TP Recherche d'informations

Sylvain Gault

8 octobre 2024

1 Introduction

Ce TP est à réaliser par groupe de 3 ou 4 en Python sous l'environnement de développement de votre choix. Il sera à rendre le lundi 14/10/2024 à 13h37 UTC. Il sera à rendre sur Teams dans le devoir *« TP note »*. Vous nommerez vos fichiers du noms des membres du groupe.

Vous rendrez un rapport en **PDF** contenant au moins les réponses aux questions ainsi que toute description nécessaire à la reproduction de vos résultats. **Numérotez les questions** auxquelles vous répondez dans votre rapport et incluez au moins une réponse pour chaque question. Pas de réponse = pas de points. N'oubliez pas de lister **les noms** de tous les membres de votre groupe.

Le but de ce TP est d'implémenter un TF-IDF pour faire un moteur de recherche de citations de Kaamelott. On utilisera la bibliothèque python spaCy pour la tokenisation.

2 TF-IDF

Exercice 1 Tokeniser avec spaCy

Référez-vous à cette URL pour installer spaCy : `https://spacy.io/usage`

Vous installerez au moins le modèle pour la langue française. Laissez les autres options avec leur valeur par défaut.

1. Écrivez un petit programme de test et importez le module `spacy` pour vérifier qu'il est bien installé.
2. Tapez la ligne `nlp = spacy.load("fr_core_news_sm")`. Remplacez éventuellement le nom du modèle par celui téléchargé. Cette variable `nlp` sera votre point d'entrée vers spaCy.
3. Parsez une chaîne de caractère avec `doc = nlp("Un jour je serai le meilleur dresseur")`. Cette fonction renvoie un objet de type `Doc` représentant votre document.
4. Itérez sur ce `doc` et affichez chaque élément. Que voyez vous ?
5. Utilisez la fonction prédéfinie `type` pour afficher le type du premier élément du document `doc[0]`.
6. Ces objets de type `Token` représentent les tokens et ont beaucoup d'attributs et méthodes intéressants. Affichez l'attribut `lemma_` de tous ces tokens. Ce sont ces lemmes qu'on utilisera dans la suite de ce TP en tant que tokens.

Exercice 2 TF

Dans cet exercice, on implémentera le calcul de la fréquence des termes dans un document.

1. Dans un nouveau programme, chargez et concaténez les deux fichiers `kaamelott_01.txt` et `kaamelott_02.txt`. On considérera que chaque ligne est un document distinct. Pour vos tests, vous pourrez vous limiter à quelques lignes seulement.
2. Séparez le texte initial en liste de lignes.
3. Parsez les lignes avec spaCy pour obtenir autant d'objets `Doc`.
4. Récupérer l'ensemble des lemmes produits par spaCy pour tous les documents. Faites-en un `set` (sans doublons).
5. Sachant qu'il y a 7447 lemmes distincts et 12711 documents (répliques), combien d'entrées votre matrice d'incidence terme-document aurait-elle ? Pour le moment, assurez-vous de travailler sur un échantillion assez petit.
6. Construisez la matrice qui compte le nombre d'occurrence de chaque mot dans chaque document. (C'est à dire dans chaque ligne du texte.) Vous aurez une ligne par token, et une colonne par document. Pensez à conserver l'ordre d'apparition des lignes dans votre matrice afin de pouvoir retrouver la ligne de texte par la suite.
7. En reprenant la formule du cours, calculez une nouvelle matrice contenant la valeur de tf pour tous les mots pour tous les documents. N'écrasez pas votre matrice de comptage, vous en aurez besoin.

Exercice 3 IDF

Dans cet exercice on calcule l'inverse de la fréquence de document. Vous aurez besoin de réutiliser votre matrice de comptage.

1. Pour chaque mot du vocabulaire, calculez la *fréquence document* df_t. C'est à dire le nombre de documents qui contiennent ce mot. Combien de documents contiennent le token *« je »* ? Combien contiennent le token *« proverbe »* ?
2. Normalisez (divisez) cette fréquence par le nombre total de documents.
3. Stockez dans un `dict`ionnaire le score idf_t pour tous les tokens.

Exercice 4 TF-IDF et recherches

Dans cet exercice on utilise ceui qui a été calculé dans les deux exercices précédents pour effectuer des recherches.

1. Mettez tout le code des deux exercices précédents dans une fonction. Elle prendra en argument une liste de documents et retournera le vocabulaire, la matrice tf ainsi que le dictionnaire idf.
2. Faites en sorte que votre programme prenne en argument sur la ligne de commande un ou plusieurs mots. Ce sera la requête de recherche à satisfaire.
3. Écrivez une fonction qui calcule le score TF-IDF pour un mot donné et un document donné. Cette fonction devra prendre en argument au moins la matrice tf le dictionnaire idf, le mot à chercher ainsi que le numéro du document.
4. Écrivez une fonction qui évaluera le score TF-IDF pour tous les mots de la requête et tous les documents. Elle pourra stocker tous les scores dans une `PriorityQueue` afin de renvoyer uniquement id des 10 documents les plus pertinents.
5. Testez sur quelques exemples avec un mot. Puis plusieurs mots.

