Rapport de Mini-Projet Graphe

Systèmes de recommandation sociale

Auteurs

Valentin FRYDRYCHOWSKI
Rémi PETIOT
Nathan STIEVANO
Ludovic TUNCAY

30 janvier 2023

Table des matières

1 Introduction

Dans le cadre de ce projet, nous avons été chargés de manipuler le système de recommandation "SocRec", qui a été conçu pour démontrer les avantages de l'intégration d'informations sociales dans le processus de recommandation sur les réseaux sociaux. Les auteurs ont introduit des concepts innovants, tels que la "régularisation sociale", pour représenter l'influence des contraintes sociales sur les recommandations et ont proposé différentes méthodes pour les implémenter. Ces nouveaux concepts on permit a ce système de produire des résultats qui dépassaient l'état de l'art au moment de la publication de l'article. Dans le cadre de ce projet, nous allons étendre ce système pour améliorer ses performances. Ce rapport se divisera en trois parties principales : la première consistant en une analyse descriptive des réseaux sociaux Epinions et Delicious, la deuxième en une présentation des extensions que nous avons implémentées et enfin, la troisième en une présentation des résultats obtenus et une comparaison avec les résultats du système de recommandation original.
Notre code peut être retrouvé sur https://github.com/vfrydrychowski/ProjetGraph. Les explications pour exécuter le code se trouve dans le readme.

2 Statistiques et analyse descriptive

Dans cette analyse descriptive, nous allons examiner plusieurs propriétés des graphes des réseaux sociaux ***Epinions*** et ***Delicious***. Les données de Epinions peuvent être représentées à l'aide de deux sous-graphes.

1. Les ratings (les notes données par les utilisateurs)
2. Les trusts (le niveau de confiance entre les utilisateurs)

Pour ce qui est des données de Delicious, nous pouvons les decomposer en 3 sous-résaux :

1. Les favoris/ratings (basé sur le nombre de fois qu'un utilisateur a mis un lien en favori)
2. Les tags (les tags donnés par les utilisateurs)
3. Les trusts (le niveau de confiance entre les utilisateurs)

Dans notre cas, on cherche à prédire les favoris (ratings) et non pas les tags. On se sert alors seulement de 2 sous-réseaux, les ratings et les trusts. Pour que les mesures soient comparables, nous avons normalisé les ratings de Delicious pour que l'intervalle de valeurs passe de [1-70] à [1-5], l'intervalle de valeur de ratings de Epinions. Nous avons utlisé une formule dérivé de la normalisation min/max :

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} * 4 + 1 \tag{1}$$

L'avantage de cette normalisation est qu'elle permet de garder au maximum les relations de distance entre les données.

On peut retrouver un tableau récapitulatif de certaines statistiques pour ces deux réseaux sociaux à la Table 1.

Une autre manière d'étudier ces réseaux est d'étudier la distribution des degrés des noeuds. On retrouvera les histogrammes de ces distributions pour les réseaux Epinions et Delicious en Figure 1 et 2 respectivement.

Sur ces figures, on peut voir que dans Epinions, cette distribution suit particulièrement bien une powerlaw, au même titre que le sous-réseau trust de Delicious. Cependant, Dans le sous-graphe ratings de Delicious, on observe un pic proche de la valeur maximale des degrés. Il est possible que cela soit dû à une tendance des utilisateurs à mettre en favoris les mêmes liens. En conséquence, l'algorithme de prédiction des favoris pourrait surpasser les résultats obtenus sur le jeu de données Epinions car l'algorithme pourrait ne prédire que les types de liens les plus communs. Information que l'on ne retrouve pas dans Epinions.

