000 implémentations ($\lambda = 1$)

Depuis TOPSIS et la centralité entre-deux ont le même top 10 noeuds influents : Ce ne sont pas seulement les utilisateurs qui ont le plus d'amis qui sont importants, les utilisateurs qui relient une géographie à l'autre sont également importants car cela permet aux utilisateurs de voir le contenu de diverses géographies. Entretemps, la centralité quantifie le nombre de fois où un noeud se trouve dans le chemin le plus court entre deux autres noeuds.

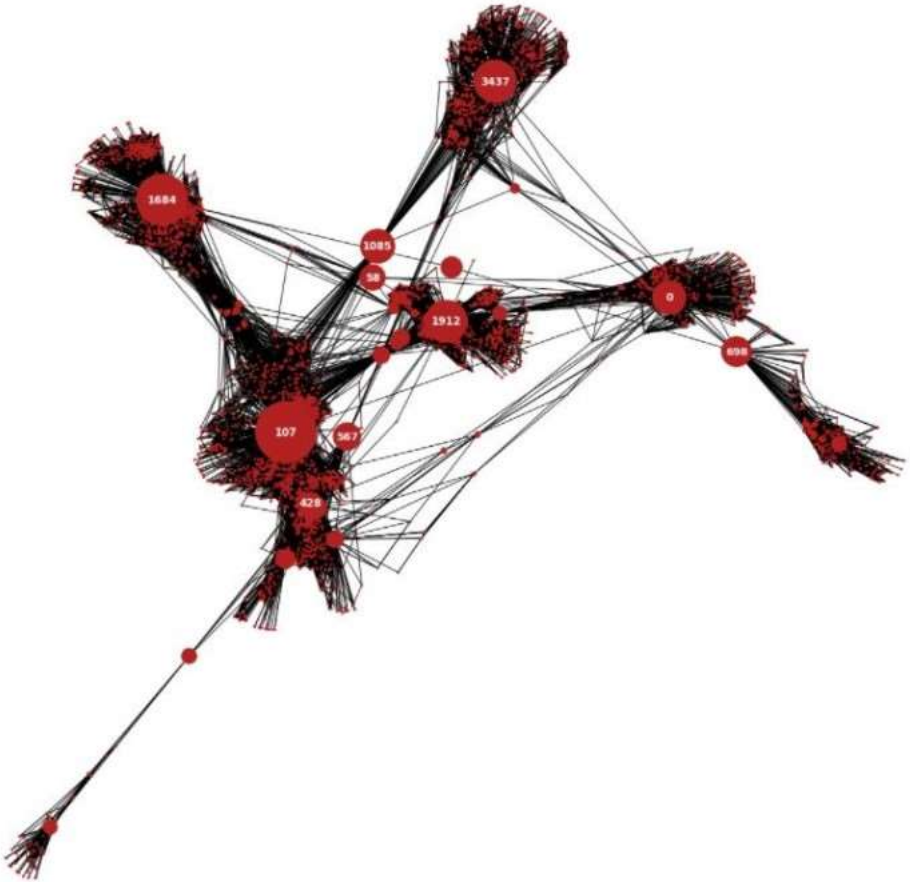

Figure 28:Affichage des nœuds sized par leur valeur betweeness

Vous pouvez voir les nœuds dimensionnés par leurs valeurs de centralité intermédiaires ici. Ils peuvent être considérés comme des passants d'information. Casser n'importe quel des nœuds avec une haute centralité entre les deux cassera le graphique en de nombreuses parties.

d- Exemple de datasets football :

Nous avons suivi même démarche que datasets de facebook

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Football Graph

```
Graph = nx.read_edgelist('Data/football.txt',create_using=nx.Graph(),nodetype=int)

print(nx.info(Graph))
```

```
Graph with 24 nodes and 30 edges
```

```
sp = nx.spring_layout(Graph)
plt.rcParams.update({'figure.figsize': (15, 8)})
nx.draw_networkx(Graph,pos=sp,node_color='blue',with_labels=False,edgecolors='black',node_size=100)
plt.show()
```

