Compte-rendu de TP de HAI824I

TP2 : Intégration de données

Asma Guermouche
Yanis Bouallouche
Sofian Ettahiri
Florian Thepaut

asma.guermouche@etu.umontpellier.fr
yanis.bouallouche@etu.umontpellier.fr
sofian.ettahiri@etu.umontpellier.fr
florian.thepaut@etu.umontpellier.fr

Master IASD

12 mars 2023

Table des matières

L'ensemble de nos travaux est disponible dans ce répertoire **git**.

1 Introduction

Dans le cadre de ce deuxième TP de l'UE Traitement Sémantique des données, dont l'objectif est de mettre en oeuvre un outil d'intégration de données structurées sous la forme de graphes de connaissances, nous avons choisi d'implémenter notre projet en Python.

2 Choix des technologies

Nous avons choisi d'implémenter notre projet en Python, car ce dernier est riche en termes de librairies, notamment :

- **rdfLib** : Elle dispose de plusieurs méthodes nous permettant de lire facilement nos fichiers d'ontologies.
- **SPARQLWrapper** : Elle permet de naviguer dans les fichiers rdf et effectuer des requêtes SPARQL afin d'extraires les données nécessaires à notre programme.
- **py_stringmatching** et **nltk** : Elles ont été utilisées pour effectuer les comparaisons entre les différentes vale urs des propriétés choisies.
- **tKinter** : nous avons utilisé cette librairie pour mettre en place l'interface graphique
- **re** : pour utiliser des expressions régulières et encore beaucoup d'autres librairies qui nous ont servi notamment à implémenter les mesures de similarités que nous utilisons.

3 Architecture du système

Notre projet se divise en cinq fichiers, dont trois principaux :

- **main.py** qui nous sert à implémenter l'interface graphique de l'application.
- **measures.py** qui nous sert à implémenter les différentes mesures de similarités que l'on va utiliser.

Et le fichier **parseRdf.py** qui nous sert à implémenter la partie algorithmique de l'application (comparaison, utilisation des mesures, calcul de la f-measure, etc ...). Les fichiers **command.py** et **bridge.py** sont des fichiers "rattachés" à l'interface graphique, dans lesquels sont implémentés des fonctions/structures dont nous nous servons.

