Projet programmation Python – Sujet 1

Vous trouverez le code de l'application à l'adresse suivante :
https://github.com/NRipahette/Projet_Python_Triki_Ripahette

Spécifications

L'outil développé pour ce projet permet à un utilisateur de récupérer des documents publiés sur Reddit et Arxiv.

L'application réalise une requête sur ces 2 bases en utilisant les APIs respectives. Les données sont instanciées dans la classe corpus qui celui-ci va instancier chaque document dans sa classe respective.

Quand les documents sont instanciés nous pouvons comparer des corpus entre eux. Pour cela nous avons développé la fonction evolution_temporelle qui nous permet de trouver les mots clés les plus importants du corpus et de voir son évolution au fil du temps.

On peut aussi comparer 2 corpus entre eux a l'aide de la fonction comparer_corpus qui nous permet de voir cote a cote les mots les plus importants des chaque corpus.

Analyse

Environnement de travail

L'environnement de travail utilisé a été Spyder avec Anaconda. Cet environnement nous a facilité le développement de l'application grâce aux nombreuses fonctionnalités qu'il offre.

Par exemple, Anaconda permet de contrôler les librairies utilisées par le programme ainsi que leur version avec le gestionnaire de paquets conda.

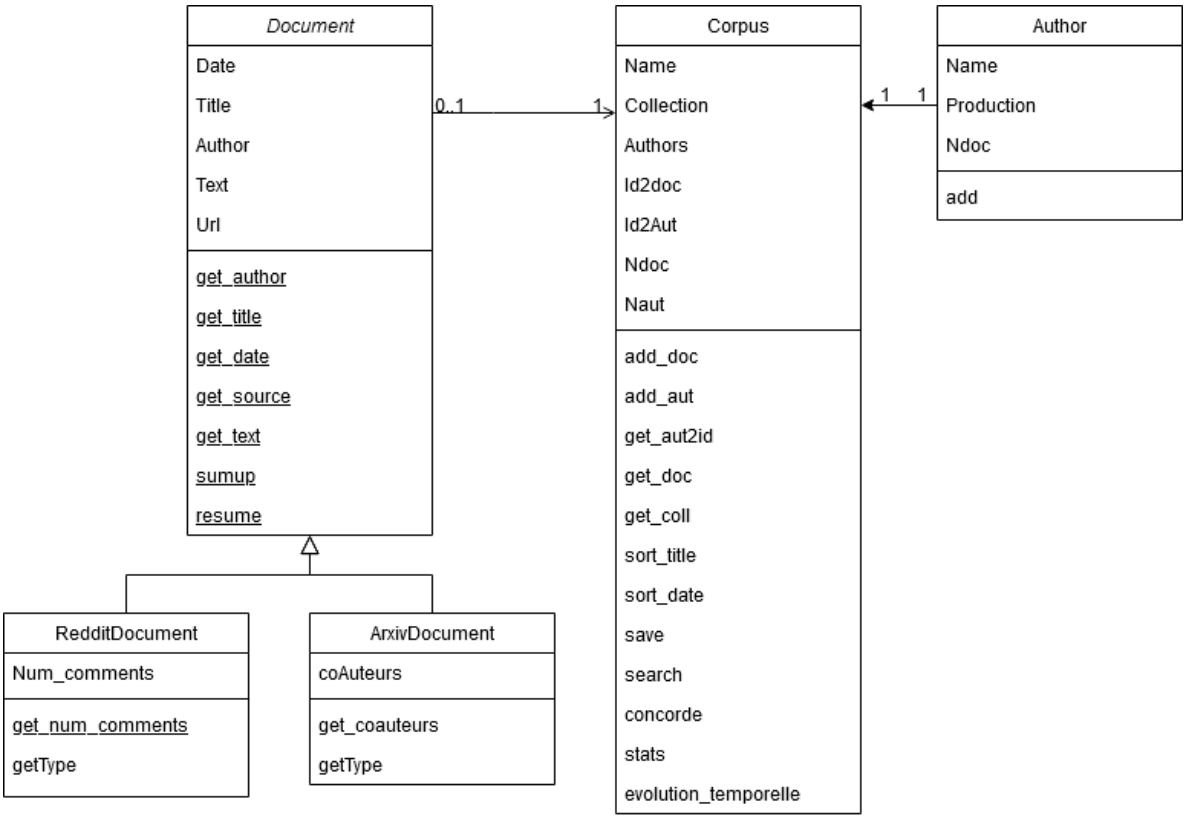
Spyder quant à lui possède un éditeur de texte classique. Cependant son avantage est la fenêtre située en haut à droite qui nous permet d'analyser les variables en temps réels ce qui nous a fait gagner du temps quand il était nécessaire de débugger.

