

UNIVERSITÉ DE MONTPELLIER



Construction d'un outil d'intégration de données

Auteurs :

SABRINE FARAH
ABDELHAMID LOUATI
LOUIS REMY

Enseignant :

KONSTANTIN TODOROV

Code source :

<https://github.com/OOKAMI22/pythonProject1>



Contenu du document.

1	Introduction	1
2	Implémentation	2
2.1	Principe	2
2.2	Bibliothèques	2
2.3	Méthodes de mesure	2
2.4	Comparaisons et calculs	3
2.5	Interface utilisateur	4
3	Validation des résultats	6
4	Conclusion	7

1 Introduction

DOREMUS (**Do**ing **Re**usable **Musical** data) est un projet qui a pour but de fournir des modèles de connaissances communs (ontologies), de référentiels partagés et multilingues ainsi que de méthodes pour publier, partager et enrichir les catalogues d'œuvres et d'événements musicaux dans le web des données. Le but de notre travail a été de créer un outil d'intégration de données structurées sous la forme de graphes de connaissances en donnant plusieurs choix à l'utilisateur tel que le choix de propriétés à comparer, les différentes mesures de similarité et le choix d'un seuil pour lequel on verra si deux URI sont les mêmes en utilisant des liens owl:sameAs.

2 Implémentation

2.1 Principe

Dans un premier temps on parse les 2 fichiers (source.ttl et target.ttl) en graphe.

Puis on extrait toutes les expressions de type F22 de nos 2 graphes en appelant la fonction **getExpressions** qui prend en paramètre un graphe.

Ensuite on extrait les propriétés rentrées par l'utilisateur et on les compare via les modes de comparaisons choisis (propriété par propriété) à l'aide de la fonction **getPropriete**, qui prend en paramètre le nom de la propriété, une expression et un graphe.

On calcule la moyenne pondérée avec les coefficients tapés par l'utilisateur, et si cette dernière est supérieure ou égale au seuil choisi, on ajoute les deux expressions au fichier resultat.ttl avec une relation owl:sameAs



Afin de lancer le programme, il suffit d'exécuter le fichier **main.py**.

2.2 Bibliothèques

Afin de mener à bien le projet, on a utilisé plusieurs API et bibliothèques python dont:

- **RDFlib**, afin de lire et gérer les graphes.
- **SPARQLWrapper**, afin d'émettre des requêtes sur les différents graphes.
- **py_stringmatching**, afin d'implémenter quelques méthodes de mesure.
- **pySimpleGui**, pour l'interface graphique utilisateur.

2.3 Méthodes de mesure

Pour les propriétés qui ont des URI, on a extrait ces dernières, et afin de ne pas biaiser nos calculs de similarité on a décidé de faire un pré-traitement pour enlever la partie domaine qui est commune à toutes les URI (puisque ce ne sont que des URL de DOREMUS).

Par exemple l'URL "http://data.doremus.org/event/c" qui deviendra après le traitement "eventc"

Pour les propriétés qui n'ont pas d'URI, on a extrait des Strings auxquels on applique aussi un pré-traitement, qui consiste à enlever les espaces (pour ne pas le compter comme caractère commun).

Voici à ce jour, les méthodes de comparaison implémentées :

- Jaro
- Jaccard
- Monge Elkan
- NumSimilarity
- Ngram
- Levenshtein
- JaroWrinkler
- uriEquality

`py_stringmatching` implémente la plupart de nos méthodes de mesures sauf **Ngram**, **NumSimilarity** et **uriEquality** qu'on a ensuite réécrit nous-même.

Voici le code de **Ngram**:

```
def Ngram(s1, s2, s):
    s1 = pretraitementURL(s1)
    s2 = pretraitementURL(s2)

    i = 0
    id = 0
    while ((i + s) <= len(s1)) or ((i + s) <= len(s2)):
        # print(s1[i:i + s], " == ", s2[i:i + s] + "      ?")
        if s1[i:i + s] == s2[i:i + s]:
            # print("OUI")
            id += 1
        i += 1
    if ((min(len(s1), len(s2)) - s) < 0):
        # print("trop grand")
        return 0
    return (id) / (min(len(s2), len(s1)) - s + 1)
```