4 Implémentation

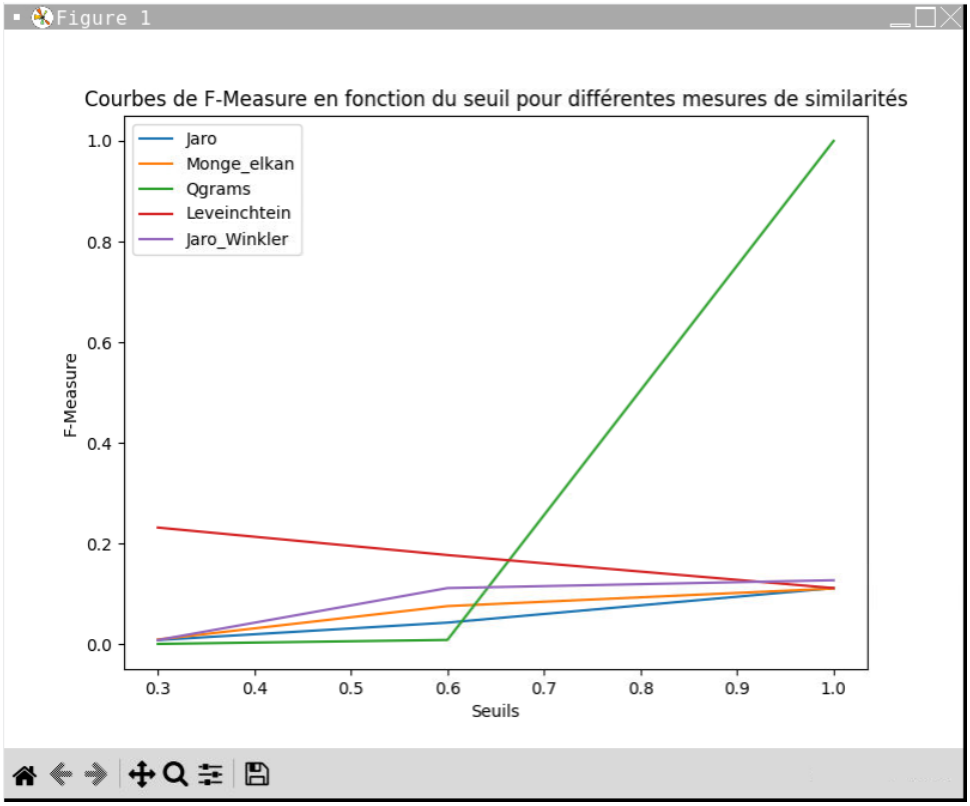
4.1 Principe de la comparaison

Concernant la comparaison, on utilise la méthode **compare** définie dans le fichier **parseRdf.py**. comme la comparaison concerne les valeurs des propriétés choisies par l'utilisateur, et que les deux fichiers **source** et **target** contiennent les mêmes propriétés : Premièrement, en utilisant la méthode **getSubObjSource** qui implémente des requêtes SPARQL, on récupère la liste des ressources de type **F22_SelfContained_Expression** (qui contient une œuvre musicale avec les attributs suivants : expression, key, graph, composer, title, note, opus, genre.), ainsi que la liste des valeurs à comparer à partir des propriétés choisies par l'utilisateur.
On compare la moyenne obtenue à partir des mesures de similarités choisies par l'utilisateur au seuil, selon le résultat de cette comparaison, le couple des deux fichiers source et cible respectivement.

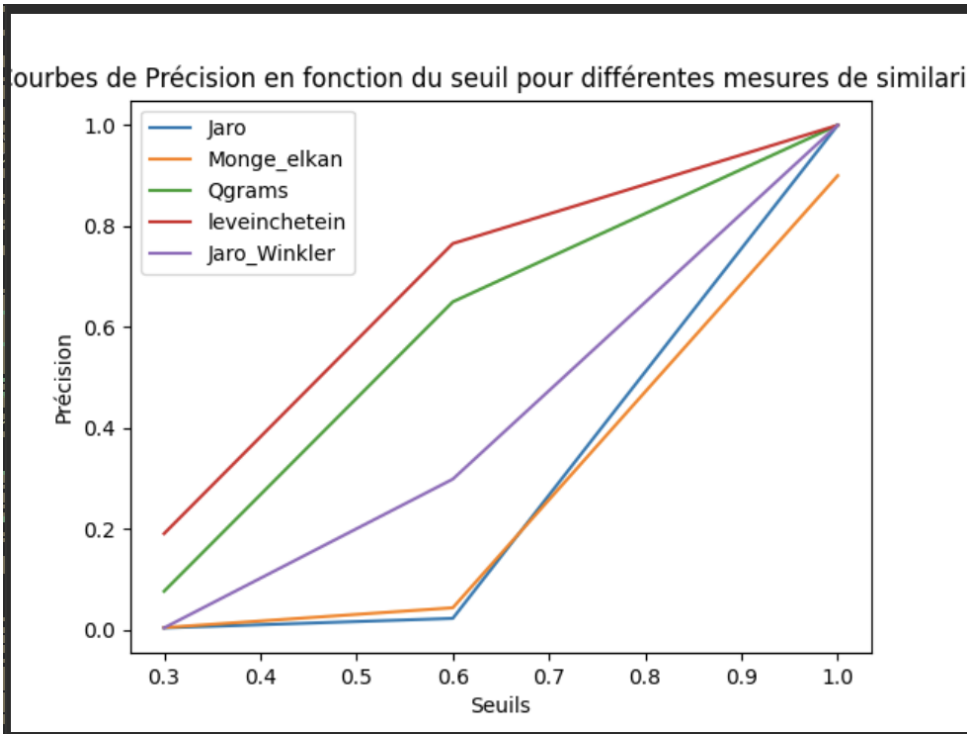
5 Evaluation

Dans le but de mesurer les performances de notre outil, nous allons avoir besoin de trois indicateurs importants, le rappel, la précision et la f-measure (qui s'obtient à l'aide des deux précédents). Pour obtenir ces données, nous allons avoir besoin d'une "référence", c'est là qu'entre en jeu le fichier d'alignement qui nous permet de savoir quels sont équivalents quelles entités le fichier source et le fichier cible.

