Projet Neo4j

À rendre : Une archive .zip comprenant :

- Un compte-rendu de projet au format **pdf** avec les réponses aux différentes questions (section 1.1 - Modélisation, section 2 - Interrogation, section 3.2 - Résultats des tests).
- Le code Python **commenté** produit pour les parties 1.2 et 3.1.

1 Cas d'étude – Modélisation

Le cas d'étude proposé concerne le site participatif d'avis Yelp décrit dans le document « Contexte d'étude - Analyse de données Yelp ». Globalement, des utilisateurs donnent leur avis sur des restaurants. Ces utilisateurs possèdent ou non des amis qui sont aussi des utilisateurs du site.

1.1 Besoins

Un grand groupe industriel vient d'implanter de nombreux restaurants thématiques dans la région du Delaware. Afin de les promouvoir et d'attirer un maximum de clientèle, ce groupe souhaite inviter dans chacun de ses retaurants des utilisateurs de la plateforme Yelp qui joueront ensuite le rôle d'influenceurs auprès de leurs amis : l'idée est que leurs amis deviennent ensuite des clients des restaurants. Il s'agit donc de trouver les utilisateurs les plus pertinents à inviter, qui en échange devront poster une review des restaurants afin « d'influencer » leurs amis.

Pour sélectionner ces utilisateurs-influenceurs pour chacun des restaurants, on se propose de calculer un score d'influence des utilisateurs de la plateforme. Soit r le restaurant pour lequel on veut trouver les bons influenceurs, et U l'ensemble des utilisateurs de la plateforme.

Étant donnés:

- v_r la ville de r
- $[a_r^1, a_r^2, ..., a_r^m]$ les m ambiances de r
- $[c_r^1, c_r^2, ..., c_r^n]$ les n catégories de r
- $pricerange(r)$ la catégorie de prix de r

l'objectif est de retourner une liste ordonnée $< (u_1, s_1), ...(u_k, s_k) >$ d'utilisateurs $u_i \in U$ avec $1 \leq i \leq k$ associés à leur score d'influence s_i (avec $1 \leq i \leq k$).

Le score s_i d'influence d'un utilisateur u_i pour r est calculé à partir de plusieurs facteurs décrits ci-après :

- facteur de centralité de l'utilisateur dans le réseau : $f_i^{centralite} = moyenne(s_i^1, s_i^2, s_i^3)$, avec :
  - s_i^1 le score social immédiat, calculé comme suit : $s_i^1 = \frac{|friends(u_i)|}{max_{u' \in U}|friends(u')|}$
    En d'autres termes, il s'agit du nombre d'amis de l'utilisateur u_i normalisé par le nombre maximal d'amis des utilisateurs de U.

- s_i^2 le score social de second niveau, calculé comme suit : $s_i^2 = \frac{|friends(friends((u_i))|}{max_{u' \in U}|friends(friends(u'))|}$
  En d'autres termes, il s'agit du nombre d'amis d'amis de l'utilisateur u_i normalisé par le nombre maximal d'amis d'amis des utilisateurs de U. Il s'agit de voir combien de personnes peuvent être touchées si un des amis de u_i devient client du restaurant et poste lui aussi une review.
- s_i^3 le score social intrinsèque, calculé comme suit: $s_i^3 = \frac{fans(u_i)}{max_{u' \in U} fans(u')}$
  Le nombre de fans de u_i correspond au champ $fans$ de la base de données Yelp : il s'agit du nombre de fois où cet utilisateur a été « liké » sur la plateforme.

- facteur de validité des commentaires $f_i^{validite} = moyenne(s_i^4, s_i^5)$, avec :
  - s_i^4 le score de validité des commentaires : $s_i^4 = \frac{\sum_{reviews(u_i)} isUseful}{review\_count(u_i)}$
    review_count est un champ de la base de données Yelp : c'est le nombre total de reviews postées par u_i[1]. *isUseful* vaut 1 si la review a été considérée comme utile (*useful*) au moins une fois. En d'autres termes, il s'agit de compter le nombre de reviews de l'utilisateur qui ont été considérées comme utiles au moins une fois.
  - s_i^5 le score de « coolitude » des commentaires : $s_i^5 = \frac{\sum_{reviews(u_i)} isCool}{review\_count(u_i)}$
    $isCool$ vaut 1 si la review a été considérée comme *cool* au moins une fois. En d'autres termes, il d'agit de compter le nombre de reviews de l'utilisateur qui ont été considérée comme cools au moins une fois.

