Rapport TP13(14-15) : Conception et implémentation d'un algorithme de recherche d'informations

TP RÉALISÉ PAR MOSBAH YASMIN EN MASTER 2 MIAGE IA2 (tp individuel)

Pour ce TP, j'ai fragmenté mon algorithme en plusieurs fonctions pour que cela soit plus simple à utiliser et surtout plus facile à lire. Lors de la réalisation du Tp, j'ai suivi point par point les étapes énoncées dans l'énoncé, donc je vais suivre le même plan mais en ajoutant plus de précision.

Etape 1 — Construction d'un corpus de documents sur le Covid-19

- Pour la construction de ce corpus, j'ai mis l'intégralité des fichiers .txt dans un répertoire.
- Ma fonction txt_doc_to_string_list(doc), a parcouru ce répertoire pour mettre dans une liste le contenu txt de tous les documents.

Etape 2 — Pré Traitement des données

- La fonction string_to_token(txt_list) prend la liste précédente en argument et utilise le tokeniseur de nltk pour scinder les documents en une liste de mots. Elle retourne une liste qui contient la liste de mots en fonction du document (chaque document à sa liste de mot, et toutes ces listes sont contenues dans une seule liste).
- J'ai ensuite une fct remove_punctuation_stopwords(txt_list_token) qui supprime la ponctuation, les stops words et les liens https de ma liste de token.
- Pour finir cette étape, j'ai une fct lemma_text(docs) qui lemmatise ma liste de document tokenisé, en leurs donnant des tag (encore avec nltk) et en utilisant WordNetLemmatizer(). La lemmatisation prend donc en compte les tags pour retourner le lemme appropriée.
- Le dictionnaire est généré par la fonction "to_dic(lemma)". Elle prend pour argument la liste de lemme générée précédemment par la fonction **"**lemma_text(docs)".. Elle retourne un dataframe qui a pour colonne la liste des mots, le numéro du document et la fréquence des mots par document.

Etape 3 - Construction de la matrice d'incidence

La fonction "matrice(df)", prend en argument le dataframe du dictionnaire relatif qui a été généré par "to_dic(docs)".

Elle récupère la liste des documents dans le dictionnaire et la liste des mots (sans doublons). La liste des documents est mise en nom de colonne et la liste des mots en index, et ensuite elle parcourt le dictionnaire et ajoute des "1", si le dictionnaire indique que le mot est présent dans le document. Ensuite, elle remplace toutes les cellules vides (NaN) par des zéros.

Etape 4 — Construction de l'index inversé

Pareil que pour la matrice, la fonction "index_inverse(df)" utilise le dataframe dictionnaire relatif pour récupérer tous les mots de tous les documents en ne gardant aucun doublons. Ensuite pour chaque mots elle parcourt le dictionnaire et récupère le numéro de tous les documents ou le mot apparaît. Le mot et la liste des documents sont mis dans un vecteurs.

La fonction retourne une liste de tous ces vecteurs.

Etape 5 — Implémentation d'un algorithme de recherche d'information

Sous-étape 5.1 — Requêtes booléennes

- **Traitement de la requête:**

La fonction "traitement_requete_bool(requete)", prends en arguments la requête (String) et construit "un arbre" dans une liste.
La requête est coupée en token, puis les lemmes des mots sont récupérés.
Elle commence par scinder la phrase en deux au niveau du "AND NOT", ensuite elle parcours les deux "branches" de l'arbre et scinde ou elle tombe sur des "and", et pour finir pour chaque bout elle coupe encore en deux si elle tombe sur des "or".
Par exemple pour la requete : (older adults AND antibodies) AND NOT (genomes OR variant)
La fct va me retourner : list = ([['genome', 'variant']], ['antibody', 'old', 'adult'])
list[0] = [['genome', 'variant']] ← Les mots qu'on ne désire pas avoir dans les documents ceux qui sont désignés par "and not". On remarque que c'est un double "[[ ]]", car les deux mots sont contenus dans un sous bloc "or".
list[1] = ['antibody', 'old', 'adult'] ← les mots qu'on cherche dans les documents.
SI on descend d' un niveau dans l'arbre on a les mots qui sont séparés par des "and" donc par exemple pour list[1] on a la liste ['antibody', 'old', 'adult']. Ici on a de simple crochet car il y a aucun bloc "or" dans la requête initiale.

- **Classement des documents :**

Pour le classement, la fonction res_requete_bool(req_bool,df) va simplement lister les documents contenant les mots non désirés et les documents contenant les mots désirés. Elle fait la soustraction des deux listes, et ensuite elle va classer les documents en fonction de la fréquence des mots désirés.
Pour cela, elle va utiliser le dictionnaire relatif, car les fréquences apparaissant dans ce dataframe.

Sous-étape 5.2 — Requêtes textuelles plus complexes

- **Traitement de la requête:**

La fonction traitement_req_complexe(req_comp), prend en argument la requête (String) et retourne une liste contenant les mots après le traitement.
Le traitement de ce type va consister à retirer les stop word, la ponctuation, et de trouver les lemmes de chaque mots.

- **Classement des documents :**

Le classement des documents est effectué par la somme des TFIDF de chaque mot. Le document ayant la somme de TFIDF le plus élevé est le mieux classé.
Les deux fonctions qui s'occupent de ce traitement sont : tfidf(df, matrice,docs,req) et traitement_doc(nomdoc).
La fonction tfidf(df, matrice,docs,req)va calculer le tfidf d'un mot, et la fonction traitement_doc(nomdoc) va l'utiliser pour l'appliquer sur tous les mots recherchés par document avant de les additionner.

Note : Si vous avez des questions n'hésitez pas à me contacter pour que je puisse vous fournir plus d'explications.
Merci de votre lecture.