KLEIN Bertrand - 2208425
GENTILI Laure – 2200986

Projets en Programmation Python

2023-2024

Chargé d'enseignement : Julien Velcin

Lien du GitHub : https://github.com/Laure69/Projet-en-Programmation-Python.git

Table des matières

1. Problématique

Comment concevoir et mettre en œuvre un moteur de recherche en utilisant le langage de programmation Python afin de créer un outil performant, bien structuré, et facilement utilisable pour la recherche et l'exploration de documents dans un corpus ?

2. Spécifications

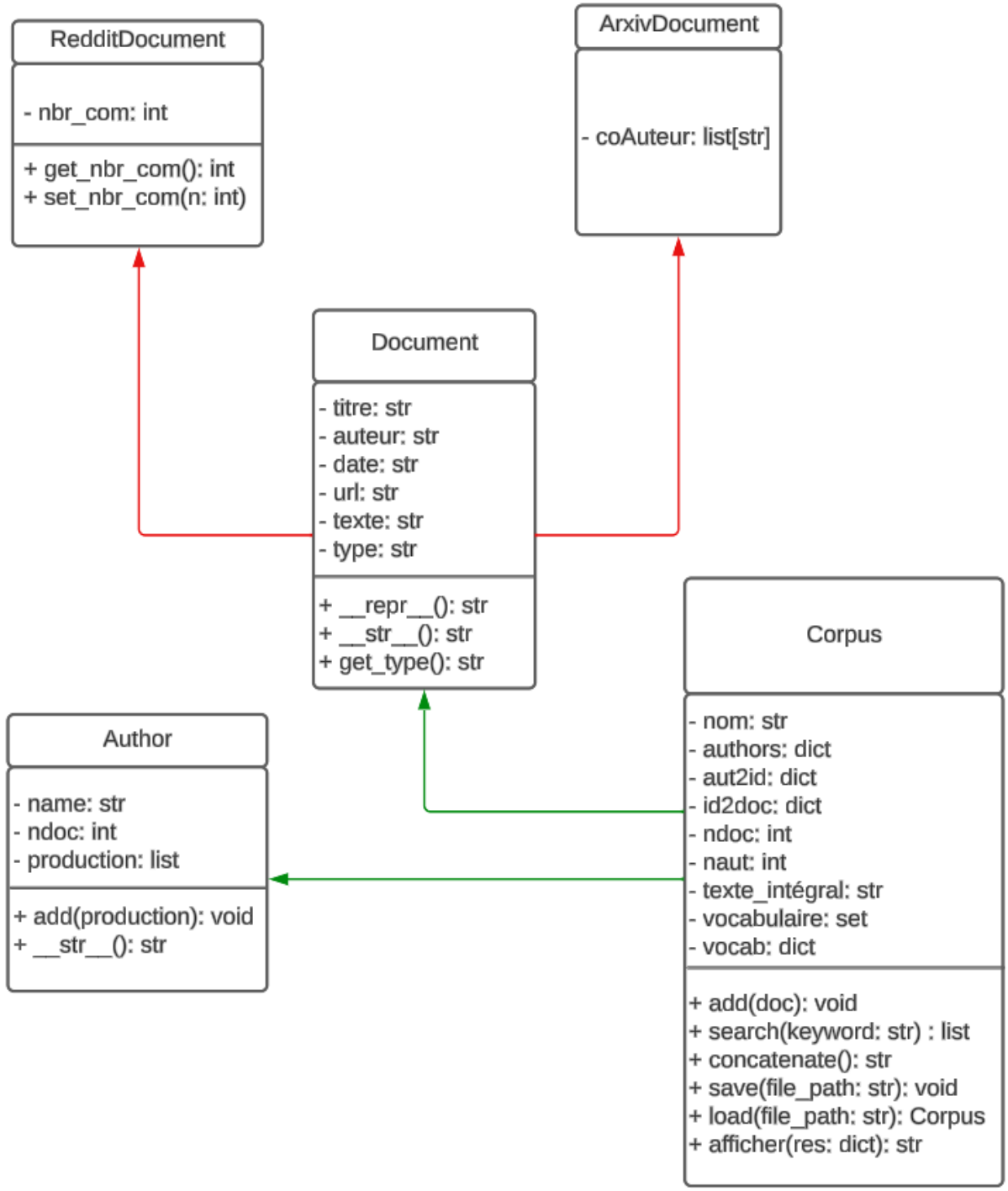
- L’utilisateur peut saisir des mots-clés dans le champ de recherche
- L’utilisateur peut saisir le nom de l’auteur dans le champ dédié
- L'utilisateur peut sélectionner une source dans la liste déroulante
- L'utilisateur peut sélectionner une date à l'aide du sélecteur de date
- L'utilisateur peut lancer la recherche en cliquant sur le bouton "Recherche"