Figure 29:read data football

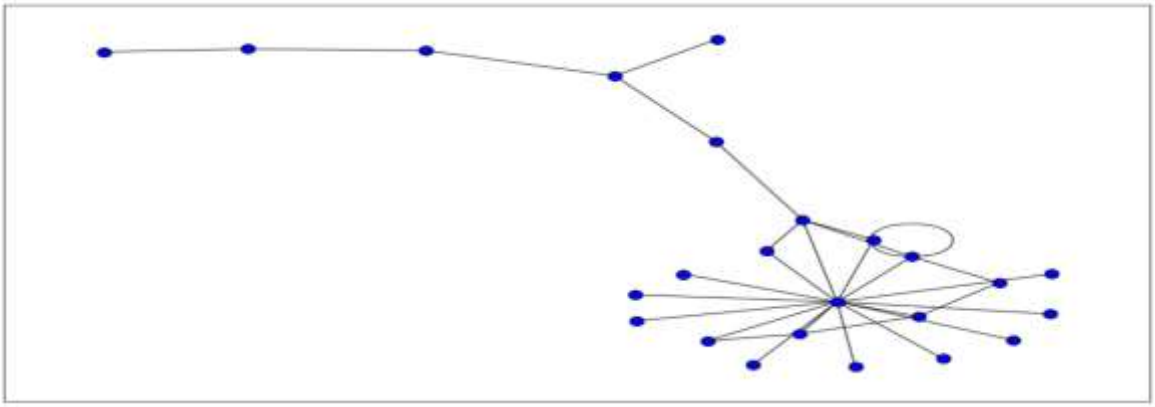

Figure 30:le graph de football

Centrality measures

```
print(nx.degree_centrality(Graph))
```

Afficher la sortie masquée

```
print(nx.betweenness_centrality(Graph))
```

Afficher la sortie masquée

```
print(nx.closeness_centrality(Graph))
```

Afficher la sortie masquée

```
print(nx.eigenvector_centrality(Graph))
```

Afficher la sortie masquée

Figure 31:les mesures de centralité de football

Build the Evaluation Matrix

```
betCent = nx.betweenness_centrality(Graph)
closCent = nx.closeness_centrality(Graph)
degCent = nx.degree_centrality(Graph)
eigCent = nx.eigenvector_centrality(Graph)


BC = []
DC = []
EC = []
CC = []
Node = []

for i in sorted(betCent):
    BC.append(betCent[i])
    Node.append(i)

for i in sorted(closCent):
    CC.append(closCent[i])

for i in sorted(degCent):
```

```
for i in sorted(degCent):
    DC.append(degCent[i])

for i in sorted(eigCent):
    EC.append(eigCent[i])


Matrix = pd.DataFrame({'Node' : Node,
                       'DC' : DC,
                       'CC' : CC,
                       'BC' : BC,
                       'EC' : EC})
Matrix
```

Figure 32:la matrice d'evaluation

	Node	DC	CC	BC	EC
0	1	0.130435	0.310811	0.312253	0.017444
1	2	0.217391	0.479167	0.411726	0.310617
2	3	0.130435	0.377049	0.003294	0.227318
3	4	0.043478	0.359375	0.000000	0.139215
4	5	0.086957	0.205357	0.086957	0.000926
5	6	0.086957	0.310811	0.001976	0.111667
6	7	0.043478	0.239583	0.000000	0.003820
7	8	0.086957	0.403509	0.000000	0.207221
8	9	0.086957	0.250000	0.166008	0.004023
9	10	0.695652	0.547619	0.772727	0.635868
10	11	0.043478	0.359375	0.000000	0.139215
11	12	0.086957	0.383333	0.355731	0.071826
12	16	0.086957	0.403509	0.000000	0.207221
13	17	0.043478	0.359375	0.000000	0.139215

Figure 33:matrice de decision

```
[ ]  # Save the Matrix as a csv file
     Matrix.to_csv("Data/Matrix.csv", index=False)
```

```
[ ]  # Convert Matrix to numpy array
     Matrix_arr= Matrix[['DC','BC','CC','EC']].to_numpy()

     Matrix_arr
```

Afficher la sortie masquée

Call Topsis class

```
[ ]  # Decision Matrix
     Evaluation_matrix = Matrix_arr
```

```
     weights = [0.2, 0.3 , 0.3, 0.2]

     criteria = np.array([True,True,True,True])

     T = Topsis(Evaluation_matrix,weights,criteria)
     T.calc()
```

```
Step 1
 [[1.30434783e-01 3.12252964e-01 3.10810811e-01 1.74435006e-02]
 [2.17391304e-01 4.11725955e-01 4.79166667e-01 3.10616505e-01]
 [1.30434783e-01 3.29380764e-03 3.77049180e-01 2.27318279e-01]
 [4.34782609e-02 0.00000000e+00 3.59375000e-01 1.39214832e-01]
 [8.69565217e-02 8.69565217e-02 2.05357143e-01 9.25773093e-04]
 [8.69565217e-02 1.97628458e-03 3.10810811e-01 1.11666792e-01]
 [4.34782609e-02 0.00000000e+00 2.39583333e-01 3.81964583e-03]
 [8.69565217e-02 0.00000000e+00 4.03508772e-01 2.07220785e-01]
 [8.69565217e-02 1.66007905e-01 2.50000000e-01 4.02264652e-03]
```

Figure 34:topsis class

Determin S+ and S-

```
[ ]  # S+ and S-
     print("best_distance\t", T.best_distance)
     print("worst_distance\t", T.worst_distance)
```

Afficher la sortie masquée

```
     print("worst_similarity\t", T.worst_similarity)
     print("rank_to_worst_similarity\t", T.rank_to_worst_similarity())
```

Afficher la sortie masquée

```
[ ]  print("best_similarity\t", T.best_similarity)
     print("rank_to_best_similarity\t", T.rank_to_best_similarity())
```

Afficher la sortie masquée

```
[ ]  #Convert S+ and S- to array numpy
     Best_Dist = np.array(T.best_distance)
```

```
#Convert S+ and S- to array numpy
Best_Dist = np.array(T.best_distance)
Worst_Dist = np.array(T.worst_distance)

#print(type(Best_Dist))
#print(len(Best_Dist))
```

Calculate Closness

```
Closnness = []
Closnness = Worst_Dist / (Worst_Dist + Best_Dist)
```

```
Closnness
#print(len(Closnness))
```

Afficher la sortie masquée

```
#Convert S+, S- and Closnness to DataFrame
Worst_Distance = pd.DataFrame(Worst_Dist,columns=['S-'])
Best_Distance = pd.DataFrame(Best_Dist,columns=['S+'])
Clos = pd.DataFrame(Closnness,columns=['C'])
```

```
# Concatenate results
Out = pd.concat([Best_Distance, Worst_Distance, Clos], axis=1)
Out
```

Figure 35:calcul de closeness

	S+	S-	C
0	0.231542	0.097571	0.296466
1	0.169613	0.151995	0.472609
2	0.278270	0.060852	0.179440
3	0.295418	0.042221	0.125048
4	0.285775	0.028304	0.090117
5	0.292580	0.034054	0.104257
6	0.308363	0.011522	0.036020
7	0.284985	0.057982	0.169061
8	0.267881	0.051895	0.162285
9	0.000000	0.310809	1.000000
10	0.295418	0.042221	0.125048
11	0.223102	0.112510	0.335238
12	0.284985	0.057982	0.169061
13	0.295418	0.042221	0.125048

```
#sort DataFrame by Closnness Column (Descending)
Out = Out.sort_values(by="C",ascending=False)
Out

# Save it into a csv file (out.csv)
Out.to_csv("Data/Out.csv",index=False)
```

```
#function return the index of the grather value(Closness vector) based on a descending order
def ranking_closness(data):
    return [i for i in data.argsort()[-len(data):][::-1]]
```

```
#Call the function Ranking_closness
C = ranking_closness(Closnness)
C
```

