Blanchon Marine, Debret Léane

Rapport de projet moteur de recherche

Python

Professeur référent : Mr Julien Velcin

Janvier 2025

Spécifications :

Ce projet a pour objectif de créer un moteur de recherche qui permet d'extraire les informations pertinentes à partir de documents récupérés sur deux plateformes : Reddit et ArXiv. Le programme doit permettre à l'utilisateur de faire une recherche par mots-clés et suite à cette recherche, un certain nombre de résultats, en fonction de la pertinence des documents par rapport aux critères définis doivent s'afficher. Cette classification permet de mieux organiser les documents et d'appliquer des filtres spécifiques en fonction de leur source. Le moteur de recherche doit aussi gérer l'affichage des résultats avec une interface graphique simple et intuitive, contenant des widgets interactifs (curseurs, champs de texte par exemple). L'interface permettra à l'utilisateur de saisir un mot-clé dans un champ de texte et d'afficher les résultats de façon claire et lisible. Le but est de faciliter l'accès rapide aux informations et d'assurer une gestion efficace des documents.

Analyse :

Environnement de travail :

Nous avons développé le projet dans un environnement Python. Nous avons utilisé des bibliothèques comme *Pandas* pour la gestion des données, *IPython* pour l'affichage interactif, et des widgets pour faire une interface graphique simple. Nous avons utilisé la bibliothèque *Pickle* pour charger et gérer les fichiers du corpus. Pour développer et tester notre projet, nous l'avons fait dans un environnement *Jupyter Notebook*, permettant une exécution interactive et afin de voir les résultats directement.

Données identifiées :

Les données principales qui sont traitées par le programme sont les documents scientifiques collectés depuis Reddit et ArXiv. À partir de ces documents, plusieurs informations sont comprises :

- Titre du document
- Auteur du document
- Date de publication du document
- URL vers le document
- Texte contenu dans le document

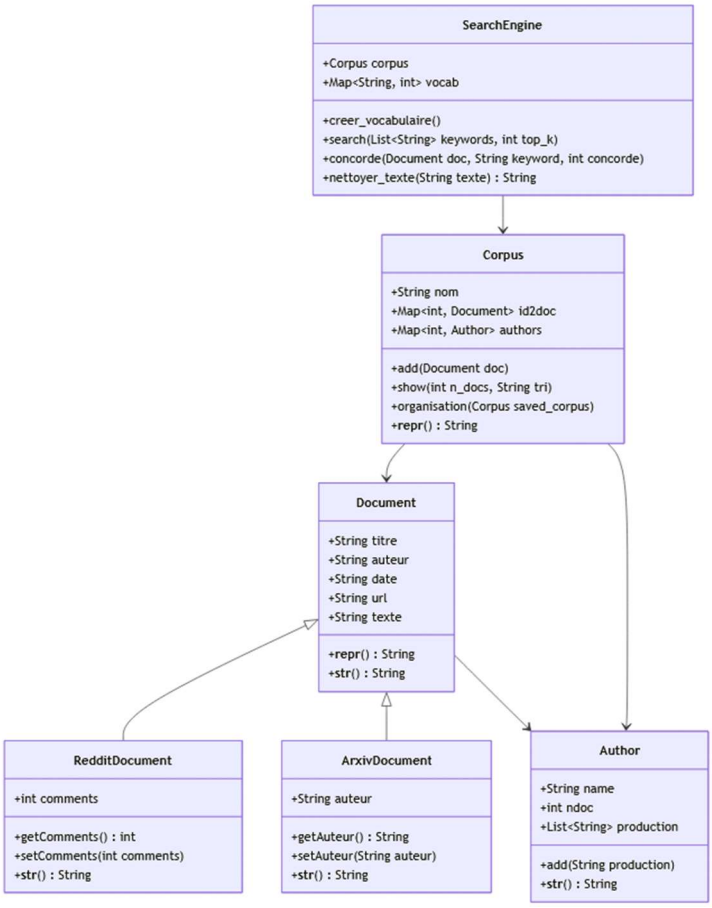
Une fois les informations extraites, les documents en question sont classés selon leur origine (que ce soit Reddit ou ArXiv), ce qui permet de gérer plus facilement le corpus de documents.

