Analyse de l'évolution des réseaux sociaux

LIFPROJET

HUTT Yannis et CADIER Julien

Sommaire

1 – Description du projet

Nous avons conçu une interface graphique sous la forme d'une fenêtre qui permet d'afficher l'exploitation des données que nous avons eu sur différentes séries.

Nous sommes parti d'une base de données constituées de plusieurs fichiers (sous le format GRAPHML) représentant pour chacun une scène de la série.

Dans chacun des fichiers, on retrouve la liste des personnages de la série (représenté par un nom et un ID) ainsi que le poids des liens qui les unit.

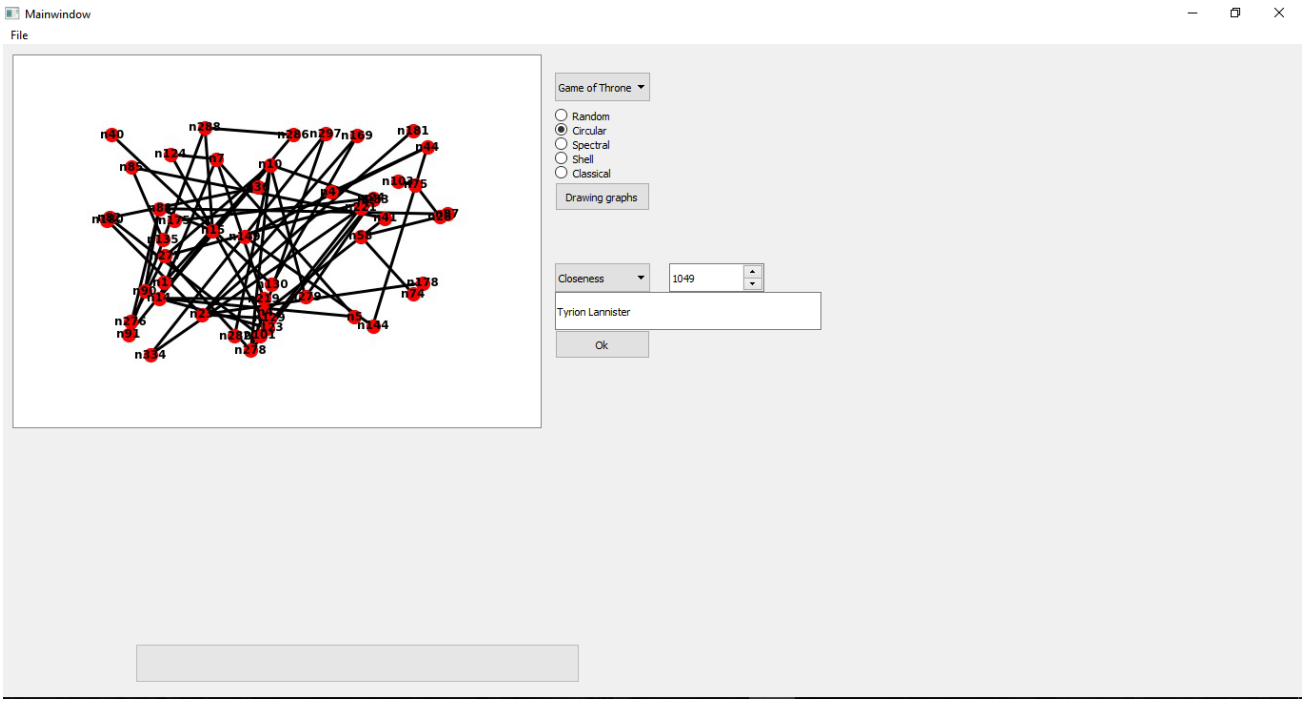
Afin de traiter ces fichiers, nous avons choisi d'utiliser le langage Python suite au conseil de notre encadrant. Après nous être familiarisé avec la documentation de Python et de Networkx (une bibliothèque) nous avons pu commencer à extraire les données qui nous intéressaient depuis les fichiers GRAPHML. Après avoir choisi la série souhaité, pour lire le fichier nous avons utiliser la fonction Read_graphml de Networkx, nous avons ensuite créer un dictionnaire afin d'associer les ID des nœuds avec les noms des personnages. Nous avons ensuite du créer un deuxième dictionnaire pour inverser cette association afin de pouvoir faire une recherche par nom de personnage plutôt que par ID.

Nous prenons alors en compte le nombre de scènes demandés par l'utilisateur et potientiellement le bornons au nombre de scènes maximum de notre base de données. Après avoir créer un tableau, on stocke la liste des données d'une scène par case. On peut donc ainsi appliqués les différentes centralités et les stocker dans de nouveaux tableaux (Voir Annexe1). On demande ensuite à l'utilisateur de choisir la centralité qu'il préfère et on l'envoie dans une nouvelle fonction avec le nom du personnage pour n'extraire que les données du personnage qui nous intéresse (fonction *recherche_nom* (Voir Annexe2)). Dans cette fonction, on convertit le nom passé en paramètre en l'ID du personnage grâce à notre dictionnaire inversé. On appelle ensuite la fonction *type_centralite_voulu* avec comme paramètres l'ID et la centralité afin d'extraire les données du personnage. On ajoute ensuite ces valeurs sur un 'plot' (nouvelle bibliothèque Seaborn). Avec ce graphe, on revient à la fonction précédente qui enregistre l'image ainsi créer et qui nettoie ensuite ce graphe pour ne pas qu'ils ne se superposent pas dans le cas de plusieurs utilisations.