2.4 Comparaisons et calculs

Afin de calculer le taux de similarité à chaque fois, nous avons créé deux dictionnaires (un pour les propriétés et l'autre pour les méthodes de mesure) qui nous permettent d'avoir les valeurs à passer en argument à notre fonction générale **compareExpressions** depuis l'interface. **compareExpressions** calcule le taux de similarité en fonction des pondérations, en appelant nos différentes méthodes de comparaison. On a aussi opté pour un code plutôt général en utilisant des fonctions de callback pour faire les différents calculs selon les mesures.

```
def compare(l1, l2, fonction):
    somme = []
    for elem1 in l1:
        for elem2 in l2:
            somme.append(fonction(str(elem1), str(elem2)))
    #LE MAX ICI REPRESENTE ELEM2 LE PLUS SIMILAIRE A ELEM1
    return max(somme)
```




Note : on a dû faire une fonction de comparaison pour ngram car elle prend 4 arguments au lieu de 3 (il faut donner le size).

Afin d'avoir un fichier en sortie, nous avons aussi implémenté une fonction **createfile** qui écrit dans un fichier (et le crée s'il n'existe pas), et à chaque fois que le taux de similarité dépasse le seuil choisi, la méthode ajoute une relation owl:sameAs dans le fichier **resultat.ttl** qui est réinitialisé à chaque execution. On s'est aussi mis d'accord sur notre traitement des ensembles vides comme étant non similaires par défaut (score de 0), car on ne peut pas comparer ce qui n'existe pas, et faire autrement d'après nous reviendrait à nuire à la précision de nos calculs. (Ne rien savoir sur une propriété d'une expression ne nous aide pas à déterminer si elle est similaire à une autre, bien au contraire).

2.5 Interface utilisateur

Notre interface a été faite sous **pySimpleGui** qui se base sur **tkinter**, elle est composée de plusieurs sections.

- Les propriétés choisies sont à cocher ou décocher si on veut les comparer ou pas, et au moins une seule propriété doit être choisie.
- Les méthodes de mesures à droite sont à cocher aussi si on veut les prendre en compte ou pas, avec leur pondération à écrire à côté. À noter aussi qu'en sélectionner plusieurs calculera la moyenne de ces dernières, et que si l'on met une pondération sans avoir coché la méthode de similarité, cette dernière ne sera pas prise en compte.
- La possibilité de mettre le seuil en bas à gauche.
- Un bouton submit une fois que tout est rempli et que l'on souhaite générer le fichier de sortie.

 Outil RDF

—

□

×

OUTIL COMPARAISON RDF (F22)

