# Mini-Projet – Systèmes de recommandation sociale
# Graph Mining & Data management

BENBASSOU Abdelhak
AMMARI Elias

2022-2023

# Table des matières

Performances de Node2Vec et Adamic Adar

Avantages:

Limites:

# Ce qu'on a acquis :

- Comprendre, analyser un modèle de recommandation à partir d'un article scientifique
- Proposer une extension de modèle de système de recommandation
- Expérimenter, interpréter et analyser les résultats d'un modèle de système de recommandation,

# Important : Ce qui a été implémenté

- Analyse statistique et Descriptive sous Python ( Dossier Analyse → fichier analyse.py)
- Analyse statistique et Descriptive sous Gephi
- La notion Node2Vec : Dossier utility → (fichiers node2vec.py et main_node2vec.py)
- La notion de similarité de Adamic Addar : Dossier model → (fichiers socreg_BENBASSOU_AMMARI.py qui contient la similatié de Adamic Addar ET Node2Vec)
- L'approche basé sur les moyennes implémentées en Bonus ( fichier (fichiers socreg_Bonus_BENBASSOU_AMMARI.py)
- Création des embeddings Node2Vec (dossier embeddings)

# Brèves descriptions:

## Epinions

L'ensemble de données Epinions est un ensemble de données en ligne qui contient des avis et des critiques écrits par des utilisateurs sur une variété de produits et de services. Il a été créé en 2003 par la société Epinions, qui a ensuite été rachetée par eBay en 2003.

Cet ensemble de données comprend des informations sur les produits, les utilisateurs et les avis eux-mêmes. Les informations sur les produits incluent

des détails tels que le nom du produit, sa catégorie, sa marque et sa description. Les informations sur les utilisateurs incluent des détails tels que leur nom d'utilisateur, leur lieu de résidence et leur activité sur le site. Les avis eux-mêmes incluent des informations telles que la date de publication, la note attribuée au produit et le texte de l'avis.

Cet ensemble de données est particulièrement utile pour les chercheurs en analyse de sentiments et en analyse des données, car il offre une grande quantité d'avis écrits sur une variété de produits et de services. Il peut également être utilisé pour l'analyse des tendances et l'analyse des données de consommation ainsi que pour les systèmes de recommandation comme le cas de notre projet.

## Delicious

Le jeu de données Delicious est un ensemble de données en ligne qui a été créé pour l'analyse des systèmes de recommandation. Il contient des informations sur les utilisateurs, les signets et les tags associés à ces signets. Le jeu de données a été créé à partir de l'historique de signets de l'utilisateur de Delicious, un service de signets en ligne qui a été lancé en 2003 et qui a été acheté par Yahoo! en 2005.

Les informations sur les utilisateurs incluent des détails tels que leur nom d'utilisateur, leur adresse e-mail et leur pays de résidence. Les informations sur les signets incluent des détails tels que l'URL du signet, la date de création du signet et la description du signet. Les informations sur les tags incluent des détails tels que les mots-clés associés au signet et le nombre de fois où chaque mot-clé a été utilisé.

Ce jeu de données est particulièrement utile pour les chercheurs en systèmes de recommandation, car il offre une grande quantité de données sur les signets des utilisateurs et les tags associés à ces signets. Il peut être utilisé pour l'analyse des tendances et l'analyse des données de consommation, ainsi que pour l'entraînement et l'évaluation des modèles de recommandation.

# Analyse Descriptive

Pour une analyse description complète, nous avons utilisé Gephi & Networkx.

# Gephi:

Gephi est un logiciel open-source de visualisation de graphes pour l'analyse des données. Il permet aux utilisateurs de charger, explorer, manipuler, analyser et visualiser des données de graphe. Il est spécialement conçu pour les graphes de grande taille et les données dynamiques. Gephi est utilisé dans de nombreux domaines tels que les réseaux sociaux, la biologie, les sciences des données et l'intelligence artificielle.

Gephi offre une variété d'outils pour l'analyse de graphes, notamment la détection de communautés, la centralité, les filtres, les statistiques, les métriques et les algorithmes de layout. Il permet également aux utilisateurs de personnaliser les visualisations en utilisant des thèmes, des styles et des mises en forme personnalisées. Les visualisations peuvent être exportées au format PNG, SVG, PDF, et GEXF.

Gephi est facile à utiliser avec une interface utilisateur intuitive et une documentation détaillée pour les utilisateurs débutants. Il est également extensible avec des plug-ins et des scripts pour les utilisateurs avancés. Il est compatible avec les systèmes d'exploitation Windows, MacOS et Linux.
En résumé, Gephi est un logiciel open-source de visualisation de graphes pour l'analyse des données. Il offre une variété d'outils pour l'analyse de graphes, une personnalisation des visualisations et une compatibilité avec différents systèmes d'exploitation. Il est utilisé dans de nombreux domaines et est facile à utiliser avec une interface utilisateur intuitive et une documentation détaillée.

# Networkx

Le package NetworkX est un module Python pour l'analyse de graphes. Il permet aux utilisateurs de créer, manipuler et étudier des graphes et des réseaux complexes. Il est utilisé dans de nombreux domaines tels que les réseaux sociaux, la biologie, les sciences des données et l'intelligence artificielle.

NetworkX offre une variété d'outils pour l'analyse de graphes, notamment la détection de communautés, la centralité, les filtres, les statistiques, les métriques et les algorithmes de parcours. Il supporte également les graphes dirigés et non dirigés, les graphes pondérés et non pondérés et les graphes multigraphiques.

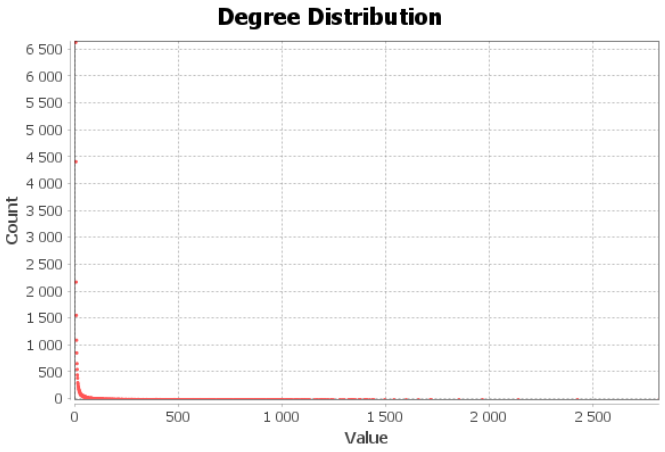
NetworkX est facile à utiliser avec une interface Python standard et une documentation détaillée pour les utilisateurs débutants. Il est également extensible avec des fonctionnalités supplémentaires pour les utilisateurs

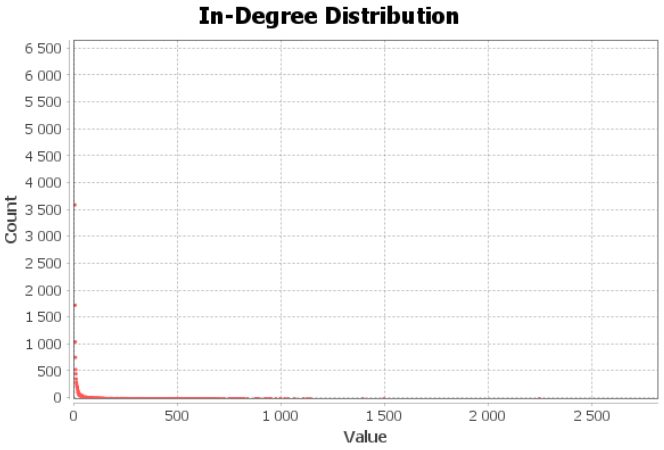
avancés. Il est compatible avec les systèmes d'exploitation Windows, MacOS et Linux.

