TELECOM ParisTech **MDI341**
2018/2019 **A. Bellet, H. Jalalzai, T. Moreau**

TP Noté n° 4 : Graphes

Le compte rendu de TP est à rendre avant mercredi 27/03/2018, 23h55. Vous devez déposer un **UNIQUE** fichier sous format `ipynb` (`ipython notebook`) sur le site pédagogique du cours, dans le dossier correspondant au TP (Travaux Pratiques > TP Graphs > Rendu TP). Merci d'effectuer **un seul rendu par groupe de 2**.

La note totale est sur **20** points, répartis comme suit :

— qualité des réponses aux questions : **17** pts,

— qualité de rédaction, de présentation et d'orthographe : **2** pts,

— absence de bug : **1** pt.

Malus : **5** pts par tranche de 12h de retard ; 2 pts pour non respect des autres consignes de rendu.

Rappel : aucun travail par mail accepté

Familiarisation avec la librairie NetworkX

Dans ce TP, on va manipuler différents réseaux en utilisant le module `networkx` chargé avec `import networkx as nx` (à installer si besoin avec Anaconda). Il est **fortement conseillé de travailler avec la version 1.11** de `networkx` (quelques incompatibilités avec le notebook fourni sont à prévoir avec les versions 2.x). Pour installer la bonne version :

- Si vous utilisez anaconda :

```
conda install networkx=1.11
```

- Si vous utilisez pip :

```
pip install --upgrade networkx==1.11
```

Le module `networkx` permet notamment de manipuler et de visualiser des graphes, de faire de l'analyse de graphe en calculant facilement des quantités variées (degrés, coefficient de clustering, centralité) et de faire de la prédiction de lien. Il est conseillé de se familiariser avec les fonctions de base du module en regardant sa documentation en ligne :

https://networkx.github.io/documentation/networkx-1.11/

Analyse de graphes

Dans cette partie, on va analyser les propriétés de différents réseaux représentés sous forme de graphe.

Graphes aléatoires

1) On va commencer par travailler sur un graphe aléatoire de type Erdös–Rényi. À l'aide de la fonction `nx.erdos_renyi_graph`, générer un graphe Erdös–Rényi avec les paramètres $n = 200$ et $p = 0.04$. Visualiser le graphe avec la fonction `nx.draw` (note : vous pouvez ajuster la taille des noeuds avec le paramètre `node_size`).
2) Afficher le nombre de noeuds et le nombre d'arêtes, ainsi que le degré minimum, maximum, médian et moyen. Commenter la valeur du degré moyen par rapport à sa valeur en espérance.
3) À l'aide de la fonction `nx.degree_histogram`, tracer la courbe de la distribution des degrés en échelle normale (degré en abscisse et fréquence en ordonnée) ainsi qu'en échelle log-log (log-degré en abscisse et log-fréquence en ordonnée) avec `plt.loglog`. Les courbes sont-elles conformes à ce que l'on attend en théorie ?

4) Calculer le coefficient de clustering global du graphe grâce à la fonction `nx.clustering`. Comparer à la valeur attendue (en espérance).

5) Peut-on parler d'effet "petit monde" pour ce graphe ? Justifier la réponse par une/des mesure(s) appropriée(s).

6) À l'aide de la fonction `nx.barabasi_albert_graph`, générer un graphe aléatoire de type Barabási–Albert avec les paramètres $n = 200$ et $m = 3$ (m donne le nombre de noeuds auquel est connecté chaque nouveau noeud). Visualiser le graphe et afficher les différentes mesures/courbes des questions 2 à 5. Commenter les différences de résultats avec le graphe Erdös–Rényi.

Graphe Karate Club

On va maintenant travailler sur un graphe réel de petite taille (34 noeuds, 78 arêtes) qui représente les liens d'amitié entre les adhérents d'un club de karaté (le graphe est intégré à la librairie networkx). Ce graphe est un classique en analyse de graphe [1] : il est intégré au module et se charge avec la commande `G_karate = nx.karate_club_graph()`.

