Traitement sémantique des données

Construction d'un outil d'intégration de données

TP noté 2

Elmaroufi Amine
Boubker Tabiti
Abdelhalim Azzouz

31 mars 2022

Table des matières

Le but de ce tp est de créer un outil d'intégration de données structurées sous la forme de graphes de connaissances.
Lien vers le git :https://github.com/Gr3yM41t3r/TP2SemFinal

1 présentation outil

on commence premierement par expliquer l'architecture du dossier de notre outils.
La racine du projet ressemble à ceci.

```
graphTools
    Fmeasure.py
    GraphTools
    requests.py
    Similarity.py
Gui
    untitled.ui
images
OutputFile
    benchmark.csv
    results.nt
Silk
    liens.nt
    silk.jar
    Silk-LSL.xml
TP2data
    refDHT.rdf
    source.ttl
    target.ttl
main.py
requirements.txt
```

.

— **graphTools :**Ce dossier contient tous les outils et données nécessaires pour le traitement des graphes de connaissance.
Puisqu'on utilise python comme langage de programmation, il existe plusieurs librairies pour le traitement des graphes. Dans notre cas, c'est la bibliothèque rdflib.
requests.py contient les requêtes sparql.
Similarity.py Pour calculer la similarité entre les chaînes de caractère, on utilise la bibliothèque strsimpy qui propose plusieurs méthodes de comparaison comme JaroWinkler et Levenstein.
Pour les méthodes de jaro et jaccard, on a préféré les écrire .
La méthode principale *calculate_avg_similarity* prend en paramètre une liste de méthodes à appliquer avec leurs pondérations et les deux textes à comparer et rend le résultat
graphtool.py : C'est le fichier principal où on fait tous les calculs de similarité en ouvrant les fichiers **Source** et **Target** et les résultats sont ensuite enregistré dans un fichier resutls.nt
Fmeasure.py Contiens les méthodes nécessaires pour calculer la précision, le rappel et la f-measure.

— **OutputFile :** fichier résultat des ensembles de liens owl :sameAs

— **TP2data :** les données graphe source et target.

— **Silk :** L'outil avec lequel on va évaluer nos données résultats

2 utilisation outil

Pour démarrer l'outil, il faut tout d'abord installer les dépendances requirements.txt

```
pip install -r requirements.txt
```

Ensuite, il faut exécuter le fichier main.py

```
python3 main.py
```

L'exécution du fichier main.py lance une fenêtre où on peut modifier les paramètres suivants.
Fichier Target : où il faut choisir le fichier target.ttl dans le dossier TP2data.
Fichier source : où il faut choisir le fichier source.ttl dans le dossier TP2data.
Propriété à comparer.
Seuil
Méthode de comparaison
Fichier de sortie : où il faut choisir le fichier data.nt dans le dossier OutputFile.