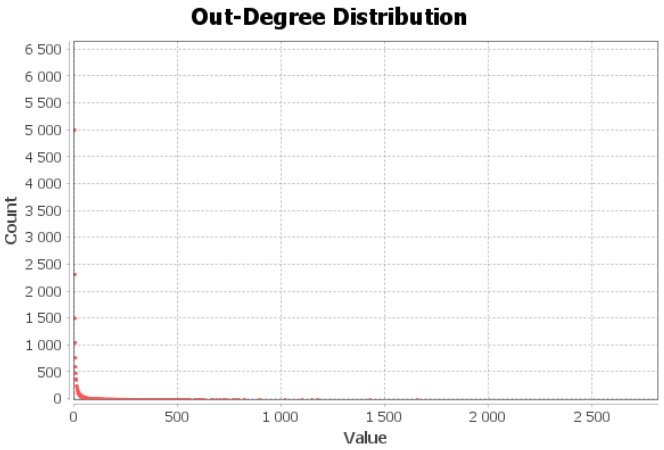
# Epinions Average Degree

Degree Report

Average Degree: 16,873



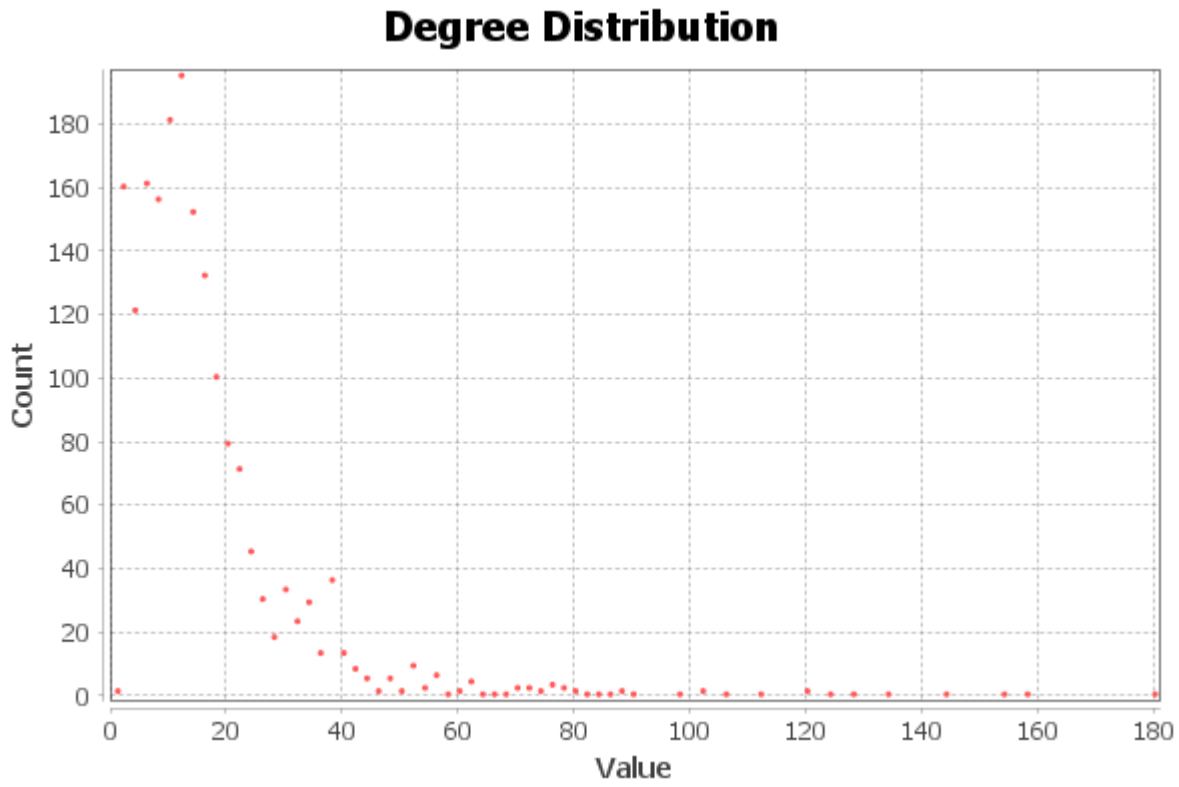




# Delicious Average Degree

Weighted Degree Report

Average Weighted Degree: 16,456




L'average degree est un indicateur de mesure de la densité d'un graphe. Il représente la moyenne des degrés de chaque nœud dans un graphe. Le degré d'un nœud est le nombre de liens ou d'arêtes qui le relient à d'autres nœuds. Pour un graphe non orienté comme delicious, le degré d'un nœud est égal au nombre de liens qui le relient à d'autres nœuds. Pour un graphe orienté, le degré d'un nœud est égal au nombre de liens entrants plus le nombre de liens sortants.

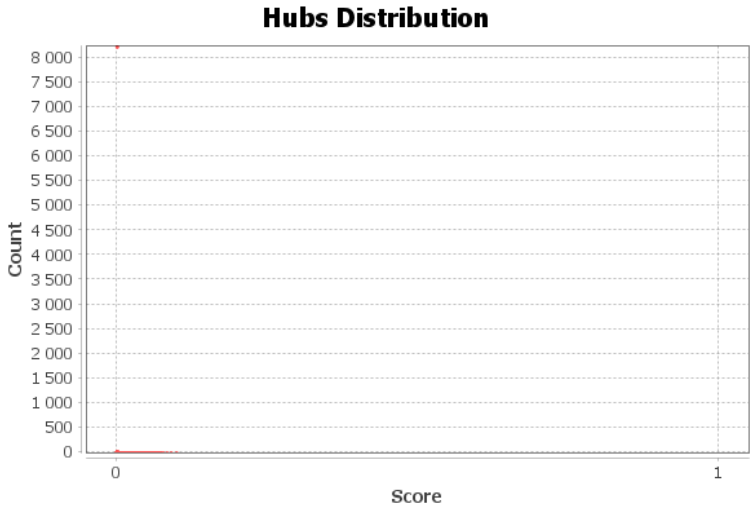
Dans notre analyse de graphe, vous avez obtenu une moyenne de degré de 16,873 pour Epinions. Cela signifie que la moyenne des degrés de tous les nœuds dans notre graphe est de 16,873. Cela signifie également que, en moyenne, chaque nœud est relié à environ 16,873 autres nœuds.

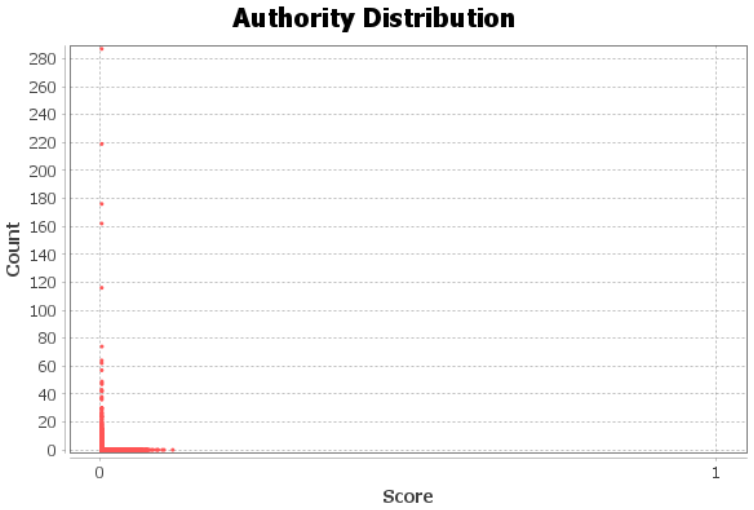
Cette valeur peut nous donner une idée de la densité de notre graphe. Si la moyenne des degrés est élevée, cela signifie que la plupart des nœuds sont reliés à de nombreux autres nœuds, ce qui indique une forte densité de liens dans le graphe. Si la moyenne des degrés est faible, cela signifie que la plupart des nœuds ne sont reliés qu'à peu d'autres nœuds, ce qui indique une faible densité de liens dans le graphe.

Il est important de noter que l'average degree n'est qu'une mesure statistique et ne peut pas être utilisée pour décrire de manière détaillée la structure globale de nos deux graphes. Il est nécessaire d'utiliser d'autres métriques pour une analyse plus approfondie de notre graphe tels que les Hits Metrics ou  les eigen centralities.

# Epinions Hits Metric Report

Parameters:
E = 1.0E-4
Results:

