

Projet Python

Spécification :

L'objectif de ce projet était de réaliser une application en python permettant à l'utilisateur d'interagir avec un corpus de documents grâce à une interface web réalisée avec la librairie Dash. Les documents ont été créés à partir du contenu de 2 API : Reddit et Arxiv, il est alors possible à travers l'interface de naviguer entre 2 pages afin d'interagir avec ces dernières. La page « Concordancier » propose d'afficher la concordance de mots ainsi que leurs présences dans les différents documents. La seconde page « Source comparaison » permet de voir l'évolution d'un mot dans le temps ainsi que de filtrer les documents selon différents critères. Une importance toute particulière a été portée sur la programmation orientée objet ainsi que sur différents design patterns.

Nous détaillerons dans un premier temps l'architecture de notre projet, puis dans un second temps l'analyse et la conception de ce dernier. Enfin, nous aborderons la validation de notre application et terminerons par une partie dédiée à la maintenance.

Le lien du dépôt git est accessible à l'url suivante : <https://github.com/FeckNeck/python-project> un fichier README permet d'avoir différentes informations sur le projet ainsi que des consignes pour l'installation de ce dernier.

Architecture du projet :

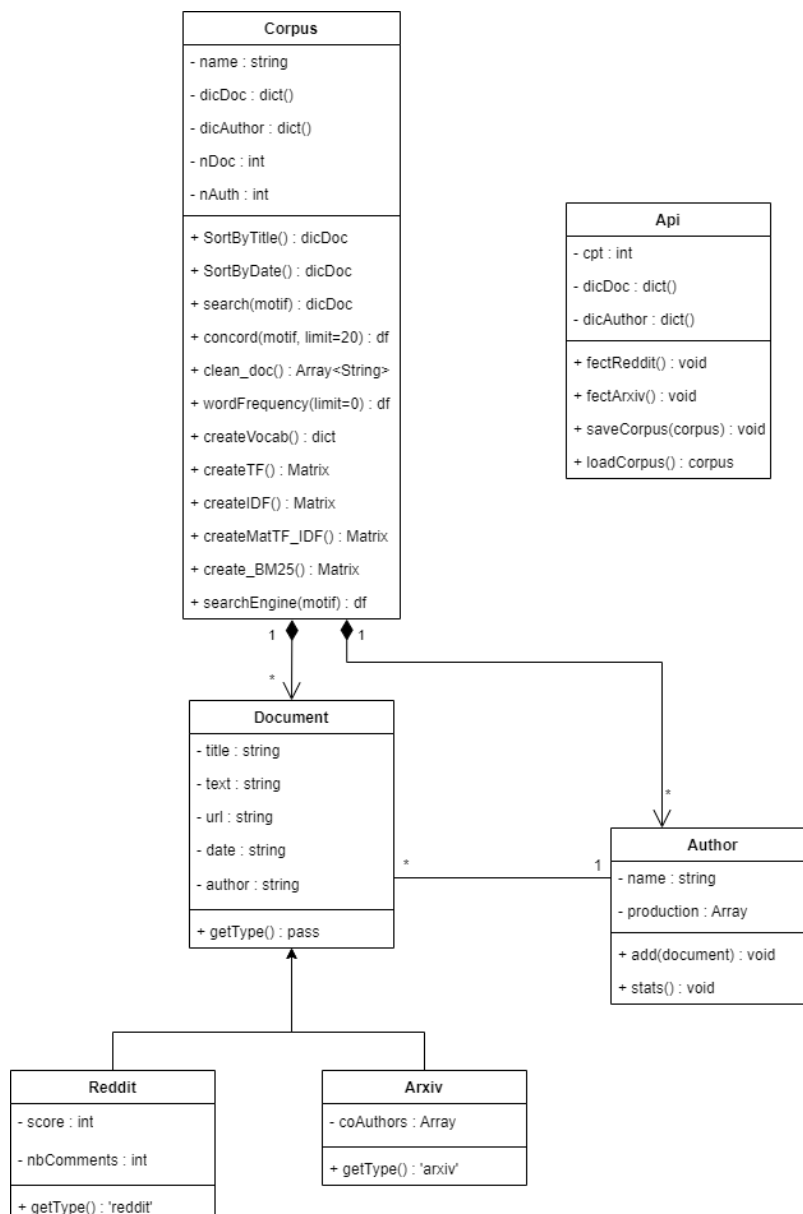
```
|   app.py
|   DOUSSE_FAIVRE.docx
|   README.md
|   tempsExec.py
|   tree.txt
|
+---assets
|   |   Nunito-Regular.ttf
|   |   style.css
|   |
|   \---Images
|           search.svg
|           wordCloud.png
|
+---data
|   corpus.pkl
|
+---modules
|   |   Author.py
|   |   Corpus.py
|   |   Document.py
|   |   DocumentGenerator.py
|   |   Json.py
|   |   Request.py
|   |   Singleton.py
|   |   __init__.py
|   |
+---pages
|   |   Concordancer.py
|   |   Source_comparaison.py
|   |
```

