M1 Informatique

Rapport de projet Python en Binôme

Nissrine Ben Ayou et Aissatou Lamarana Barry

Lien vers le dépot GitHub : https://github.com/Nirsenn/python-project

1. Les spécifications

Dans notre cas, elles ont été rédigées à partir des énoncés des TDs et des questions posées à l'enseignante.

L'objectif principal de ce projet est la mise au point d'un moteur de recherche d'information agissant sur un corpus de documents provenant des sites internet Reddit et ArXiv. Les fonctionnalités attendues par le client sont le chargement des données, leur sauvegarde, leur analyse, et enfin la production d'une interface permettant l'utilisation de ces fonctionnalités.

Chargement des données :
Le programme doit pouvoir extraire des documents à partir d'une source externe. Le binôme doit faire le choix d'une thématique identifiable en mots clés, et extraire des documents en ligne en lien avec la thématique choisie à travers des requêtes API sur deux sources (Reddit et ArXiv).

Sauvegarde des données :
Le programme doit pouvoir stocker les informations extraites des APIs afin de ne pas avoir à générer des requêtes à chaque exécution. Pour cela, les données devront être organisées dans une structure principale, le corpus, qui doit être formée par :

- Les métadonnées liées aux textes des documents
- Les métadonnées liées aux auteurs des document

Analyse du contenu :
Le programme doit pouvoir offrir à l'utilisateur les outils nécessaires pour la visualisation, l'exploration et l'analyse des données contenues dans le corpus. Cela doit être fait via un moteur de recherche de mots clés entrés par l'utilisateur.

Interface :
Enfin, le programme doit proposer une interface utilisateur simple et visuellement intuitive, afin d'y effectuer l'analyse de contenu mentionnée précédemment. Le binôme est libre sur la forme de celle- ci et est encouragé à améliorer les fonctionnalités demandées ou à en ajouter d'autres.

Chaque fonctionnalité doit être implémentée sans utiliser de solutions "pré-faites", comme par exemple la librairie scikit-learn, ceci afin de permettre au binôme de mettre en oeuvre leur compétences en programmation mais également d'apprendre le processus derrière ces librairies.

2. L'analyse

Environnement de travail

Le projet a été entièrement développé en **Python 3.12**.
Deux environnements de développement ont été utilisés selon les besoins : **Google Colab** et **Visual Studio Code**.

Google Colab a été choisi pour sa simplicité d'utilisation, la présence de nombreuses bibliothèques déjà installées et la facilité de sauvegarde et de partage des fichiers via Google Drive.
Visual Studio Code, utilisé en local ou en environnement virtuel, offre un cadre plus familier et permet de travailler sans connexion Internet.

Le projet a été versionné sur **GitHub**, outil imposé par le client mais également choisi pour la traçabilité des modifications, la collaboration et le partage du code.

Données identifiées

Les données proviennent de deux sources distinctes :

- **Reddit**, un réseau social basé sur des communautés,
- **ArXiv**, une encyclopédie d'articles scientifiques.

Une clé API est nécessaire pour les requêtes Reddit, tandis qu'ArXiv est accessible via une librairie Python.