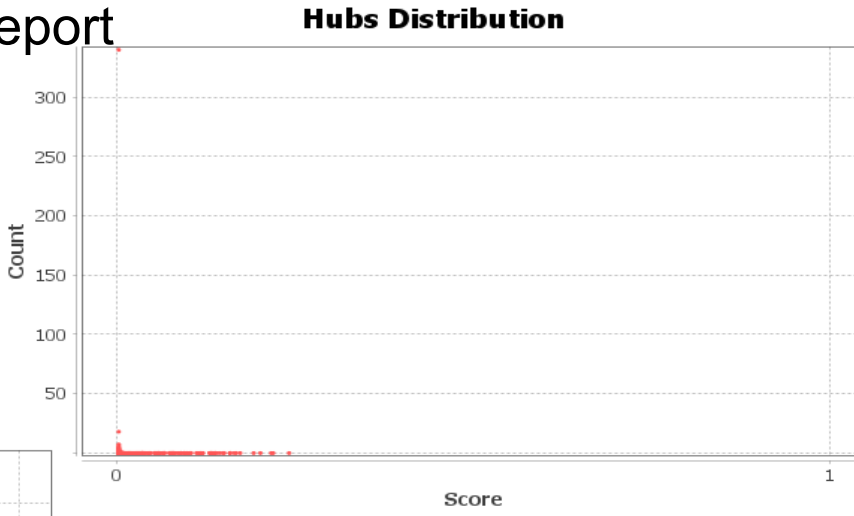


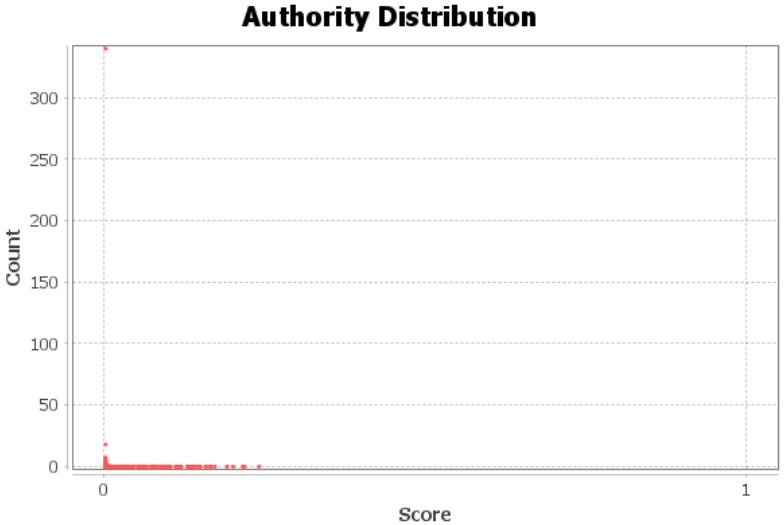




# Delicious Hits Metric Report

Parameters:
E = 1.0E-4

Results:



Algorithme utilisé:

Jon M. Kleinberg, *Authoritative Sources in a Hyperlinked Environment*, in Journal of the ACM 46 (5): 604–632 (1999)

Les distributions de hubs et d'autorités sont deux métriques utilisées pour l'analyse de réseaux.

La distribution de hubs se réfère à la répartition des nœuds dans un graphe qui ont un degré élevé. Les hubs sont des nœuds qui ont beaucoup de liens vers d'autres nœuds. Ils sont souvent considérés comme des points centraux dans un réseau, car ils ont un grand nombre de connexions. La distribution de hubs peut donner une idée de la connectivité globale d'un graphe et de la présence de points centraux importants.

La distribution d'autorités se réfère à la répartition des nœuds dans un graphe qui ont une forte influence sur les autres nœuds. Les autorités sont des nœuds qui ont un grand nombre de liens sortants vers d'autres nœuds. Ils sont souvent considérés comme des sources d'informations fiables et influentes dans un réseau. La distribution d'autorités peut donner une idée de la répartition de l'influence dans un graphe et de la présence de sources d'informations importantes.

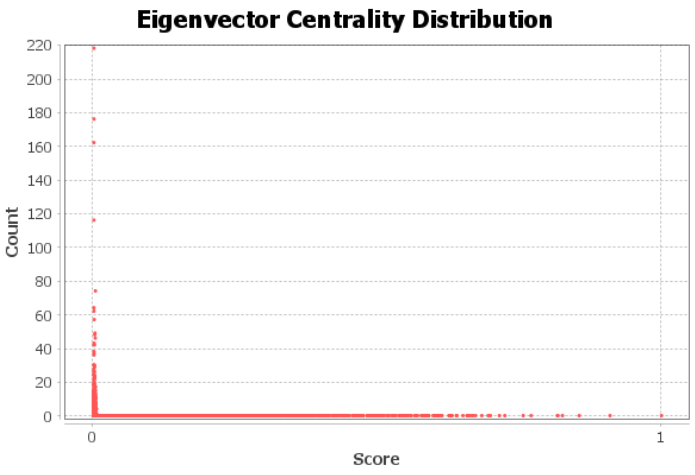
Dans notre cas, l'analyse des Hubs et Authority distribution indique la présence d'un nombre assez important de nœuds ayant beaucoup de liens vers d'autres  nœuds. Nos deux graphes contiennent des nœuds avec des degrés importants et des points centraux importants. Ceci confirme aussi l'average degree qui pivote autour de 16 pour nos deux graphes.

# Epinions  Eigenvector Centrality Distribution

Parameters:

Network Interpretation: directed
Number of iterations: 100
Sum change: 0.2048577906407468

Results:



# Delicious Eigenvector Centrality Distribution

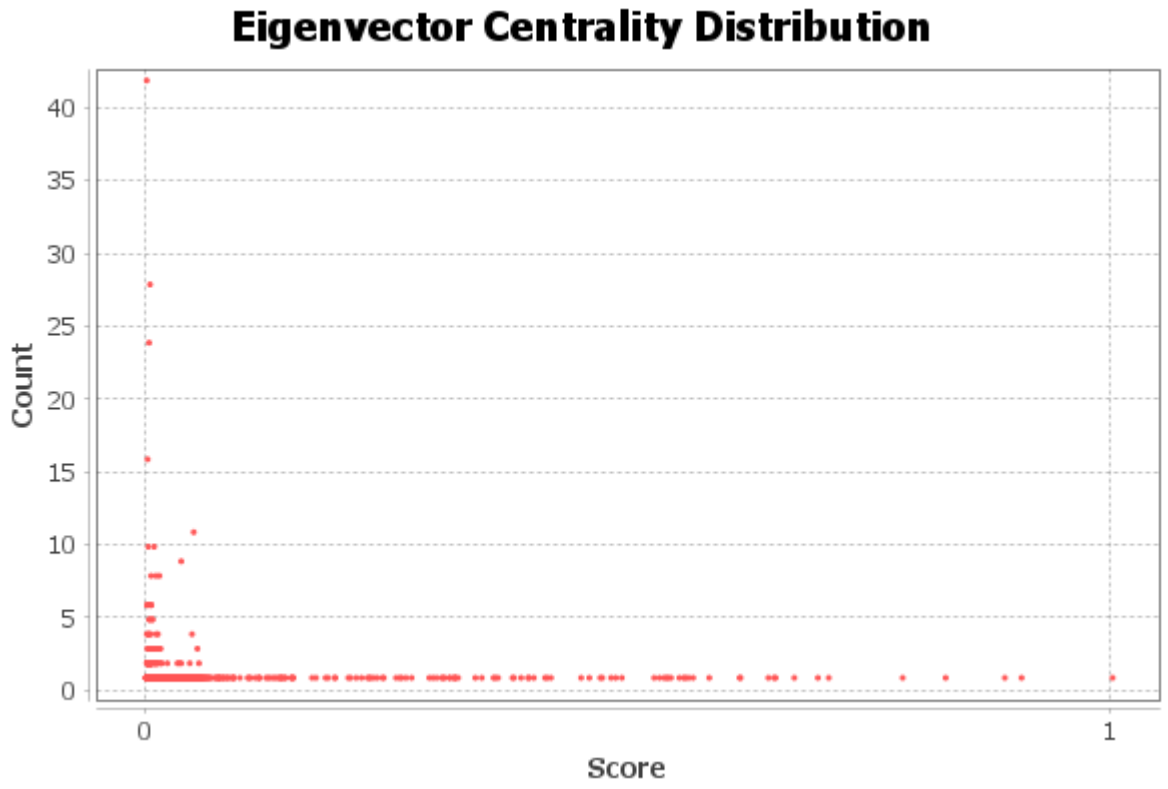
Parameters:

Network Interpretation: undirected

Number of iterations: 100

Sum change: 0.10754601299527061

Results:

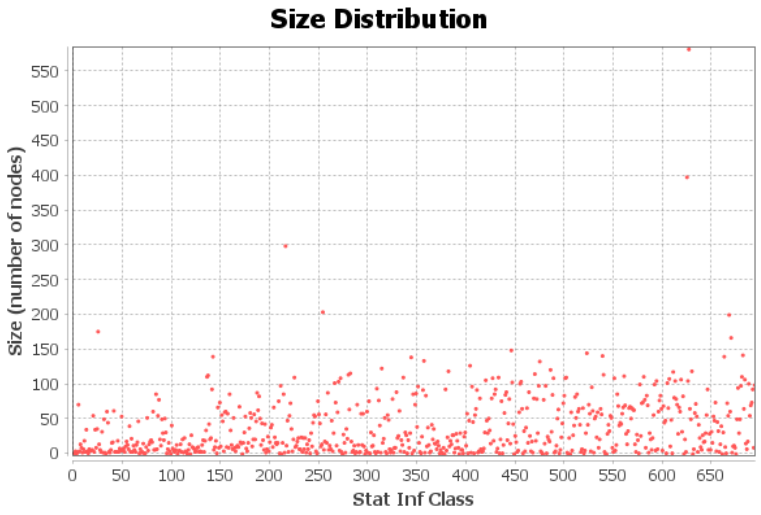



L'indice de centralité de vecteur propre (eigenvector centrality) est une mesure de l'importance d'un nœud dans un graphe en se basant sur les liens de ses voisins. Il permet de mettre en évidence les nœuds les plus importants dans un graphe en fonction de leur degré de centralité.

