Rapport Projet Python

1. Les spécifications :

L'objectif du projet est de concevoir une application interactive permettant d'explorer et analyser un corpus de discours politiques américains.
Cette application ne s'adresse pas à des utilisateurs informaticiens mais plutôt à des chercheurs, linguistes, sociologues etc… souhaitant analyser rapidement un corpus volumineux

L'application doit fournir trois fonctionnalités principales en complément d'une interface utilisateur :

1.1 Recherche par mots-clés

L'utilisateur doit pouvoir :

- saisir un ou plusieurs mots-clés
- choisir le nombre de résultats à afficher
- lancer une requête via un bouton "rechercher"
- obtenir les phrases du corpus les plus pertinentes selon un score TF-IDF

La fonction Recherche doit afficher les résultats sous forme de tableau comprenant :

- Le numéro du Discours
- La phrase où apparaît le mot clé choisi
- La source (Reddit ou Arxiv)
- Le score TF-IDF

1.2 Analyse comparative d'un mot dans le corpus

L'utilisateur doit pouvoir analyser un mot (seul). Pour un mot choisit, le programme doit calculer et afficher :

- Sa fréquence totale dans l'ensemble du corpus
- Son importance moyenne selon son score TF-IDF moyen
- Une analyse automatique du mot selon sa fréquence totale et son score TF-IDF moyen

Une interface dédiée permet à l'utilisateur de saisir le mot et d'afficher les résultats de l'analyse.

1.3 Identification du discours utilisant le plus le mot

L'utilisateur doit pouvoir savoir quel est le discours où le mot choisi apparaît le plus souvent. L'application doit afficher :

- Le numéro (titre) du discours
- L'auteur du discours
- La date du discours
- le texte complet du discours concerné

Cette fois aussi une interface dédiée permet à l'utilisateur de saisir le mot et d'afficher les résultats de l'analyse.

1.4 Interface utilisateur

L'outil doit être utilisable dans un environnement Jupyter Notebook. Il doit intégrer une interface simple, épurée et intuitive basée sur la bibliothèque ipywidgets, incluant :

- Des zones de saisie pour les requêtes et les analyses
- Un slider
- Des boutons pour lancer les recherches et les analyses
- Une zone de sortie mise à jour dynamiquement suivant les résulta

2. Analyse :

2.1 Environnement de travail

Le développement a été mené en Python 3 dans l'éditeur Visual Studio Code. VS Code a été choisi car il permet de travailler à la fois avec des fichiers Python classiques et des notebooks Jupyter (via l'extension *Jupyter*), ce qui est utile pour :

- L'exécution interactive et itérative
- L'édition de modules (Corpus.py, Document.py, SearchEngine.py)
- L'intégration avec git et la gestion de projet

Les bibliothèques principales utilisées sont pandas, numpy, scipy.sparse (matrices creuses), et ipywidgets pour l'interface.