- App est notre fichier de démarrage de l'application, c'est à travers ce dernier que le serveur Dash est démarré.
- tempsExec est un fichier qui permet de calculer le temps d'exécution pour le calcul des matrices TF, IDF et BM25
- Assets comprend les éléments visuels à l'application Dash tel qu'un fichier css, des images ainsi qu'une police.
- Data est composé du fichier corpus intégrant la classe du même nom au format binaire.
- Modules contient l'intégralité de nos classes.
- Pages intègre les pages de l'application Dash. Nous avons fait le choix d'ajouter un système de pagination à notre application afin qu'une même page ne possède pas trop de fonctionnalités ce qui aurait aussi surchargé le code d'une page.

Analyse :

Nous avons fait le choix de développer l'application en utilisant l'IDE Spyder avec le gestionnaire de package Anaconda. Nous nous sommes orientés vers Spyder car c'est l'IDE que nous avons toujours utilisé pour développer avec le langage python. Anaconda nous permet de créer des environnements virtuels afin d'y intégrer des packages. Cela permet notamment de ne pas télécharger une nouvelle fois à chaque lancement les packages nécessaires à notre application.

En ce qui concerne les données de notre application, 2 sources peuvent être identifiées : Les données de l'API Reddit ainsi que Arxiv, possédant sensiblement les mêmes données. Néanmoins, Reddit possède par exemple un « score » (pour le nombre de vote du post) et un document Arxiv peut avoir des co-auteurs. Ces différences nous ont amené à modifier l'architecture du projet afin de les distinguer.



L'une des premières classes que nous avons créée est la classe Document afin de stocker les résultats des requêtes sous la forme d'un objet. Afin de faire la différence entre les documents Reddit et Arxiv, il a été ensuite nécessaires de créer 2 classes filles à Document pour chaque source. Reddit possède un score et Arxiv une liste de co-auteurs. Document peut d'ailleurs être considérée comme une classe abstraite car elle n'est pas sensée être instanciée.

La classe auteur a été créée en parallèle afin de stocker les auteurs. Un document peut avoir 1 ou plusieurs auteurs en fonction de sa source. Un auteur peut avoir quant à lui plusieurs documents.

Afin de créer les documents Reddit et Arxiv nous avons mis en place le design pattern factory qui instancie un document de type Arxiv ou Reddit en fonction du type qu'on lui donne.

La classe Corpus nous permet de stocker nos documents et nos auteurs à travers des dictionnaires. Cette classe est une composition de Document et d'Auteurs car elle ne peut exister sans ces 2 entités.

Enfin Api permet de requêter sur les API Reddit et Arxiv ainsi que de sauvegarder le corpus sur le disque et enfin, le charger en mémoire. Cette classe n'était pas requise mais nous avons décidé de la créer afin de mieux organiser notre code.