Metrique	Epinions		Delicious	
	Rating	Trust	Rating	Trust
Nombre de noeuds	139738	49288	69870	1861
Nombre de liens	664825	487183	104799	15328
in-degree min	1	1	0	1
out-degree min	0	0	0	1
in-degree max	2026	2589	29	90
out-degree max	1023	1760	95	90
in-degree median	1	2	1	6
out-degree median	0	1	0	6
in/out-degree moyen	4.76	9.88	1.5	8.24
ecart type in-degree	20.02	40.10	1.06	8.16
ecart type out-degree	21.29	32.85	9.93	8.16
densité	0.00003	0.00020	0.00002	0.00443
Nombre de triangles (3-cliques)	99969	1928214	153	16272
degré clique max	6	6	3	3
closeness centrality min	0.167	0.108	0.10	0.09
closeness centrality max	0.367	0.405	1	1
closeness centrality moyen	0.238	0.254	0.17	0.33
closeness centrality median	0.236	0.254	0.17	0.21
betweenness centrality min	0	0	0	0
betweenness centrality max	1.17×10^9	6.21×10^7	2.51×10^6	1.98×10^5
betweenness centrality moyen	1.67×10^5	1.15×10^5	1.48×10^3	5.31×10^3
betweenness centrality median	0	19.8	0	300
PageRank min	4.93×10^{-6}	4.74×10^{-6}	1.40×10^{-5}	1.06×10^{-4}
PageRank max	1.17×10^{-3}	3.58×10^{-3}	9.11×10^{-5}	3.95×10^{-3}
PageRank moyen	7.16×10^{-6}	2.03×10^{-6}	1.43×10^{-5}	5.37×10^{-4}
PageRank median	5.17×10^{-6}	7.01×10^{-6}	1.42×10^{-5}	5.15×10^{-4}
clustering coef min	0	0	0	0
clustering coef max	1	1	1	1
clustering coef moyen	0.0032	0.1808	0.00111	0.48963
clustering coef median	0	0	0	0.46667