Diagramme des classes :

Notre projet contient 3 classes principales : *Document*, *RedditDocument* et *ArxivDocument*. *SearchEngine*, *Corpus* et *Author* sont des classes supplémentaires. La classe *Document* contient les attributs communs de tous les types de documents (titre, auteur, date, url, texte). Elle utilise __*repr*__ et __*str*__ pour donner une représentation simple et claire des documents. Cette classe est essentielle car elle permet de réunir les informations communes et de garder une structure cohérente pour tous les types de documents. La classe *RedditDocument* hérite de la classe *Document*. Son attribut *comments* correspond au nombre de commentaires associés à une publication Reddit. La classe utilise les méthodes *getComments* et *setComments* pour manipuler le nombre de commentaires et rédéfinit la méthode __*str*__ pour une représentation propre à Reddit. La classe *ArxivDocument* hérite elle aussi de la classe *Document*. Elle a l'attribut *auteur*, spécifique à ArXiv et utilise les méthodes *getauteur* et *setAuteur* pour manipuler l'auteur. Tout comme la classe précédente, elle redéfinit la méthode __*str*__ mais cette fois-ci pour ArXiv.

La classe *Author* représente un auteur et contient donc les informations liées aux auteurs dans le Corpus. Elle contient les attributs *name, ndoc et production.* Elle utilise la méthode *add* pour ajouter un titre aux productions de l'auteur en question et __*str*__ pour afficher les informations liées à l'auteur. La classe *Corpus* gère l'ensemble des documents et auteurs. Elle contient les attributs *nom, id2doc, et authors.* Elle utilise les méthodes *add* pour ajouter un document au Corpus, *show* pour montrer des documents, *organisation* pour structurer le corpus selon une version enregistrée, et __*repr*__ pour une représentation claire du Corpus. Enfin, la méthode *SearchEngine* implémente le moteur de recherche pour faire de requêtes dans le Corpus. Ses attributs sont *corpus* et *vocab*. Elle utilise la méthode *créer_vocabulaire* pour générer un vocabulaire basé sur les documents, *search* pour faire une recherche sur des mots-clés en classant les résultats, concorde pour afficher le contexte des mots-clés dans un document et *nettoyer_texte* pour arranger le texte pour la recherche.

Les méthodes *RedditDocument* et *ArxivDocument* héritent de *Document*. Elles partagent donc des attributs et méthodes, tout en rajoutant des fonctionnalités spécifiques aux types de documents. La classe *Corpus* utilise *Document* pour stocker les documents du projet et *Auhtor* pour gérer les auteurs. La classe *SearchEngine* utilise *Corpus* pour faire des recherches sur tous les documents.



Conception:

Répartition des tâches :

Le projet a été réalisé en binôme et la répartition des tâches s'est faite progressivement. Dès le début, nous avons travaillé de façon indépendante sur les premiers TDs. Au fil des cours, nous avons compris que ces TDs s'inscrivaient dans un ensemble qui menait directement à la création de ce projet final.

Au début du projet, nous avons donc fait les premiers TDs séparément. Nous avons toutes les deux eu du mal à nous y mettre : nous ne visualisions pas forcément le fil directeur de ce projet. Cependant, au fur et à mesure des TDs, nous avons commencé à comprendre plus clairement l'importance de chaque étape et compris où les TDs commençaient à nous mener. C'est à partir de ce moment que nous avons pris une réelle motivation à travailler sur le projet. À partir du TD5, après avoir compris que tous ces TDs faisaient partie du projet et seraient utilisés pour la finalité du projet, nous avons décidé de poursuivre ces TDs en binôme.

Nous avons utilisé les premiers TDs pour poser des bases solides, en mettant en commun nos travaux faits séparément et nous assurer que nous partions toutes les deux dans la même direction. Nous avons pris les premiers TDs corrigés comme point de départ pour le projet, pour uniformiser nos deux approches et garantir que notre code serait cohérent. Ensuite, nous avons fusionné les TDs suivants que nous avions faits individuellement. Nos deux méthodes étaient assez proches, ce qui nous a grandement facilité la tâche.

