TME 1 - Indexation

Exercice 1 – Exercice de compréhension : indexation d'un petit jeu de données

On considère les documents présentés lors du TD1 :

- Doc 1 : the new home has been saled on top forecasts
- Doc 2 : the home sales rise in july
- Doc 3 : there is an increase in home sales in july
- Doc 4 : july encounter a new home sales rise

Ainsi qu'une liste de mots vides : the, a, an, on, behind, under, there, in, on.

Q 1.1 Ecrire le code qui à partir de la chaîne de caractère du document 1 : 1) sépare les mots, les transforme en minuscule, compte le nombre d'occurrences par mot dans le texte, 2) supprime les mots vides et 3) stocke le résultat sous la forme :

```
{'new': 1, 'home': 1, 'ha': 1, 'been': 1, 'sale': 1, 'top': 1, 'forecast': 1}
```

Remarque : pour la normalisation des termes, on s'aidera du fichier `porter.py`. Pour compter le nombre d'occurrences d'un terme, on utilisera la librairie `Counter` de `collection`.

Q 1.2 Réaliser les fichiers index et index inversé pour toute la collection de documents.

```
# fichier index :
{0: {'new': 1, 'home': 1, 'ha': 1, 'been': 1, 'sale': 1, 'top': 1, 'forecast':
    1},
1: {'home': 1, 'sale': 1, 'rise': 1, 'juli': 1},
2: {'is': 1, 'increas': 1, 'home': 1, 'sale': 1, 'juli': 1},
3: {'juli': 1, 'encount': 1, 'new': 1, 'home': 1, 'sale': 1, 'rise': 1}}

#fichier index inverse
{'new': {'0': '1', '3': '1'},
'home': {'0': '1', '1': '1', '2': '1', '3': '1'},
'ha': {'0': '1'},
'been': {'0': '1'},
'sale': {'0': '1', '1': '1', '2': '1', '3': '1'},
'top': {'0': '1'},
'forecast': {'0': '1'},
'rise': {'1': '1', '3': '1'},
'juli': {'1': '1', '2': '1', '3': '1'},
'is': {'2': '1'},
'increas': {'2': '1'},
'encount': {'3': '1'}}
```

Q 1.3 Modifier le code pour effectuer une pondération tf-idf.

Exercice 2 – Parsing d'un fichier

On considère le fichier cacmShort.txt qui contient une collection de documents. On souhaite récupérer le contenu des balises ".I" (identifiant du document) et ".T" (texte du document).

Remarques : seule la balise ".I" est obligatoire, il est possible que ".T" n'existe pas pour chaque document. Dans la balise ".T", les textes ne sont pas tous sur la même ligne.

Q 2.1 Ecrire une fonction `buildDocCollectionSimple` qui lit le fichier ligne par ligne et récupère le contenu des balises. Pour chaque document, afficher son identifiant et son texte.

Q 2.2 Ecrire une fonction `buildDocumentCollectionRegex` qui lit l'intégralité d'un fichier, sépare les documents en fonction de la balise ".I" et récupère le contenu des balises ".I" et ".T" à partir d'expressions régulières.

Exercice 3 – Projet "Moteur de Recherche" : Etape Indexation

Sur l'ensemble des 5 premières séances de TME du semestre (et à poursuivre à la maison), vous aurez à développer les différents composants d'un moteur de recherche.
Dans ce TME, nous nous intéressons au package `indexation` permettant l'indexation de différentes collections.

1 Jeux de données

Vous devez obligatoirement considérer les deux collections CACM et CISI. Mais vous pouvez en considérer d'autres si vous le souhaitez. Ces deux repertoires contiennent trois fichiers chacun contenant, selon l'extension :

- .txt : Les documents de la collection (cisi.txt contient 2460 documents, cacm.txt en contient 4204) ;
- .qry : Des jeux de tests de requêtes (cisi.qry contient 112 requêtes, cacm.qry en contient 64) ;
- .rel : Les jugements de pertinence pour les requêtes de test.

Les fichiers des deux bases suivent le même format. Les documents donnés dans les fichiers cisi.txt et cacm.txt, suivent un format défini selon les balises suivantes :

- La balise .I repère le début d'un document en en donnant l'identifiant (important car utilisé par les jugements de pertinence des requêtes test) ;
- La balise .T donne le titre du document ;
- La balise .B donne la date de publication ;
- La balise .A donne l'auteur du document ;
- La balise .K donne les mots-clé du document ;
- La balise .W donne le texte du document ;
- La balise .X donne les liens du document : une ligne par lien, seul le premier nombre (qui correspond à l'identifiant du document pointé par le lien) est utile.

Attention : excepté la balise .I qui est toujours présente, tous les autres champs sont facultatifs et peuvent être absents de la définition d'un document (penser donc à rester suffisamment flexible lors de la lecture des collections). Nous détaillerons les deux autres fichiers dans d'autres TPs.

2 Parsing de la collection

On souhaite ici parser la collection pour récupérer l'identifiant et le texte du document, et éventuellement d'autres données.

Q 3.1 Les documents sont stockés sous forme d'objets `Document` pour lesquels on peut accéder aux valeurs de l'identifiant et du texte. Vous pouvez également stocker les autres métadonnées qui permettront d'avoir un affichage plus complet. Implémenter la classe `Document` qui servira de base au parsing de la collection.

Q 3.2 Implémenter la classe `Parser` qui permet de parser la collection stockée sous la forme d'un dictionnaire de `Documents`.

3 Indexation

On souhaite ensuite construire les fichiers index (index + index inversé) des collections qui ont été parsées.

Les fichiers index devront être écrits dans des fichiers.

Q 3.3 Implémenter la classe `IndexerSimple` qui permet d'indexer une collection dans une méthode `indexation`. Pour cela, vous pourrez utiliser la classe `TextRepresenter.py` associée à la classe `porter.py` qui permet de normaliser une chaîne de caractères.

Q 3.4 Modifier la méthode indexation pour qu'elle permette de construire en plus l'index et l'index inversé normalisé.

Q 3.5 La classe `IndexerSimple` doit en outre contenir à minima les méthodes suivantes :

- Une méthode `getTfsForDoc` qui retourne la représentation (stem-tf) d'un document a partir de l'index ;
- Une méthode `getTfIDFsForDoc` qui retourne la représentation (stem-TFIDF) d'un document a partir de l'index ;
- Une méthode `getTfsForStem` qui retourne la représentation (doc-tf) d'un stem a partir de l'index inverse ;
- Une méthode `getTfIDFsForStem` qui retourne la représentation (doc-TFIDF) d'un stem a partir de l'index inverse ;
- Une méthode getStrDoc qui retourne le texte du document

Q 3.6 Tester l'ensemble de votre code dans un fichier `Test.py`

4 Bonus

On notera que l'indexation a été codée de façon à garder en mémoire l'index, ce qui est possible pour des petites collections de documents. Dans le cas d'une indexation sur une grosse collection, une des solutions est la suivante : utiliser notre propre structure, que nous définissons à l'aide de flux de lecture à accès direct tels que proposés par la classe File en Python : un accès direct à une position spécifique du fichier se fait en O(1) par la méthode seek. Pour créer un index inversé dans ce contexte, une solution est de parser deux fois l'ensemble des données d'entrée : une première passe permet de calculer les positions des différents éléments dans le fichier, une seconde passe permet d'écrire effectivement les informations aux positions prédéfinies.