Les données utilisées

Comme mentionné précédemment, les données sont extraites de Reddit et Arxiv. Quand l'application reçoit la réponse de la requête de l'API, celle-ci va créer une instance de chaque document qui en suite sera ajouter à la classe Corpus grâce à la méthode corpus.add_doc(). En fonction de la source du document, celui-ci sera soit de la classe fille RedditDocument() soit ArxivDocument()

L'application aussi crée instancie des objets de la classe Author lors de l'ajout d'un document dans le corpus si l'auteur n'existe pas.

A partir des ses classes nous pouvons utiliser les méthodes ci-dessous pour les récupérer des informations sur les corpus et documents :



Conception

Répartition des taches

Noé Ripahette	Arthur Triki
- Fonction de mesure TF-IDF - Implémentation des fonctions dans l'interface - Interface Tkinter et graphiques (app.py)	- Finalisation de la partie 4 ○ Functions search, concorde, stats, summarize, et nettoyer_texte - Fonction de comparaison des corpus - Evolution temporelle d'un mot donné par période

Explication des algorithmes

L'algorithme de mesure TF-IDF

La mesure TF-IDF est séparée en deux parties, TF (*term frequency*) et IDF (*inverse document frequency*) wikipedia.org nous dit concernant TF que : « La fréquence "brute" d'un terme est simplement le nombre d'occurrences de ce terme dans le document considéré » et concernant IDF : « La fréquence inverse de document (*inverse document frequency*) est une mesure de l'importance du terme dans l'ensemble du corpus. Dans le schéma TF-IDF, elle vise à donner un poids plus important aux termes les moins fréquents, considérés comme plus discriminants. Elle consiste à calculer le logarithme de l'inverse de la proportion de documents du corpus qui contiennent le terme »

La fonction tf_idf() ici fonctionne avec la classe Corpus, elle prend en paramètres deux instances de la classe Corpus, de là on vas faire quelques opération pour extraire les textes, d'abord on crée des "sacs" de mots car c'est comme ça que nous allons pouvoir compter et reconnaitre les mots dans les textes. Et depuis ces sacs on crée une liste de tous les mots répertoriés dans les deux corpus. Comme le « vocabulaire » du TD.

```
def tf_idf(CorpusA,CorpusB):
    #On crée un sac de mot par corpus de textes.
    sacA=[]
    for i in range(0,CorpusA.ndoc):
        text = nettoyer_texte(CorpusA.get_doc(i).get_text())
        sacA.extend(text.split(" "))

    sacB=[]
    for i in range(0,CorpusB.ndoc):
        textb = nettoyer_texte(CorpusB.get_doc(i).get_text())
        sacB.extend(textb.split(" "))

    #On crée une liste des mots qui appartiennent aux corpus
    listeMots = set(sacA).union(set(sacB))
```

Ensuite le calcul de fréquence des mots pour chacun des corpus on stocke les scores dans *TFDictA* et *TFDictB*

```
#On compte le nombre d'occurences de chaque mot dans chaque corpus
NbMotA = dict.fromkeys(listeMots,0)
NbMotB = dict.fromkeys(listeMots,0)

#On ne comptabilise pas les mots qui font parties des stopwrods comme "the" "and" "or" etc....
for mot in sacA:
    if mot in stopwords.words('english'):
        pass
    else:
        NbMotA[mot] +=1
for mot in sacB:
    if mot in stopwords.words('english'):
        pass
    else:
        NbMotB[mot] +=1

#Calcul de la Fréquence du mot (TF)
TFDitcA={}
TFDitcB={}
tailleSacA = len(sacA)
tailleSacB = len(sacB)

for mot,n in NbMotA.items():
    TFDitcA[mot] = n /float(tailleSacA)
for mot,n in NbMotB.items():
    TFDitcB[mot] = n /float(tailleSacB)
```