3. Analyse

- Pour le développement de ce moteur de recherche, nous avons choisi d’utiliser l’environnement Conda, gestionnaire de paquets, en combinaison avec l’interface Anaconda, qui sert de gestionnaire d'environnement. Cette décision, basée sur l’utilisation du langage Python, s'inscrit dans la continuité de notre apprentissage en classe, motivant ainsi notre choix.

  De plus, nous avons choisi d’utiliser Jupyter Notebook en raison de son interface interactive et visuelle, offrant une prise en main plus accessible par rapport à Dash, que nous ne maîtrisons pas encore.

  Enfin, le travail collaboratif a été facilité grâce à GitHub, permettant de suivre l'évolution et l'avancement du binôme tout au long du projet.



Dans ce diagramme, les classes utilisées sont Document, RedditDocument, ArxivDocument, Author, et Corpus. Les relations d'héritage sont indiquées par les flèches rouges reliant Document à RedditDocument et ArxivDocument. Les relations d'utilisation entre les classes sont représentées par les appels de méthodes ou d'attributs, tels que l'utilisation de méthodes de Document et Author dans Corpus.

4. La conception

- La répartition des tâches au sein du binôme :

  Dans un premier temps, nous avons travaillé de manière individuelle sur les premiers travaux dirigés pour assimiler les concepts et acquérir des compétences nécessaires pour le projet. Par la suite, nous nous sommes retrouvés pour échanger de l'avancement des TD, partager les résultats obtenus et discuter des problèmes rencontrés. Nous avons mis en communs nos programmes en piochant ce qu'il y avait de mieux chez l'un ou l'autre. Nous étions alors rendus au TD7 qui devait finaliser l'implémentation de notre moteur de recherche avant de nous attaquer à l'interface graphique. Comme nous avions besoin de finaliser le TD7 pour continuer la suite, nous avons préféré travailler ensemble sur les mêmes fonctionnalités, soit en codant à deux, soit en se relayant. Nous avons ensuite abouti à une version du moteur de recherche suffisamment satisfaisante pour passer à l'étape suivante.
  Nous nous sommes ensuite attaqués à l'interface graphique. Là encore notre répartition n'était pas particulièrement réfléchie, chacun faisait ce qu'il était capable de faire. Bertrand s'est occupé de la sélection/création du corpus, ainsi que de la recherche par mots-clés, et Laure a amélioré cette dernière en y ajoutant des options (date, nom de l'auteur, source du document). Laure s'est également occupé de la présentation du notebook et nous nous sommes relayés pour la rédaction du rapport.

- Algorithmes spécifiques à l'application :

  La méthode **recherche(self, query)** implémente un algorithme de recherche de similarité entre une requête de mots-clés et les documents du corpus. Cette méthode est la clé de voute de notre moteur de recherche. Elle renvoie en tableau de la taille du Corpus étudié qui contient les scores de similarité entre la requête et chaque document du corpus. Pour se faire, il faut vectoriser la requête en fonction du vocabulaire du corpus, c'est-à-dire qu'on crée un vecteur de la taille du vocabulaire qui contient des 0 et des 1 : 1 lorsqu'à l'indice correspondant ce mot est dans le vocabulaire, 0 sinon. On procède de la même manière pour les mots de chaque document, mais à la place des 1, on y met la valeur de la matrice TFxIDF au même indice. On fait ensuite un produit scalaire sur chaque vecteur document et cela nous donne un tableau de score qu'il faut trier.