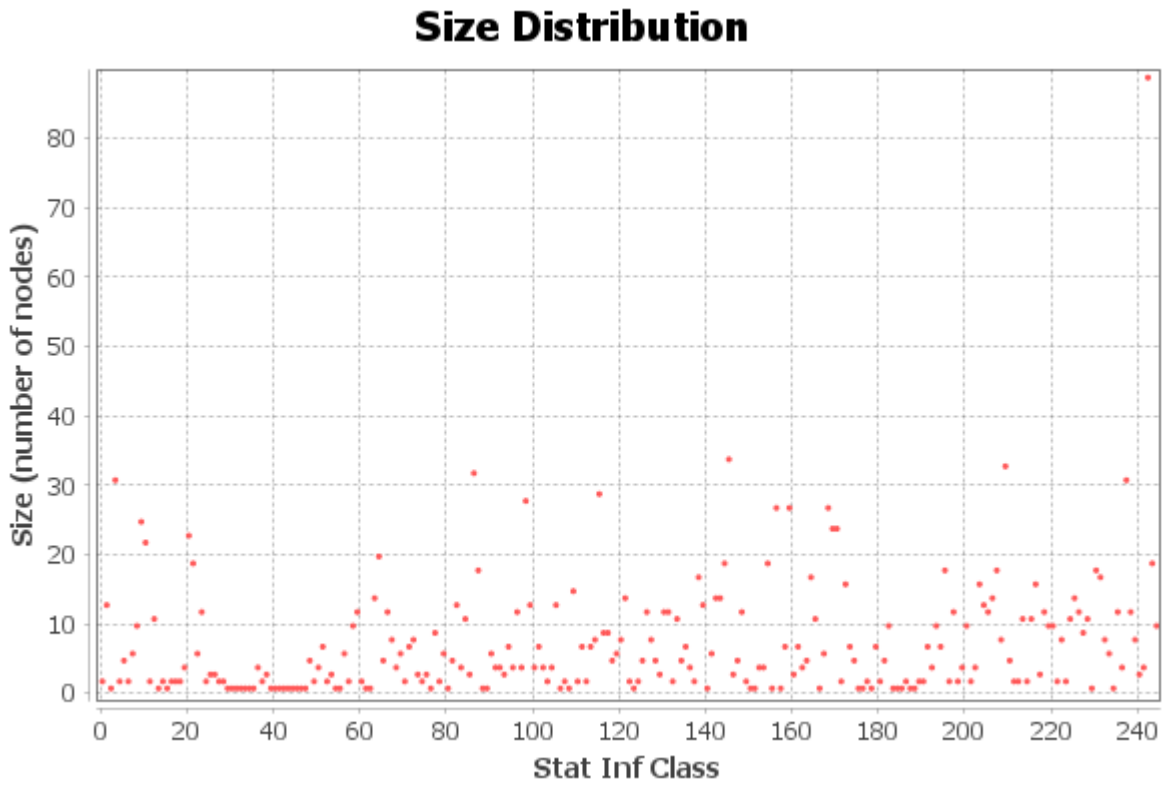
Il est important de noter que l'indice de centralité de vecteur propre est relatif à d'autres nœuds dans le graphe. Il peut varier considérablement d'un graphe à l'autre, même si les nœuds en question sont identiques. Il est donc important de l'interpréter en comparant les scores de centralité des différents nœuds dans le graphe

# Epinions Statistical Inference Report

Description Length: 1616558,400
Number of Communities: 694




# Delicious Statistical Inference Report




## Algorithm:

Statistical inference of assortative community structures
Lizhi Zhang, Tiago P. Peixoto
Phys. Rev. Research 2 043271 (2020)
https://dx.doi.org/10.1103/PhysRevResearch.2.043271

# Delicious Clustering Coefficient Metric

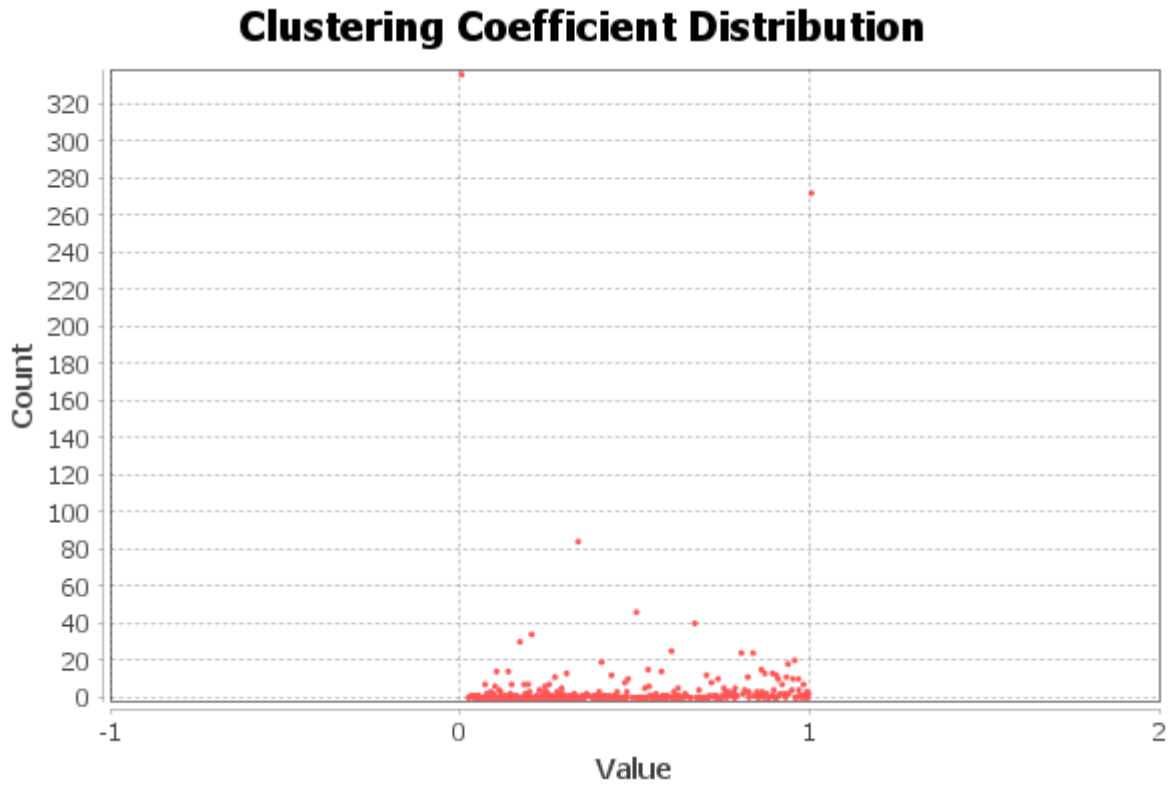
Parameters:

Network Interpretation: undirected

Results:

Average Clustering Coefficient: 0,536

Total triangles: 16272




Algorithm:

Matthieu Latapy, *Main-memory Triangle Computations for Very Large (Sparse (Power-Law)) Graphs*, in Theoretical Computer Science (TCS) 407 (1-3), pages 458-473, 2008

L'indice de coefficient de clustering moyen (CCM) est un indicateur de la tendance des nœuds d'un graphe à former des groupes ou des communautés. Il mesure le pourcentage de triangles qui existent parmi les nœuds d'un graphe par rapport au nombre total de triangles possibles. Plus précisément, il est calculé en divisant le nombre total de triangles dans un graphe par le nombre total de trios de nœuds (c'est-à-dire les combinaisons de trois nœuds distincts) dans ce graphe.

Dans notre analyse, le CCM pour le deuxième graphe (Delicious) est de 0,536, ce qui signifie qu'environ 53,6% des trios de nœuds dans ce graphe sont des triangles. Cela indique que les nœuds dans ce graphe tendent à former des groupes ou des communautés fortement connectés entre eux.

Le nombre total de triangles dans un graphe est un indicateur de la densité de ce graphe. Il mesure le nombre total de triangles dans un graphe, c'est-à-dire le nombre total de groupes de trois nœuds qui sont tous connectés entre eux. Dans notre analyse, le nombre total de triangles pour le premier graphe est de 16272, ce qui signifie qu'il y a 16272 groupes de trois nœuds qui sont tous connectés entre eux dans ce graphe.

.