Une fois cette f-measure obtenu, on va pouvoir observer l'effet du seuil de similarité sur nos résultats et sur la performance de notre application. Pour cela, on va exécuter notre outil en choisissant une seule propriété. Pour chaque mesure de similarités, et pour chaque valeur de seuil allant de 0.3 à 1, on va relever la f-measure mesurée et, on va tracer un graphique sur lequel chaque courbe représentera l'évolution de la f-measure, la précision ainsi que le rappel en fonction du seuil pour différentes mesures de similarité.

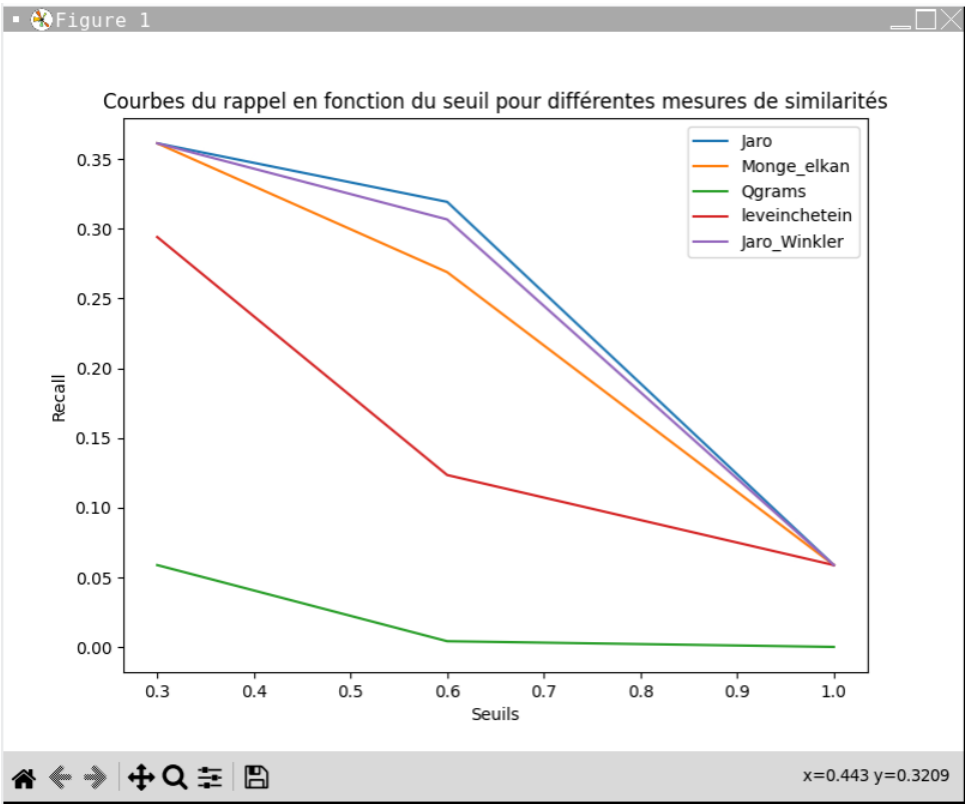


Ici, on obtient le graphique d'évolution de la f-measure en fonction du seuil, pour plusieurs des mesures de similarités que l'on a utilisé.



D'après le graphe qui représente l'évolution de la précision en fonction du seuil, on peut remarquer que plus le seuil est haut, plus on a une précision plus importante. Cela s'ex-