Pour la suite du projet, nous avons choisi de toujours travailler ensemble sur les TDs pour éviter toute confusion et être sûres que nous restions sur la même longueur d'onde. Nous avons utilisé un ordinateur pour coder et un autre pour lire les TDs et faires des recherches nécessaires sur Internet. Cette méthode nous a permis de gagner beaucoup de temps : nous avons pu avancer rapidement, en nous concentrant ensemble sur le travail de recherche et de codage.

À la fin de certains TDs, quand nous n'étions pas satisfaites de notre progression pendant le cours, nous décidions de poursuivre notre avancée dans une salle à côté pour continuer à travailler, tant que nous avions encore les idées fraîches. Cette façon de faire nous a permis de bien avancer sur le projet et d'atteindre nos objectifs dans les temps.

Une fois le dernier TD terminé, nous avons décidé de nous réunir chez nous, en dehors des cours pour finaliser le script, faire les dernières finalisations du bon fonctionnement et le ajouter les derniers éléments concernant l'interface. Enfin, nous avons rédigé ce rapport ensemble, en ayant chacune pris des notes sur ce que nous voulions dire dans ce rapport avant de nous retrouver pour tout mettre en commun et le rédiger.

Pendant le projet, nous avons chacune exploré des pistes différentes. Bien que nous connaissions les notebooks, nous avons envisagé d'autres alternatives. Après en avoir discuté lors d'un TD, nous avons décidé ensemble que le notebook semblait être la solution la plus adaptée aux besoins du projet en fonction de nos connaissances, de la faisabilité et du temps imparti.

Algorithmes :

L'algorithme principal repose sur la recherche de mots-clés dans le texte des documents. Pour chaque mot-clé, une mesure de pertinence est calculée pour chaque document. Cette pertinence est ensuite utilisée pour trier les résultats par ordre décroissant de pertinence.

Problèmes rencontrés et solutions :

Pendant le développement du projet, plusieurs difficultés sont apparues, principalement liées à la gestion des documents et à la gestion des types de données.

Le premier problème que nous avons rencontré était sur la gestion des documents. Etant donné que nous avons traité des fichiers ArXiv ou Reddit, il était important que ces documents soient correctement classés et catégorisés. Pour ça, nous avons utilisé des classes et des objets, ce qui nous a permis de mieux organiser et gérer toutes les informations liées à chaque document. Mais cette méthode nous a apporté plusieurs défis.

Le projet nécessitait plusieurs fonctions, qui ont souvent été écrites individuellement dans le cadre des TDs suivis en cours tout au long du projet, qui étaient testées séparément pour vérifier qu'elles fonctionnaient bien. Cela nous a permis d'avoir des fonctions fonctionnelles isolées. Mais quand nous avons dû les intégrer, nous avons rencontré un problème de compatibilité et de liaison entre ces fonctions. L'intégration de toutes ces fonctions dans un seul système cohérent a donc été un défi pour nous.

La gestion des types des données a également été un point sur lequel il a fallu plus travailler. Au moment de l'implémentation du moteur de recherche, quand on saisissait un mot-clé pour faire la recherche dans les documents, nous avons rencontré des erreurs de type. Les objets que nous manipulions, comme les instances de *RedditDocument* et *ArxivDocument*, étaient plus complexes que des simples chaînes de caractères ou listes. Cela posait donc problèmes : certaines de nos fonctions, qui étaient conçues pour fonctionner avec des chaînes de caractères, recevaient des objets en entrée, ce qui renvoyait des erreurs. Le problème était que certaines fonctions censées chercher un mot dans une phrase se retrouvaient confrontées à un objet entier plutôt que de recevoir une chaîne de caractères.

Ce problème de type nous a demandé beaucoup de temps et d'efforts pour être résolu. Nous avons à ce moment utilisé l'Intelligence Artificielle (*ChatGPT*) afin de mieux comprendre la source du problème, et comprendre comment la résoudre. Nous avons donc fait des conversions et ajustements continus des types de données pour nous assurer que les fonctions puissent fonctionner correctement avec les objets. Nous avons dû ajuster nos méthodes pour garantir qu'elles acceptaient les bons types d'entrée, ce qui a été une source de confusion et d'erreurs pour nous pendant un certain temps.

