Extraction de mots clés et de termes

24 septembre 2019

—Lexicologie, terminologie, dictionnairique—

— informations, lecture :
  — Evert et Krenn (2003) : *Computational approaches to collocations. Introductory course at the European Summer School on Logic, Language, and Information*
  — Pazienza et al. (2005) : *Terminology extraction : an analysis of linguistic and statistical approaches*
  — chapître 3 de QasemiZadeh (2015) : *Investigating the Use of Distributional Semantic Models for Co-Hyponym Identification in Special Corpora*. PhD thesis
— logiciels : TermSuite (extraction statistique), GATE (extraction linguistique, annotation)

Exercices.

1. Ouvrez et complétez le fichier exo1.py selon les instructions en commentaire pour préparer les listes de fréquence.

2. Complétez exo1.py pour écrire les deux listes (sortie standarde) un mot par ligne, en ordre décroissant selon la fréquence. Exécutez-le avec le fichier acl.txt en entrée. Sauvegardez les résultats.

3. Parcourez les 100 mots unigram les plus fréquents. Ne gardez que les mots grammaticaux, supprimez les mots lexicaux et les informations de fréquence. Nous cherchons à obtenir une liste des *stopwords* fréquents, un mot par ligne.

4. Modifiez exo1.py de façon à ce qu'il lise le fichier de stopwords, et lors du calcul il ignore les mots stopwords et les bigrams qui contiennent un stopword. (Attention : les variables N et B devront changer aussi par rapport à 1) !). Re-exécutez exo1.py avec acl.txt et comparez les bigrams les plus fréquents.

5. Complétez exo1.py avec une fonction de normalisation "normalize(word)" qui prend un mot en entrée et
   — transforme les majuscules en minuscules
   — coupe les ponctuations attachées
   Vous pouvez utiliser le module *re* pour la substitution avec une expression régulière. Appelez la fonction à l'endroit approprié dans exo1.py (attention : le nombre des types de mots devra changer) et ré-exécutez. Faites en sorte qu'il ne reste pas de mots qui correspondent à des strings vides après la suppression des ponctuation, ni parmi les mots unigram, ni parmi les composants de bigrams.

6. Complétez exo1.py de façon à ignorer les mots et les bigrams avec une fréquence inférieure à 5. (Attention : N et B changent encore).

7. Nous remplaçons la fréquence par une mesure d'association, la PMI :

$$PMI(a,b) = log\frac{P(a,b)}{P(a) \times P(b)} \tag{1}$$

où la probabilité d'un bigram P(x,y) est estimé à partir de sa fréquence relative dans le corpus :

$$P(x,y) = log\frac{freq(x,y)}{\sum_{bigram} freq(bigram)} = log\frac{freq(x,y)}{B} \tag{2}$$

et la probabilité d'un mot P(x) est estimé :

$$P(x) = log\frac{freq(x)}{\sum_{x} freq(x)} \tag{3}$$

Complétez le programme pour calculer la PMI de chaque bigram et afficher les bigrams en ordre décroissant de PMI.
Pour le calcul de log vous pouvez utiliser *math.log* du module *math*, qui calcule par défaut le logarithme naturel (ln). Utilisez le log à base 10.

8. Spécificité (*termhood*)

— mesurer la spécificité des termes par rapport à un document
tf-idf (term frequency - inverse document frequency) :

$$tf - idf_{t,d} = tf_{t,d} \cdot idf_{t,D} \tag{4}$$

où $tf_{t,d}$ est la fréquence du terme t dans document d et

$$idf_i = log\frac{|D|}{|\{d \in D : t \in d\}|} \tag{5}$$