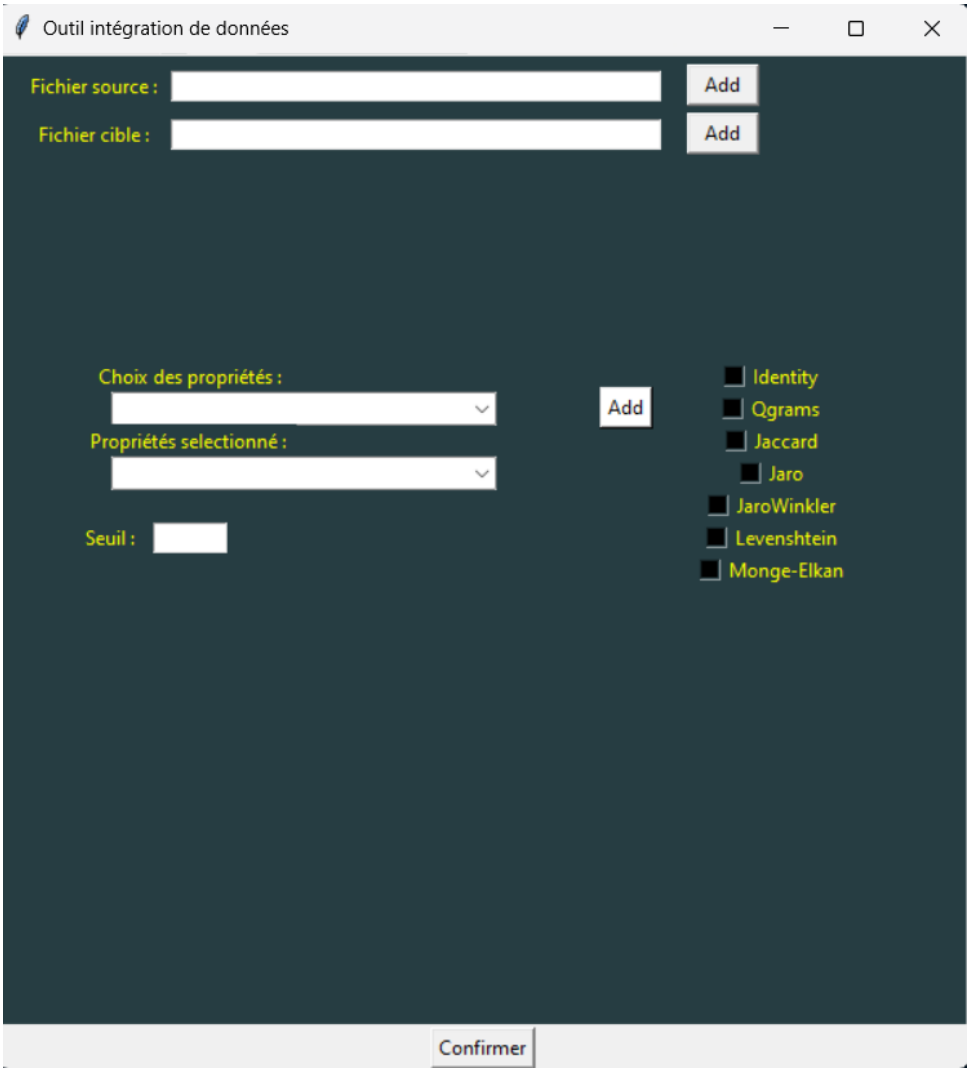
plique par le fait qu'on a la somme du vrais_positifs et du faux_positifs qui diminue, donc la précision augmente.



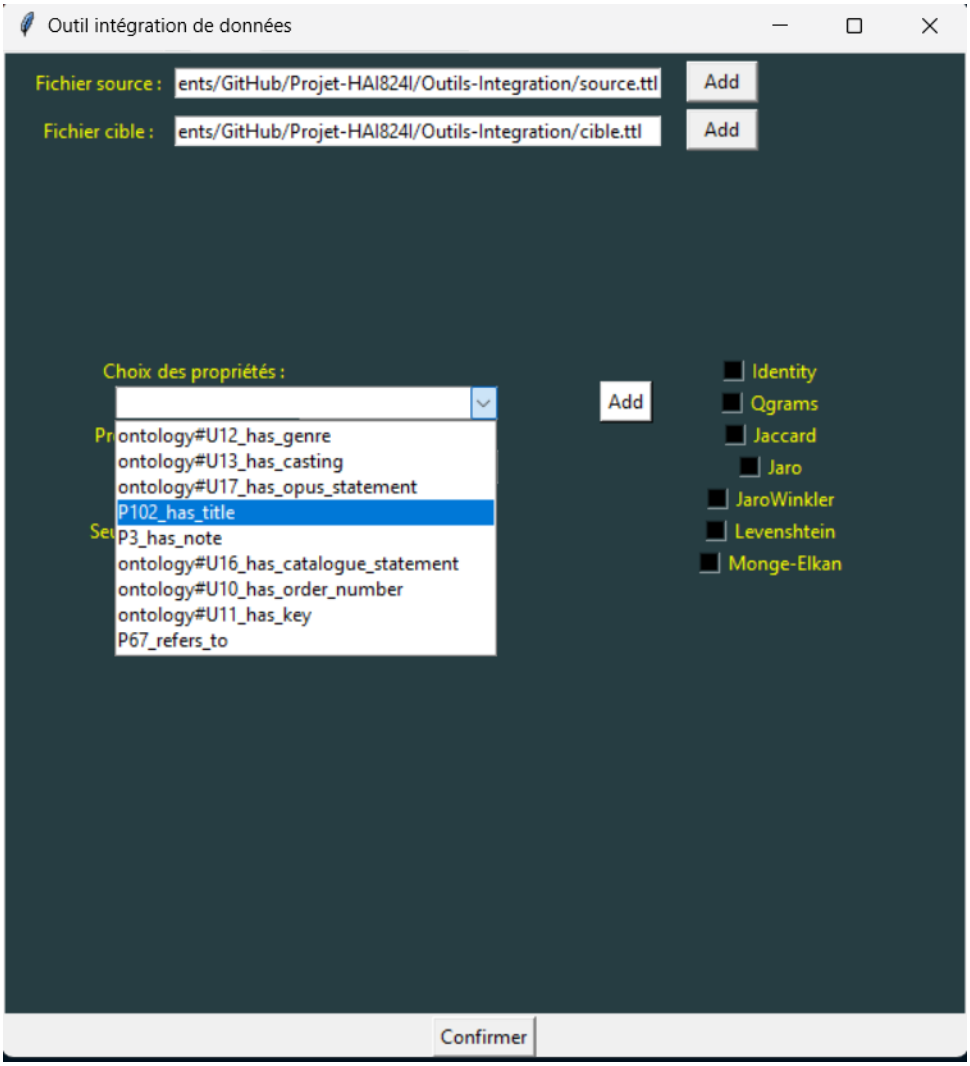
Ici le graphique du rappel en fonction du seuil pour chacune des mesures.
Contrairement au seuil, le rappel diminue