2.2 Données identifiées

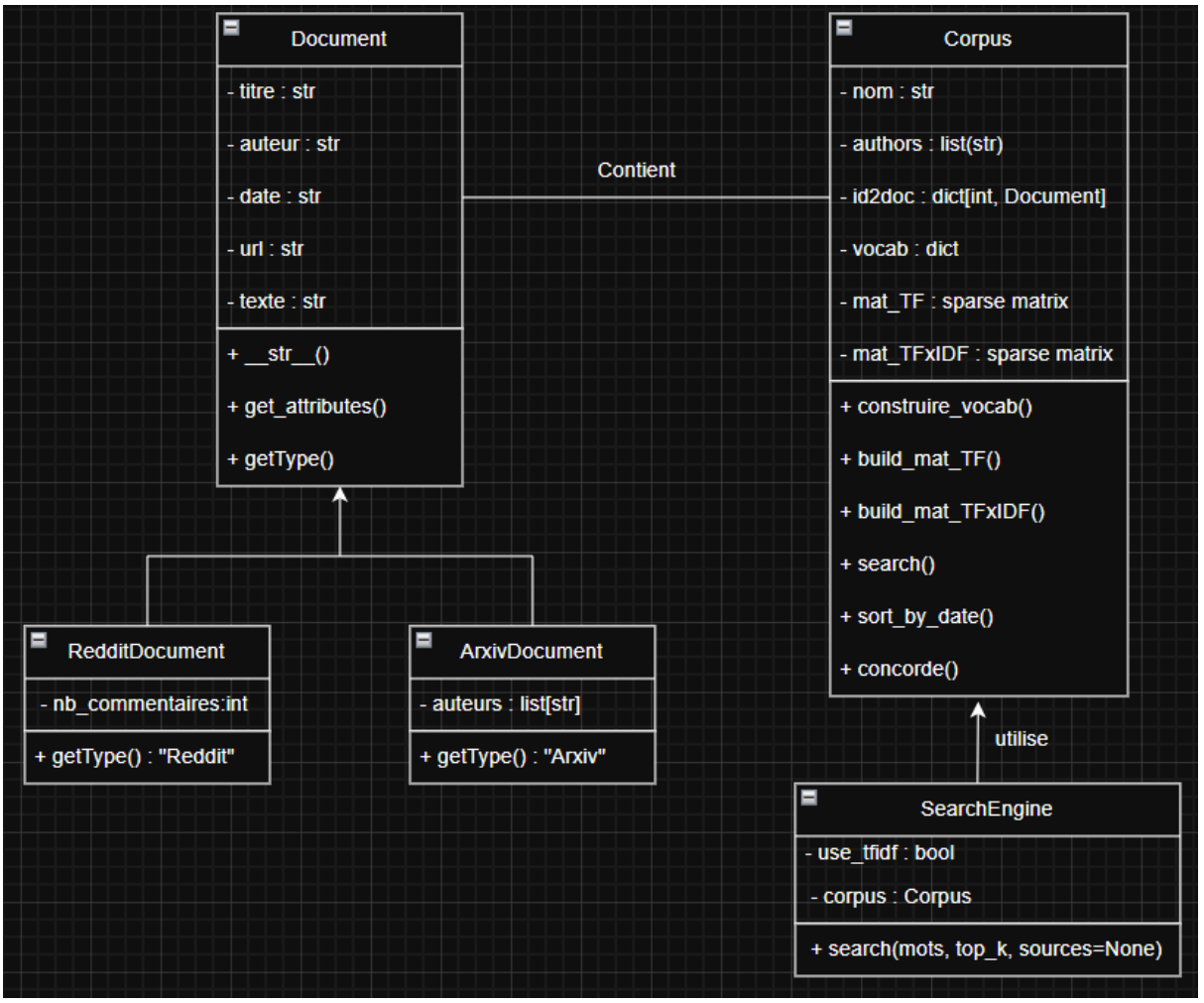
Voici les sources de données utilisées pour mener à bien ce projet :

- discours_US.csv (jeu de données local fourni) : colonnes : speaker, text, date, descr, link
- Reddit — posts récupérés via praw : titre, texte, nombre de commentaires, auteur, date, url
- ArXiv — flux exporté via l'API ArXiv : titre, résumé, auteurs, date, url

Voici cette fois, les données manipulées en interne :

- Document : Représente une unité textuelle (dans notre cas, une phrase). Il contient : *titre*, *auteur*, *date*, *URL*, *texte*.
- Corpus : Ensemble de tous les documents collectés. Il regroupe :
    - Un dictionnaire `id2doc` qui associe chaque identifiant à son document
    - La liste des auteurs présents
    - Le vocabulaire global
    - Deux matrices décrivant la représentation vectorielle du corpus (TF et TF-IDF)
- Vocabulaire : Dictionnaire où chaque mot est associé à 3 informations essentielles :
    - Son identifiant numérique
    - Son tf_total (fréquence globale dans le corpus)
    - Son df (nombre de documents où il apparaît)
- Matrices de représentation :
    - TF (Term Frequency) : Fréquence des mots dans chaque document.
    - TF-IDF : Pondération tenant compte de l'importance du mot dans le corpus.
      Les deux matrices sont stockées en format CSR (Compressed Sparse Row) pour optimiser la mémoire et accélérer les calculs.

2.3 Architecture & diagramme des classes (en texte pour l'instant)



Explications :

- Document est la classe mère représentant une unité textuelle.
- RedditDocument et ArxivDocument héritent de Document et ajoutent des attributs spécifiques liés à leur source.
- Corpus regroupe l'ensemble des documents, construit le vocabulaire global et les matrices de représentation TF et TF-IDF.
- SearchEngine utilise un Corpus pour effectuer les recherches par mots-clés et retourner les documents les plus pertinents.
- L'interface utilisateur (via ipywidgets) n'est pas modélisée comme une classe métier, elle utilise directement SearchEngine et Corpus.