Afficher la sortie masquée

```
#retrieve the nodes that correspond to the indices returned by the function above
Tab = []
for i in range(len(C)):
        Tab.append(Matrix['Node'].loc[C[i]])
```

```
for i in range(len(C)):
        Tab.append(Matrix['Node'].loc[C[i]])

Tab
```

Afficher la sortie masquée

```
#Convert Tab above to DataFrame
Tab_Data=pd.DataFrame(Tab,columns=["Node"])

Tab_Data["Node"].to_csv("Data/Topsis.csv",index=False)
```

- The nodes depending on the relative closeness to the ideal solution

```
Final_Tab = pd.concat([pd.read_csv("Data/Out.csv"),Tab_Data],axis=1)

Final_Tab.head(10)
```

	S+	S-	C	Node
0	0.000000	0.310809	1.000000	10
1	0.169613	0.151995	0.472609	2
2	0.223102	0.112510	0.335238	12
3	0.231542	0.097571	0.296466	1
4	0.251492	0.085035	0.252684	35
5	0.278270	0.060852	0.179440	3
6	0.272943	0.059421	0.178784	24
7	0.284985	0.057982	0.169061	16
8	0.284985	0.057982	0.169061	8
9	0.267881	0.051895	0.162285	9

▾ User

```
[ ]  K = int(input("Enter the number of the top influential nodes:"))

     for i in range(K):
         print("The",i+1,"top influential node is ",Final_Tab["Node"][i])
```

```
Enter the number of the top influential nodes:5
The 1 top influential node is  10
The 2 top influential node is  2
The 3 top influential node is  12
The 4 top influential node is  1
The 5 top influential node is  35
```

Voici le model SI de football :

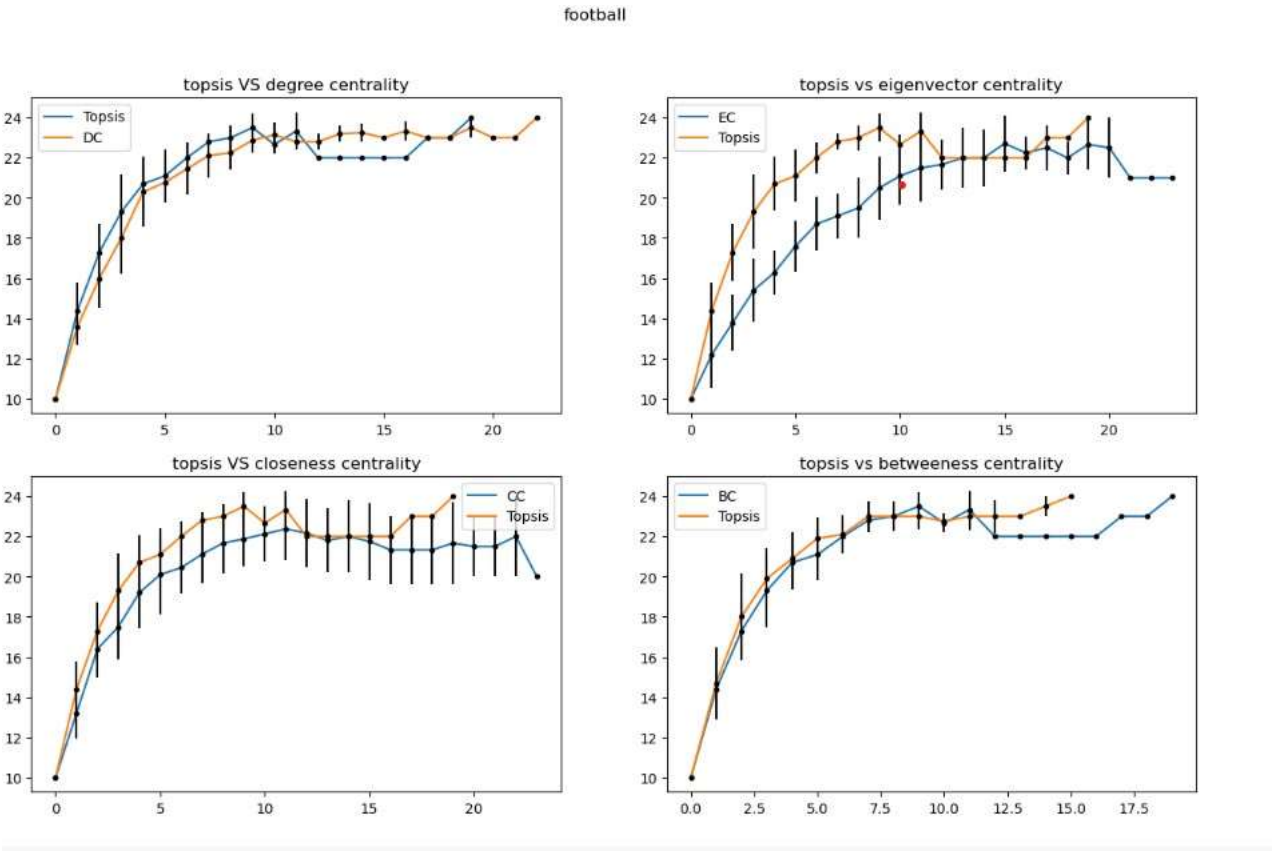

Figure 36:le model si de football

e- Application sur datasets Zachary :

Nous avons appliqué même démarche sur datasets de Zachary :

Voici le model SI de datasets zachary :