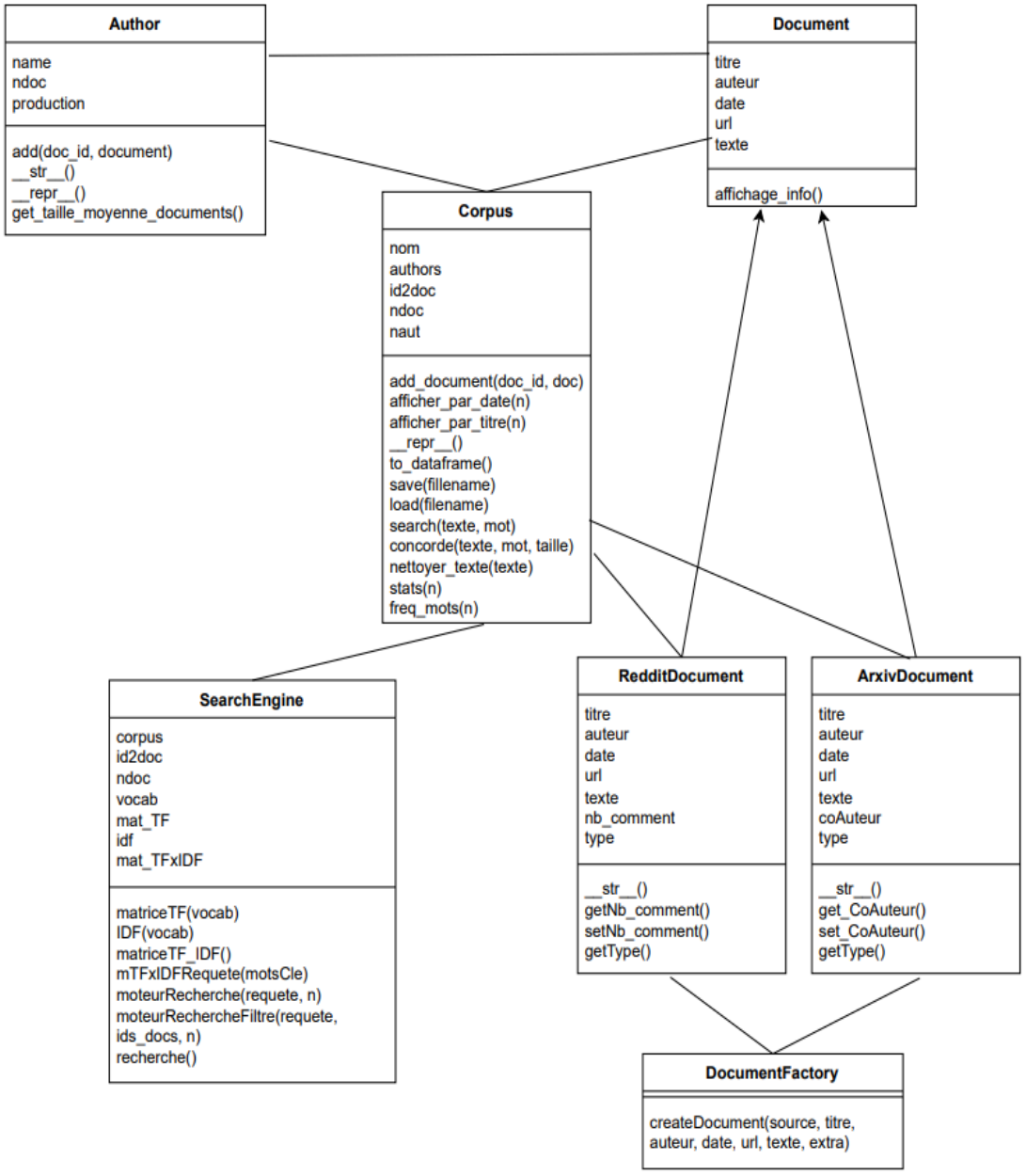
Classes et structure du projet

Pour la création du corpus, plusieurs classes sont nécessaires :

- **Document**, qui à pour attributs *titre, auteur, date, url, texte, type* et est distinguée par deux classes filles qui possèdent chacune un attribut spécifique

    - **ArxivDocument** (nom des co-auteurs de l'article)
    - **RedditDocument** (nombre de commentaires sous le post).
- **Document Factory**, qui permet d'instancier un objet document en fonction de sa source (Reddit ou ArXiv).
- **Author**, qui gère les métadonnées liées aux auteurs des documents (nom, nombre de documents écrits, textes).
- **Corpus**, qui permet de rassembler tous les documents (nom du corpus, objets Author, objets Document, nombre de documents, nombre d'auteurs).
- **SearchEngine**, qui permet, à partir d'un corpus, la construction d'une matrice Documents x Termes, exploitée par le moteur de recherche et plusieurs méthodes d'analyse de texte.

un diagramme des classes simplifié :



3. La conception

Pour implémenter les fonctionnalités mentionnées, nous avons séparé chaque classe dans son propre fichier .py pour garder un code organisé et structuré.