Conclusion de la partie Analyse

La conception vise la clarté, la réutilisabilité et la simplicité d'usage. L'architecture propose des classes bien séparées (Documents / Corpus / SearchEngine) et une interface Jupyter accessible pour l'utilisateur final. La gestion multi-source est traitée via un champ origine ou par instanciation de plusieurs Corpus selon le niveau de séparation souhaité.

3. La conception :

3.1 Répartition du travail

Le projet a été réalisé en binôme. Le travail a été réparti de manière complémentaire afin de travailler efficacement et de manière égale. La plupart des TD, nous séparions les tâches en 2, puis nous travaillions chacuns de nos côtés en s'entraidant si besoin à la rencontre d'un problème. Par exemple, l'un s'est occupé de Reddit et l'autre de Arxiv au premier TD. Nous mettions en commun le moment venu et apportions les modifications nécessaires.

3.2 Construction et préparation des données

Les données textuelles proviennent de plusieurs sources (discours politiques américains, Reddit, ArXiv). Chaque document est découpé en phrases, ce choix permettant :

- une recherche plus fine et plus pertinente
- une meilleure interprétation des résultats par l'utilisateur
- une réduction de l'effet de documents trop longs sur les scores

Chaque phrase est ensuite encapsulée dans un objet Document (ou une de ses classes filles), puis stockée dans le dictionnaire id2doc du corpus.

3.3 Construction du vocabulaire

Une fois l'ensemble des documents chargé, un vocabulaire global est construit. Le corpus est parcouru afin d'identifier l'ensemble des mots distincts présents dans les textes. Pour chaque mot, les informations suivantes sont stockées :

- un identifiant numérique unique
- sa fréquence totale dans le corpus (TF global)
- son nombre de documents d'apparition

Ce vocabulaire constitue la base de la représentation vectorielle du corpus et permet un alignement cohérent entre les documents et les requêtes utilisateurs.

3.4 Représentation vectorielle : TF et TF-IDF

Le corpus est représenté sous forme de deux matrices creuses :

- une matrice TF (Term Frequency) indiquant la fréquence des mots dans chaque document
- une matrice TF-IDF pondérant la fréquence des mots par leur importance relative dans l'ensemble du corpus

Les matrices sont stockées au format CSR (Compressed Sparse Row), ce qui permet :

- une réduction significative de l'utilisation mémoire
- des accès rapides aux lignes et colonnes lors des calculs de similarité

Cette représentation est essentielle pour assurer des performances correctes lors des recherches sur des corpus de taille importante.

3.5 Moteur de recherche

Le moteur de recherche est implémenté dans la classe SearchEngine. Lorsqu'un utilisateur saisit une requête constituée d'un ou plusieurs mots-clés :

1) Les mots de la requête sont transformés en un vecteur aligné sur le vocabulaire du corpus
2) Le moteur sélectionne la matrice appropriée (TF ou TF-IDF)
3) Un score de similarité est calculé entre le vecteur requête et chaque document du corpus.Les documents sont triés par score décroissant
4) Les top_k documents les plus pertinents sont retournés

Les résultats sont renvoyés sous forme de pandas.DataFrame, facilitant leur affichage et leur exploitation dans l'interface graphique.

3.6 Analyse comparative d'un mot

Une fonctionnalité spécifique permet d'analyser un mot indépendamment d'une requête classique. Pour un mot donné, le programme calcule :

- sa fréquence totale dans l'ensemble du corpus
- son score TF-IDF moyen
- une interprétation automatique basée sur ces deux indicateurs

Cette analyse permet de distinguer les mots très fréquents mais ordinaires et peu impactants des mots plus rares mais spécifiques à certains discours.

3.7 Identification du discours le plus représentatif

Afin d'enrichir l'analyse, l'application permet également d'identifier le discours dans lequel un mot apparaît le plus fréquemment. Pour cela, le programme :

- parcourt l'ensemble des documents du corpus
- calcule le nombre d'occurrences du mot pour chaque discours
- sélectionne le discours ayant la fréquence maximale

Les métadonnées associées (titre, auteur, date, texte) sont ensuite affichées afin de contextualiser l'analyse.

3.8 Interface utilisateur

L'interface utilisateur a été développée dans un environnement Jupyter Notebook à l'aide de la bibliothèque ipywidgets. Elle repose sur :

- des zones de saisie pour les mots-clés et les analyses
- un slider permettant de choisir le nombre de résultats
- des boutons déclenchant les traitements
- une zone Output mise à jour dynamiquement

Ce choix permet une interaction simple et intuitive, adaptée à des utilisateurs non informaticiens, tout en conservant la flexibilité du langage Python.