Choisissez les propriétés à comparer

☐ Clef

☐ Titre

☐ Note

☐ Genre

☐ Compositeur

☐ Opus

Seuil

Choisissez les mesures

☐ Jaro

☐ Jaccar

☐ Monge Elkan

☐ NumSimilarity

☐ Ngram

Ngram Size

☐ Levenshtein

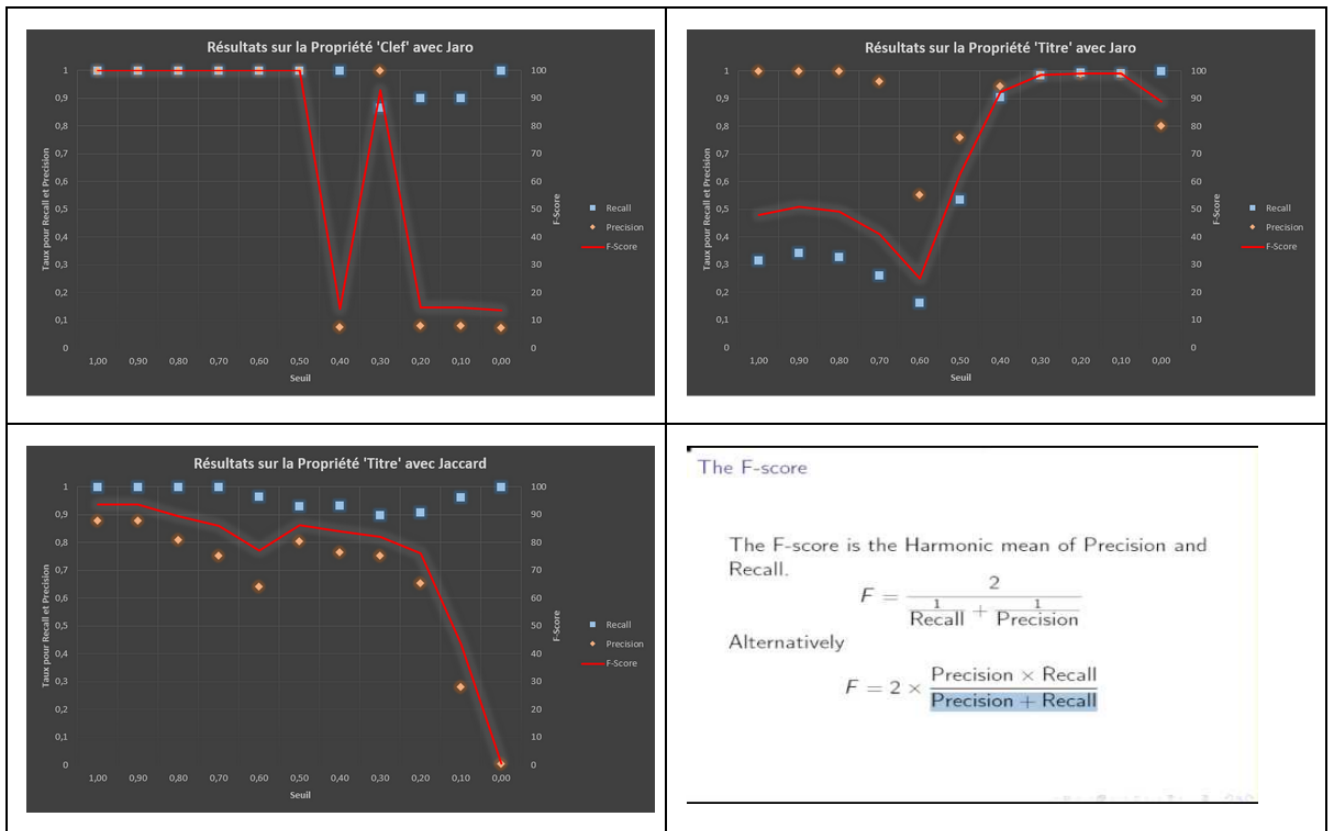
☐ JaroWrinkler

☐ uriEquality

Submit

3 Validation des résultats

Nous comparons nos résultats avec ceux fournis **Silk**, afin de comparer avec différents seuils autres que "1", car dans le fichier fourni en énoncé, les résultats se basaient seulement sur un seuil de "1" et voulant être plus précis dans nos tests on génère des fichiers avec **Silk** en les paramétrant de la même façon que les fichiers générés par notre programme. Etant donné la quantité de données et de possibilités, nous avons choisi de faire notre étude comparative des résultats seulement sur les propriétés: **Titre** représentée par des String et **Clef** par des Uri. Dans un premier temps avec la mesure de **Jaro** puis nous avons remarqué que les résultats pour **titre** n'étaient pas aussi bons que pour **Clef** donc on a relancé les tests avec la mesure **Jaccard** qui utilise la **Tokenisation** sur ses paramètres et les résultats étaient meilleurs par rapport à certains seuils (Nous avons testé les seuils de 0 à 1 par paliers de 0,1. Après calcul du F-score). Voici ce que nous obtenons :



Nous remarquons donc d'après les graphes obtenus que les résultats varient et dépendent des types de mesures utilisées ainsi que des propriétés appelées.



Note : Nous avons faits des tests pour le fichier donné en énoncé mais quand on coche toutes les mesures ainsi que toutes les propriétés le fichier reste vierge après que la machine ait tourné pendant plus de 40 minutes.

4 Conclusion

Ce projet nous a permis d'explorer une nouvelle façade du traitement sémantique des données web, cependant nous aurions aimé améliorer certains aspects tels que donner le choix à l'utilisateur de traiter d'autres expressions autres que celles du type F22, d'implémenter d'autres mesures de similarité ou même d'aller plus loin et améliorer notre interface utilisateur sans oublier le point le plus important qui est faire plus de test sur les différentes mesures implémentées et essayées d'améliorer le code au fur et à mesure.