# Encore plus de metrics

| Metric | Dataset | Valeur | Définition | Interprétation |
|---|---|---|---|---|
| Nombre de noeuds | Epinions | 26486 | | |
| | Delicious | 1863 | | |
| Nombre de edges | Epinions | 446906 | | |
| | Delicious | 7665 | | |
| Local Clustering Coefficient (LCC) | Epinions | 0.165 | Fractions of pairs of the node's friends are friends with each other | How likely a node's friends are connected to each other |
| | Delicious | 0.536 | | |
| Average local clustering coefficient: | Epinions | 0.165 | Measuring how likely the nodes are going to cluster on the whole network Average local clustering coefficient over all nodes in the graph. | Overall, what's the probability of adjacent nodes being connected. |
| | Delicious | 0.536 | | |
| Transitivity (number of triangles) | Epinions | 692365 | Ratio of number of triangles and number of "open triads" in a network | Overall, what's the probability of triangles formed over all the possible triangles in the network |
| | Delicious | 16272 | | |
| Average Distance (Average Shortest Path Length) | Epinions | 3.7609 | Measuring the length of the shortest path between a node A and a node B. the average shortest path of all the connected node pairs | On average, how far away the nodes are from each other. |
| | Delicious | 5.37 | | |
| Eccentricity | Epinions | | For a node n, eccentricity is the largest distance between n and all other nodes | For each node, what's the most far away distance to connect a node |
| | Delicious | | | |
| Radius | Epinions | 0 | For the entire network's eccentricity values, radius is the minimum eccentricity; (min of the all the nodes' largest distance between itself and all other nodes). A disconnected graph therefore has infinite radius. | To reach all the nodes, what's the shortest distance needed for existing nodes in the graph. It's like the radius of a circle but here it's the graph. |
| | Delicious | 1 | | |
| Diameter | Epinions | 11 | maximum shortest distance between any pair of nodes of the entire network | In the network, what's maximum shortest path distance to connect any 2 nodes |
| | Delicious | 16 | | |
| Center | Epinions | | set of nodes that have eccentricity = radius | : For a graph, what nodes are closest to other nodes and can reach all other nodes with shortest distances. |
| | Delicious | | | |
| Periphery | Epinions | | set of nodes that have | For a graph, what nodes are |

| | | | | |
|---|---|---|---|---|
| | Delicious | | eccentricity = diameter | farest away from other nodes and can reach other nodes with the longest shortest distance. |
| Strongly connected | Epinions | FALSE | for every pair nodes u and v,there is<br>a directed path from u to v and<br>a directed path from v to u. | Are all the nodes reachable from every other node in the network? |
| | Delicious | FALSE | | |
| Weakly connected | Epinions | TRUE | a directed graph is weakly connected if replacing all directed edges with undirected edges produces a connected undirected graph | If not considering the direction of the connections, is the graph is connected? |
| | Delicious | TRUE (63) | | |
| Density | Epinions | 0.001 | the ratio between the count of existing edges and all the possible edges' count | How many percent of the total possible edges are observed in a given network; help us understand how connected the network is compared to how connected it might be.<br>Second, when comparing two networks with the same number of nodes and the same type of relationships, it can tell us how the networks are different.<br>Graphs with large node and edge connectivity are more robust to the loss of nodes and edges |
| | Delicious | 0.004 | | |
| centralities eigenvector | Epinions | 0.2048 | Voir l'analyse descriptive | Voir l'analyse descriptive |
| | Delicious | 0.1075 | | |
| degré moyen | Epinions | 16.873 | Voir l'analyse descriptive | Voir l'analyse descriptive |
| | Delicious | 8.229 | | |
| degré moyen pondéré | Epinions | 18.351 | Voir l'analyse descriptive | Voir l'analyse descriptive |
| | Delicious | 16.456 | | |
| Nombre de communauté | Epinions | 694 | Voir l'analyse descriptive | Voir l'analyse descriptive |
| | Delicious | 245 | | |

# Evaluation SocReg

Le papier Recommender Systems with Social Regularization  a implémenté deux approches de régularisation sociale.

D'après notre compréhension, les avantages de l'approche de régulation sociale basée sur la moyenne (average-based regularization) sont les suivants:

1. Simplicité: Cette approche est simple à mettre en œuvre car elle ne nécessite pas de modèles complexes ou de grandes quantités de données pour fonctionner efficacement.

2. Précision: Les résultats de cette approche sont généralement considérés comme étant précis, car elle utilise des informations faciles à obtenir (notes moyennes des amis) pour réguler les prédictions.

3. Peut être utilisé en combinaison avec d'autres techniques: Cette approche peut être utilisée conjointement avec d'autres techniques de régulation sociale ou de filtrage collaboratif pour améliorer encore les résultats.

Les inconvénients de l'approche de régulation sociale basée sur la moyenne sont les suivants:

1. Limited to friends : Cette approche ne tient pas compte des relations sociales en dehors des amis directs, ce qui peut entraîner des recommandations moins précises pour les utilisateurs qui ont peu d'amis ou qui ont des relations sociales très éloignées.

2. Peut être sujet à une influence de groupe: Si les amis d'un utilisateur ont des opinions similaires, cela peut entraîner des recommandations qui reflètent uniquement ces opinions, plutôt que les préférences individuelles de l'utilisateur.

3. Difficulté à gérer les données manquantes: Si des données sur les notes moyennes des amis d'un utilisateur sont manquantes, cela peut entraîner des résultats moins précis pour cet utilisateur.

En opposition à cette approche, la régularisation basée sur les individus est une méthode utilisée dans les systèmes de recommandation pour intégrer des informations sociales plus "individuelles" dans le processus de recommandation. Contrairement à l'approche de régularisation basée sur la moyenne, où les notes moyennes des amis ou des utilisateurs similaires d'un utilisateur sont utilisées pour régulariser les notes de cet utilisateur, cette approche prend en compte les notes individuelles de chaque utilisateur. Cela permet de mieux capturer les différences individuelles et les dynamiques

sociales complexes, mais il peut ne pas être aussi précis que l'approche basée sur la moyenne dans certaines situations.

Pour Douban, Epinions et Delicious, nous avons testé l'algorithme implémenté dans le papier, à savoir l'approche basée sur les individus.

| Dataset | Training | Metrics | SR2 pcc |
|---|---|---|---|
| Douban | 80% | MAE | 0.5786 |
| | | RMSE | 0.7175 |
| | 60% | MAE | 0.5931 |
| | | RMSE | 0.7375 |
| | 40% | MAE | 0.606 |
| | | RMSE | 0.7475 |
| Epinions | 90% | MAE | 0.8034 |
| | | RMSE | 1.0376 |
| | 80% | MAE | 0.8292 |
| | | RMSE | 1.1237 |
| Delicious | 90% | MAE | 0.7453 |
| | | RMSE | 0.834 |
| | 80% | MAE | 0.7523 |
| | | RMSE | 0.8423 |

# Interprétation

Pour les mêmes paramètre des modèles, nous obtenons des valeurs de MAE et RMSE légèrement différentes:

Il est possible d'avoir des valeurs différentes de MAE et RMSE pour le même jeu de données (Epinions et Douban) , les mêmes paramètres de modèle, la même implémentation et le même prétraitement sur différents ordinateurs. Cela peut être dû à des différences dans les performances des ordinateurs, comme la vitesse du processeur, la quantité de mémoire disponible, la configuration du système, etc. Ces différences peuvent entraîner des différences dans les temps d'exécution des calculs, ce qui peut entraîner des différences dans les

résultats finaux. Il peut également y avoir des différences dans les bibliothèques utilisées pour l'implémentation de l'algorithme, qui peuvent avoir des erreurs numériques différentes. Enfin, il y a des différences dans les versions des bibliothèques utilisées, qui peuvent entraîner des différences dans les résultats. Il est donc important de conserver des informations sur les configurations utilisées pour les calculs pour pouvoir les comparer correctement.

- Différences matérielles : Les différents ordinateurs peuvent avoir des puissances de traitement, des mémoires et des capacités de stockage différentes, ce qui peut affecter les performances du modèle.

- Différences logicielles : Les différents ordinateurs peuvent avoir des versions différentes du logiciel ou des bibliothèques utilisées pour exécuter le modèle, ce qui peut entraîner de légères variations dans les résultats.

- Différences de prétraitement des données : Si les étapes de prétraitement des données ne sont pas exactement les mêmes sur les deux ordinateurs, cela peut également affecter les résultats.

- Aléatoire : Certains modèles impliquent de l'aléatoire, comme l'initialisation aléatoire des poids dans les réseaux de neurones, cela peut entraîner des résultats différents lors de l'exécution du même modèle sur différents ordinateurs.

# Explications (Comparaison Delicious & Epinions)

Nous obtenons une performance nettement meilleure sur Delicious que sur Epinions. Il est possible que plusieurs facteurs contribuent à la différence de performance entre les deux ensembles de données. Voici quelques possibilités :

- L'ensemble de données Delicious peut avoir une structure de réseau plus dense (0.001 pour Epinions vs 0.004 pour Delicious) . Cela signifie qu'il y a plus d'arêtes reliant les nœuds dans l'ensemble de données Delicious, ce qui permet à l'algorithme de recommandation de trouver plus facilement des éléments similaires en fonction des connexions entre les utilisateurs.