- facteur d'adéquation au restaurant $f_i^{adequation} = moyenne(s_i^6, s_i^7, s_i^8)$, avec:
  - s_i^6 le score d'adéquation aux ambiances de r : $s_i^6 = \frac{\sum_{j=1}^{m} \frac{|reviewPos(j)|}{|reviews|}}{m}$
    m est le nombre d'ambiances du restaurant r , $|reviews|$ est le nombre de reviews faites par u_i à votre disposition, et $|reviewPos(j)|$ est le nombre de fois où l'utilisateur u_i a fait une review positive ($stars \geq 4$) pour un restaurant de l'ambiance j. En d'autre termes, on cherche à voir si u_i aime les restaurants proposant la ou les mêmes ambiances que r.
  - s_i^7 le score d'adéquation aux catégories de r : $s_i^7 = \frac{\sum_{j=1}^{n} \frac{|reviewPos(j)|}{|reviews|}}{n}$
    n est le nombre de catégories de r et $|reviewPos(j)|$ est le nombre de fois où l'utilisateur u_i a fait une review positive ($stars \geq 4$) pour un restaurant de la catégorie i. En d'autre termes, on cherche à voir si u_i aime les restaurants de la ou des mêmes catégories que r.
  - s_i^8 le score d'adéquation aux tarifs pratiqués par r : $s_i^8 = \frac{|reviewPos(pricerange(r)|}{|reviews|}$
    $pricerange(r)$ est la catégorie tarifaire de r et $|reviewPos(pricerange(r))|$ est le nombre de fois où l'utilisateur u_i a fait une review positive ($stars \geq 4$) pour un restaurant de cette catégorie tarifaire. En d'autre termes, on cherche à voir si u_i fréquente (et aime) des restaurants de la même catégorie tarifaire que r.

- facteur géographique $f_i^{geographique} = s_i^9$, avec $s_i^9 = \frac{|reviews(friends(u_i, v_r)|}{|reviews(friends(u_i)|}$
  avec $|reviews(friends(u_i, v_r)|$ le nombre de reviews des amis d'u_i concernant la ville v_r de r et $|reviews(friends(u_i)|$ le nombre total de reviews des amis d'u_i . En d'autres termes, il s'agit de savoir si les amis de u_i ont l'habitude de fréquenter des restaurants de la même ville que r.

Finalement, le score d'un utilisateur u_i est calculé comme suit:

$$s_i = \alpha \times f_i^{centralite} + \beta \times f_i^{validite} + \gamma \times f_i^{adequation} + \delta \times f_i^{geographique}$$

- Question - Proposez une modélisation permettant de couvrir ce cas d'étude et l'ensemble des requêtes de la section 2.

[1]Ce chiffre est supérieur au nombre de reviews $|reviews|$ avec lequel vous allez également travailler, car nous avons extrait un sous-ensemble de la base Yelp concernant le Delaware.

1.2 Transformation des données

Pour pouvoir importer un grand nombre de données sous Neo4j, il est usuel de créer des fichiers `.csv` d'import. Vous trouverez des indications ici : `https://neo4j.com/developer/guide-import-csv/`.

- Question - Écrivez une « moulinette » python permettant de transformer les fichiers `yelp_restaurants.json`, `yelp_user.json` et `yelp_review.json` en autant de fichiers `.csv` que nécessaire.

Quelques indications supplémentaires :

- Les données .json qui sont mises à votre disposition ne concernent que les restaurants du Delaware et les utilisateurs ayant fait des reviews sur ces restaurants. Il ne faut donc pas filter les restaurants, les utilisateurs ou les revues : ils sont tous potentiellement intéressants.
- Il faut un fichier csv par type de noeud et un fichier csv par type de relation.
- Si certaines données de type String contiennent des ", il faut les doubler, puis mettre la chaîne entière entre " ".
- Prévoyez des fichiers avec une ligne d'en-tête.