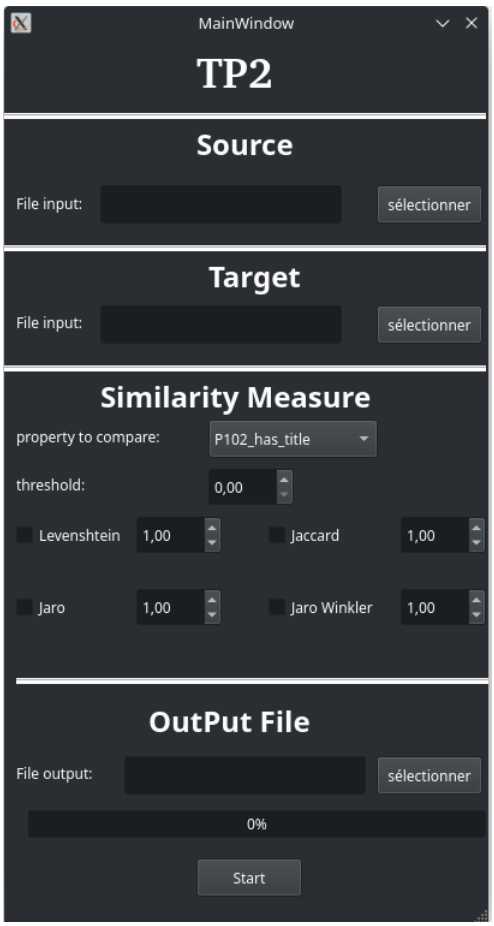

FIGURE 1 – page principale

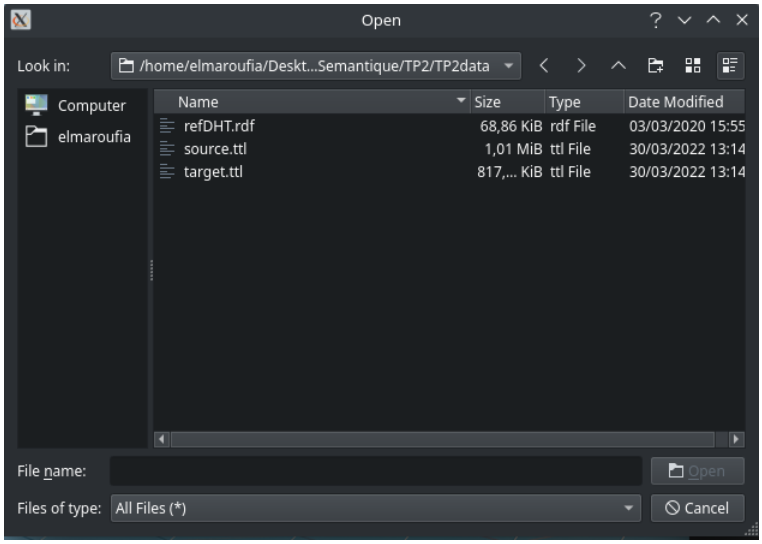

FIGURE 2 – cChoix des paramètres

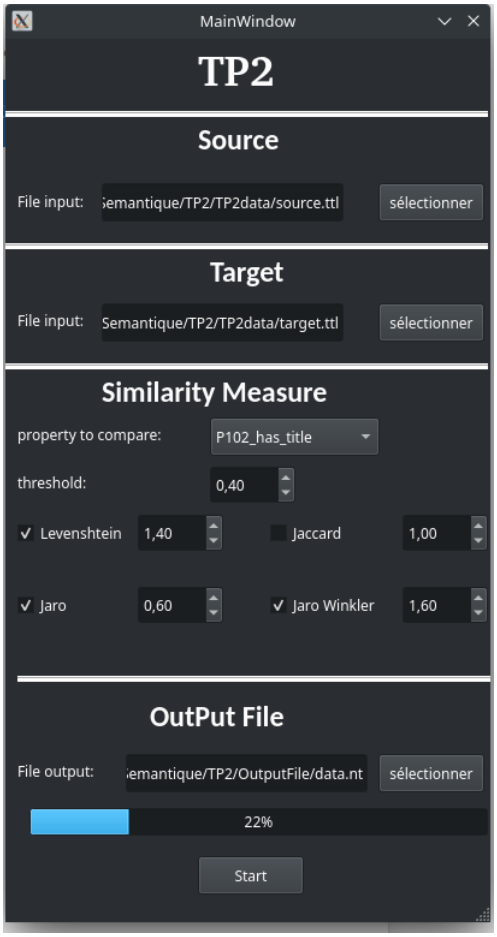

FIGURE 3 – Exécution du programme

2.1 Similarity measure

Dans cette partie de la fenêtre, on coche les méthodes qu'on veut utiliser pour la comparaison, avec leur seuil de pondérations. **"1 par defaut"**

— **Levenshtein.**
— **Jaro**
— **Jaro Winkler**
— **Jaccard**

On tient à préciser que toutes les méthodes retournent un nombre compris ente 0 et 1 où 0 les deux chaînes de caractère sont similaire et 1 sinon.

2.2 property to compare

On propose les 6 propriétés présente dans **F22 Self-Contained Expression**
— **P102_has_title**
— **P3_has_note**
— **U12_has_genre**
— **U11_has_key**
— **U13_has_casting**
— **U16_has_catalogue_statement**

3 Exécution

Lorsque on appuie sur le bouton start. On crée une instance Graphtools avec les fichiers Source, Target et results.nt fournit dans le form, et on récupère la propriété et l'approche choisit pour lancer la bonne méthode de comparaison. Dans le cas où on choisit p102 has title, on lance la méthode function_comp_title

```
        threshold = round(self.doubleSpinBox_5.value(),2)
        if selectedProperty == "P102_has_title":
            one.function_comp_title(selectedMeasure, threshold, self.progressBar)
```

Cette méthode va ensuite exécuter la requête appropriée sur le deux graphe crées. Et pour chaque couple de valeur du fichier source et Target, on calcule la distance entre les deux chaînes de caractère, si cette distance est inférieure au seuil, on l'ajoute dans le fichier output.

```
            for s in source:
            counter += 1
            progressbar.setValue((counter * 100) / total)
            for t in target:
                score = round(calculate_avg_similarity(similarity_method, str(s["P102_has_title"]),
                                                       str(t["P102_has_title"])), 2)
                if score <= threshold:
                    line = "<" + str(s["p"]) + ">  <http://www.w3.org/2002/07/owl#sameAs>  <" + str(
                        t["p"]) + "> .\n"
                    if line not in line_seen:  # not a duplicate
                        line_seen.add(line)
                        self.fichier.write(line)
```

Avant d'ajouter la ligne, on vérifie tout d'abord si cette ligne n'existe pas déjà dans nos résultats pour ne pas avoir de doublant.