- L'ensemble de données Delicious peut avoir des utilisateurs plus homogènes. Si les utilisateurs de l'ensemble de données Delicious ont tous des goûts similaires, il peut être plus facile pour l'algorithme de prédire les éléments qu'un utilisateur donné

aimera en fonction de ce que d'autres utilisateurs similaires ont aimé.

- L'ensemble de données Delicious peut contenir davantage d'éléments très bien notés par les utilisateurs. S'il y a plus d'éléments bien notés, l'algorithme disposera de plus de données pour faire des prédictions précises.

- L'ensemble de données Delicious est moins sparse (voir les coefficients de clustering sur le tableau d'analyse ) que l'ensemble de données Epinions. Si un ensemble de données est peu dense, c'est-à-dire s'il contient de nombreuses entrées vides, il peut être plus difficile pour l'algorithme de faire des prédictions précises.

- L'ensemble de données Delicious peut présenter moins de bruit que l'ensemble de données Epinions. Le bruit désigne des données aléatoires ou non pertinentes qui peuvent avoir un impact négatif sur les performances de l'algorithme.

# Extension

## Première Similarité: Node2Vec

### Un peu de théorie

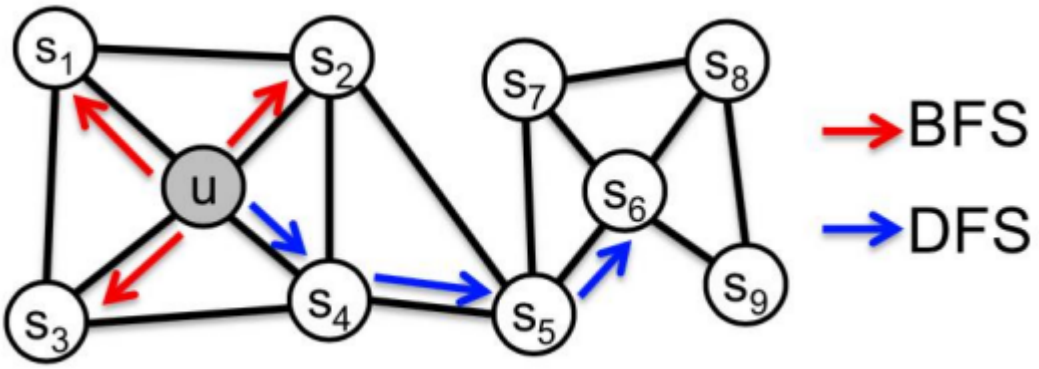



*Figure 1 BFS and DFS search strategies*

Les deux propriétés principales qui sont souhaitables à capturer dans un graphe sont l'homophilie et l'équivalence structurelle. L'homophilie se réfère à la propriété selon laquelle les nœuds similaires et appartenant à des grappes de réseaux similaires ont des plongements similaires. Dans la figure précédente, la propriété d'homophilie peut être vue pour les nœuds u, s1, s2, s3 et s4. L'équivalence structurelle se réfère à la propriété selon laquelle les nœuds ayant des rôles similaires ont des propriétés structurelles similaires dans le graphe.
Dans la figure 1, la propriété d'équivalence structurelle peut être vue pour les nœuds u et s6 qui jouent tous les deux le rôle de hubs dans le réseau. La prochaine section explore comment ces deux propriétés peuvent être capturées dans les plongements de nœuds (embeddings).

## Sampling Strategies

La stratégie de parcours aléatoire pour node2vec intègre deux techniques d'échantillonnage : la recherche en largeur (BFS) et la recherche en profondeur (DFS). La figure 1 montre à la fois les techniques d'échantillonnage BFS et DFS sur le graphe donné. Ces techniques d'échantillonnage sont utilisées pour échantillonner le voisinage d'un nœud donné u donné par Ns (u). La technique d'échantillonnage BFS a tendance à restreindre l'échantillonnage aux nœuds qui se trouvent dans les environs immédiats de la source. L'échantillonnage BFS à partir du nœud source u donne les voisins s1, s2 et s3 qui sont les voisins immédiats de u. L'échantillonnage DFS donne les nœuds qui se trouvent à des distances croissantes de la source. L'échantillonnage DFS démarrant à partir

du nœud u donne les nœuds s4, s5 et s6 qui sont séquentiellement à des distances croissantes du nœud source.

BFS capture le voisinage local du nœud, ce qui signifie qu'il capture l'équivalence structurelle entre les nœuds. DFS capture la relation entre les différents nœuds, ce qui signifie qu'il a tendance à trouver des ponts entre les différentes structures de communauté dans le réseau. BFS donne une vue micro du voisinage et DFS, d'autre part, donne une vue macro du voisinage. L'échantillonnage DFS amène le parcours à explorer de plus grandes parties du réseau en s'éloignant du nœud source. Un réseau a tendance à avoir un mélange d'homophilie et d'équivalence structurelle et il est donc important d'intégrer les deux techniques d'échantillonnage BFS et DFS pour générer des parcours aléatoires. La prochaine sous-section parle de la façon dont les deux techniques peuvent être utilisées pour introduire un biais dans la génération de parcours aléatoires.

# Random walk

La génération de parcours aléatoires décrite ci-dessous interpole entre l'échantillonnage BFS et DFS pour générer un contexte. Laissons ci désigner le ième nœud dans le parcours, en commençant par c0 = u. L'échantillonnage des nœuds utilise une distribution de probabilité donnée par :

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

où πvx est la probabilité de transition non normalisée entre les nœuds v et x et Z est la constante de normalisation. La probabilité de transition est calculée par πvx = αpq (t, x) · wvx où

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

Ici, wvx est le poids de l'arête entre les nœuds v et x. Pour les graphes non pondérés, wvx aura une valeur par défaut de 1,0. αpq (t, x) est appelé le biais de

recherche entre les nœuds t et x avec des paramètres p et q qui sont appelés paramètre de retour et paramètre In-out respectivement. dtx représente la distance la plus courte entre les nœuds t et x. Notez que la valeur de dtx est limitée à {0, 1, 2} ce qui limite le nombre de paramètres à deux. Les paramètres p et q sont utilisés pour interpoler entre les techniques d'échantillonnage BFS et DFS respectivement.

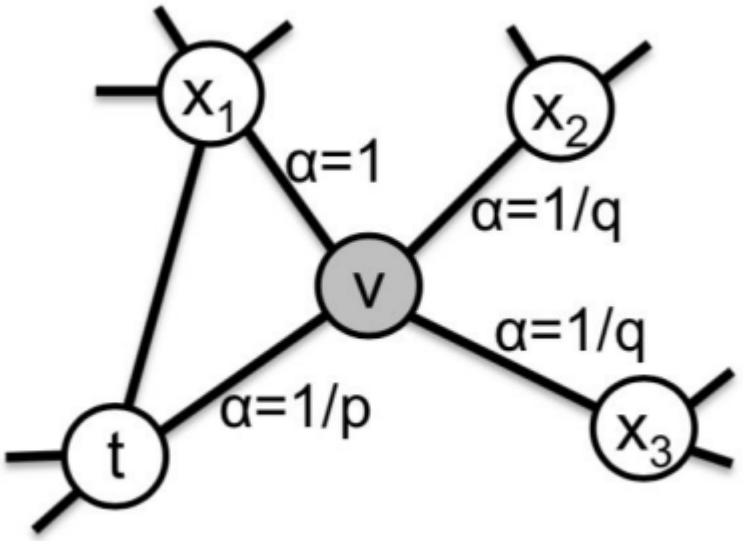



*Figure 2 Transition probabilities α for the next step after the walk rooted at t transitioned to v.*

Le paramètre de retour p est utilisé pour contrôler la probabilité que le parcours revisite le nœud précédent.
En fixant p à une valeur élevée (> max (q, 1)), le parcours sera moins susceptible de rééchantillonner un nœud déjà visité. Inversement, en fixant p à une valeur faible (< min (q, 1)), le parcours sera plus susceptible de retourner au nœud précédent, simulant ainsi une traversée BFS. En fixant q> 1, le parcours sera plus susceptible de rééchantillonner des nœuds proches du nœud source et en fixant q <1, le parcours rééchantillonnera des nœuds plus éloignés du nœud source, ce qui entraîne une exploration vers l'extérieur. Ainsi, q est utilisé pour simuler une traversée DFS. Les parcours aléatoires générés par cette stratégie sont ensuite alimentés en word2vec [1]. Cela traite les parcours aléatoires comme des phrases et génère des plongements pour chaque nœud dans le réseau.