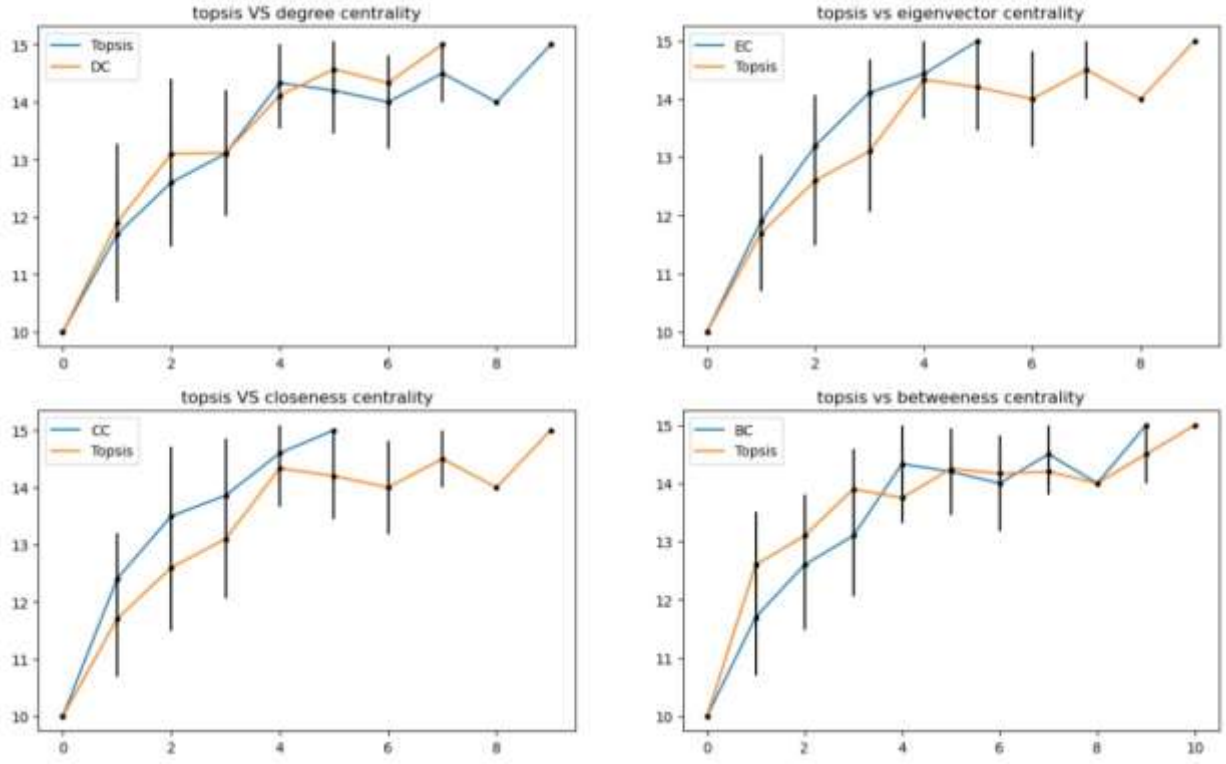

Figure 37:si zachary

Nous remarquons que topsis travaille très bien avec les datasets volumineux qui a beaucoup des nœuds.

f-kMeans :

Première étape : nous allons appliquer l'algorithme K Means à la matrice de décision qui contient les valeurs des mesures de centralité et la méthode proposée.

	DC	BC	CC	EC
0	0.085934	1.463059e-01	0.353343	3.391796e-05
1	0.004210	2.783274e-06	0.261376	6.045346e-07
2	0.002476	7.595021e-08	0.261258	2.233461e-07
3	0.004210	1.685066e-06	0.261376	6.635648e-07
4	0.002476	1.840332e-07	0.261258	2.236416e-07
...	...	...	...	...
4034	0.000495	0.000000e+00	0.183989	2.951270e-10
4035	0.000248	0.000000e+00	0.183980	2.912901e-10
4036	0.000495	0.000000e+00	0.183989	2.931223e-10
4037	0.000991	7.156847e-08	0.184005	2.989233e-10
4038	0.002229	6.338922e-07	0.184047	8.915175e-10

4039 rows × 4 columns

Figure 38:matrice de decision

Deuxième étape : initialisation de kmeans

	Node	DC	BC	CC	EC	S+	S-	C
0	107	0.258791	0.480518	0.459699	2.606940e-04	0.019029	0.200396	0.913277
1	1684	0.196137	0.337797	0.393606	7.164260e-06	0.061948	0.141539	0.695566
2	1912	0.186974	0.229295	0.350947	9.540696e-02	0.102563	0.100961	0.496063
3	3437	0.135463	0.236115	0.314413	9.531613e-08	0.103349	0.098846	0.488865
4	0	0.085934	0.146306	0.353343	3.391796e-05	0.140349	0.061395	0.304321
5	1085	0.016345	0.149015	0.357852	3.164082e-06	0.142845	0.060458	0.297379
6	698	0.016840	0.115330	0.271189	1.116876e-09	0.155715	0.046803	0.231106
7	567	0.015602	0.096310	0.328881	9.932295e-06	0.163087	0.039165	0.193646
8	58	0.002972	0.084360	0.397402	5.898120e-04	0.168330	0.034346	0.169464
9	428	0.028479	0.064309	0.394837	5.990065e-04	0.174851	0.026805	0.132923

Figure 39:les tops nodes influents

Comme l'étalement des centroïdes initiaux est considéré comme un objectif louable, k-means poursuit ceci en attribuant les centroïdes initiaux à l'emplacement des nœuds influents sélectionnés dans la méthode proposée, puis en choisissant les centroïdes suivants parmi les autres points de

données basés sur une probabilité proportionnelle à la distance au carré d'un point donné le plus proche du centroïde existant.

