L2 - Données web

Francois.Rioult@unicaen.fr

15 février 2018

1 Cours

On dispose d'un *corpus* constitué d'une liste de *texte*, issus de l'aspiration de pages web. Les éléments HTML ont été retirés, mais il subsiste une mise en forme avec de la ponctuation et des lignes vides pour délimiter les paragraphes.

Pour valoriser ces textes, un moteur de recherche est mis en place. Il répond à une requête de l'utilisateur, qui est une liste de mots. Dans un premier temps, le moteur doit renvoyer la liste des textes contenant les mots de la requête, mais ce n'est pas suffisant pour l'utilisateur : il faut lui présenter d'abord les documents les plus *pertinents*.

1.1 Pretinence d'un mot

Pour un texte retourné par le moteur, la pertinence d'un mot de la requête, relativement au corpus, est défini par le score TF-iDF du mots, construit à partir des mesures suivantes :

1. soit $TF(word, doc)$ (*term frequency*) le nombre d'occurrences du mot dans le document
2. soit $DF(word)$ (*document frequency*) le nombre de documents dans lequel le mot apparaît
3. soit $nDocument$ le nombre de documents du corpus

Un mot est d'autant plus pertinent dans un document que :
— sa fréquence est élevée ;
— le nombre de document qui le contient est faible.

La pertinence doit donc croître avec le TF mais décroître avec le DF, soit croître avec l'inverse du DF. Dans la pratique, on utilise la formule suivante :

$$TFiDF(word, doc) = TF(word, doc) * \log \frac{nDocuments}{DF(word)}$$

2 Pertinence d'un document

Il est nécessaire de déterminer, parmi les documents qui contiennent la requête, ceux qui ont la meilleure similarité avec la requête. La requête va donc être considérée comme un document, à comparer aux autres.

Un document est représenté par le vecteur des TF-iDF des mots de la requête. La similarité entre deux vecteurs est donnée par le cosinus [1] de l'angle entre les deux vecteurs, qui fournit une valeur entre -1 et 1 :

— 1 : c'est la meilleure similarité
— 0 : les vecteurs sont orthogonaux, il n'y a rien de plus dissemblable
— -1 : les vecteurs suivent des directions opposées

$$\cos(u, v) = \frac{\langle u, v \rangle}{\|u\|.\|v\|}$$

Pour le vecteur de la requête, on prend la liste des iDF des mots ($TF = 1$).

3 TP1

On souhaite appréhender les textes comme des sacs de mots, que l'on appellera des *documents*, *i.e.* comme des ensembles de mots avec leur nombre d'occurrences. On commence donc par traiter les fichiers originaux pour n'en conserver qu'une liste de mots en minuscules que l'on va compter.

3.1 Création du corpus

Récupérez le mini-corpus : une collection de quelques textes.

3.2 Calcul des documents

En utilisant `sed`, transformer un texte en une liste de mots.
— `cat ...` tr [:upper :] [:lower :]| passe en minuscule

3.3 Compter les mots d'un texte

Réaliser une commande comptant le nombre d'occurrences de chaque mot en triant le document (`sort`) puis en comptant le nombre de ligne contenant chaque mot [2]

3.4 Compter les mots du corpus : DF

Réalisez les opérations précédentes sur tout le corpus, pour obtenir les DF.

3.5 Calculer les TF-iDF

Fusionner les résultats en combinant les TF de chaque documents aux DF.

1. dit de Salton
2. en awk, détecter le changement de mot et afficher le compte du précédent, ne pas oublier le dernier mot

3.6 Analyse et commentaire

— Quels sont les mots les plus pertinents pour chaque document ? [3]
— Quels sont les mots les moins pertinents ?
— En temps, quelle est la complexité empirique de cette méthode ? [4]
— En théorie, comment se comporte le calcul de TF-iDF selon le nombre de documents ?
— Quelles sont les pistes d'amélioration en terme de filtrage initial ? [5]

4 TP2

4.1 Calcul de l'index

Pour répondre à une requête, il faut disposer d'un index associant les mots aux documents.

— pour chaque document, générer la liste des mots associés à l'identifiant du document
— fusionner(`cat`), trier (`sort`), compter [6]

4.2 Répondre à une requête

On considère une requête [7] comme une liste de mots, un par ligne, dans un fichier. Écrire une commande qui calcule le TF-iDF de la requête en allant piocher dans les DF.

4.3 Calculer la pertinence de la requête

3. utiliser `sort -k2,2nr` pour définier une clé de tri reverse (r) numérique (n) sur la deuxième colone (de 2 à 2)
4. Testez sur 10, 100, 1000, 10000, 100 000 textes
5. taille des mots, nombre d'occurrence, etc.
6. en awk, signaler le changement dans la première colonne et sortir la liste mémorisée
7. ex. « music again »