TABLE 1 – Statistiques des réseaux sociaux Epinions et Delicious

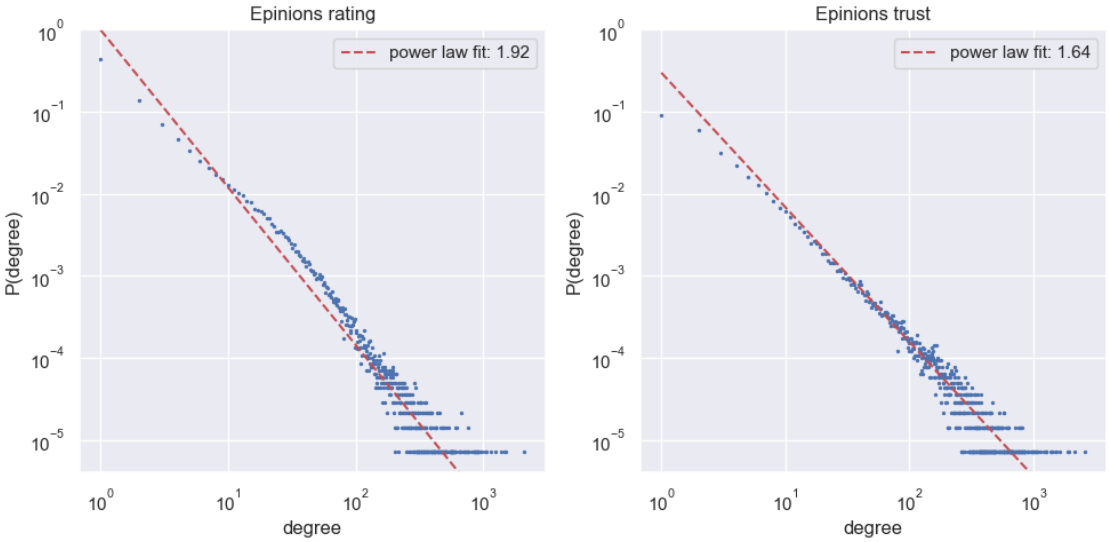

FIGURE 1 – Distribution des degrés des noeuds du réseau social Epinions

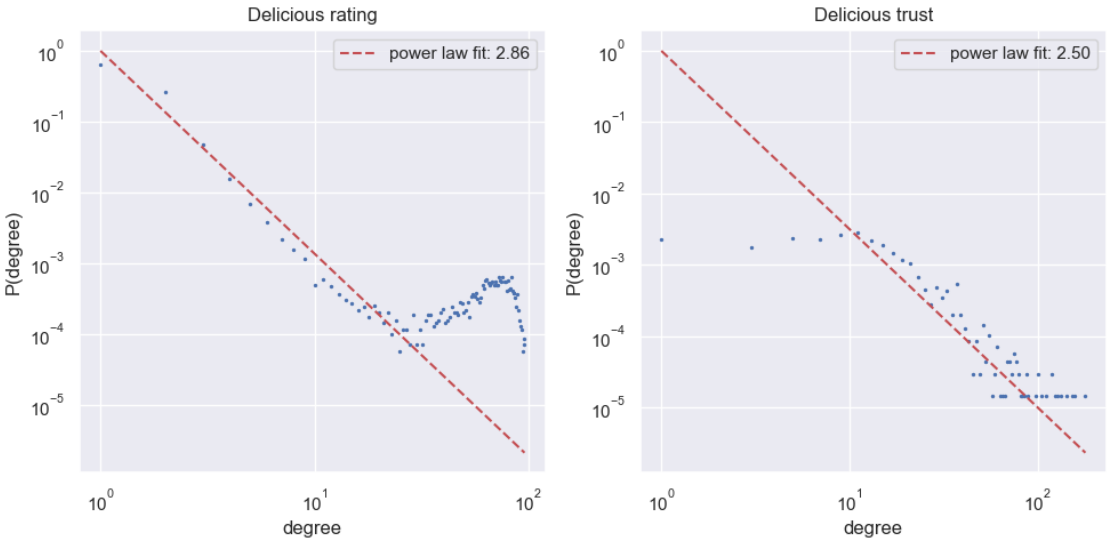

FIGURE 2 – Distribution des degrés des noeuds du réseau social Delicious

3 Extensions du modèle

Pour améliorer les performances du modèle de recommandation, nous avons entrepris une révision de la base algorithmique en passant d'une exécution séquentielle à une exécution parallèle et en créant des fichiers de configurations pour nous faciliter la tâche. Nous avons également exploré différentes méthodes de calcul de la similarité entre les utilisateurs. Cela incluait l'implémentation de la mesure de similarité de Jaccard et l'essai de deux extensions au modèle SR2pcc, à savoir la similarité entre les top-K utilisateurs d'une communauté et l'utilisation de l'algorithme Node2Vec.

3.1 Fichiers de configurations

Au cours des premiers tests de différents algorithmes sur le dépôt Git, nous avons identifié une inadéquation dans les fichiers de configuration. Plus précisément, un grand nombre de paramètres étaient codés de manière statique dans les algorithmes, tels que les chemins de données. En outre, il n'était pas possible d'utiliser plusieurs fichiers de configuration simultanément. Ce manque de flexibilité rendait difficile la collaboration sur différents ensembles de données. Par conséquent, nous avons procédé à des modifications sur les algorithmes de chargement de données et les algorithmes de prédiction afin d'intégrer correctement les hyperparamètres et les chemins de données précisés dans les fichiers de configuration. Ce processus était fastidieux en raison de la nécessité de comprendre une grande partie du code non-commenté.

3.2 Parallélisation

Nous avons constaté que les algorithmes de référence s'exécutaient lentement sur nos systèmes informatiques (parfois supérieur à vingt-quatre heures). Nous avons utilisé des validations croisées pour évaluer nos modèles, mais chaque évaluation devait être effectuée séquentiellement, alors qu'elle sont en réalité indépendantes. C'est pourquoi nous avons décidé de paralléliser cette partie du code pour accélérer son exécution. Nous avons divisé les données en k blocs, partagés entre les k processus parallèles. Nous avons utilisé le module ***multiprocessing***, qui nous a permis d'exécuter les différentes phases de la validation croisée en parallèle. Par conséquent, notre code s'exécute plus rapidement sur nos systèmes.

3.3 Jaccard Similarity

Une première idée que nous avons eu a été de modifier la fonction de similarité utilisée dans l'algorithme de social regularisation. En effet, les fonctions de similarité utilisée dans l'article sont la Vector Space Similarity (VSS) et le Pearson Correlation Coefficient (PCC). Nous avons voulu, en modifiant ces fonctions de similarité, comparer l'efficacité de celles-ci à la fonction de similarité de Jaccard.

Soit deux ensembles A et B, l'indice de Jaccard est :

$$J(A,B) = \frac{A \cap B}{A \cup B} \tag{2}$$

3.4 Top-K Similarity

La première des deux évolutions que nous avons décidé d'implémenter pour étendre le modèle est de ne pas prendre tout les liens sociaux mais plutôt de détecter les communautés et d'y prendre les top-k influenceurs.