6 Cas d'usage

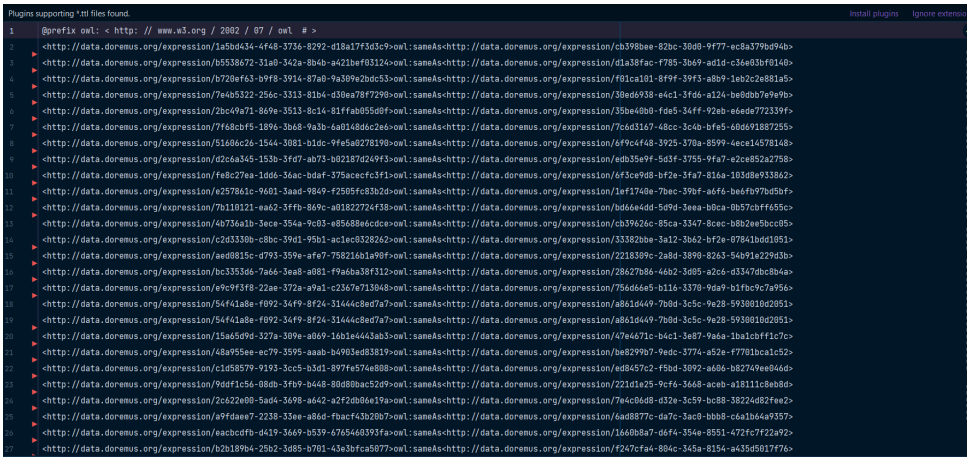
L'outil d'intégration se présente donc comme cela :



Pour l'utiliser, il suffit donc de sélectionner le fichier source et le fichier cible en appuyant sur Add. Ensuite, il faut choisir les propriétés à comparer :



Les propriétés seront visibles dans la liste du bas. Ensuite, il faut sélectionner les mesures à utiliser pour calculer la similitude entre les valeurs des propriétés ainsi que le seuil à appliquer. Une fois cela fait, la fonction **compare** se lance, et à la fin de l'exécution, vous pourrez visualiser les triplets liés par un lien owl :sameAs. Voici un exemple :



7 Conclusion

Le développement de cette application, que nous avons utilisé pour explorer l'alignement ontologique, nous a permis de mieux comprendre les fichiers d'ontologies et les outils disponibles pour les parcourir, tels que SPARQL ou rdflib. Le domaine étant tout nouveau pour nous, le travail s'est révélé assez désarçonnant au début. Cependant, ce fut un bon exercice d'introduction au monde de l'ontologie, et nous avons pu progresser dans notre compréhension de ce sujet. Enfin, nous avons également pu utiliser certaines mesures de similarités que nous avions vues en cours, et qui nous seront sans doute utiles dans d'autres projets, que ce soit en Machine Learning ou en Langage naturel.