# Méthodologie:

Tout d'abord : Installer les pré-requis: pip3 install -r req.txt

Nous avons codé 2 fichiers (Dossier utility):

## node2vec.py

permet de:
Compute transition probabilities for all the nodes. (2nd order Markov chain)

```python
@timer('Computing probabilities')
def compute_probabilities(self):

    G = self.graph
    for source_node in G.nodes():
        for current_node in G.neighbors(source_node):
            probs_ = list()
            for destination in G.neighbors(current_node):

                if source_node == destination:
                    prob_ = G[current_node][destination].get('weight',1) * (1/self.p)
                elif destination in G.neighbors(source_node):
                    prob_ = G[current_node][destination].get('weight',1)
                else:
                    prob_ = G[current_node][destination].get('weight',1) * (1/self.q)

                probs_.append(prob_)

            self.probs[source_node]['probabilities'][current_node] = probs_/np.sum(probs_)

    return
```

Generate biased walks based on probabilities

```python
def generate_random_walks(self):

    G = self.graph
    walks = list()
    for start_node in G.nodes():
        for i in range(self.max_walks):

            walk = [start_node]
            walk_options = list(G[start_node])
            if len(walk_options)==0:
                break
            first_step = np.random.choice(walk_options)
            walk.append(first_step)

            for k in range(self.walk_len-2):
                walk_options = list(G[walk[-1]])
                if len(walk_options)==0:
                    break
                probabilities = self.probs[walk[-2]]['probabilities'][walk[-1]]
                next_step = np.random.choice(walk_options, p=probabilities)
                walk.append(next_step)

            walks.append(walk)
    np.random.shuffle(walks)
    walks = [list(map(str,walk)) for walk in walks]

    return walks
```

Ce code définit une classe appelée Graph qui prend en entrée un graph (un graphe), des probabilités, p, q, le nombre maximum de chemins aléatoires à générer, la longueur de ces chemins et le nombre de travailleurs.
La classe contient également deux fonctions : compute_probabilities() et generate_random_walks().

La fonction compute_probabilities() calcule les probabilités de transition entre les différents noeuds du graphe en utilisant les paramètres p et q. Pour chaque noeud source, pour chaque noeud actuel qui est un voisin du noeud source, les probabilités de transition vers chaque destination (voisin du noeud actuel) sont calculées en utilisant la formule suivante:

- si source_node est égal à destination: probabilité = poids de l'arête (current_node, destination) * (1/p)
- si destination est un voisin de source_node: probabilité = poids de l'arête (current_node, destination)
- sinon: probabilité = poids de l'arête (current_node, destination) * (1/q)

Les probabilités sont ensuite normalisées de sorte qu'elles somment à 1 pour chaque noeud actuel.
La fonction generate_random_walks() génère des chemins aléatoires à partir de chaque noeud du graphe en utilisant les probabilités de transition calculées précédemment. Pour chaque noeud de départ, des chemins sont

générés jusqu'à ce que le nombre maximum de chemins soit atteint. Pour chaque étape suivante d'un chemin, le prochain noeud est sélectionné en utilisant les probabilités de transition calculées précédemment. Les chemins générés sont ensuite mélangés aléatoirement et retournés.

## main node2vec.py

→ permet de générer les embeddings

```python
@node2vec.timer('Generating embeddings')
def generate_embeddings(corpus, dimensions, window_size, num_workers, p, q, input_file, output_file):

    model = Word2Vec(corpus, vector_size=dimensions, window=window_size, min_count=0, sg=1, workers=num_workers)
    #model.wv.most_similar('1')
    w2v_emb = model.wv

    if output_file == None:
        import re
        file_name = re.split('[. /]', input_file)
        output_file = 'embeddings/' + file_name[-2] + '_embeddings_'+'dim-' + str(dimensions) + '_p-'+ str(p)+'_q-'+str(q)+'.txt'

    print('Saved embeddings at : ',output_file)
    w2v_emb.save_word2vec_format(output_file)

    return model, w2v_emb
```

Ce code définit une fonction nommée "generate_embeddings" qui utilise l'algorithme Word2Vec pour générer des embeddings à partir d'un corpus de mots (c'est-à-dire, une liste de listes de mots). Word2Vec est un algorithme de traitement automatique du langage qui permet de générer des vecteurs de mots (appelés embeddings) qui peuvent être utilisés pour des tâches telles que la classification de texte ou la détection de similarité.
La fonction prend plusieurs paramètres en entrée :

- corpus : la liste de listes de mots à partir de laquelle les embeddings doivent être générés.
- dimensions : le nombre de dimensions dans lesquelles les embeddings doivent être générés.
- window_size : la taille de la fenêtre utilisée pour déterminer les contextes dans lesquels les mots apparaissent (c'est-à-dire, les mots qui apparaissent dans les environs immédiats d'un mot donné).
- num_workers : le nombre de threads à utiliser pour générer les embeddings.
- p, q : les paramètres de l'algorithme de génération de marche aléatoire utilisé pour générer les corpus.
- input_file : le chemin vers le fichier d'entrée contenant les données.

- output_file : le chemin vers le fichier de sortie où les embeddings générés doivent être enregistrés.

Dans cette fonction, un objet de modèle Word2Vec est créé en utilisant les paramètres fournis. Le paramètre "sg" est fixé à 1 pour indiquer que le modèle utilisé est "skip-gram" (contrairement à "CBOW"). Les embeddings sont ensuite générés en utilisant le corpus fourni.
Ensuite, une variable "w2v_emb" est définie pour stocker les embeddings générés par le modèle. Si aucun nom de fichier de sortie n'est fourni, un nom de fichier de sortie est créé automatiquement en utilisant les paramètres fournis, le nom du fichier d'entrée et le répertoire "embeddings/".
Enfin, les embeddings sont enregistrés dans le fichier de sortie spécifié en utilisant la fonction "save_word2vec_format" de l'objet "w2v_emb" et le nom du fichier de sortie est affiché à l'écran. La fonction retourne également les objets "model" et "w2v_emb" pour permettre à l'utilisateur de les utiliser ultérieurement.

# Exécution

Pour exécuter avec les arguments par défaut:
python3 utility/main_node2vec.py

Notre code supporte plusieurs arguments lors de l'exécution, pour consulter la liste:
python3 src/main.py -h

```
usage: main.py [-h] [--input INPUT] [--output OUTPUT] [--p P] [--q Q]
               [--walks WALKS] [--length LENGTH] [--d D] [--window WINDOW]
               [--workers WORKERS] [--directed]

node2vec implementation

optional arguments:
  -h, --help          show this help message and exit
  --input INPUT       Path for input edgelist
  --output OUTPUT     Path for saving output embeddings
  --p P               Return parameter
  --q Q               In-out parameter
  --walks WALKS       Walks per node
  --length LENGTH     Length of each walk
  --d D               Dimension of output embeddings
  --window WINDOW     Window size for word2vec
  --workers WORKERS   Number of workers to assign for random walk and word2vec
  --directed          Flad to specify if graph is directed. Default is undirected.
```

Exemple d'exécution:

