TP Accès et Recherche d'information

Ergi Sala, Tom Solvery - INFO4 Polytech

I. Introduction

Lors de ces séances de TP nous avons vu comment construire un modèle de recherche d'information à partir d'un corpus de documents (cacm). Nous avons compris que avant pouvoir traiter une requête, il y a un certain travail à faire, que nous allons détailler par la suite.

II. Prerequis

Avant d'exécuter les programmes, Il faut créer des répertoires suivant cette hiérarchie:

```
|--antiDict/
|--clean/
|--json/
|--split/
|--src/
   |--split_cacm.py
   |--tokenize_cacm.py
   |--courbe.py
  ...
|--cacm.all
|--common_words
...
```

La fenêtre du terminal ou s'exécute le programme ne doit pas changer de taille.

Si vous utilisez Mac, il faut remplacer "xdg-open" par "open" dans la ligne 49 du fichier recherche.py

III. Loi de Zipf

On commence d'abord par repatrier les données. Avec split_cacm.py, en utilisant la fonction ExtractionDesFichiers fournie, on coupe le fichier cacm.all en petits fichiers du type CACM-XX, qui se trouvent dans le répertoire split. C'est la base qu'on utilisera pour notre corpus de documents net plus tard.

Ensuite avec tokenize_cacm.py, en utilisant le tokenizer fourni, on sauvegarde dans un autre fichier du type CACM-XX.flt, dans le répertoire clean, les mots qui commencent par une lettre et qui ne contiennent que des lettres et des chiffres.

Enfin dans courbe.py, en utilisant la fonction zipf, on calcule la fréquence d'apparition de tous les termes de la collection et on les ordonne par le nombre d'apparition décroissant afin d'afficher

a) Les 10 termes les plus fréquents dans l'ordre décroissant avec leur nombre d'occurrences
b) La taille du vocabulaire
c) La valeur lambda théorique calculée.
d) La courbe qui présent le nombre total d'occurrences en fonction du rang de tous les termes.
e) La courbe de Zipf théorique pour les rangs de 1 au nombre de termes du vocabulaire.

IV. Vocabulaire et Présentation

Cette partie débute par le fichier anti-dict.py, contenant une fonction qui applique l'anti-dictionnaire (fichier common-words) sur les mots en minuscule du répertoire clean en enlevant tous les termes de ces fichiers qui apparaissent dans common-words. Le résultat du filtrage est mis dans un fichier portant le même nom que ces fichiers avec une nouvelle extension .sttr. Ces derniers se trouvent dans le répertoire antiDict.

La prochaine étape est de construire le vocabulaire associé à la collection .sttr en considérant tous les termes qui apparaissent au moins une fois. C'est le but du programme vocabjson.py. Le résultat est stocké dans le fichier vocabulaire.json, dans le répertoire json en format {terme: document_frequency}.

Après cela, dans le programme vect.py on fait la représentation vectorielle de tous les documents d'après le modèle vectoriel de Salton. Le type de représentation choisi est une représentation creuse (sparse) de chaque document d avec un dictionnaire de couples (terme, tf.idf du terme dans d), ou idf_i = ln(N/df_i). Le résultat est stocké dans le fichier vect.json dans le répertoire json. On ne se préoccupe pas encore de la normalisation.

On arrive maintenant au point le plus important, la construction de l'index inversé à partir de la représentation vectorielle des documents. En soi la réalisation n'est pas très difficile, parce qu'on a toute l'information qu'il nous faut dans les fichiers vocabulaire.json et vect.json. Il suffit de changer l'organisation des données en format terme: {idDoc: tf.idf dans le doc}. Cela est fait dans le programme indexInverse.py et le résultat est stocké dans le fichier indexInverse.json dans le répertoire json.

Pour préparer le traitement des requêtes, on calcule aussi le norme de chaque document, $||d|| = \sqrt{\Sigma(tf.idf)^2}$. Cela est fait dans le programme norme.py et le résultat est stocké dans le fichier norm.json dans le répertoire json.

V. Recherche

Finalement on va pouvoir traiter des requêtes, avec le programme recherche.py . Pour récupérer une requête tapée à la main dans le terminal, on utilise la librairie python assez pratique curses. On suit la démarche indiquée dans l'énoncé du TP:

1. Charger l'index inversé, le vocabulaire et les normes des documents
2. Boucler tant que requête non vide
   a) Acquérir une requête q (tapée à la main)
   b) Transformer q en une vecteur vq (dictionnaire) avec pondération tf.idf, et stocker la norme de la requête q (= sqrt(somme(carré)))
   c) Calculer le produit scalaire entre requête et document. Grâce à l'index inversé on sait quels documents contiennent le terme et son tf.idf dans le document.
   d) Diviser ces scores par “||d|| * ||q||”
   e) Trier les réponses par ordre de pertinence décroissante
   f) Afficher les M premiers documents les plus pertinents la réponse.

VI. Extensions

Nous avons ajouté deux extensions au programme de base.

1. Dans les résultats de la requête, on affiche sous chaque numéro de document, une quinzaine de mots (ou moins) qu’il contient. Les mots clés (ceux qui font partie de la requête) apparaissent avec une couleur différente du reste.

2. Si l’utilisateur fournit un numéro dans le range des documents du corpus, on va ouvrir le document avec un éditeur de texte pour le visualiser en totalité (bien sûr idéalement sans droit de modification du contenu).

VII. Conclusion

Rechercher de l’information dans un corpus de documents n’est pas si simple qu’il paraît. Il faut d’abord bien organiser et filtrer les documents pour qu’ils contiennent de l’information utile, et puis il faut choisir ceux les plus pertinents en suivant le modèle de recherche.