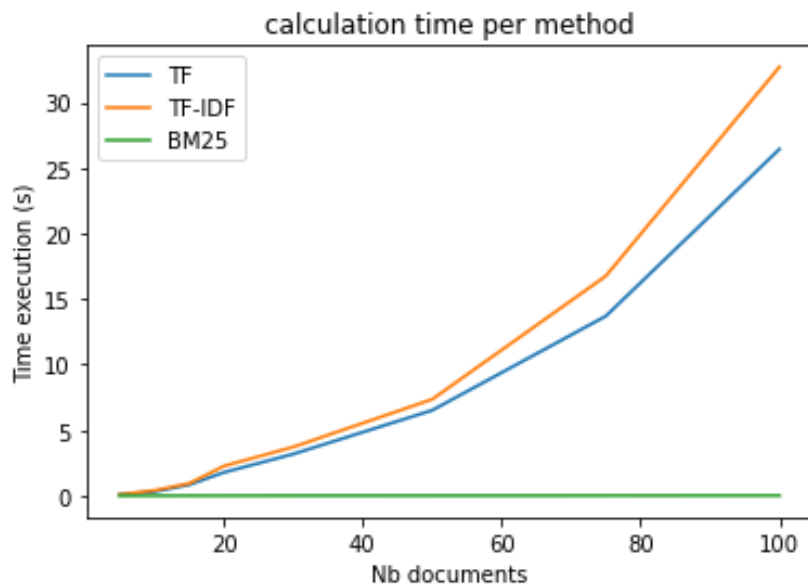
C'est dans la classe `Corpus` que nous réalisons la plus grosse partie des traitements des documents comme nettoyer les textes pour en faire ressortir les mots les plus présents. Afin de réaliser cette transformation nous avons réalisé une première méthode utilisant en grande partie des regex afin de nettoyer les documents. A côté de cela, nous avons vu en cours d'ingénierie des données des procédés afin de nettoyer des textes notamment en utilisant la librairie `sklearn`. Le cours n'étant pas un cours de text mining, nous n'allons pas nous attarder sur cette méthode mais nous vous proposons de voir les nuages de mots en fonction des deux méthodes :



Toujours dans la classe `Corpus`, nous réalisons les matrices TF, IDF et TF-IDF afin de calculer selon plusieurs procédés, la fréquence d'un mot dans le corpus. Afin de calculer ces matrices nous utilisons un tableau à 2 dimensions. Or, il aurait été préférable d'utiliser des matrices sparses (creuses) adaptées à ce genre de problème. En effet, les matrices sparses ne prennent pas en compte les 0 lors de calcul ce qui réduit ainsi les coûts. Néanmoins, nous n'arrivons pas à réaliser de produit scalaire et d'autres opérations arithmétiques avec les matrices sparses. C'est pourquoi nous avons utilisé un tableau à 2 dimensions et cela fonctionne très bien.

Page 5 sur 11

Nous nous sommes posé la question du temps d'exécution du calcul des matrices TF, TF-IDF et BM25. Nous avons décidé de réaliser le graphe suivant qui permet de mettre en avant le temps d'exécution de chaque méthode en fonction du nombre de documents :



Nous pouvons voir que le temps d'exécution pour le calcul des matrices TF et IDF est exponentiel contrairement à BM25 qui est linéaire. L'une des raisons à cela est que pour BM25 on calcule uniquement un vecteur de score, on ne réalise aucun calcul matriciel contrairement à TF et IDF. Il est possible que le temps d'exécution aurait été moindre si on avait utilisé les matrices sparses comme nous avons pu le citer précédemment. Enfin, il est très probable que nos fonctions peuvent être grandement optimisées ce qui réduirait le temps d'exécution notamment en utilisant des librairies ou en pensant le code autrement. Néanmoins, dans l'application final, le nombre de documents est de 30. Ainsi le temps d'exécution des différentes méthodes ne se fait pas tellement ressentir.

En ce qui concerne la répartition des tâches, nous avons tous les 2 réussis à aller jusqu'au TD n° 9-10 donc nous avons tous les 2 implémentés l'orienté objet et plusieurs fonctions de la classe Corpus. L'un de nous ayant une appétence pour le développement web, cette personne à davantage mis son attention sur l'esthétique de l'interface ainsi que sur les interactions utilisateur. Nous avons chacun réalisé une page s'entraînant en fonction de nos qualités et faiblesses.

Utilisation du programme :

Nous avons décidé de réaliser une vidéo présentant l'utilisation de notre programme accessible à l'url suivante : https://youtu.be/WLqa_DlIGZA

Dans le cas où vous ne pouvez voir la vidéo, voici de manière écrite un cas concret d'utilisation de notre programme.

Vous pouvez accéder à l'interface en lançant l'application depuis le projet en exécutant le fichier app.py puis en accédant à l'url : <http://127.0.0.1:8050/>.

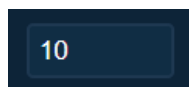
Une fois sur l'interface, vous êtes sur la page « Concordancer » par défaut. L'objectif de cette page est d'afficher la présence d'un ou plusieurs mots dans le corpus.

La page possède un tableau affichant les documents du corpus selon leur titre, texte et date. Un concordancier permet d'afficher les documents possédant un mot clef ainsi que leur position dans le document. Enfin, un graphe en barre permet selon un mot clé et une mesure, d'afficher l'importance d'un mot dans les documents.