Pour créer nos communautés, nous avons utilisé l'algorithme de Louvain qui est un algorithme de détection de communauté qui s'appuie sur la méthode de la modularité (mesure de la qualité de la partition d'un réseau en communautés) et qui est un des algorithme les plus rapide et efficace dans la détection de communautés. L'algorithme de Louvain implémente une approche itérative consistant à optimiser la modularité en assignant des noeuds à des communautés et en les fusionnant par la suite. On peut exprimer la modularité avec la formule suivante :

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j) \tag{3}$$

Avec A la matrice d'adjacence, m le nombre d'arêtes du graphe, k_i la somme des poids des arêtes connecté au noeud i, c_i la communauté attribuée à i et δ la fonction de Kronecker :

$$\delta(c_i, c_j) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon.} \end{cases}$$

L'algorithme de Louvain se compose de trois étapes :

- Initialisation : Les noeuds sont assignés à des communautés individuelles
- Boucle itérative :
  - Étape de recherche locale : Pour chaque noeud, le transfert du noeud à une autre communauté est évalué pour maximiser la modularité
  - Étape de regroupement : Les communautés sont fusionnées pour créer de nouveaux groupes
- Terminaison : L'algorithme se termine lorsqu'il n'est plus possible d'optimiser la modularité.

Une fois nos communautés crées, nous les avons intégrées dans la mesure de similarité. Pour être plus précis, dans l'algorithme de base la similarité se fait entre tous les followees d'un user pour chaque user. Or, nous avons trouvé plus intéressant de prendre seulement les top-k followees les plus influents (users qui sont le plus suivis) d'un user appartenant à la même communauté que ce dernier et ça pour chaque user.

3.5 Node2Vec

Dans le papier, on considère que deux users sont similaires s'ils ont les mêmes ratings. Ici nous proposons que 2 users soient similaires s'ils sont proches dans le graphe de trust, c'est-à-dire qu'ils se suivent mutuellement et/ou qu'ils suivent les mêmes personnes. Nous faisons donc un embedding du graphe de trust et nous appelons une mesure de similarité cosinus sur les vecteurs users obtenues.
Soit deux vecteurs A et B, on note la similarité cosine :

$$cos\,\theta = \frac{A \cdot B}{||A||\,||B||} \tag{4}$$

Pour l'implémentation de NodeToVec, nous utilisons le module python node2vec, un module qui permet de :

- effectuer une random walk depuis chaque noeuds d'un graphe
- transformer des séquences pour l'adapter au module word2vec
- lancer l'entrainement du modèle skip-gram pour apprendre le contexte d'un noeud dans un graphe
- fournir directement une similarité cosinus pour comparer 2 noeuds

Il est à noter que notre random walk se constitue de 100 marches aléatoires par noeud d'une longueur de 16 sauts. Ces deux paramètres à eux seuls peuvent changer l'efficacité de la similarité.

4 Résultats

4.1 Métriques

Dans leur article, les auteurs utilisent deux métriques pour évaluer leurs modèles : la Mean Absolute Error (MAE) et la Root Mean Square Error (RMSE). Nous avons décidé de conserver ces deux métriques pour pouvoir comparer nos résultats avec ceux de l'article.

La formule de la MAE est définie comme suit :

$$MAE = \frac{1}{T} \sum_{i,j} |R_{i,j} - \widehat{R}_{i,j}|$$

Où $R_{i,j}$ est la note (rating) réelle de l'utilisateur i pour l'item j, $\widehat{R}_{i,j}$ est le rating prédit par le système de recommandation pour l'utilisateur i et l'item j et enfin, T est le nombre de ratings sur lesquels on évalue notre modèle.

La formule de la RMSE est définie comme suit :

$$RMSE = \sqrt{\frac{1}{T} \sum_{i,j} \left(R_{i,j} - \widehat{R}_{i,j}\right)^2}$$

Dans les deux cas, les valeurs des métriques sont positives (ou nulles) et une valeur plus faible indique un modèle plus performant.