7) La classe `AgglomerativeClustering` du module `sklearn.cluster` permet de réaliser un clustering hiérarchique du graphe à partir de la matrice `distances` contenant la longueur des plus court chemins entre les noeuds du graphe (cette matrice est précalculée pour vous dans le notebook fourni). Pour réaliser le clustering, il faut utiliser l'option `affinity='precomputed'` et fitter le modèle sur la matrice `distances`. Effectuer le clustering avec 2 puis 4 clusters en utilisant le critère de linkage de votre choix. Visualiser les communautés obtenues en colorant les noeuds du graphe en fonction de leur cluster (option `node_color` de `nx.draw`).

8) Calculer les scores de centralité des noeuds selon les 4 notions de centralité vues en cours (degree, eigenvector, closeness et betweenness) grâce aux fonctions dédiées. Visualiser ces scores en colorant les noeuds du graphe et expliquer les différences entre les 4 notions en prenant exemple sur ces visualisations.

Bonus : graphe des produits Amazon

On va manipuler ici un graphe de grande taille (plus de 300k noeuds et 900k arêtes) dont les noeuds sont des produits vendus par Amazon, et deux noeuds sont connectés si les deux produits sont fréquemment achetés ensemble. [2] Le code pour charger ce graphe est inclus dans le notebook fourni. **Attention :** ce graphe est volumineux, n'essayer pas de le visualiser sous peine de bloquer votre machine !

9) **(bonus +2pts)** Reproduire l'analyse globale du réseau réalisée plus haut sur les graphes aléatoires, et ajouter une courbe affichant le coefficient de clustering moyen en fonction du degré des noeuds en échelle log-log. Commenter et donner une interprétation possible des résultats pour l'application considérée (compréhension des mécanismes d'achats simultanés de produits).

Prédiction dans les graphes

Dans cette partie, on va expérimenter avec la prédiction de lien et la propagation de label dans un *ego network*, c'est-à-dire un sous-graphe d'un réseau social centré sur un utilisateur. Ici on va travailler avec un ego-network Facebook qui représente les amis d'un utilisateur donné (sans ce dernier) et les liens d'amitié entre eux. [3] Le jeu de données contient 10 ego-networks, on va travailler avec l'un d'entre eux seulement qui se prête bien à la visualisation. Le code pour charger et visualiser ce graphe est inclus dans le notebook fourni.

10) Nous allons nous intéresser à la prédiction de lien. Le notebook fourni contient un code permettant de créer un graphe d'apprentissage `G_fb_train` qui est une copie du graphe original auquel on a retiré 20% des arêtes choisies aléatoirement. Scorer les arêtes non-existantes de `G_fb_train` avec les 3 similarités vues en cours et implémentées dans `networkx` (Jaccard coefficient, Adamic-Adar index et Preferential attachment). Pour chaque similarité, calculer l'Aire sous la Courbe ROC

1. Les curieux pourront consulter la page Wikipedia : https://en.wikipedia.org/wiki/Zachary%27s_karate_club
2. Plus de détails ici : https://snap.stanford.edu/data/com-Amazon.html
3. Plus de détails ici : https://snap.stanford.edu/data/egonets-Facebook.html

(AUC) (`sklearn.metrics.auc`) et afficher la courbe ROC (`metrics.roc_curve`). Commenter les résultats.

11) Pour finir, nous allons utiliser l'algorithme de propagation de labels semi-supervisée vu en cours. Les noeuds de l'ego network ont différents attributs que l'on peut utiliser comme étiquette. On utilisera ici un attribut qui vaut $+1$ si la personne a étudié dans une école donnée, et -1 sinon (cet attribut est relativement lisse sur le graphe). Le code pour charger l'attribut en question, ainsi que pour sélectionner un sous-ensemble de noeuds étiquetés, est fourni dans le notebook. Appliquer la propagation de label avec le paramètre `alpha=0.7` et afficher le score de prédiction ainsi que les labels prédits.