IN104 Projet

DELOFFRE Arthur, CARUEL Pierrick

Mai 2021

Github :

https : $//github.com/a-deloffre/IN104\_Caruel\_Pierrick-Deloffre\_Arthur.git$

Table des matières

1 Introduction

Durant des années, les questions de santé ont été étudiées conjointement par les médecins et les chercheurs. Ceux ci utilisent les nouvelles technologies afin de répandre rapidement les connaissances et les différents progrès médicaux, notamment à l'aide du web.

Cependant, la situation pandémique actuelle a forcé la mobilisation de milliers d'institutions et de scientifiques afin de lutter contre le virus, et la documentation sur le sujet a considérablement augmenté.

2 But du projet

Le but du prjet est de créer un moteur de recherche spécialement conçu pour la recherche d'articles portant sur le Covid 19. En effet, le nombre d'articles médicaux sur ce sujet a augmenté de manière exponentielle, approximant les 100 000. La hausse du nombre de documents numériques sur le web rend donc la recherche d'articles de plus en plus complexe, et un outil afin de faciliter la recherche de données permettrait un gain de temps considérable.

Le moteur codé pour répondre aux attentes du moment va prendre une base de données composée d'articles scientifiques, les simplifier et effectuer une recherche dans ces nombreux articles à l'aide de mots clés.

3 Répartition des tâches

Durant ce projet, nous avons travailler conjointement afin d'être productif et de pouvoir rendre un travail abouti. Cependant, rien ne s'est passé comme prévu. En effet, Arthur a eu beaucoup de mal à utiliser Python à partir de Cygwin, ce qui l'a énormément retardé au niveau de l'avancée des séances. De mon côté, cela marchait encore moins qu'Arthur et ai donc commencé à coder le projet sur Spyder. J'ai donc réalisé en majorité la Partie 1 avec l'aide d'Arthur qui essayait de m'aider tout en résolvant ses problèmes. Dès qu'il a fini, il s'est tout de suite penché sur la 2e partie du projet, pendant que je finalisais la première. Ayant fini avant qu'il finisse la deuxième partie, je me suis donc occupé d'écrire le rapport. Arthur a ensuite préparé la présentation orale.

Nous aurions préféré avancer ensemble sur le projet, mais cela n'a malheuresement pas été possible et avons dû nous adapter tout au long des séances. Cependant, il s'avère que nous avons réalisé la même quantité de travail et que tout s'est bien déroulé.

4 Méthodes utilisées

4.1 Partie 1

Dans cette partie, nous avons manipulé 4 fichiers différentes : *Preprocess-Data, Utils, Extractdata* et *Main*. Le but de cette partie est, à partir d'un fichier texte, de créer un nouveau fichier texte plus simple afin de faciliter la recherche du moteur de recherche codé.

La classe *ExtractData* va récupérer toutes les données intéressantes d'un fichier, c'est-à-dire sont identifiants, son titre et son contenu.Pour récupérer le contenu et coder cette récupération, nous aurions dû regarder dès le début à quoi ressemblaient les articles scientifiques de la base de données (*.xml.json). En effet, la partie intéressante de ceux-ci était ce qui était appelé *text*, mais nous avons mis du temps avant de le remarquer. Cela induisait beaucoup d'erreurs lors de la compilation. Avec l'aide de notre tutrice, nous avons pu mieux comprendre comment lire ces fichiers, et ainsi corriger nos erreurs. Il a ensuite suffit de coder une boucle *for* parcourant l'article afin de soutirer les informations.

preprocess_data est une classe qui contient toutes les fonctions de traitements de texte, à savoir le passage en minuscule, la suppression des chiffres, des caractères spéciaux, de la ponctuation et des mots non essentiels, la *lemmatization*...Pour la coder, il a suffit d'écrire une fonction différente pour chaque caractère particulier que nous voulions supprimer (*stop_words*, ponctuation, chiffre... etc). Ces fonctions de suppression commencent par vérifier que leur argument est bien une chaîne de caractères avant de la parcourir mot à mot pour supprimer les caractères et mots indésirables. Cette fonction a été améliorée par la suite (cf [5.4])

Utils est un fichier qui contient des fonctions permettant de traiter du texte, notamment voir si un caractère est un chiffre, ou encore lire le fichier initial et créer un nouveau fichier que l'on a écrit. Cette dernière fonctionnalité a du être codée. Pour cela nous avons dû ouvrir un nouveau document à l'aide de la fonction *open* (cf [5.2]) et utiliser la fonction *file.write()* afin de pouvoir créer le nouveau fichier texte .