L'utilisateur peut saisir un mot clé à l'aide du filtre de recherche :



Le filtre suivant permet de choisir le nombre de lettres qui seront affichées dans le concordancier. Exemple :



contexte gauche	motif trouve	contexte droit
...Please	post	your ques...
...Hey peeps,	post	s here and...
...Hey folks,	post	s/tunemyai...
...Hi everyon	post](https://...

Pour 30 :

contexte gauche	motif trouve	contexte droit
...Please	post	your questions here instead o...
...Hey peeps, I know I know ChatG	post	s here and there about it but ...
...Hey folks, I built [TuneMyAI]	post	s/tunemyai). Thanks & Happy H...
...Hi everyone! For the 8th (!)	post](https://tryolabs.com/blog/20...

Enfin, le dernier filtre permet de choisir la mesure d'importance d'un mot dans le corpus, ce qui modifie directement le résultat du graphe en barre :

Recherche du mot « post » :

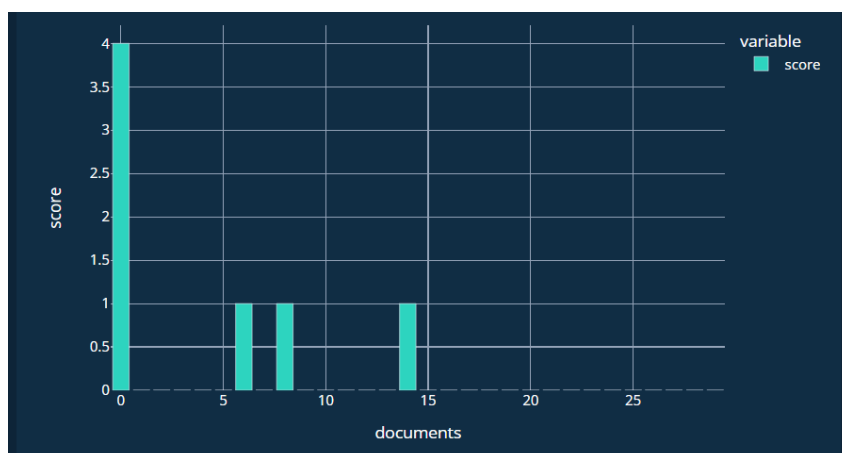


Figure 3 - Mesure TF

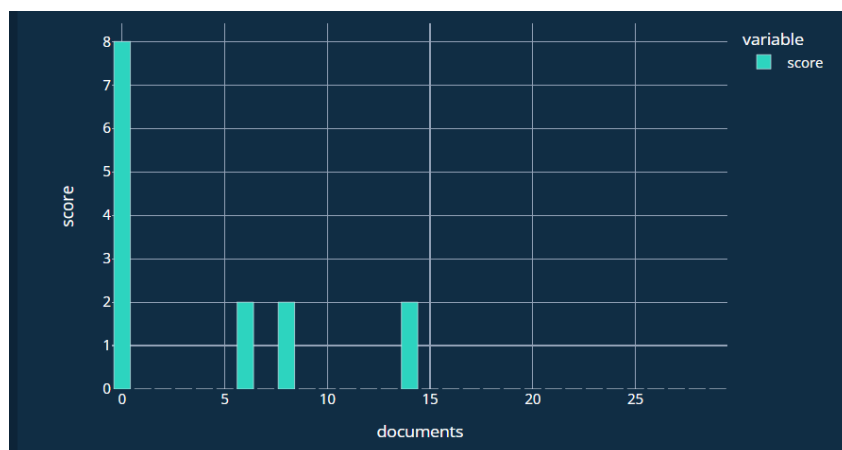


Figure 4 - Mesure IDF

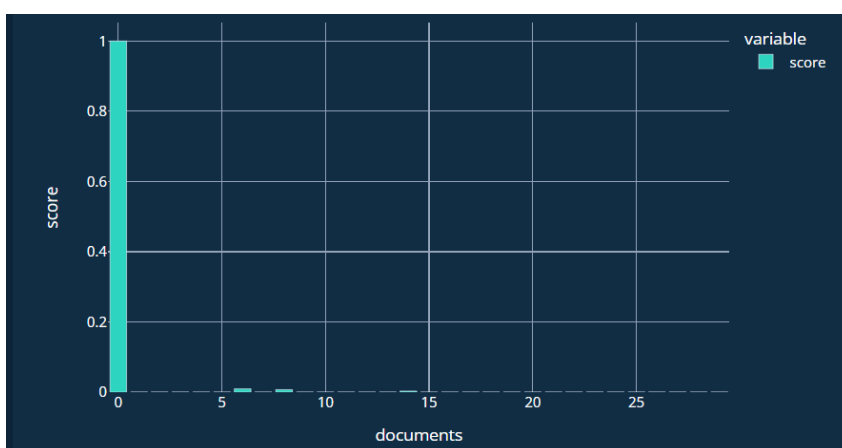


Figure 5 - Mesure BM25

Vous pouvez ensuite naviguer sur la page « Source comparaison ».