Nous avons créé un compte Reddit afin de pouvoir générer une clé API pour effectuer les requêtes. Cette clé est utilisée dans une boucle qui effectue une recherche des 10 articles les plus tendances du subReddit “Machine Learning”. Nous avons filtré la recherche afin que le résultat n’incluent pas les posts avec peu de contenu textuel, ou les posts provenant du modérateur bot de la communauté. Pour les documents ArXiv, la librairie requests fut suffisante afin de récupérer les 10 meilleurs articles obtenus lorsque l’on cherche Machine Learning.
Les boucles nous permettent de récupérer, résultat par résultat, les données nécessaires à la création d’un objet Document et d’un objets Author. Ces objets sont ensuite stockés dans deux dictionnaires distincts qui seront utilisés pour la création d’un objet Corpus.

Pour l’interface utilisateur, nous avons utilisé la librairie ipywidgets qui permet l’affichage de boutons et autres contenus à la façon HTML. Nous y avons intégré les fonctions de recherche sur le corpus, ce qui permet à n’importe quel utilisateur de faire parler l’outil de manière aisée.

Concernant le partage de tâches, lors de chaque cours en TD nous nous sommes accordées afin qu’elles soient réalisées équitablement, cela après avoir échangé sur notre compréhension des fonctionnalités demandées et des résultats attendus. Ainsi, notre temps de travail fut grandement optimisé.

Lors du développement, nous avons rencontré divers problèmes qui ont pu ralentir l’avancement de notre travail. Parmis eux notamment :

- La gestion des noms d’auteur Arxiv. Contrairement aux posts Reddit qui n’ont qu'un seul auteur, les articles ArXiv peuvent en avoir plusieurs. Il a alors fallu gérer le format par lequel l’API renvoie ces données, et les traiter afin de séparer le premier auteur, qui figurera comme auteur, et tous les auteurs suivants qui seront considérés comme co-auteurs et stockés dans l’attribut nb_comments de la classe ArxivDocument.
- La gestion du format de date. Le format de dates renvoyé par les requêtes API est différent sur Reddit et Arxiv, ce qui générait des problèmes lorsque nous avons voulu développer les méthodes pour exploiter ou analyser les années de parution des documents.

4. La conception

Le projet a été conçu pour permettre la **création, l'analyse et la recherche de documents** issus de différentes sources, notamment **Reddit et ArXiv** via leurs **API respectives**. Il illustre l'ensemble du processus, de la collecte de données à la visualisation des résultats.

1. Extraction des données et création des auteurs

Les documents sont d'abord récupérés automatiquement via :

- **L'API Reddit** pour collecter des posts et commentaires depuis des subreddits spécifiques.

- **L'API ArXiv** pour récupérer des articles scientifiques et leurs métadonnées.

Chaque document est ensuite associé à un **auteur**, et ses informations principales (titre, texte, date, subreddit ou identifiant ArXiv) sont stockées. Cette étape assure la structuration initiale des données et leur préparation pour l'analyse.

2. Construction du corpus

Les documents récupérés sont regroupés dans un **corpus**, qui constitue la structure centrale pour l'exploration et l'analyse. Le corpus permet de trier, filtrer et manipuler les documents de manière uniforme, quel que soit leur type ou leur source.

3. Prétraitement et analyse textuelle

Avant l'indexation, les textes sont **nettoyés et prétraités** : suppression de la ponctuation et des chiffres, normalisation de la casse, et tokenisation. Des statistiques de base sont générées, comme la fréquence des mots et la composition du vocabulaire.

4. Indexation et moteur de recherche

Le projet construit une **matrice Documents × Termes**, calculant les scores TF et TF-IDF. Cette étape permet de lancer des **recherches par mots-clés** et d'obtenir les documents les plus pertinents, classés par similarité avec la requête.