Troisième étape : convertir Dataframe en tableau numpy

```
#convert it to numpy array
Y = inf10[["DC","BC","CC","EC"]]
Yrr = Y.to_numpy()
Yrr
```

Figure 40:convertir Dataframe en tableau numpy

Quatrième étape : Appliquer l'algorithme K Means et afficher les communautés

```
kmeans = KMeans(n_clusters=10, init=Yrr, max_iter=20000, n_init=10, random_state=0)
```

```
#Communities
label=kmeans.fit_predict(X)
for i in (label):
    print(i)
```

Figure 41:application de kmeans

Nous avons utilisé l'algorithme kmeans qui se trouve dans package sklean

- Nombre de clusters initialisés est 10
- Kmeans.fit_predict(X) est utilisé pour calculer les centres de clusters et prédire l'indice de cluster pour chaque échantillon où X représente notre matrice de décision.

Cinquième étape : Affichage des communautés.

```
gk = community.groupby('center')
gk.first()
```

center	Node	DC	BC	CC	EC
0	107	0.258791	0.480518	0.459699	2.606940e-04
1	1684	0.196137	0.337797	0.393606	7.164260e-06
2	1912	0.186974	0.229295	0.350947	9.540696e-02
3	3437	0.135463	0.236115	0.314413	9.531613e-08
4	0	0.085934	0.146306	0.353343	3.391796e-05
5	58	0.002972	0.084360	0.397402	5.898120e-04
6	3980	0.014611	0.024820	0.225448	4.722356e-08
7	1	0.004210	0.000003	0.261376	6.045346e-07
8	136	0.032937	0.026870	0.294186	3.894375e-03
9	34	0.001238	0.003602	0.303313	4.074185e-06

Sixième étape : Regroupez les communautés dans une liste, à l'intérieur de chaque liste, nous trouvons les nœuds appartiennent à une communauté.

```
comm = []
for i in range(10):
    comm.append(gk.get_group(i)['Node'].tolist())
```

Figure 42:la table de communautés

```
colors=['red','blue','yellow','green','lightblue','white','brown','grey','purple','pink','orange','gold','olive']
for i in range(10):
    nx.draw_networkx(Graph, pos=sp, nodelist=comm[i], node_color= colors[i],with_labels=False,node_size=70, font_size=8,
        edgecolors='black')
```

Figure 43:graphe de construction

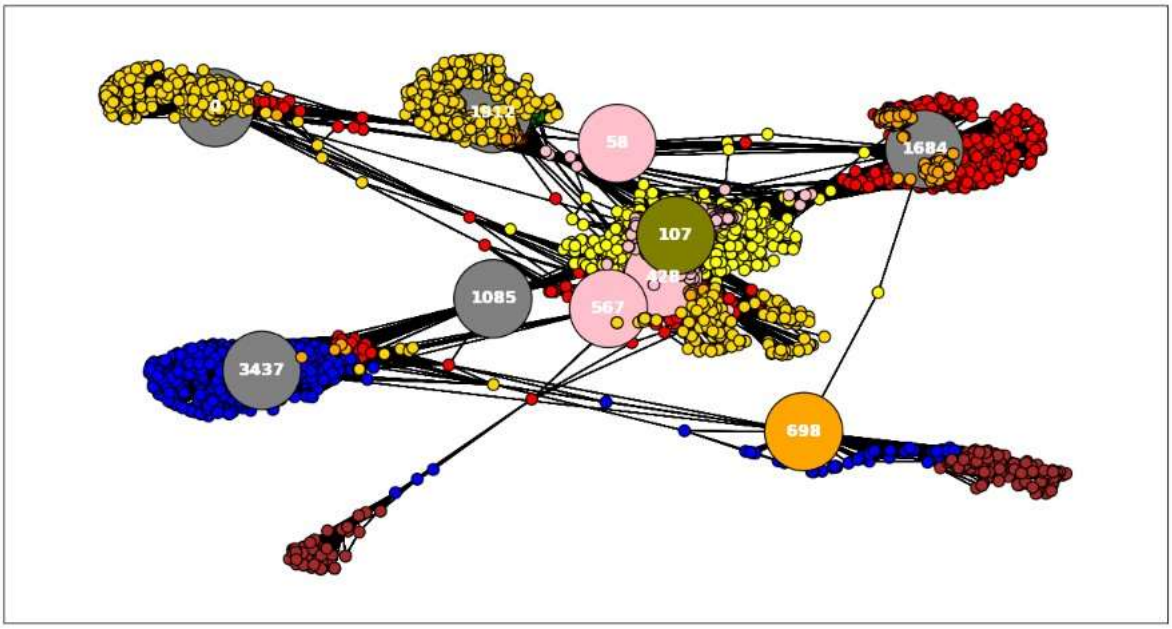

Figure 44:graphe kmeans

Visualiser les graphes des communité (Exemple communité 2)

et appliquer Topsis pour la communité 2

```
G2 = nx.read_edgelist('Data/comm2.txt',create_using=nx.Graph(),nodetype=int)

print(nx.info(G2))
```

```
Graph with 786 nodes and 4937 edges
C:\Users\DELL\AppData\Local\Temp\ipykernel_8752\1149726737.py:3: DeprecationWarning: info is deprecated and will be removed in version 3.0.

  print(nx.info(G2))
```

```
sp = nx.spring_layout(G2)
```

```
plt.rcParams.update({'figure.figsize': (15, 8)})
nx.draw_networkx(G2,pos=sp,node_color='blue',with_labels=False,edgecolors='black',node_size=100)
plt.show()
```