Enfin, le *Main* permet de lier les 3 fichiers précédents : il ouvre tout d'abord chaque fichier un par un à l'aide de $read\_json()$ de *Utils*, il utilise *extractData* afin de trouver les différentes informations du fichier qu'il doit traiter (le titre, l'identité et le contenu), puis traite le contenu avec *preprocess_data* (passe en minuscule, supprime ce qui est indésirable et *lemmatize*) et *Utils* (crée le nouveau fichier épuré et le sauvegarde). A la fin, un nouveau fichier épuré et pratique pour la recherche a été généré.

4.2 Partie 2

La partie 2 de ce projet permet la mise en place du moteur de recherche à proprement parler. En effet, c'est cette partie qui va permettre à l'utilisateur de récupérer les emplacements des articles contenant le mot (ou la suite de mots) qu'il recherche. Dans notre cas, le module *whoosh* est utilisé afin de réaliser cette recherche, et correspond aux premières difficultés que nous avons rencontrées. Le fonctionnement d'un moteur de recherche avec *whoosh* nécessite de mettre en place un schéma et de construire un index. Bien que la fonction réalisant ces tâches dans notre programme (cf. *index.py*) était déjà implémentée, il nous fallait la comprendre (ou du moins essayer) afin de mieux maîtriser les objets utilisés. Néanmoins, cela a nécessité plusieurs recherches sur Internet et discussions avec d'autres groupes, et la compréhension de ce module a été un réel casse-tête. Notre connaissance de *whoosh* est d'ailleurs encore aujourd'hui limitée : nous en comprenons désormais les grandes lignes, ce qui est malgré tout suffisant ici.

Voici donc un résumé du fonctionnement de ce moteur de recherche.

Il se découpe en 3 fichiers distincts : *index*, *query* et *main_part*2. A noter que le fichier prédéfini *redefineIndex* n'a pas été conservé dans le projet final, puisqu'il réalise à nos yeux une tâche pouvant être programmée directement dans la fonction qui l'utilise.

Index

Comme précisé ci-dessus, la fonction *create_index* a été la plus difficile à comprendre globalement sur le projet, en raison des nombreux objets et fonctions qui nous étaient inconnus. Elle a pour mission de construire un index à partir des paramètres qui le définissent (emplacement du dossier, nom, nombre de documents traités et leurs noms, schéma de base). Tout d'abord, on vérifie qu'un index n'a pas déjà été créé dans le dossier spécifié : si ce n'est pas le cas, on utilise la fonction *create.in* de *whoosh.index*, qui retourne un index dans ce dossier à partir du schéma.

Rq. : ici, le schéma est du type (emplacement sur le PC ; contenu texte simplifié de l'article).

Un premier problème s'est alors posé : notre manque de compréhension au départ a fait que nous ne supprimions pas l'index précédent avant de relancer une nouvelle fois le code, ce qui provoquait des erreurs que l'on ne comprenait pas. En discutant avec d'autres groupes, nous avons compris notre erreur, et nous avons pu nous focaliser sur les autres problèmes de notre code.

Lorsque l'index est introduit, on peut y « ajouter » chaque document un par un (dans la limite du nombre de documents au sein desquels on veut effectuer une recherche). Pour cela, il est nécessaire de récupérer son emplacement (que l'on connait déjà) et son contenu texte. Chaque article va être ouvert avec la fonction *open*, et lu avec la méthode *read*, qui nous retourne bien le contenu texte du document. On peut donc ensuite utiliser *writer.add_document*, pour...

ajouter le document à l'index. Lorsque tous les documents sont traités, on ouvre le dossier de l'index afin de le manipuler, et ce grâce à *open_dir*. Dans cette partie de la fonction, il était important de bien préciser la manière dont le contenu des documents était récupéré. A l'origine, un fichier supplémentaire complétait le dossier de base pour participer à la réalisation de cette tâche. Néanmoins, son intérêt ne nous semblait pas clair, puisqu'il était possible de faire sans. Ainsi, nous ne l'avons finalement pas utilisé, pour garder une totale compréhension de notre projet. En revanche, il nous a semblé judicieux de conserver le *timer* introduit dans la fonction originelle. En effet, celui-ci permet de connaître la durée pendant laquelle le code tourne, ce qui fait partie des informations importantes pour juger de la qualité du code. L'affichage qui lui était associé a aussi été conservé par souci esthétique.