L'objectif de cette page est de permettre à l'utilisateur de voir l'évolution d'un mot dans le temps. Cette page possède le même tableau de corpus que l'autre page ainsi qu'un « line graph » afin de visualiser la présence d'un mot selon le score et la date.

L'ensemble des filtres changent de manière dynamique le graphe ainsi que le tableau.

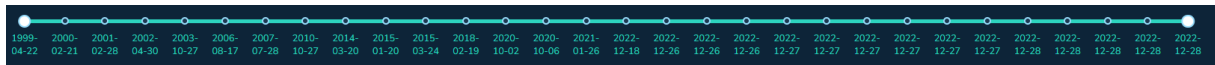
Tout comme l'autre page, il est possible de saisir un mot clé avec le filtre de recherche.

Le filtre qui suit permet de trier le tableau par date ou titre.

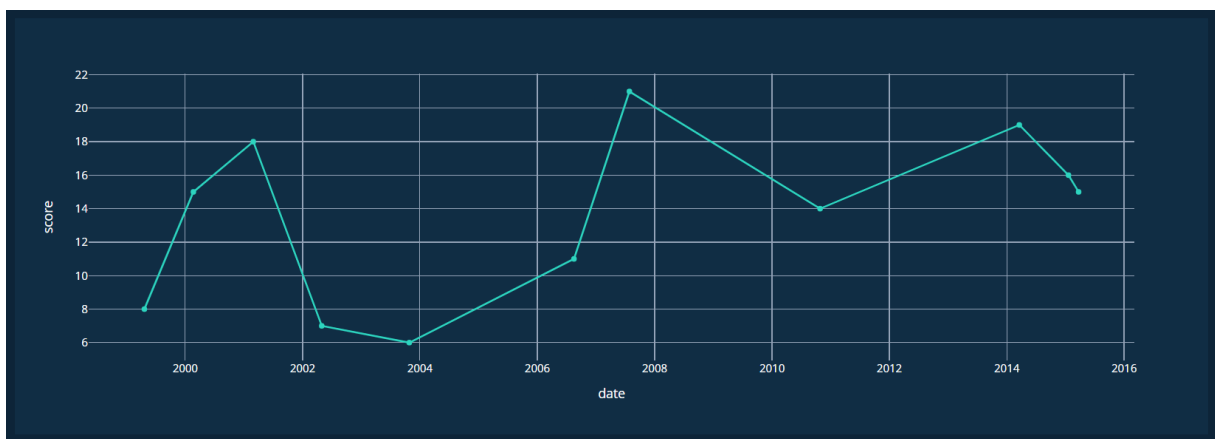
Source permet de trier le tableau selon la source : Reddit ou Arxiv.

Enfin, le dernier filtre permet de choisir un auteur dans le corpus.

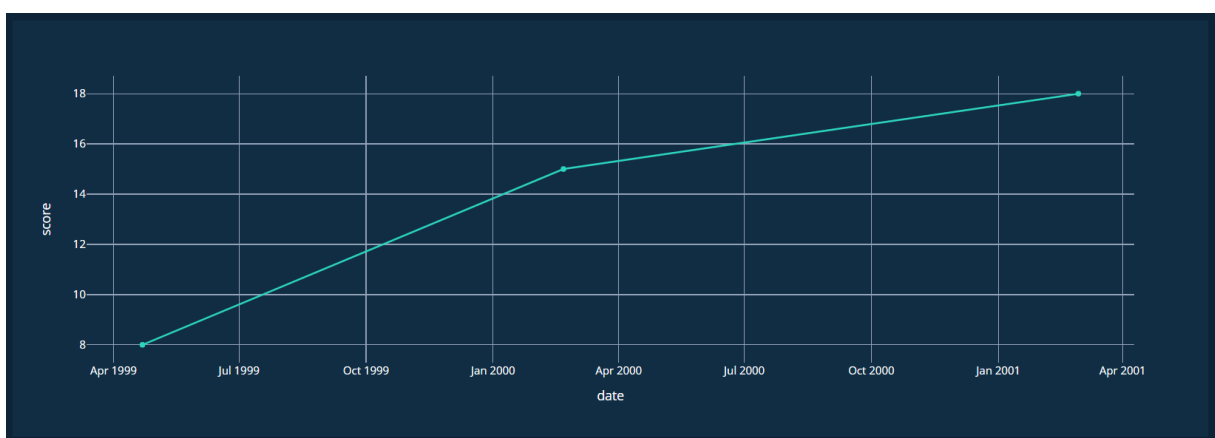
Il est possible de filtrer par date en choisissant une plage avec l'élément suivant :