python3 src/main.py --p 0.4 --q 1 --walks 20 --length 80 --d 256

# Application de Node2Vec sur le Dataset Epinous

PS: Dimension = 128, walks = 10 , taille de chaque walk = 10

Exemple pour les noeuds 495 et 8

```
495
-0.12645127 -0.07638421 -0.042400215 0.30057606 -0.5519958 0.08388195 0.06808508 -0.20986712 -0.022341974 -0.34228596 0.2752651 -0.10055243 -0.27374822 -0.36514282
-0.21167974 -0.04414307 0.06210799 0.08323062 0.2718307 -0.19242026 0.115572475 -0.10228249 0.03220879 -0.18051492 0.15339713 0.012119505 -0.2592918 0.03588797
0.23655628 0.192855 0.15245758 -0.055986438 0.12699425 -0.17807966 -0.029023262 0.2782648 0.42723668 0.07592923 0.048288446 -0.1361606 -0.11836408 -0.32697788
0.096790664 0.03980337 0.017965028 -0.07065265 0.24608421 0.035676092 -0.066405326 -0.047477335 -0.092854805 -0.24679375 0.0075426074 -0.010879341 -0.21826637
0.3044349 -0.110380605 0.1472484 0.36088976 0.13716182 0.13805169 -0.15735863 -0.23244111 -0.14285506 -0.15951748 -0.13782501 -0.07133317 0.0885589 -0.016483271
0.04072008 0.23744279 0.077646926 -0.45505556 -0.0986511 0.05349847 0.25363976 0.16926043 0.081417724 -0.32008338 0.025960324 -0.11045421 0.15654865 0.21583183
0.08796368 -0.041561678 0.11153539 -0.33126256 -0.09335876 -0.26625568 -0.12144671 -0.02142585 0.11898937 -0.23450738 0.11681103 0.07079314 0.18020347 0.058392182
-0.166258 0.005075445 0.29788354 -0.09633925 0.04384161 0.3760815 -0.06909686 0.17919722 -0.17783006 0.19548453 0.011138315 0.2264241 -0.22813433 0.24238893
0.25252205 -0.26449907 0.152432 0.04891452 0.05592059 0.031043751 0.39209643 -0.054920174 -0.16800572 -0.012726394 -0.07829467 0.114105776 -0.06516698
-0.23134813 -0.19436118 -0.18424661 -0.12989934

8
 -0.3600062 -0.018566184 -0.0848674 -0.015747832 -0.42549527 0.015791925 -0.1715076 -0.3488328 0.13564788 0.037808433 0.13624671 -0.26820007 -0.25111154 -0.31212205
0.15009406 -0.036020882 0.15434349 0.23774086 -0.09738374 -0.05044802 -0.0073395777 0.14155355 -0.0025752706 -0.2320596 -0.09068746 0.11943921 -0.20843428 -0.0073186965
0.085467845 0.1143938 0.22113203 0.10478218 0.08245446 -0.10632979 0.061615895 0.037845325 -0.010053441 -0.018887758 0.062889874 -0.13512707 0.14076303 -0.08239479
0.20129296 -0.08392684 0.24187917 -0.11107248 0.08572399 0.005966464 0.22666751 -0.08199622 0.0044114217 -0.05084175 0.024878059 -0.14516564 -0.3058893 0.08285085
-0.048680063 0.30373597 0.027812669 -0.095383406 0.10001634 -0.09544286 -0.2548972 0.19157565 0.07061668 -0.1688805 -0.13263285 0.0020250939 0.09248785 0.05421301
0.038794465 -0.04937463 -0.22352678 -0.119955786 -0.022491347 -0.030851986 0.049383376 -0.021574654 -0.26742175 0.20253201 -0.15981652 -0.11425779 -0.01752947
-0.18890934 0.1910104 0.03482928 -0.18871784 0.34236118 0.06746785 0.10626693 -0.20170763 -0.07684856 0.033870317 -0.08515462 0.13715303 0.28193247 -0.00212165
-0.009503452 0.120369665 0.24357489 -0.09706571 -0.07176225 0.11795659 -0.012347241 -0.019880831 -0.20628548 0.16347227 0.0022092727 -0.07545794 -0.07858564
-0.10713502 0.15178612 0.008162965 0.012802446 0.044008072 -0.03384872 0.14404064 0.06093455 -0.062077563 -0.07293455 0.050184473 0.035729334 -0.3108152 0.026932009
-0.14633614 -0.27502155 -0.35847268 -0.07314369
```

# Deuxième Similarité: Adamic-Adar

```python
######## Adamic Adar  ##########

    Graph, init_probabilities = read_graph('data/trust_data.edgelist')
    def my_adamic_adar_index(self, Graph, u, v):
        return sum(1 / log(Graph.degree(w)) for w in nx.common_neighbors(Graph, u, v))


######## Adamic Adar ##########
```

Ce code définit une fonction appelée "my_adamic_adar_index" qui prend en entrée un graphe (appelé "Graph"), ainsi que deux noeuds (appelés "u" et "v") dans ce graphe. La fonction retourne un score de similarité basé sur l'indice d'Adamic-Adar.

L'indice d'Adamic-Adar mesure la similarité entre deux noeuds dans un graphe en utilisant les voisins communs qu'ils partagent. Plus précisément, il calcule la somme des termes 1/ln(degré(w)), où "w" est un voisin commun des noeuds "u" et "v" et "degree(w)" est le degré de ce voisin.

La fonction utilise la bibliothèque NetworkX pour obtenir les voisins communs des noeuds "u" et "v" en utilisant la fonction "common_neighbors (Graph, u, v)". Ensuite, elle utilise une compréhension de liste pour calculer la somme des termes 1/ln(degré(w)) pour chaque voisin commun. Le résultat final est retourné par la fonction.

Plus précisément, cette fonction mesure la similarité de deux noeuds dans un graphe en utilisant l'indice d'Adamic-Adar qui est basé sur les voisins communs qu'ils partagent. Il calcule la somme des termes 1/ln(degré(w)) où "w" est un voisin commun des noeuds "u" et "v" et "degree(w)" est le degré de ce voisin.

# Performances de Node2Vec et Adamic Adar

| Dataset | Training | Metrics | SR2 pcc | SR2 adamic adar | SR2 Node2Vec |
|---|---|---|---|---|---|
| Epinions | 90% | MAE | 0.8034 | 0.81332 | 0.741 |
| | | RMSE | 1.0376 | 1.15365 | 0.9326 |
| | 80% | MAE | 0.8292 | 0.8324 | 0.7491 |
| | | RMSE | 1.1237 | 1.1632 | 0.996 |
| Delicious | 90% | MAE | 0.7453 | 0.7521 | 0.7201 |
| | | RMSE | 0.834 | 0.8365 | 0.830 |
| | 80% | MAE | 0.7523 | 0.7526 | 0.732 |
| | | RMSE | 0.8423 | 0.862 | 0.839 |

Nous pouvons remarquer que la notion de Node2Vec implémenté a le mieux performé parmi les autres méthodes. Nous détaillerons les avantages et inconvénients de cette méthode dans la section suivante.

# Avantages:

Les avantages de l'algorithme node2vec incluent la capacité à capturer des propriétés structurales des graphes tout en gardant une flexibilité pour explorer différents types de voisinages. Cela permet de générer des vecteurs de noeuds qui peuvent être utilisés pour une variété de tâches telles que la classification de noeuds, la recommandation de noeuds et la détection de communautés.

Un des avantages clés de node2vec est sa capacité à capturer des propriétés de similarité entre les noeuds. En utilisant des marches aléatoires biaisées pour explorer les voisinages des noeuds, node2vec est capable de capturer des propriétés de similarité structurelle qui peuvent être perdues en utilisant des méthodes traditionnelles de calcul de similarité. Cela permet de générer des vecteurs de noeuds qui sont similaires pour les noeuds qui ont des structures de voisinage similaires, ce qui est utile pour des tâches telles que la recommandation de noeuds et la détection de communautés.

# Limites:

Les limites de l'algorithme node2vec incluent son utilisation de marches aléatoires pour explorer les graphes, ce qui peut entraîner des résultats imprécis si les paramètres de biais sont mal choisis. Il peut également être coûteux en termes de mémoire et de temps de calcul pour des graphes de grande taille. Enfin, il nécessite une certaine expertise pour choisir les bons paramètres et interpréter les résultats obtenus
Les limites de ce code peuvent être liées aux hypothèses faites pour la génération des random walks (biais p et q), qui peuvent ne pas être adaptées à tous les types de graphes. De plus, le choix des paramètres (taille de la fenêtre, nombre de worker) peut avoir un impact sur la qualité des embeddings générés. Il est donc important de les ajuster en fonction du graphe considéré et de la tâche visée. Enfin, cette approche nécessite une quantité importante de données de random walks pour obtenir des embeddings de qualité, ce qui peut être un frein pour des graphes très volumineux..

En résumé,  Il convient également de noter que la génération de parcours aléatoires peut facilement être parallélisée, de sorte que le modèle est évolutif pour les grandes réseaux. Node2vec fait un bon travail pour capturer les propriétés d'homophilie et d'équivalence structurelle en interpolant entre les techniques d'échantillonnage BFS et DFS. En plus de la classification multi-étiquette présentée ci-dessus, les auteurs originaux de node2vec ont

également adopté node2vec pour générer des plongements pour les arêtes. Ces plongements d'arêtes peuvent alors être utilisés pour des tâches de prédiction de lien. Node2vec a connu un succès dans diverses applications et a également inspiré d'autres modèles proposés par la suite, tels que struc2vec

# En Bonus : Approche basée sur les moyennes

(fichier socreg_Bonus_BENBASSOU_AMMARI.py)

Le code présenté est un exemple d'un système de recommandation social basé sur la régularisation basée sur la moyenne.

Il utilise une bibliothèque appelée MF pour créer un modèle de filtrage collaboratif basé sur la factorisation de matrices. Il utilise également une classe appelée TrustGetter pour obtenir les relations de confiance entre les utilisateurs. Il utilise des fonctions de similarité telles que pearson_sp, cosine_sp, JaccSim_HD et Adar pour calculer la similarité entre les utilisateurs.

Enfin, il utilise la génération d'Embedding pour générer des vecteurs d'utilisateurs. La régularisation basée sur la moyenne est utilisée pour tenir compte des relations de confiance entre les utilisateurs lors de la prédiction des notes d'évaluation.

| Dataset | Training | Metrics | SR2 pcc | SR1 pcc |
|---|---|---|---|---|
| Epinions | 90% | MAE | 0.8034 | 0.80332 |
| | | RMSE | 1.0376 | 1.04365 |
| | 80% | MAE | 0.8292 | 0.8124 |
| | | RMSE | 1.1237 | 1.1331 |
| Delicious | 90% | MAE | 0.7453 | 0.7532 |
| | | RMSE | 0.834 | 0.8423 |
| | 80% | MAE | 0.7523 | 0.7525 |
| | | RMSE | 0.8423 | 0.8536 |