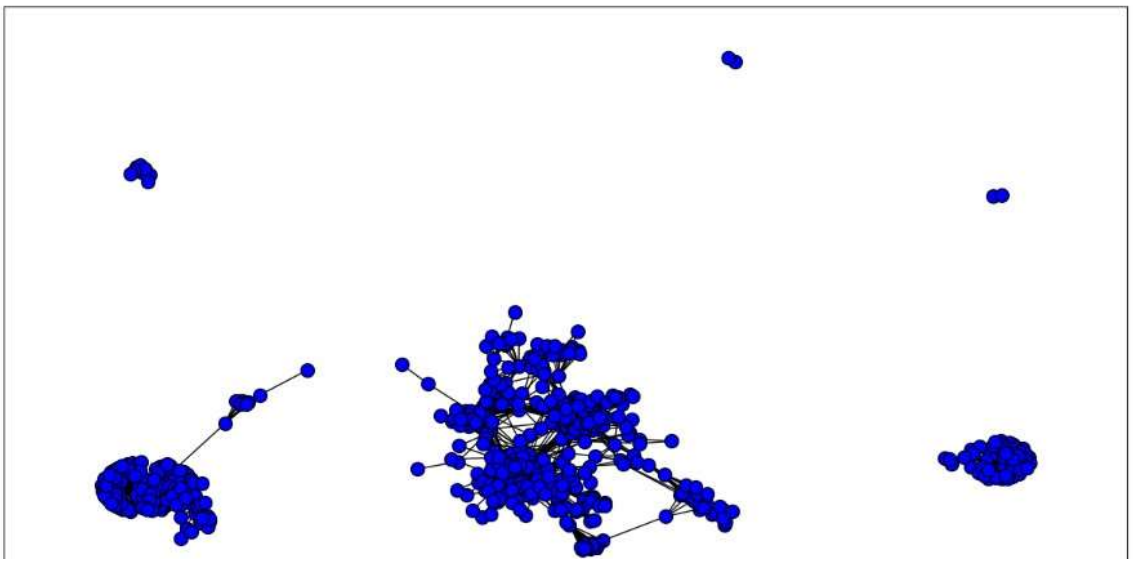

Figure 45:graphe de communautés 2

Nous remarquons que nous avons 3 couleurs dominantes dans le graphique. On a donc 3 principaux

Communautés Concernant les nœuds influents découverts par la méthode proposée.

Plot de communautés : la répartition des nœuds dans chaque communauté n'est pas bien répartie : les communautés 8, 7 et 9 contiennent la majorité des nœuds, tandis que les autres communautés se partagent le reste des nœuds avec de petits pourcentages.

```
plt.scatter(community['Node'], community['center'])
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

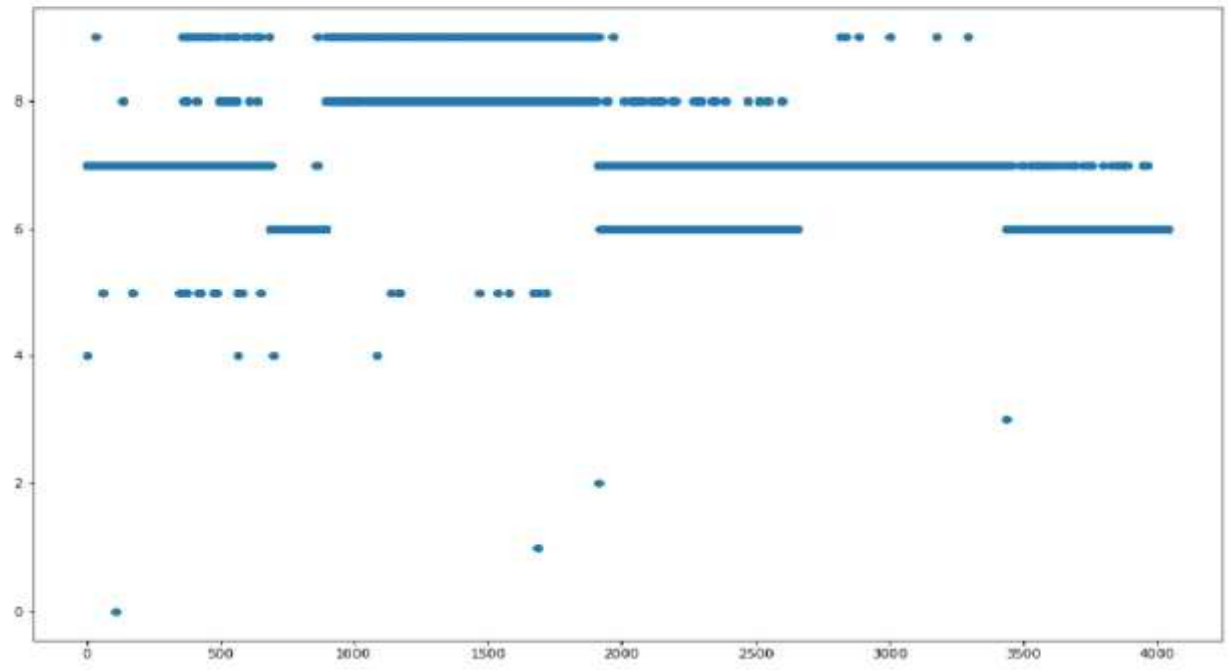

Figure 46:Diagramme des communautés K Moyennes basé sur la matrice de décision

Nous avons appliqué le model SI sur la communauté 2 :

```
[ ] import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np
```

```
[ ] import networkx as nx
    import ndlib.models.ModelConfig as mc
    import ndlib.models.epidemics as ep
```

Read Data

```
[ ] Graph = nx.read_edgelist('Data/comm2.txt',create_using=nx.Graph(),nodetype=int)
```

Read Decision Matrix

```
[ ] #Read our decision Matrix
    res = pd.read_csv("Data/resComm2.csv")
    res.head(10)
```

	Node	DC	CC	BC	EC	S+	S-	C
0	686	0.216561	0.218421	0.048057	2.701246e-07	0.038454	0.142088	0.787010
1	687	0.001274	0.115666	0.000000	1.216925e-08	0.047809	0.128745	0.729210
2	688	0.001274	0.115666	0.000000	1.216925e-08	0.056551	0.118178	0.676352
3	689	0.001274	0.115666	0.000000	1.216925e-08	0.079416	0.084373	0.515132
4	690	0.001274	0.115666	0.000000	1.216925e-08	0.091859	0.073181	0.443414
5	691	0.001274	0.115666	0.000000	1.216925e-08	0.095394	0.071124	0.427126
6	692	0.001274	0.115666	0.000000	1.216925e-08	0.101845	0.062259	0.379388
7	693	0.001274	0.115666	0.000000	1.216925e-08	0.106434	0.063799	0.374776
8	694	0.001274	0.115666	0.000000	1.216925e-08	0.103636	0.058976	0.362678
9	695	0.001274	0.115666	0.000000	1.216925e-08	0.112790	0.047034	0.294284

Read data

The top-10 ranked nodes by Topsis and Centrality measures

