Visualisation graphique des relations en dépendances.

Mot d'introduction

Nous arrivons à la dernière étape du projet.

Pour celle-ci, on part du fichier TSV que nous sommes maintenant capables d'obtenir en sortie du script de la semaine précédente. Un exemple d'une telle sortie est toutefois disponible sur iCampus pour pouvoir faire ce TP quel que soit l'état du dépôt de projet qui vous est assigné.

À partir de ce fichier qui contient des informations sur des couples *prédicat – argument* avec une valeur d'information mutuelle pour chaque couple, nous allons construire des graphes pondérés et utiliser Gephi-lite pour les visualiser et les analyser.

On va donc écrire un dernier script qui prendra en argument le nécessaire pour construire le graphe propre à l'item lexical à étudier, et produira un fichier **gefx** qui est un format XML de description de graphe.

On chargera ensuite les fichiers obtenus dans l'interface en ligne pour construire des visualisations.

Pour rappel

— un ultime groupe gitlab vous a été attribué aléatoirement pour le reste du semestre. Vous y trouverez un dépôt à cloner. Votre rapport final décrira ce dépôt sur sa branche **doc** mais vous pouvez commencer à travailler sur les graphes avec le fichier fourni sur iCampus.
— la branche **main** sert à l'avancée du projet et des exercices, mais elle ne doit contenir que du code finalisé. il faut y pousser le moins souvent possible.
— une branche **doc** sert au rendu du journal de bord (un fichier markdown *différent* par semaine), et au rapport final.
— chaque semaine, vous devrez créer des branches individuelles réservées au travail de chaque membre du groupe.
— un tag xxx-fin doit être utilisé pour indiquer qu'un exercice et terminé et un tag xxx-relu indiquera qu'il a été relu par un tiers et est prêt à être fusionné (**merge**). Un dernier tag indiquera que le travail sur la branche **main** est terminé.
— pour rappel, les **xxx** d'un tag seront à remplacer par **xy-sTrN**, où xy sont vos initiales, T le numéro de la séance et N votre rôle.

Exercice 1 Construction du graphe

Un graphe est un objet mathématique constitué d'un ensemble de **sommets** ou **nœuds** (*vertices* ou *nodes* en anglais) et d'un ensemble de **liens** (ou *edges* en anglais) qui relient certains couples de sommets.

En python, ces deux ensembles sont aisément représentés par des **set()**, les liens peuvent être décrits comme des **tuples** contenant deux sommets et éventuellement un poids qui indique la force de l'association.

Les lignes du fichier TSV produit précédemment peuvent donc s'interpréter comme décrivant chacune un lien entre un prédicat et un argument.

Ainsi le tableau donné en exemple la semaine dernière décrit un graphe qui pourrait être visualisé dans gephi comme illustré par la figure 1.

Le fichier XML correspondant est disponible sur iCampus pour l'exemple.

Si le graphe ci-dessous ne comporte que 2 liens, le fichier tsv sur lequel on va travailler en décrit 35755 ! On va donc pour cela procéder à la sélection d'une sous-partie de ces données en fonction d'une **requête utilisateur donnée en argument**.

On voudra par exemple «*construire le graphe autour de VERB-lutter-contre*».

On pourra alors retenir le **nœud** *VERB-lutter-contre* lui-même, puis tous ses voisins, ainsi que les voisins de ces voisins.

On souhaitera proposer différents filtres en amont :
— une option du script doit permettre d'ignorer les liens en dessous d'un certain seuil sur l'Information Mutuelle ;
— une option du script doit permettre d'ignorer les liens qui contiennent un hapax (un mot qui n'apparaît qu'une fois dans le corpus) ;
— une option du script doit permettre d'ignorer les liens uniques (un mot qui n'est rattaché qu'à un seul prédicat et inversement).

On proposera d'aller plus loin (voisins des voisins des voisins…) en option du script.

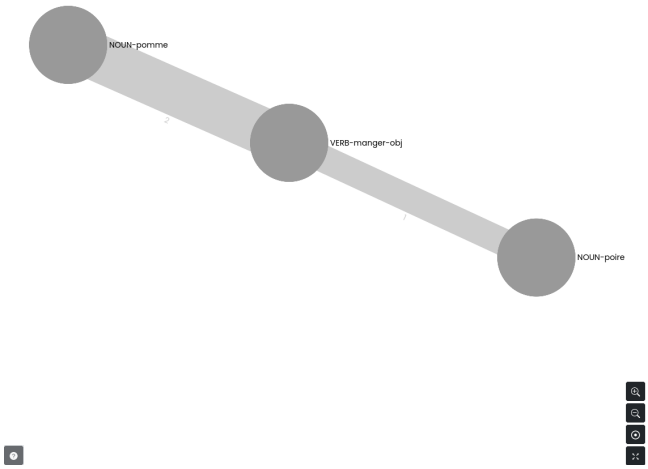

FIGURE 1 – graphe du mini corpus de la séance 8.

Consultez les fichiers gefx donnés en exemple et chargez les dans gephi [1] pour vous en inspirer.

Vous devrez travailler en vous concertant pour que tout fonctionne, mais voici trois tâches qu'on peut isoler pour répartir le travail :

R1 travaillera sur l'interface du script (arguments et options) ainsi que sur le filtrage pour sélectionner les données strictement nécessaires à la construction du graphe.

R2 travaillera à produire une une sortie XML à partir des données filtrées par R1. voir `https://gexf.net/primer.html` et les fichiers donnés en exemple.

R3 explorera la bibliothèque python **networkx** [2] pour trouver des alternatives aux solutions de R1 et R2.

Exercice 2 visualisation

Un fois capables de produire des fichiers XML, vous chargerez ceux-ci dans **gephi-lite** (`https://gephi.org/gephi-lite/`) pour produire les visualisations.

Rappels des fonctionnalités à privilégier :

— Dans le menu **statistiques**, lancez le calcul du **PageRank** et du **Degree**
— Dans le menu **apparence**, rendez la taille des nœuds proportionnelle au **PageRank**
— Si le graphe comporte trop de sommets, créez un filtre sur l'attribut **Degree**
— pour la spacialisation, essayez surtout **ForceAtlas2**

Bonus : Faites en sorte de colorier différemment les nœuds des **prédicats** et les nœuds des **attributs**

Exercice 3 Mise en production

Comme pour les semaines précédentes, validez le code d'un(e) de vos camarade et ajoutez un tag **xxx-relu** quand le résultat est satisfaisant.

Finalement, fusionnez vos travaux et proposez une version finale combinant les différentes contributions sur la branche **main**.

Indiquez que le travail du groupe est terminé au moyen d'un tag.

Exercice 4 Mise à jour du journal de bord

Pour ce travail, chaque membre renseigne sa partie du journal de bord, qui sera hébergé sur la branche **doc**. Commentez :

1. vos difficultés
2. vos solutions
3. les choix lors des *merges*

1. `https://gephi.org/gephi-lite/`
2. `https://networkx.org/`