TD3 : web scraping et introduction à la recherche d'information dans un corpus de textes

Télécharger l'archive *TD3.zip* depuis l'espace-cours Cursus et décompresser cette archive dans un répertoire personnel. Pour la partie webscraping, depuis les machines du réseau MASS vous devrez passer un proxy en fournissant vos identifiants : utiliser le script *td3_scraping_proxy.py*. Sinon utiliser le script *td3_scraping.py.*

Exercice 1 : récolte d'information sur le web

On souhaite obtenir un corpus de textes sur les énergies renouvelables depuis des ressources du web.

Pour cela on se propose de partir de la page Wikipédia présentant la catégorie « Énergie renouvelable » disponible à cette adresse :

https://fr.wikipedia.org/wiki/Catégorie:Énergie_renouvelable

Les pages de catégorie de Wikipédia contiennent une première partie présentant d'éventuelles sous-catégories, puis une deuxième partie présentant des pages associées à la catégorie.

Pour simplifier la tâche, nous n'explorerons que les pages de la catégorie elle-même, en laissant de côté les sous-catégories.

1. A l'aide du module BeautifulSoup, écrire le code Python permettant de récupérer une liste de toutes les URLs des pages de la catégorie « Énergie renouvelable »
2. A partir de cette liste d'URL, écrire le code Python permettant de récupérer le titre de chaque page, et l'ensemble du texte de la division d'attribut id="mw-content-text", à l'exception du texte contenus dans les éléments de classe 'toc' (zone pour le sommaire), 'mw-editsection' (zones pour les liens [modifier | modifier le code]), 'mwe-math-element' (zones éventuelles de formules mathématiques), 'bandeau-portail' (bandeau de navigation vers les autres portails Wikipédia, en bas de la page).
3. Mémoriser l'ensemble des informations récoltées (url, titre et texte) dans un dictionnaire « docs » possédant trois clés associées aux trois listes d'informations :

   ```
   doc = {"url" : [] , "titre" : [], "texte":[]}
   ```

   et enregistrer ce dictionnaire dans un fichier à l'aide du module pickle.

Exercice 2 : recherche d'information dans un corpus de texte

Dans cet exercice, on se propose de simuler de façon simplifié le fonctionnement d'un moteur de recherche.

Notre ensemble de documents est celui récupéré sur le web à l'exercice 1. Nous allons construire à partir de ce corpus une matrice document-terme (DTM : Document-Term Matrix) représentant la fréquence de chaque terme du corpus dans chaque document du corpus.

Le fichier *td5_analyse.py* contient déjà une fonction *getTokens( )* prenant en argument un texte et retournant la liste des termes de ce texte, en minuscule, et en enlevant la ponctuation.

1. Écrire une classe DTM dont le constructeur prend en paramètre le dictionnaire enregistré à la fin de l'exercice 1 et possédant les attributs suivants :

- url : listes des URLs des documents
- title : liste des titres des documents
- data : une *DataFrame* de pandas représentant la matrice document-terme du corpus. Les indices de ligne de cette DataFrame seront les indices des documents et les intitulés de colonne seront les termes du corpus. Remplacer, dans cette DataFrame, les valeurs manquante NaN par la valeur 0 (cas où aucune occurrence du terme représenté par l'intitulé de colonne n'a été trouvée dans le document de la ligne correspondante).

2. Ajouter à cette classe la méthode *__repr( )__* pour que l'affichage d'un objet DTM renvoie la *DataFrame* contenu dans l'attribut data.
3. Ajouter à la classe DTM une méthode *nBest( )* prenant en argument un nombre entier *N* et renvoyant la liste des *N* termes les plus fréquents dans le corpus entier, avec leur fréquence, par ordres décroissant des fréquences. *Indication : utiliser la méthode sum() des DataFrame pandas.*
4. Rajouter une deuxième méthode *nBestDoc( )* prenant en deuxième argument l'indice (entier) d'un document et renvoyant la liste des *N* termes les plus fréquents dans ce document, avec leur fréquence, par ordres décroissant des fréquences.
5. Testez les deux méthodes précédentes en observant les 10 termes les plus fréquents pour le corpus entier et pour différents documents. Concluez.
6. Utilisez la liste de mots contenus dans le fichier *mots_vides.txt* pour les exclure de la matrice document-terme. Ces mots sont appelés mots-vides car il ne porte pas de sens en eux-même et ne servent pas à définir le sujet d'un texte. Ils sont donc exclus de l'analyse.

   Ajoutez la liste de mots-vide comme nouvel attribut de la classe DTM : attribut *stopWords*.

   Testez à nouveau les deux méthode nBest() et nBestDoc() après avoir exclu ces mots-vides.
7. Ajouter à la classe DTM une méthode *query( )* prenant en paramètre une requête (chaîne de caractères) représentant une liste de mots séparés par des espaces, et renvoyant la liste des documents contenant l'ensemble des mots de la requête.
8. Pour pouvoir classer les résultats renvoyés, il faut être capable de leur attribuer un score, c'est à dire donner à chaque terme de la requête un score pour chaque document (le score global sera la somme des scores de chaque terme). On utilise souvent comme score d'un terme pour un document, l'indicateur appelé **tf.idf** :
   - **tf** est la « term frequency » qui représente la fréquence du terme dans le document. On utilisera une version normalisée de cette fréquence pour prendre en compte la longueur du document, par exemple la fréquence brute divisée par la fréquence du terme le plus présent dans le document.
   - **idf** est la « inverse document frequency » qui représente l'inverse du nombre de document dans lesquels apparaît le terme. Il permet de donner plus d'importance à des termes rares dans le corpus au travers des documents.

     La formule finale utilisée sera la suivante pour un terme *t* et un document *d* :

     - $\text{tf .idf}_{td} = (\text{tf}_{td} / \max_i(\text{tf}_{id})) \times \log(N/\text{df}_t)$ où N est le nombre total de documents.

Remplir la matrice document-terme (attribut *data* des objets DTM) avec le **tf.idf** ci-dessus et utiliser ces scores pour classer les documents résultats de la requête, en sortie d'une nouvelle méthode *queryScore()*. Le score global de la requête sera la somme des scores **tf.idf** de chaque terme.