4.2 Méthodologie

Nous avons évalué les différents modèles en utilisant une validation croisée en $k = 5$ blocs. Les données du dataset ont été divisées en 80% pour l'entraînement et 20% pour l'évaluation, pour les datasets Epinions et Delicious. Pour réduire la variance de nos résultats, nous avons effectué 5 tests indépendants pour chaque méthode. Nous avons mené une recherche approfondie des paramètres optimaux, mais malheureusement nous n'avons pas pu reproduire les résultats du papier. Cependant, nous avons trouvé des valeurs assez proches sans pour autant les atteindre. Il y a plusieurs hypothèses qui pourraient expliquer cela, telles que l'utilisation de valeurs de paramètres non optimales (tous les paramètres n'étaient pas donné dans le papier), le code disponible différant de celui utilisé dans le papier, et une différence dans le choix des données d'entraînement et de test (nous avons choisi aléatoirement). Nous pensons que la dernière hypothèse est la plus probable, mais nous ne pouvons la vérifier. Par conséquent, nous avons comparé les résultats obtenus pour les extensions aux résultats que nous avons pu obtenir avec le modèle de base. Les résultats peuvent être consultés dans la Table 2 de la partie 4.3.

4.3 Résultats et analyse

Dataset	Métrique	SR2pcc-A	SR2pcc-R	SR2-Jaccard	SR2-Top-K	SR2-Node2Vec
Epinions	MAE	0.8443	0.8580	0.8582 (0.03% ↑)	0.8579 (0.01% ↓)	0.8519 (0.7% ↓)
	RMSE	1.0954	1.2671	1.2670 (0.00% ↓)	1.2670 (0.00% ↓)	1.2519 (1.2% ↓)
Delicious	MAE		0.6157	0.6156 (0.02% ↓)	0.6157 (0.00% =)	0.6122 (0.6% ↓)
	RMSE		0.5965	0.5965 (0.00% =)	0.5966 (0.02% ↑)	0.5901 (0.9% ↓)

TABLE 2 – Comparaison des résultats des différents modèles. SR2pcc-A pour les résultats présentés dans l'article, SR2pcc-R pour les résultats que nous avons produit nous même, SR2-Jaccard losque nous avons changé la similarité pour une similarité de Jaccard, SR2-Top-K quand nous avons sélectionné les top-K users de la communauté pour le calcul des similarités et enfin, SR2-Node2Vec lorsque nous avons utilisé Node2Vec.

Étant donné que nous n'avons pas réussi a reproduire les résultats du papier, nous avons décidé de comparer nos résultats avec ceux de SocRec (SR2pcc-R) que nous avons pu produire sur nos machines. Dans tous les cas, les différences sont assez, voire très, faibles. Cependant, nous avons souhaité vérifier la significativité de ces résultats résultats. Pour ce faire nous avons réalisé un test (bilatéral) de Student avec comme hypothèse nulle $H0$ l'égalité des moyenne des métriques et p-valeur seuil égale à 0.05 (une p-valeur inferieure à ce seuil nous permettrait de rejeter l'hypothèse $H0$). Nous n'observons pas de différences significatives entre le modèle de base (SR2pcc-A), le modèle utilisant une similarité de Jaccard (SR2-Jaccard) et celui utilisant les Top-K (SR2-Top-K). Cepdandant, on observe une différence significative lorsqu'on utilise Node2Vec pour nous aider au calcul des similarités et ce pour les deux datasets ainsi que pour les deux métriques. Nous pouvons alors affirmer que nous avons réussi à améliorer le système de recommandation

5 Conclusion

En conclusion, pour modéliser les systèmes de recommandation sociale de manière plus réaliste, nous avons réalisé un algorithme efficace pour identifier le groupe d'amis le plus approprié avec l'implémentation des top-k users. De plus, les fonctions de similarité jouent un rôle important pour distinguer les différents goûts des utilisateurs. Dans l'article "Recommender Systems with Social Regularization", les auteurs utilisent le PCC et le VSS traditionnels pour les calculs de similarité. À cela nous avons ajouté la similarité de Jaccard. Les améliorations apportées au système de recommandation, principalement l'implémentation de Node2Vec, ont eu un impact assez significatif sur la qualité des recommandations proposées. Grâce à l'utilisation de différents algorithmes les recommandations sont plus précises et plus adaptées aux préférences des utilisateurs. Pour améliorer notre système de recommandation, nous pourrions intégrer davantage d'informations contextuelles dans nos extensions provenant de beaucoup plus de données.