- Problèmes rencontrés :

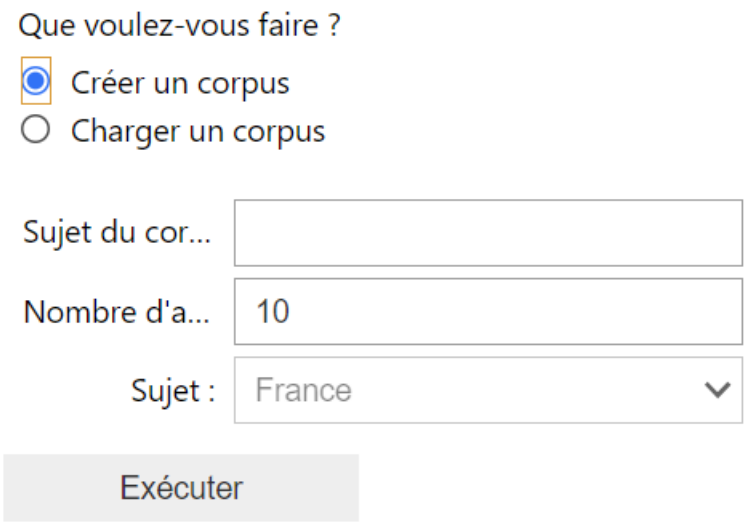
  Premièrement, nous avons rencontré un problème dans la mise en place du singleton pour la classe Corpus. Lorsque le singleton était implémenté, cela créait des problèmes lors de la sauvegarde du corpus. Nous ne sommes pas parvenus à trouver de solution donc nous avons simplement enlevé le singleton de la classe Corpus. Ceci rend possible la création de plusieurs instances de la classe mais nous n'avons pas eu l'impression en effectuant les tests que cela entravait le fonctionnement du programme. Peut-être que pour de futures fonctionnalités du programme, dans l'optique d'un développement de l'application vers quelque chose de plus complet, cela poserait un problème.

  Deuxièmement, nous avons fait face à un problème lors de l'affichage des résultats de la recherche au niveau de l'interface. En effet nous voulions dans un premier temps que

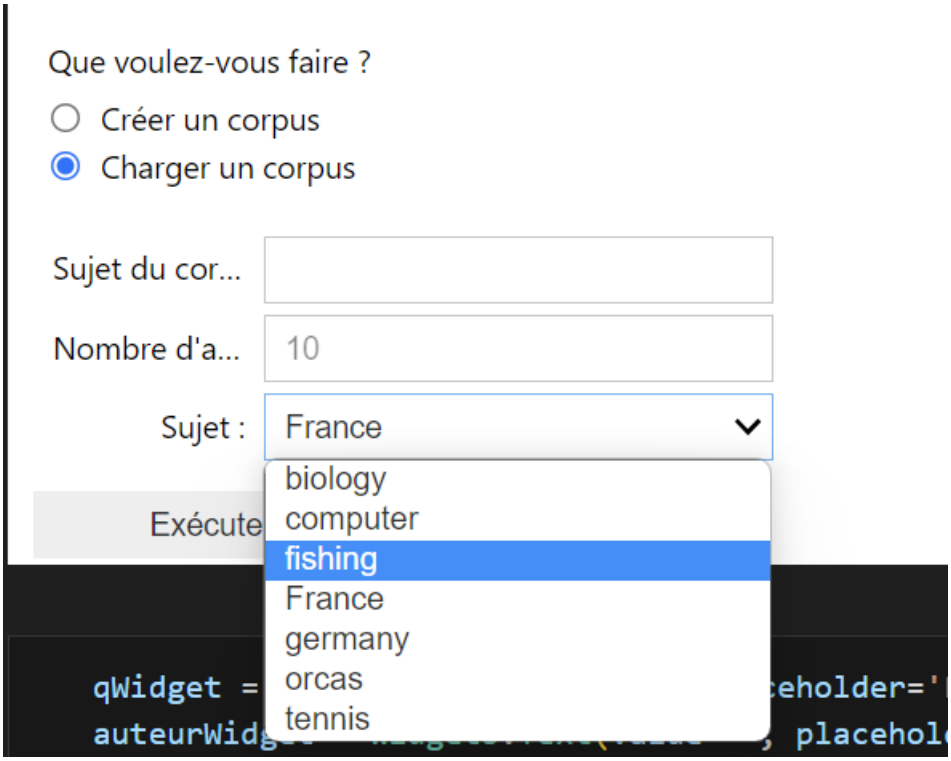
la fonction d'affichage soit une méthode de la classe Corpus, qui produirait du code HTML, affiché ensuite par l'interface, cependant cela donnait lieu à un affichage peu lisible car le texte ne revenait pas à la ligne et « dépassait » de la fenêtre, il était donc impossible de lire le contenu du document intégralement. Nous avons résolu ce problème en implémentant l'affichage directement sur le notebook. Cependant, la cause de ce problème reste inconnue pour nous car c'est virtuellement le même code HTML qui est produit, aucune balise ni aucun contenu n'est différent, mais l'affichage n'est pas le même.