Query

Aucune intervention n'a été nécessaire pour ce fichier, qui permet de définir une classe *Query* à partir de notre requête. Cette classe contient deux fonctions : *init* et *get_query*, qui va permettre la recherche dans l'index à l'aide des fonctions *parse* et *search*. La première prend en argument la requête, qui est une suite de mots, et la divise en « résultats ». Par exemple, la requête « number of cases in the UK » devient avec *parse* : *(content : number OR content : case OR content : uk)*. La seconde fonction réalise la recherche dans la variable *searcher* (qui est ici *ix.searcher*, *ix* correspondant à notre index), et retourne une suite d'emplacemnts ('*paths*') correspondant aux contenus contenant la requête ('*texts*'). Finalement, on affiche ces adresses dans une liste qu'on retourne. Le résultat de *get_query* est donc une liste d'adresses relatives aux articles concernés par la requête.

*Main_part*2

Là aussi, peu de choses à dire sur la fonction en elle-même, puisqu'elle consiste à :

1. récupérer les paramètres utiles : emplacement du dossier contenant les articles modifiés, emplacement du dossier index et son nom, le nombre de documents traités, le nombre maximum de documents affichés, la requête.
2. fabriquer le schéma et l'index
3. appliquer *get_query* à notre requête, remodelée au préalable en classe
4. Afficher le résultat

Un problème est malgré tout apparu lors du test ; lorsque l'on affichait *lst_index_docs*, on apercevait un double antislash au lieu d'un slash classique. Notre code étant défectueux, nous avons décidé de modifier manuellement la syntaxe du path des documents.

5 Autres problèmes rencontrés et résolutions

5.1 Utilisation de Python avec Cygwin

Nous avons eu tout les deux des problèmes pour tout d'abord télécharger Anaconda (qui a pu être résolu), puis pour utiliser le shell de Cygwin afin d'écrire des commandes Python. En effet, l'installation du module via le terminal était impossible. Ce problème a provoqué un certain retard dans la progression du code dès le départ.

Afin de pallier ce problème, nous avons utiliser Spyder (Python 3.8) qui était présent dans Anaconda, et n'avons plus eu de problème particulier.

5.2 Changement entre ASCII et Unicode

Dans le fichier *Utils*, un de nos problèmes a été que le texte que la fonction recevait était mal encodé. En effet, il était affiché à cause de notre fonction *open* :
"UnicodeEncodeError : 'charmap' codec can't encode character '\u03b1' in position 5424 : character maps to $< undefined >$".
Il en a été déduit qu'il y avait un problème dans l'encodement, chose que nous n'avions jamais vu auparavant dans les différents cours d'informatique. Cependant, nous avons tout de même trouvé comment régler ce problème en ajoutant un troisième paramètre à notre fonction (encoding='utf8') :

```
with open(path_file + filename + ".txt",'w+', encoding='utf-8') as file:
```

FIGURE 1 – Correction *open*

Cela nous a donc permis de régler ce problème. En effet, il devait y avoir une erreur d'encodement entre ASCII et Unicode qui nous donnait le message d'erreur précédent.

5.3 Code

Le code à écrire n'était pas aussi long que nous le pensions originellement. Malheureusement il était beaucoup plus compliqué que prévu... En effet, les parties de code que nous devions écrire étaient très précises et nous demandaient beaucoup de recherches sur le web ou de débuggage. Cela fut assez frustrant pour nous car nous passions beaucoup de temps sur des petites choses qui nous semblaient faciles, mais que nous mettions une heure à résoudre.

De plus, il y avait parfois des morceaux de code à modifier que nous n'avions pas écrit nous même, ce qui a pu nous complexifier la tâche.

5.4 Trop lent d'analyser tout les fichiers

De plus, compte tenu du nombre de fichier considérable à analyser, l'algorithme mettait beaucoup de temps à compiler, ce qui poussait à croire qu'il n'était pas bien optimisé. Nous avons vu que dans la code de la classe *preprocess_data*, toute les fonctions sont utilisées par le *main*, et chacune d'entre elles a besoin de parcourir mot à mot le fichier texte afin de faire les changements nécessaires. Ainsi, nous avon donc pensé qu'utiliser une seule fonction au lieu de plusieurs permettrait de diminuer le temps de compilation et ainsi d'optimiser en partie notre algorithme. Voici les modifications apportées à l'algorithme initial :

```
def remove_all(self, text):
    """
    It removes the stopwords
    :param text: it is the text of the article
    :type: str
    :return text: article text without stop words
    :rtype: str
    """
    if isinstance(text, str):
        lst_words = [word for word in word_tokenize(text) if word not in self.stop_words and word not in self.punctuation and not is_number(word) and word.isalnum ()]
    return ' '.join(lst_words)
```