```
[ ] #The top-10 ranked nodes by degree centrality (DC)
    DCR = res[["Node", "DC"]]
    DCR = DCR.sort_values(by='DC', ascending=False)
    DCR.rename(columns = {'Node':'DCN'}, inplace=True)

    DCR["DCN"].to_csv("Data/DCRComm2.csv",index=False)
```

```
[ ] #The top-10 ranked nodes by betweenness centrality (BC)
    BCR = res[["Node", "BC"]]
    BCR = BCR.sort_values(by='BC', ascending=False)
    BCR.rename(columns = {'Node':'BCN'}, inplace=True)

    BCR["BCN"].to_csv("Data/BCRComm2.csv",index=False)
```

```
[ ] #The top-10 ranked nodes by closeness centrality (CC)
    CCR = res[["Node", "CC"]]
    CCR = CCR.sort_values(by='CC', ascending=False)
    CCR.rename(columns = {'Node':'CCN'}, inplace=True)

    CCR["CCN"].to_csv("Data/CCRComm2.csv",index=False)
```

```
[ ] #The top-10 ranked nodes by eigenvector centrality (EC)
    ECR = res[["Node", "EC"]]
    ECR = ECR.sort_values(by='EC', ascending=False)
    ECR.rename(columns = {'Node':'ECN'}, inplace=True)

    ECR["ECN"].to_csv("Data/ECRComm2.csv",index=False)
```

```
    fac = pd.concat([pd.read_csv("Data/DCRComm2.csv"),pd.read_csv("Data/BCRComm2.csv"), pd.read_csv("Data/CCRComm2.csv"), pd.read_csv("Data/ECRComm2.csv")],axis=1)
```

```
    #Concate the results
    comm2 = pd.concat([fac,pd.read_csv("Data/TopsisComm2.csv")],axis=1)

    comm2 = comm2.head(10)
    comm2
```

	DCN	BCN	CCN	ECN	Node
0	686	3604	3604	3545	3604
1	3545	3830	3545	3596	3830
2	3596	686	3830	3838	686
3	3604	3521	3521	3838	3521
4	3838	3705	3596	3604	3705
5	3830	3628	3869	3811	3546

Figure 47 :top ranked nodes

```
[ ]
   6   713  3918  3617  3488  3628
   7  3638  3545  3722  3804  3596
   8  3980  3722  3672  3790  3918
   9  3521  3566  3705  3824  3722
```

Application SI Model

```
def SI(nodes):
    n = 786
    Lst = []
    model = ep.SIModel(Graph)
    cfg = mc.Configuration()
    cfg.add_model_parameter('beta', 0.3)
    cfg.add_model_initial_configuration('Infected', nodes)
    model.set_initial_status(cfg)
    res = pd.DataFrame(columns=['iteration', 'Nb_Susceptible', 'Nb_infected'])
    for i in range(n):
        iteration = model.iteration()
        res.loc[len(res.index)] = [iteration['iteration'],iteration['node_count'][0], iteration['node_count'][1] ]
        if iteration['node_count'][1] == n:
            break

    Lst.append(res['iteration'].tolist())
    Lst.append(res['Nb_Susceptible'].tolist())
    Lst.append(res['Nb_infected'].tolist())

    return Lst
```

```
[ ] Rank_DC = set(comm2['DCN'])
    Rank_BC = set(comm2['BCN'])
    Rank_CC = set(comm2['CCN'])
    Rank_EC = set(comm2['ECN'])
    Rank_Topsis = set(comm2['Node'])
```

```
[ ] DClst = []
    BClst = []
    CClst = []
    EClst = []
    TOPSISlst = []
    for i in range(10):
        DClst.append(SI(Rank_DC))
        BClst.append(SI(Rank_BC))
        CClst.append(SI(Rank_CC))
        EClst.append(SI(Rank_EC))
        TOPSISlst.append(SI(Rank_Topsis))
```

Figure 48 : Application si model

```
[ ]  def Std_mean(lis):
        dc = {}
        sd = []
        mean = []
        for i in range(10):
            for j in range(len(lis[i][0])):
                dc[lis[i][0][j]]=[]
        for i in range(10):
            for j in range(len(lis[i][0])):
                dc[lis[i][0][j]].append(lis[i][2][j])
        for i in dc:
            sd.append(np.std(dc[i]))
            mean.append(np.mean(dc[i]))
        return mean, sd
```

```
[ ]  DCmean, DCsd = Std_mean(DClst)
     BCmean, BCsd = Std_mean(BClst)
     CCmean, CCsd = Std_mean(CClst)
     ECmean, ECsd = Std_mean(EClst)
     TOPSISmean, TOPSISsd = Std_mean(TOPSISlst)
```

Plot the results