4 Cas avec plusieurs mesure de similarité

Dans le cas où l'utilisateur choisirait plusieurs méthodes de similarité, notre outil crée une liste avec des couples de valeur : "Mesure : pondération" Et passe cette liste dans le paramètre de la méthode **calculate_avg_similarity** qui va ensuite itérer sur chaque élément de la liste et renvoie le résultat pondéré

5 Validation des résultats :

Le but de cette étape et de vérifier la performance de notre outil Pour l'évaluer, on va créer plusieurs fichiers d'alignement pour les différentes propriétés et seuil grâce à l'outil **silk**.

5.1 configuration Silk

Dans le fichier **Silk-LSL.xml** on configure notre fichier de tel manière qu'il prend les expressions à partie de **F22_Self-Contained_Expression** et compare dans un premier temps avec la méthode Levenshtein et un seuil de 0 la propriété has title

5.2 génération fichier alignement

Après qu'on ait terminé la configuration, on lance l'outil Silk avec la commande suivante, ce qui crée un fichier liens.nt

```
1    java -DconfigFile=Silk-LSL.xml -jar silk.jar
```

5.3 F-measure

5.3.1 Precision

$$P_{\%} = \frac{true\_Matches\_found}{all\_found}$$

Pour calculer cette mesure On définit la méthode precison() dans le fichier Fmeasure.py Où on va déterminer le nombre des résultats qu'on a trouvé et qui sont vraiment vrais sur le nombre total des resultas trouvés. Et pour faire, on fait l'intersection des données qu'on a trouvées sur le fichier result.nt avec le fichier produit à partir de silk

5.3.2 Recall

$$R_{\%} = \frac{true\_Matches\_found}{all\_true\_Matches}$$

de la même façon que précision la fonction recall retourne le résultat de la division du nombre des triplets qui sont vraie qu'on a trouvé, sur le nombre total qui est vrai du fichier d'alignement

5.3.3 Fmeasure

$$Fmeasure_{\%} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5.4 resultat et courbe

Maintenant, qu'on a les outils pour calculer la performance, on crée les fichiers benchmark avec diffèrent seuil et différentes méthodes de mesure et on représente les données dans un graphe.

5.4.1 comparaison avec des seuils variables

Dans cette analyse, on va dessiner la courbe en fonction des seuils variables pour les deux outils, le notre et Silk.
C'est-à-dire à chaque fois qu'on donne un seuil x à notre outil, on va mettre ce même seuil dans le fichier de configuration XML Et on compare ensuite les résultats pour voir si notre outil trouve les mêmes triplets ou pas.

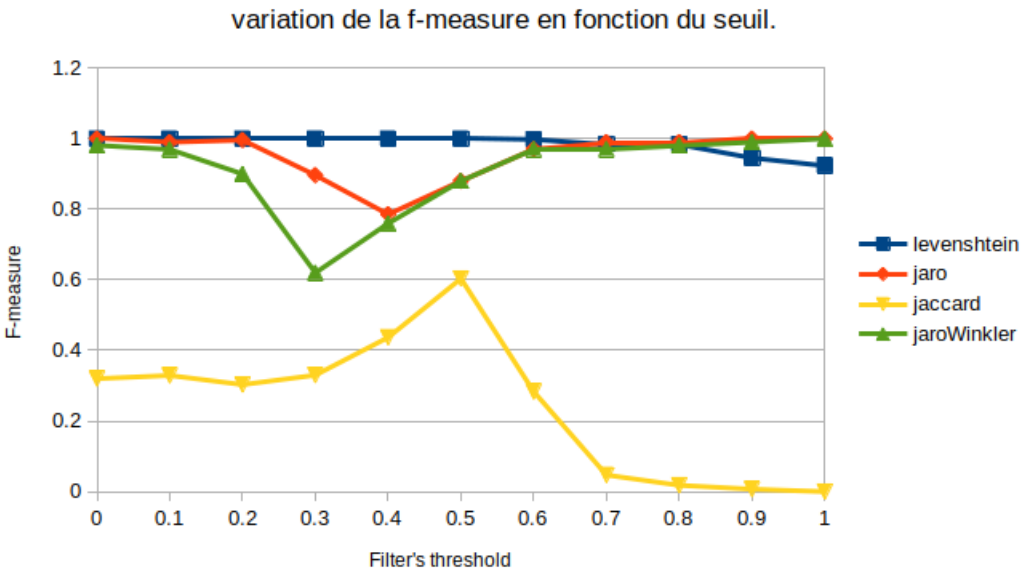

FIGURE 4 – la courbe de la variation de la f-measure en fonction du seuil.