3.9 Problèmes rencontrés et solutions

Le développement du projet a donné lieu à plusieurs difficultés techniques, certaines liées tant à la manipulation des données textuelles qu'à la mise en place des structures internes et de l'interface utilisateur. Cette section détaille les principaux problèmes rencontrés ainsi que les solutions apportées.

Problèmes liés à l'hétérogénéité des données

Les documents proviennent de sources différentes (discours politiques, Reddit, ArXiv), avec des formats et des métadonnées parfois variables. Cela a posé des difficultés lors de la création d'une structure de données commune.
En effet, certains documents possédaient des champs absents ou facultatifs (URL, auteurs multiples, nombre de commentaires).
On a donc défini une classe Document générique contenant uniquement les attributs communs à toutes les sources. Enfin, on a fixé certaines valeurs par défaut (ex. nombre de commentaires à 0) afin d'éviter les erreurs lors de l'instanciation des objets.

Problèmes liés aux matrices creuses (TF et TF-IDF)

L'utilisation de matrices creuses a été nécessaire pour des raisons de performance, mais a introduit une complexité supplémentaire. En effet, on a eu pas mal d'erreurs de type : matrix is not subscriptable. De plus, on a eu des difficultés à récupérer correctement une colonne correspondant à un mot donné. On a donc décidé de convertir systématiquement les matrices au format CSR et d'utiliser des méthodes adaptées (getcol, toarray, sum) pour accéder aux valeurs. Puis on a centralisé les opérations matricielles dans la classe Corpus.

Problèmes liés au moteur de recherche

La construction du moteur de recherche a nécessité plusieurs ajustements. Les résultats étaient non pertinents lorsque plusieurs mots-clés étaient fournis par l'utilisateur. De plus, l'application avait beaucoup de mal avec la gestion des requêtes contenant des mots inconnus. On a donc normalisé les scores avant le tri et ignoré automatiquement les mots absents du vocabulaire afin d'éviter les erreurs.

3.10 Exemple d'utilisation du programme

L'utilisateur lance le notebook et accède à l'interface graphique. Il peut saisir une requête composée de mots-clés, sélectionner le nombre de résultats souhaité, puis lancer la recherche. Les phrases les plus pertinentes sont affichées avec leur score TF-IDF. L'utilisateur peut ensuite analyser ou comparer un mot spécifique afin d'en observer l'importance dans le corpus et identifier le discours où il apparaît le plus.

4. La validation :

4.1 Tests unitaires

Les tests unitaires ont été réalisés de manière progressive lors du développement, tout d'abord à chaque fin de TD mais aussi à d'autres moments stratégiques principalement par des scripts de test simples dans Visual Studio Code.

Tests de la classe Document et de ses sous-classes

Objectif : Vérifier que les documents sont correctement instanciés et que leurs attributs sont accessibles.
Tests effectués :

- Vérification de la présence des attributs : titre, auteur, date, url, texte
- Vérification des attributs spécifiques :
    - nb_commentaires pour RedditDocument
    - auteurs (liste) pour ArxivDocument
- Test de la méthode getType() pour identifier la source du document

Résultat : Les objets sont correctement créés et les attributs sont accessibles sans erreur

Tests de la classe Corpus

Objectif : Valider la construction correcte du corpus et de ses structures internes.
Tests effectués :

- Vérification du nombre de documents (ndoc) après chargement
- Test de la méthode construire_vocab() :
    - Taille du vocabulaire cohérente avec les données
    - Présence des mots attendus
- Test de la méthode build_mat_TF() :
    - Dimensions correctes de la matrice (documents × termes)
    - Valeurs positives et cohérentes
- Test de la méthode build_mat_TFxIDF() :
    - Valeurs nulles pour les mots trop fréquents
    - Pondération correcte des termes plus spécifiques

Résultat : Les matrices sont construites sans erreur et correspondent aux attentes théoriques.

Tests du moteur de recherche (SearchEngine)

Objectif : S'assurer que le moteur de recherche retourne des résultats justes et stables.
Tests effectués :

- Requêtes avec un mot unique
- Requêtes avec plusieurs mots-clés
- Requêtes contenant des mots absents du vocabulaire
- Comparaison entre recherche TF et TF-IDF

Résultat :

- Les résultats sont correctement triés par score décroissant.
- Les mots inconnus sont ignorés sans provoquer d'erreur
- Les résultats retournés sont cohérents avec le contenu du corpus

4.2 Tests globaux via l'interface utilisateur

Les tests globaux ont été réalisés directement via l'interface interactive construite avec ipywidgets.