Exemple de résultat avec pour mot clé « électron » et pour plage de date 1999 - 2015



Toujours avec le même mot clé mais pour la plage de date 1999 – 2001 :



Validation :

En ce qui concerne les tests unitaires, nous n'en avons pas réalisé à proprement parler notamment dans un fichier .py.

Nous avons néanmoins testé les fonctions de calcul des matrices TF, TF-IDF et BM25 selon différents nombres de documents.

La fonction qui permet de créer le concordancier a été testée pour plusieurs mots avec toutes les limites (de 10 à 30).

Enfin les filtres ont été testés selon toutes les configurations possibles afin de vérifier leur bon fonctionnement :

- Pour la page « Concordancer » avec 0, 1 ou plusieurs mots avec une limite de 10 à 30 et selon les 3 méthodes d'importance d'un mot.
- En ce qui concerne la page « Source comparaison », avec 0, 1 ou plusieurs mots, selon différentes sources avec différentes plages de dates et avec ou sans auteurs.

Nous avons aussi testé les filtres de l'interface avec plusieurs nombres de documents. Nous avons vu que le temps d'exécution des mesures d'importance d'un mot pouvait devenir assez élevé dans le cas d'un grand nombre de documents. Le temps que l'interface s'actualise devient d'autant plus long en fonction du nombre de documents. Lorsque l'utilisateur réalise une recherche, l'interface s'actualise à chaque pression de touche. Ça rend l'interface dynamique mais pour un nombre élevé de documents, cela ajoute des problèmes de lenteur car les matrices sont calculées à chaque pression de touches.

Maintenance :

Pour conclure sur ce projet, nous sommes très satisfaits du travail réalisé. L'un des principaux aspects qui a demandé le plus de travail a été l'interface Dash et nous avons réussi à réaliser les idées que nous avions. Nous étions un peu septiques vis à vis de Dash et de son support des interactions utilisateurs comparé à un langage comme JavaScript. Après avoir développé ce projet, nous trouvons que Dash est un bon outil pour créer une interface web dynamique à partir de python et il propose une documentation bien fournie. Etant passionné par le développement web, j'ai trouvé que Dash intégrait très bien la création de balise html mais aussi l'ajout de style css. Nous avons pu réaliser l'interface avec le style visuel que nous souhaitons.

En ce qui concerne les évolutions possibles du logiciel. L'une des premières modifications à faire serait d'optimiser certains algorithmes de l'application afin de réduire leur complexité pour avoir de meilleure performance. Un autre aspect concernerait l'architecture du projet. En effet, nous avons fait le choix d'utiliser des pages pour décomposer le code afin de rendre le tout plus maintenable. Nous aurions pu aller plus loin en créant des composants à l'intérieur des pages ce qui aurait réduit la duplication de code et amélioré la maintenabilité de notre projet.

Par manque de temps et de documentation à ce sujet nous n'avons pas pu mettre en place cette idée. L'une des fonctionnalités à laquelle nous avons pensé était de permettre à l'utilisateur d'entrer un titre ainsi qu'un texte et de prédire la source du document. Cette idée rejoint nos cours d'ingénierie des données où nous réalisons des choses similaires. Néanmoins, il aurait fallu beaucoup de données afin de réaliser une bonne prédiction ce qui aurait demandé d'avoir soit un fichier corpus contenant les documents très volumineux ou alors de requêter sur les apis avec un grand nombre de documents. Une autre option aurait pu être de laisser choisir le modèle de prédiction. En termes de code, l'idée ne nous semble pas si compliqué à implémenter mais demande une base de données conséquente pour apprendre.