Exercice 5 ***(Bonus)*** **Index inversé**

Dans cet exercice on changera les matrices par un index inversé afin de ne pas stocker les innombrables 0.

1. Au lieu de stocker le nombre d'occurrences dans une matrice avec une ligne par token et une colonne par document, utiliser une liste de `dict`ionnaires. Ce dictionnaire associera un numéro de document (numéro de ligne du texte initial) au nombre d'occurrences de ce token dans ce document. Il ne stockera pas les 0.
2. Modifiez votre matrice de fréquences pour avoir la même structure.
3. Modifiez vos autres fonctions qui utilisent ces structures de données si besoin.
4. Testez de charger des morceaux de texte de plus en plus grand. Notez que spaCy prendra lui aussi de plus en plus de temps. Mettez un `print` quand le parsing par spaCy est terminé afin de savoir si le temps est pris par votre code ou par spaCy.

3 Spherical K-Means

Exercice 6 ***(Bonus)*** **Spherical K-Means**

Dans cet exercice, on implémentera un Spherical K-Means et on clusterisera les répliques de Kaamelott basé sur leur vecteurs calculés avec TF-IDF. Vous pourrez faire usage de numpy si vous savez l'utiliser. Sinon, des listes python suffiront pour représenter les vecteurs et les listes de listes représenteront les matrices.

1. Écrivez une fonction `norm` qui calcule la norme d'un vecteur. C'est à dire qui calcule la somme du carré de ses composants, puis retourne la racine carrée de cette somme.
2. Écrivez une fonction `normalize` qui prend en paramètre un vecteur et renvoie ce vecteur normalisé. C'est à dire qu'il faut diviser chaque composant du vecteur par la norme du vecteur.
3. Écrivez une fonction `cosdist` qui donne la distance cosinus entre deux vecteurs. Testez-la sur quelques exemples simples.
4. Écrivez une fonction `assign_one` qui, étant donné un point et une liste de centroïdes (sous forme de matrice ou liste de liste) renvoie l'index du centroïde dont le point est le plus proche selon la distance cosinus.
5. Écrivez une fonction `assign` qui prend en paramètre une liste de N points (sous la forme d'une matrice ou liste de liste) et une liste de K centroïdes et renvoie une liste d'entiers compris entre 0 et K de même taille que la liste de points. Ces entiers représentent le centroïde affectés à chacun des points.
6. Écrivez une fonction `barycenter` qui prend en paramètre une liste de points et renvoie le point qui correspond au barycentre des points d'entrée.
7. Écrivez une fonction `kmeans` qui prend en paramètre une liste de points et un entier K et qui normalise les points. Faites attention à faire une copie de vos points afin de ne pas écraser les valeurs existantes.
8. Complétez votre fonction `kmeans` pour sélectionner K points au hasard pour être les centroïdes initiaux.

9. Dans votre fonction `kmeans` utilisez la fonction `assign` pour récupérer la liste des numéros des centroïdes affectés aux points.
10. Écrivez une fonction `select` qui prend en paramètre une liste de points, une liste d'affectations et un entier. Elle renverra la liste des points dont le numéro d'affectation est égal à l'entier. En d'autres termes, elle extrait les points appartenant à un cluster.
11. Dans votre fonction `kmeans` rajoutez le calcul du barycentre de vos K clusters. Vous aurez besoin de la fonction `select` pour extraire chacun des ensembles de points affectés aux centroïdes. Ils deviendront par la suite vos nouveaux centroïdes, mais n'écrasez pas encore vous anciens centroïdes. Sauvegardez vos barycentres dans une nouvelle variable.
12. Normalisez ces barycentres.
13. Comparez ces barycentres normalisés avec les anciens centroïdes. Si au moins l'un d'entre eux a bougé, écrasez vos anciens centroïdes avec ces barycentres normalisés, etrecommencez les étapes à partir de la question 9.
14. Si aucun centroïde n'a bougé, retournez la liste d'affectation des points aux clusters.
15. Testez votre fonction `kmeans` en l'appliquant à quelques vecteurs TF-IDF représentant les répliques de Kaamelott.
16. Appliquez maintenant votre fonction à plus de documents, affichez les différents clusters de répliques. Faites varier K jusqu'à ce que les clusters soient cohérents.

Relisez les modalité de rendu dans l'introduction de ce document et assurez-vous de remplir toutes les conditions. :)