Nous avons fait face à plein d'autres petits problèmes dans l'implémentation de notre interface. Par exemple, les documents affichés ne se remettait pas à jour lorsqu'on changeait les mots-clés et qu'on exécutait une nouvelle recherche. Pour que le notebook mette à jour l'affichage il a fallu implémenter un objet Output, afficher le résultat dans l'output et vider l'output à chaque nouvelle exécution. Aussi, la liste déroulante permettant de choisir un corpus ne se mettait pas à jour non plus (quand on créait un corpus par exemple), donc nous avons implémenté une fonction qui met à jour la liste déroulante, en regardant les fichiers présents dans notre répertoire all_corpus, et nous avons fait en sorte qu'elle se déclenche à chaque fois qu'on clique sur le bouton d'exécution.

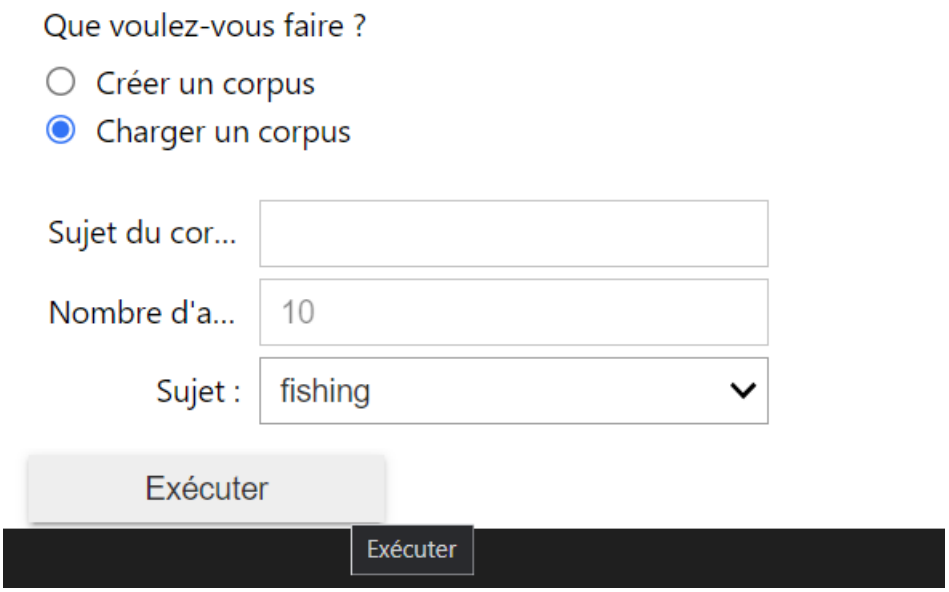
- Exemple d'utilisation :