Quelques exemples minimaux :

Noeuds user

```
user_id:ID(user),name,review_count
1,Weili,90
2,Robyn,518
...
```

Relations friend

```
:START_ID(user),:END_ID(user)
2,42
2,33
...
```

1.3 Importation des données

Une fois vos fichiers .csv créés, vous allez pouvoir les importer dans une base Neo4j. Pour ce faire, appliquez le protocole suivant :

1. Créez une nouvelle base de données locale (nom et mot de passe à votre convenance). Ne la démarrez pas !
2. Dans la zone correspondant à votre nouvelle base de données, cliquez sur les ..., puis sur `Terminal`
3. Entrez la commande suivante avec **tous** les fichiers `.csv` à charger :

```
./bin/neo4j-admin import --database=graph.db
    --nodes=labelDuNoeud=cheminComplet/monFichierDeNoeuds.csv
    --relationships=labelDeLaRelation=cheminComplet/monFichierDesRelations.csv
    --trim-strings=true --force --multiline-fields=true
```

Il y a autant de `-nodes=labelDuNoeud=cheminComplet/monFichierDeNoeuds.csv` que de types (labels) de noeuds à charger et donc de fichiers .csv de noeuds et autant de `-relationships=labelDeLarelation=cheminComplet/monFichierDesRelations.csv` que de types (labels) de relations à charger et donc de fichiers .csv de relations. Votre ligne de commande est donc potentiellement très longue, essayez au fur et à mesure !
Le `-force` supprime l'import existant pour refaire un nouvel import.
Le `-multiline-fields=true` permet de gérer des valeurs avec des retours à la ligne. Il est donc fortement conseillé de le mettre.
Le `-database=graph.db` est obligatoire tel quel (ne changez pas graph.db).

4. Si aucun message d'erreur n'apparaît, vous avez réussi !
5. Il ne vous reste plus qu'à démarrer la base et rouvrir le navigateur pour commencer à manipuler le graphe

1.4 Découverte des données

À l'aide des commandes de base vous pouvez vérifier que les données importées sont correctes par rapport à la modélisation définie pour répondre aux besoins applicatifs de recommandation d'utilisateurs.

2 Interrogation des données

- Question - Maintenant que les données sont insérées, vous devez définir les requêtes neo4j permettant de répondre aux besoins suivants :

1. Donner le nombre de nœuds par label;
2. Donner le nombre de relations par type;
3. Donner la ou les catégories du restaurant *8th & Union Kitchen*;
4. Donner la ou les ambiances du restaurant *8th & Union Kitchen*;
5. Donner les reviews du restaurant *Tokyo Sushi*;
6. Donner les couples d'utilisateurs amis qui ont reviewé le restaurant *Tokyo Sushi*;
7. Qui est l'utilisateur qui a le plus d'amis ?
8. Qui sont les amis des reviewers du restaurant *Tokyo Sushi* qui ont reviewés des restaurants avec les mêmes ambiances ?
9. Combien de paires de personnes ont au moins 10 restaurants aimés en commun ? (un restaurant est 'aimé' si la review a une note (*stars*) >= 4)

3 Application : moteur de recommandation d'influenceurs

La dernière partie de ce TP a pour objectif la construction du moteur de recommandation d'influenceurs.

3.1 Codage de l'application

- Question - Nous vous demandons de produire une application en Python reposant sur la base de données Neo4j, prenant en entrée des informations sur un restaurant à promovoir :

- ville (chaîne de caractères)
- ambiances (tableau de chaines de caractères)
- catégories (tableau de chaines de caractères)
- catégorie tarifaire (entier)

et renvoyant les 10 utilisateurs les plus intéressants à inviter pour ce restaurant.

Le code ci-dessous vous montre comment interfacer vos programmes Python avec une base Neo4j. Notez bien que pour l'authentification, le `username` est `neo4j`, et le mot de passe est celui de la base, que vous avez défini au moment de sa création (voir TP de prise en main).

```
from py2neo import Graph

graph = Graph("bolt://localhost:7687", auth=("neo4j", "votremotdepassepourlabase"))

q = "match (u:User) return u.name as username"
res = graph.run(q).to_table()

for record in res :
   print(record[0])
```

Un peu d'aide: `https://neo4j.com/developer/python/`. Il vous faut bien sûr récupérer la librairie `Py2neo`.

3.2 Test de l'application

- Question - Dans votre rapport, vous indiquerez la sortie (liste d'utilisateurs et scores associés) obtenue pour les tests suivants, avec $\alpha = 0.3$, $\beta = 0.3$, $\gamma = 0.3$ et $\delta = 0.1$:

- 'Wilmington', ['casual'], ['Pizza','Burgers','Italian'], 1
- 'Wilmington', ['casual, 'romantic'],['Chinese'], 2
- 'Wilmington',['hipster']['Nightlife','Bars'],1
- 'New Castle', ['casual','classy'],['Coffee & Tea'],2
- 'New Castle', ['classy'],['Seafood'],1