Les autres traitements que l'on peut faire c'est le dessin des graphs avec les liens entre les personnages, nous proposons différents types de graphiques (circulaire,aléatoire…)

Concernant l'interface graphique nous l'avons fait avec la bibliothèque PYQT5, qui permet de concevoir des fenêtres pour pouvoir choisir les traitements à faire sur le jeu de

données. Le code se compose d'une classe principale qui permet d'initialiser la fenêtre principale ainsi que les différents élements du programme, on y trouve aussi des fonctions interne de traitement qui permette de relier la partie graphique avec la partie traitement expliquée en haut, l'interface permet aussi de charger une image depuis un fichier et de l'afficher. Nous nous sommes inspirés du Modèle VueContrôleur.



2 – Objectifs

Pour commencer, nous avions pour objectif d'apprendre le langage Python nottament grâce à de la documentation en ligne.

Nous nous sommes ensuite documenter sur la bibliothèque Networkx que notre encadrant nous avait conseillé d'utiliser afin de traiter les fichiers GRAPHML au plus simple.

Nous avons par la suite décider de séparer notre partie Modèle (traitement) et notre partie VueControleur (graphique).

Nous avons chercher une bibliothèque permettant d'afficher nos graphes et nous sommes tombés sur Seaborn et NetworkX.

Pendant ce temps, nous nous sommes renseigner sur la documentation PYQT5 et avons commencer à afficher nos graphes dans une fenêtre.

3 – Choix de conception

Nous avons choisi Python pour la diversité de ses bibliothèques et sa simplicité d'inclusion de celles-ci.
C'est également un langage interprété léger.
Il permet aussi de faire de la programmation orienté objet.

Nous avons choisi QT car c'est une bibliothèque graphique très complète qui contient beaucoup d'éléments graphiques.

4 – Conclusion

Ce projet nous a permis d'appréhender le monde de la Data Science et d'apprendre un nouveau langage de programmation avec ces subtilités et ces bibliothèques externes qui permettent un traitement et une visualisation dans une interface qui proposent différent type de traitement.

```python
def start_main(Nomserie,nbScenes, nom, operations) :

    if Nomserie == "Game of Thrones":
        """ On crée un dictionnaire qui associe les noms des persos à leur numéro de noeud """
        Gl=nx.read_graphml("data/got/GoT_S05E09_1039.graphml")
        serie ="got"
    if Nomserie ==  "Breaking Bad":
        Gl=nx.read_graphml("data/bb/BB_S03E11_598.graphml")
        serie = "bb"
    if Nomserie == "House of Card":
        Gl=nx.read_graphml("data/hoc/HoC_S02E13_879.graphml")
        serie = "hoc"
    dicoNumToNom = nx.get_node_attributes(Gl,"label")
    for cle in dicoNumToNom :
        dicoNomToNum[dicoNumToNom[cle]] = cle


    plage = nbScenes
    if serie == 'got' :
        if plage > 1039 :
            plage = 1039
    if serie == 'bb' :
        if plage > 599 :
            plage = 599
    if serie == 'hoc':
        if plage > 879 :
            plage = 879
    liste_graph = []
    for element in os.listdir('data/'+serie+'/' )[:plage]:
        liste_graph.append(nx.read_graphml("data\\"+serie+"\\"+ str(element)))


    for l in liste_graph :
        #l.remove_nodes_from(nx.isolates(l))  ==> Pour enlever les isolates mais on doit les garder po
        #nx.draw_networkx(l,pos=nx.spring_layout(l))
        """ |== >  Affichage du graphe de la case liste_graph(l)"""
        tableau_degree_centrality.append(nx.degree_centrality(l))  # une façon de lire la centralité
        tableau_closeness_centrality.append(nx.closeness_centrality(l))
        tableau_pagerank.append(nx.pagerank(l))
        tableau_betweeness.append(nx.betweenness_centrality(l))

#    graph_style(liste_graph)
    recherche_nom(nom, operations)
```

Annexe 1

```python
    liste = operations
    i = 0
    tab =[]


    if 'd' in liste:
        tab.append([x[numNoeud] for x in tableau_degree_centrality])
        sns.tsplot(tab[i], color = "blue")
        i+=1
    if 'c' in liste:
        tab.append([x[numNoeud] for x in tableau_closeness_centrality])
        sns.tsplot(tab[i], color = "red")
        i+=1
    if 'p' in liste:
        tab.append([x[numNoeud] for x in tableau_pagerank])
        sns.tsplot(tab[i], color = "black")
        i+=1
    if 'b' in liste:
        tab.append([x[numNoeud] for x in tableau_betweeness])
        sns.tsplot(tab[i], color = "green")
        i+=1


def recherche_nom(nom, operations) :

    nomNoeud = str(nom)
    numNoeud = dicoNomToNum[nomNoeud]

    type_centralite_voulu(numNoeud, operations)

    plt.savefig("images/" + nom + operations + ".jpg")

    plt.clf()
```

Annexe 2