5. Recherche et exploration interactive

Une **interface Jupyter Notebook** offre un environnement interactif pour :

- Rechercher des mots-clés ou expressions

- Filtrer par **auteur**, **type de document** ou **période**

- Visualiser le contexte des mots grâce à un concordancier

6. Visualisation et comparaison

Le projet inclut également des outils de visualisation :

- Comparaison de plusieurs corpus (par exemple Reddit vs ArXiv)

- Analyse de l'évolution temporelle des mots

- Génération de **Word Clouds** pour illustrer les mots les plus fréquents dans le corpus

Commentaires

- L'extraction via API permet d'automatiser le processus et de récupérer des documents à jour.

- Les étapes du projet restent **modulaires**, ce qui permet d'ajouter de nouveaux documents à tout moment.

- Le moteur de recherche peut utiliser TF ou TF-IDF selon les besoins de l'utilisateur.

- L'ensemble du processus est pensé pour être **intuitif, pédagogique et interactif**, adapté à l'analyse de corpus variés.

5. Tests globaux

L'interface Jupyter Notebook a été testée sur plusieurs **cas représentatifs** afin de vérifier la robustesse et la fiabilité du projet. Les tests ont porté sur différents aspects: collecte des données, traitement, recherche et visualisation.

1. Test de récupération des documents

- **Reddit** : extraction de posts et commentaires depuis plusieurs subreddits avec des volumes différents (petits vs grands nombres de posts).

- **ArXiv** : récupération d'articles avec des métadonnées complètes et certaines manquantes (ex. absence de résumé ou de date).
  Objectif : vérifier que le programme gère correctement les données provenant de sources variées et les ajoute au corpus sans erreur.

2. Test de prétraitement et tokenisation

- Documents contenant **ponctuation, chiffres, majuscules/minuscules**.

- Textes très longs et textes courts.
  Objectif : s'assurer que le nettoyage et la tokenisation fonctionnent pour tous les types de textes.

3. Test du moteur de recherche

- Requêtes simples : un seul mot-clé.

- Requêtes complexes : plusieurs mots-clés ou expressions.

- Requêtes avec **mots absents du corpus**.
  Objectif : vérifier que le moteur renvoie les documents pertinents, même lorsque la requête ne correspond à aucun document, et que les résultats sont correctement classés par TF ou TF-IDF.

4. Test des filtres interactifs

- Filtrage par **auteur**, **type de document** (Reddit/ArXiv), et **période**.

- Combinaison de filtres multiples.
  Objectif : vérifier que les filtres fonctionnent correctement et que les documents affichés correspondent aux critères sélectionnés.

5. Test des visualisations

- **Concordancier** : visualisation du contexte des mots-clés dans différents documents.

- **Word Clouds** : génération avec corpus de taille variable.

- **Comparaison de corpus** : identification des mots spécifiques ou communs.
  Objectif : s'assurer que les visualisations reflètent correctement les données et restent lisibles pour l'utilisateur.

Conclusion des tests

Les tests ont montré que le programme :

- Gère correctement différents types de documents et sources.
- Prétraite et analyse le texte de manière fiable.
- Fournit un moteur de recherche robuste avec filtres interactifs.
- Génère des visualisations pertinentes et adaptées à l'analyse.

5. Maintenance et évolutions possibles du logiciel

Le projet est organisé en plusieurs classes, ce qui facilite sa compréhension et sa maintenance.

Il est donc possible de faire évoluer le logiciel sans modifier toute sa structure.

Par exemple, on pourrait **ajouter de nouvelles sources de données** en créant de nouveaux types de documents, sur le même modèle que Reddit et ArXiv.
Le **moteur de recherche** pourrait aussi être amélioré avec des méthodes d'analyse plus avancées.

L'interface, actuellement sous forme de notebook Jupyter, pourrait évoluer vers une **interface web** plus simple pour l'utilisateur.

Enfin, grâce au versionnement avec Git et à l'utilisation de tags pour les différentes versions du projet, la maintenance du code est facilitée. Les évolutions peuvent être ajoutées progressivement sans remettre en cause le fonctionnement global du programme.