FIGURE 2 – Amélioration *preprocess_data*

Ce qui revenait à modifier un peu notre programme *main* comme ceci :

```
    # Remove punctuation, number , stop words, special characters
    new_text=obj_preprocess.remove_all(new_text)
```

FIGURE 3 – Amélioration *main*

Ceci a permis d'améliorer notre algorithme

5.5 Query et Search

Dans la deuxième partie du projet, il est question à un moment d'utiliser Query afin de trouver les mots clés recherchés dans les articles en base de données. Cependant, cela n'a pas marché au début. En effet le problème était le suivant : la liste d'articles trouvés contenant les mot clés est vide. Cela ne peut permettre le bon fonctionnement de l'algorithme comme le montre la photo ci dessous :

```
In [22]: runfile('C:/cygwin64/home/pierr/in104/projet/IN104_Caruel_Pierrick-Deloffre_Arthur/Project/
main_part2.py', wdir='C:/cygwin64/home/pierr/in104/projet/IN104_Caruel_Pierrick-Deloffre_Arthur/Project')
Reloaded modules: redefineIndex, index, query
début
on recherche : viruses body
ce qui est cherché par l algorithme : (content:virus OR content:bodi)
résultat de la fonction search : <Top 0 Results for Or([Term('content', 'virus'), Term('content',
'bodi')]) runtime=0.00013720000015382539>
liste de documents trouvés contenant les mots clefs : []
Fin main
```

FIGURE 4 – Erreur renvoyée

Nous pouvons tout d'abord apercevoir que les mots clef entrés ne sont pas entièrement pris en compte (il manque la fin du mot *viruses* et *body* est mal écrit), et la liste est bel et bien vide. l'algorithme ne rentre donc pas dans la boucle *for* du *main* de la partie 2. Nous pensons que cela venait de la fonction *Search* qui était sensée repérer les articles contenant les mots clefs entrés.
Contrairement à ce que nous pensions, l'erreur venait plus simplement de la syntaxe du *path* des documents que nous avons dû modifier manuellement.

6 Conclusion

Pour conclure, nous avons plutôt apprécié travailler sur ce projet, même si nous avons eu parfois du mal à régler nos problèmes et à avancer. Nous avons beaucoup bloqué sur des petites choses et avons passé beaucoup de temps à les résoudre, mais cela n'a pas nuit à la finalité de notre projet. Le moteur de recherche pour le Covid-19 ainsi créé devrait permettre aux scientifiques et chercheurs de trouver plus rapidement des articles sur le virus, articles qui correspondraient parfaitement à ce qu'ils cherchent.
Cependant, il faudrait que la base de fichiers soit régulièrement mise à jour. En effet, des avancées se font fréquemment et afin d'endiguer la pandémie, il faut que les différentes connaissances se répandent, et donc que tout le monde (surtout les chercheurs) soient au courant des résultats des différentes études faites dans le monde entier. Afin de parfaire le moteur de recherche, il devrait être couplé avec une base de données sur laquelle les scientifiques pourraient ajouter leurs études comme bon leur semble.

7 Discussion

Durant ce projet, nous avons appris à manier le langage Python et de nombreuses fonctionnalités qui nous étaient encore inconnues. En effet, la notion de classe et de méthode sont très peu abordées en classe préparatoire d'où nous provenons, et travailler en Python nous a permis de consolider nos acquis. De plus, nos différentes erreurs nous ont obligés à chercher de la documentation par nous-même sur le web, chose qui ne nous était pas habituel en cours d'informa-

tique, où nos avions l'habitude de nous reposer sur le polycopié du professeur. Mais c'est là aussi le but d'un projet, qui nous a permis de gagner et de nous habituer à l'autonomie.
Ce projet nous a tenu en haleine jusqu'au bout. En effet, il nous restait quelques toutes petites erreurs qui ont finalement pu être corrigée et cela a permis le bon fonctionnement du programme final.

Enfin, ce projet nous a aussi permis de découvrir de nouveaux outils de travail, utiles pour des ingénieurs dans un monde de plus en plus numérisé.