Un autre point que nous estimions important à relever est la création du corpus. Ce n'est pas un point qui a été autant problématique que la gestion des types, mais il nous a également pris beaucoup de temps. Le corpus représente l'ensemble des documents que nous devons organiser et traiter, il est donc essentiel que sa création soit correctement faite pour que l'ensemble du projet fonctionne. Si le corpus est mal structuré, il est ensuite difficile de le manipuler et ça fausse tous les traitements qui sont faits, notamment la recherche. Nous avons donc pris soin de bien organiser les documents et de les classer correctement selon leur type (Reddit ou ArXiv), mais cela a demandé plusieurs ajustements pour que tout soit en ordre.

Exemple d'utilisation :

L'utilisateur rentre un mot-clé dans le champ de texte associé. Il ajuste ensuite le curseur en fonction du nombre de résultats qu'il veut avoir. Une fois que c'est fait, il peut cliquer sur le bouton recherche. Cela déclenchera l'affichage des résultats sous forme de liste : chaque document affiché présente son titre, auteur, date et un extrait du texte dans le contexte du mot-clé qui a été recherché.

Validation :

Concernant les tests, nous avons opté pour une approche manuelle tout au long du développement. En effet, chaque méthode et chaque classe ont été testées individuellement, avec des valeurs fictives ou simulées pour vérifier leur bon fonctionnement avant des passer à la suite et de les intégrer définitivement dans le projet. Cette façon de faire nous a permis d'avancer sereinement dans le projet et de contrôler chaque étape de l'implémentation pour être sûres que tout était cohérent avant de passer aux étapes suivantes.

Au fur et à mesure de l'avancement du code, nous avons testé chaque nouvelle fonctionnalité dès qu'elle était implémentée pour nous assurer qu'elle correspondait bien à la spécification. Par exemple, quand nous avons créé la classe *RedditDocument*, nous avons manuellement vérifié que les méthodes comme *getComments()* et *setComments()* fonctionnaient correctement en introduisant des valeurs de test, comme des fausses listes de commentaires. Cela nous a permis de facilement détecter les erreurs quand il y en avait et de les corriger avant de les utiliser dans des contextes plus difficiles.

Nous avons appliqué la même méthode pour tester la classe *ArxivDocument*, notamment la gestion de l'attribut *auteur* et son interaction avec les autres objets *Author*. En testant ces fonctionnalités individuellement, nous avons évité les risques liés à des erreurs qui s'accumulent ce qui aurait pu causer des dysfonctionnements plus difficiles à localiser sans cela.

Cette approche nous a aussi permis de mieux comprendre comment chaque méthode et chaque classe interagissent entre elles. En testant par petites étapes, nous avons vu les effets de chaque modification et avons ajusté le code en fonction des résultats (bons ou non) que nous obtenions.

En résumé, bien que nous n'ayons pas intégré de fichier de tests automatisés dans notre rendu de projet, notre approche manuelle de tests itératifs a été essentielle pour s'assurer que le programme fonctionne bien et pouvoir nous donner à la fin un code fonctionnel.

Maintenance :

Evolutions possibles :

Afin d'améliorer notre projet, il aurait été intéressant d'ajouter d'autres systèmes de classification des documents. Actuellement, les documents sont classés pas leur source seulement. Nous aurions pu ajouter d'autres méthodes de classification comme une classification par type de publication par exemple (article scientifique, discussion, …). Nous avons également essayé de créer une interface graphique simple et intuitive sur le thème de la NASA mais pour améliorer le projet, il aurait également été intéressant d'ajouter des filtres

plus détaillés. Par exemple, nous aurions pu ajouter la possibilité de trier par date ou par nombre de commentaires. Une autre possibilité aurait été d'étendre le moteur de recherche à d'autres plateformes de publication.

Conclusion :

Ce projet a permis de mettre en œuvre un moteur de recherche simple mais fonctionnel pour explorer des documents scientifiques provenant de plusieurs sources. Les résultats renvoyés sont pertinents et l'interface permet une navigation fluide grâce aux widgets interactifs que nous avons intégrés. Cependant, il reste évidemment des améliorations possibles, par exemple sur l'extension des types de documents et l'amélioration de l'interface utilisateur.