Cas de test principaux

Recherche simple

- Saisie d'un mot-clé fréquent
- Résultats affichés instantanément sous forme de tableau
- Les phrases retournées contiennent bien le mot recherché

Recherche multi-mots

- Saisie de plusieurs mots-clés
- Résultats combinant les scores des différents termes
- Classement pertinent des phrases

Variation du nombre de résultats

- Utilisation du slider Top k
- Vérification que le nombre de résultats affichés correspond à la valeur sélectionnée

Champ vide

- Lancement de la recherche sans mot-clé
- Message d'erreur clair affiché à l'utilisateur

Tests de l'analyse d'un mot

Objectif : Valider les fonctionnalités d'analyse d'un mot.
Tests effectués :

- Calcul de la fréquence totale d'un mot
- Calcul du score TF-IDF moyen

- Identification du discours utilisant le plus le mot

Résultat :

- Les valeurs calculées sont cohérentes avec les données brutes
- Les résultats sont compréhensibles et exploitables par un utilisateur non informaticien

4.3 Cas particuliers testés

- Mot très fréquent (ex : the, and…)
- Mot très rare apparaissant dans un seul discours
- Mot inexistant dans le corpus
- Exécution répétée de recherches sans recharger le corpus

Dans tous les cas, l'application se comporte de manière stable et prévisible.

4.4 Conclusion de la validation

L'ensemble des tests unitaires et globaux permet de conclure que :

- Le logiciel respecte les spécifications définies
- Les résultats produits sont fiables et cohérents
- L'interface utilisateur est stable et utilisable par des non-informaticiens

La validation montre que l'outil est fonctionnel et prêt à être utilisé pour l'analyse de corpus textuels.

5. La maintenance :

5.1 Ajout de nouvelles sources de données

Il est possible d'enrichir l'application avec de nouvelles sources textuelles (presse, discours politiques européens, réseaux sociaux, etc.). Pour cela il suffirait de créer une nouvelle classe héritant de Document (par exemple NewsDocument), implémenter les attributs spécifiques à la source et enfin ajouter les documents au Corpus existant ou à un nouveau corpus.
Difficulté **:** faible. Grâce à l'héritage et à la structure générique de la classe Corpus, l'ajout de nouvelles sources est relativement simple.

5.2 Amélioration de l'interface utilisateur

L'interface actuelle est simple mais peut être enrichie. En effet, les améliorations possibles seraient de :

- Sélectionner la source (Reddit / ArXiv / Discours) via des menus déroulants
- Mise en forme avancée des résultats (surlignage des mots-clés etc…)
- Ajout de graphiques interactifs

Difficultés **:** faible (pour les 2 premiers) / moyenne (pour la troisième). L'utilisation d'ipywidgets permet des évolutions progressives sans refonte complète.

5.3 Ajout de nouvelles mesures statistiques

D'autres mesures de pondération ou d'importance des mots pourraient être intégrées. Par exemple, on pourrait faire un regroupement thématique simple.
Difficulté **:** moyenne. Cette ajout concerne principalement la partie algorithmique et peut être intégré sans trop modifier l'interface existante.

Conclusion

Ce projet avait pour objectif de concevoir un outil permettant l'exploration et l'analyse d'un corpus textuel volumineux pour des utilisateurs non-informaticiens. À travers les différentes étapes du projet, nous avons mis en œuvre une approche progressive mélangeant modélisation des données, traitement automatique du langage et interaction utilisateur.

Nous avons développé une application capable de charger et structurer des corpus différents (discours politiques, Reddit, ArXiv), de représenter les documents sous forme vectorielle avec des mesures statistiques classiques (TF et TF-IDF), et de proposer des fonctionnalités de recherche et d'analyse accessibles via une interface graphique simple basée sur ipywidgets dans un environnement Jupyter Notebook.

Le projet nous a permis de mettre en pratique les notions abordées pendant les TD, notamment la programmation orientée objet, le calcul de similarités, ainsi que la structuration d'une application en modules réutilisables.

Malgré certaines difficultés rencontrées, notamment liées à la gestion des formats de données, à la cohérence des classes et à l'intégration progressive des fonctionnalités, les solutions mises en place ont permis d'arriver à une application qui fonctionne, possiblement extensible et conforme aux consignes du projet.

Enfin, l'architecture adoptée rend l'application plutôt facilement maintenable et ouverte à des évolutions futures, telle que l'intégration de nouvelles mesures statistiques.