```
fig, ax = plt.subplots(2, 2,figsize=(15, 9))
ax[0, 0].set_title('topsis VS degree centrality')
ax[0, 0].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[0, 0].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=TOPSISsd, fmt ='.k')
ax[0, 0].plot(list(range(len(DCmean))) , DCmean, label='DC')
ax[0, 0].errorbar(list(range(len(DCmean))), DCmean, yerr=DCsd, fmt ='.k')
ax[0, 0].legend()

ax[1, 0].set_title('topsis VS closeness centrality')
ax[1, 0].plot(list(range(len(CCmean))) , CCmean, label='CC')
ax[1, 0].errorbar(list(range(len(CCmean))) , CCmean, yerr=CCsd, fmt ='.k')
ax[1, 0].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[1, 0].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=TOPSISsd, fmt ='.k')
ax[1, 0].legend()

ax[0, 1].set_title('topsis vs eigenvector centrality')
ax[0, 1].plot(list(range(len(ECmean))) , ECmean, label='EC')
ax[0, 1].errorbar(list(range(len(ECmean))) , ECmean, yerr=ECsd, fmt ='.k')
ax[0, 1].plot(list(range(len(TOPSISmean))), TOPSISmean, label='Topsis')
ax[0, 1].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=TOPSISsd, fmt ='.k')
ax[0, 1].legend()

ax[1, 1].set_title('topsis vs betweeness centrality')
ax[1, 1].plot(list(range(len(TOPSISmean))), TOPSISmean, label='BC')
ax[1, 1].errorbar(list(range(len(TOPSISmean))), TOPSISmean, yerr=TOPSISsd, fmt ='.k')
ax[1, 1].plot(list(range(len(BCmean))) , BCmean, label='Topsis')
ax[1, 1].errorbar(list(range(len(BCmean))) , BCmean, yerr=BCsd, fmt ='.k')
```

Figure 49:plot les résultats

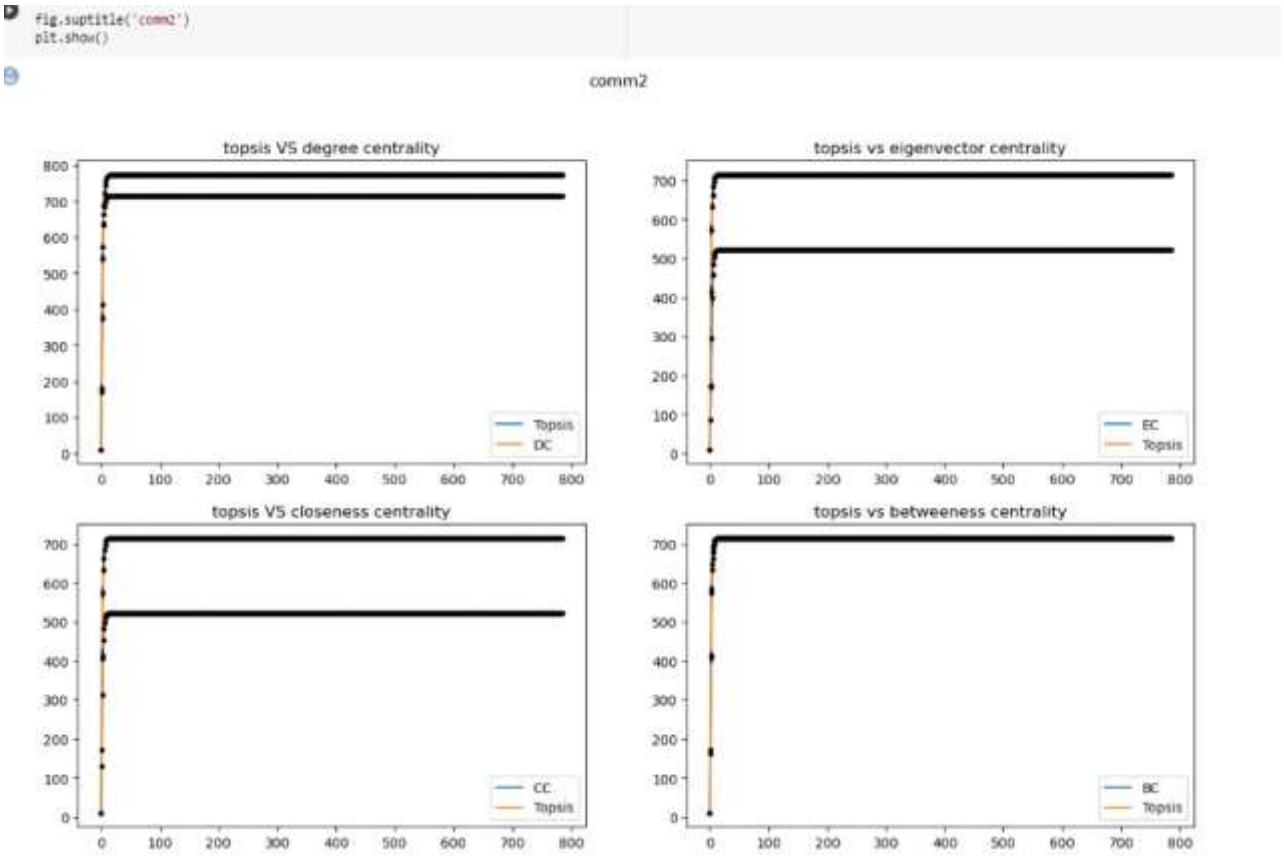

Figure 50:si model

Conclusion :

Dans le réseau, nous avons considéré plusieurs mesures de centralité différentes comme le multi-attribut de réseau complexe dans TOPSIS. Il est utilisé pour agréger les multi-attributs afin d'obtenir l'évaluation de l'importance du nœud de chaque nœud, qui ne se limite pas à une seule mesure de centralité, mais considère synthétiquement différentes mesures de centralité. Pour évaluer les performances, nous avons utilisé le modèle SI afin d'estimer l'influence de propagation des nœuds les mieux classés par différentes méthodes. Les résultats expérimentaux sur le réseau d'ego Facebook montrent que notre méthode peut bien identifier les nœuds influents. En comparaison avec d'autres mesures de centralité, la nouvelle méthode proposée fonctionne bien mieux que eigenvecor, assez similaire au betweness et presque aussi bonne que la centralité d'intermédiaire et degré de centralité. L'expérience de l'ensemble de données sur l'ego de Facebook prouve que le KMeans dynamique a une valeur raisonnable performances et peuvent convenir au traitement de données réseau complexes à grande échelle. Cependant, le choix de la meilleure méthode à utiliser dépend

du jeu de données choisi. En revanche, il est difficile de vérifier si le clustering est correct pour des données réseau complexes.