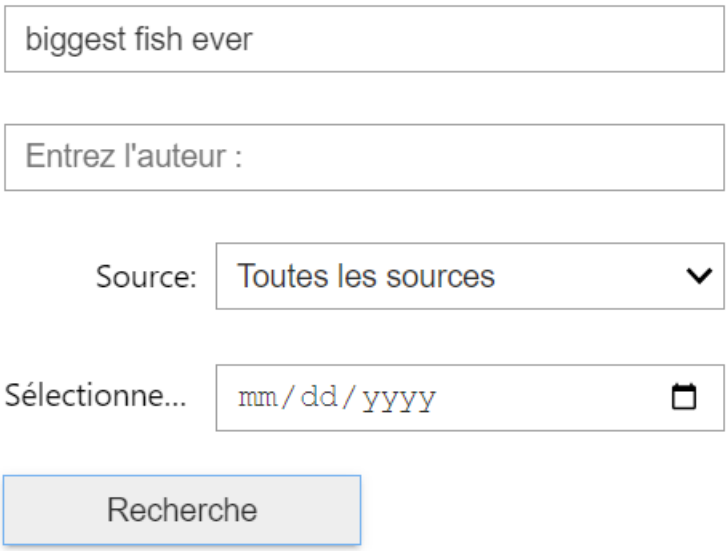
L'interface offre le choix de créer un nouveau corpus ou de charger un corpus préexistant. Lors de la création d'un nouveau corpus, vous choisissez le nombre d'articles qui le constitueront.



Si vous chargez un corpus, une liste déroulante s'affiche avec les corpus déjà existants.



Exécuter une fois votre choix fait.



Effectuez votre recherche par mots-clés, vous pouvez définir la source des documents, leur date, ainsi que l'auteur.

5. La validation

a. Tests unitaires :

Nous avons effectué des tests unitaires à l'aide de la librairie "unittest". Notre focalisation s'est concentrée sur la classe Corpus afin d'établir les tests principaux. Pour faciliter ces tests, nous avons eu recours à ChatGPT pour garantir le bon fonctionnement de notre code. Voici quelques exemples :

i. '**test_save' :** Ce test vérifie que la méthode '**save'** sauvegarde correctement le corpus dans un fichier binaire, puis le charge à nouveau avec '**load'** pour vérifier que le contenu est préservé.
ii. '**test_concatenate':** Ce test vérifie que les documents sont ajoutés correctement au corpus et que la méthode '**concatenate'** renvoie une chaîne de texte contenant le contenu de tous les documents.
iii. '**test_search':** Ce test vérifie que la recherche d'un mot-clé spécifique dans le corpus donne le résultat attendu.

6. La maintenance

La possibilité d'effectuer une analyse comparative de deux corpus, tels qu'Arxiv et Reddit, représente une piste d'amélioration significative pour notre moteur recherche. Elle offre aux utilisateurs une compréhension plus approfondie des différences et similitudes entre deux ensembles de documents. En explorant ces différences et similitudes, les utilisateurs seront en mesure d'identifier des tendances, des thèmes récurrents, et des variations qui peuvent être cruciaux pour leurs recherches.

Aussi, pour améliorer la performance de notre moteur de recherche, il aurait pu être intéressent de construire un vocabulaire lemmetisé et de lemmetiser les mots-clés de notre requête. Cela implique de revoir beaucoup de chose dans le code car il faut reconstruire tout un vocabulaire, et une matrice TFxIDF à partir de ces documents lemmetisés. Le score de similarité pourrait même être amélioré en prenant en compte le nombre de mot issu d'un même lemme présent dans le document. Cette approche permettrait de mieux discriminer les documents en fonction du sens de leur contenu par rapport aux mots-clés renseignés par l'utilisateur. Cependant, cela reste à tester car cela pourrait peut-être entrainer d'autres problèmes, comme un manque de précision dans les termes.