Puis, le calcul de l'IDF. Le score calculé ici est influencé par les deux corpus (nombre de textes par corpus), chaque mot à donc son score général et donc il n'y a qu'un dictionnaire regroupant les scores à la fin de cette étape (*dictIDF*)

```
#Calcul IDF
# nombre de textes dans lesquels un mot apparait / nombre de textes
dictIDF = dict.fromkeys(listeMots,0)
for doc in range(0,CorpusA.ndoc):
    motsA=[]
    stopDict = dict.fromkeys(listeMots,0) # permet de savoir si le mot à déjà été vu dans le document
    text = nettoyer_texte(CorpusA.get_doc(doc).get_text())
    motsA.extend(text.split(" "))
    for mot in motsA:
        if stopDict[mot] == 0:
            dictIDF[mot] += 1
            stopDict[mot] = 1

for doc in range(0,CorpusB.ndoc):
    motsB=[]
    stopDict = dict.fromkeys(listeMots,0) # permet de savoir si le mot à déjà été vu dans le document
    text = nettoyer_texte(CorpusB.get_doc(doc).get_text())
    motsB.extend(text.split(" "))
    for mot in motsB:
        if stopDict[mot] == 0:
            dictIDF[mot] += 1
            stopDict[mot] = 1

for mot, n in dictIDF.items():
    dictIDF[mot] = math.log((CorpusA.ndoc + CorpusB.ndoc)/ float(n))
```

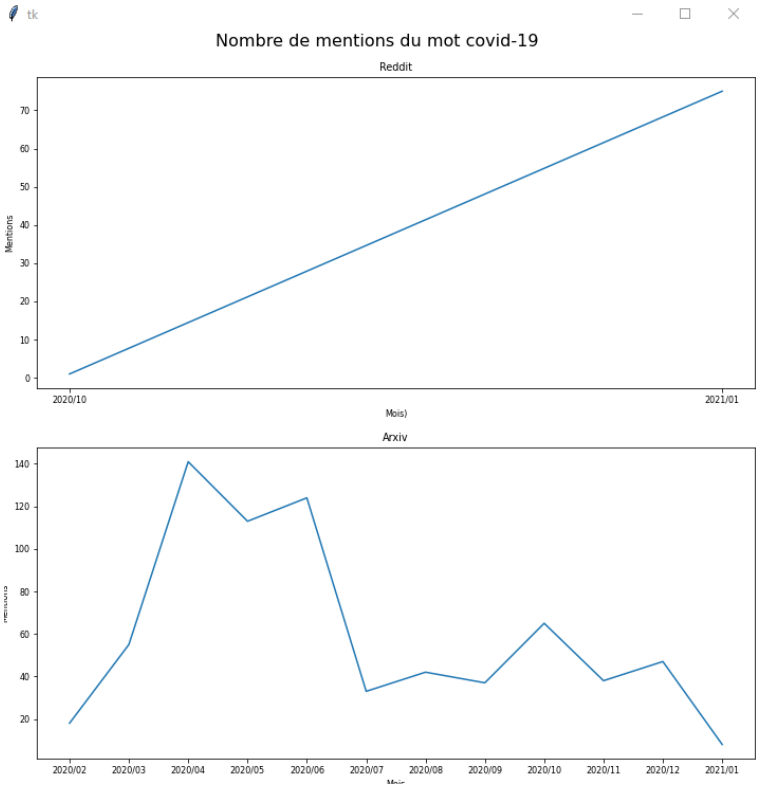
Et enfin, on met en commun les deux scores pour donner le score d'importance final de chaque mot du corpus. Retourné dans un nouveau dictionnaire par corpus.

```
#Calcul TF-IDF
tfidfA = {}
tfidfB = {}

for mot , score in TFDitcA.items():
    tfidfA[mot] = score * dictIDF[mot]
for mot , score in TFDitcB.items():
    tfidfB[mot] = score * dictIDF[mot]

return tfidfA, tfidfB
```

Problèmes rencontrés

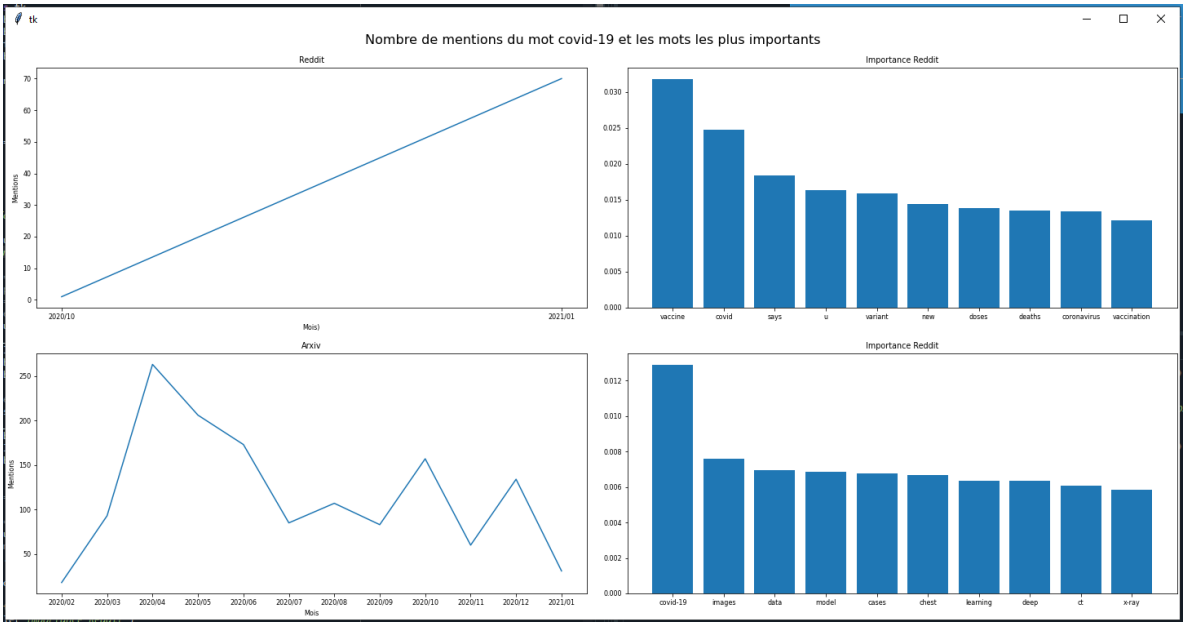
Lors de la création du corpus nous interrogeons l'API, puis pour chaque article dans le subreddit Coronavirus on récupère les données nécessaires (auteur, titre, …). Cependant tous les articles qu'apparaissent dans ce subreddit concret n'ont pas de champs « body » (ils redirigent vers une autre page) donc nous avons utilisé que le nom de l'article pour rechercher les mots les plus fréquents.

Cela impacte directement nos analyses puisque sur un titre nous il est très compliqué de retrouver les mots importants de la phrase et donc avoir une analyse correcte. Nous pouvons observer dans le graphe ci-dessous que le mot clé « covid-19 » n'apparait que dans quelques publications en Octobre 2020 et Janvier 2021.



Exemple d'utilisation du programme

L'utilisation conseillée est d'exécuter le script app.py , ce qui ouvrira l'interface avec les représentations graphiques des résultats.



Donc après avoir récupéré et traité les données de Reddit et Arxiv on peut observer le nombre de fois que le mot clé est apparu mois par mois et voir les termes les plus importants de chaque corpus.

Par la suite on peut à l'aide de la méthode evolution_temporelle observer le nombre de fois que le mot clé est apparu mois par mois.

En haut du script "app.py" on peux modifier le mot dont on veut connaitre la fréquence durant 2020. Ici *mot = "covid-19"*.

```
1   # -*- coding: utf-8 -*-
2
3   import tkinter as tk
4   import matplotlib
5   import matplotlib.pyplot as plt
6   from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
7   import Corpus
8   import collections
9
10
11  mot = "covid-19"
```

Améliorations

Afin d'améliorer l'application nous pourrions développer une interface graphique plus complète qui permettrai de d'indiquer le corpus que nous voulons rechercher, la période pour la quelle nous voulons voir l'évolution temporelle du mot clé et le nombre de résultats que nous voulons obtenir de chaque source.

L'intégration d'un ou plusieurs champs de paramétrage des requêtes dans l'interface plutôt que de devoir manipuler les scripts

Ces évolutions sont faciles à mettre en place avec un peu de connaissances d'une librairie graphique. Pour cela l'application récupèrera le corpus que nous voulons chercher et le passera en paramètre a la boucle qui crée le corpus et les documents.

Il est aussi nécessaire de développer des tests unitaires et globaux pour vérifier que l'application fonctionne correctement.