UNIVERSITÉ LUMIÈRE LYON 2

Institut de la communication

Master 1 Informatique

Exploration d'un Corpus de Discours Politiques US

Moteur de Recherche Textuel avec Analyse Temporelle

Réalisé par :

Cyrine Nighaoui

Année Universitaire 2025–2026

15 décembre 2025

Table des matières

Table des figures

Chapitre 1

Présentation générale du projet

1.1 Contexte académique et cadre pédagogique

Ce projet s'inscrit dans le cadre du module **Programmation Python** du Master 1 Informatique.

L'ensemble du développement a été réalisé en Python, en respectant les bonnes pratiques de programmation, notamment :

— la modularité du code,

— la séparation des responsabilités,

— la lisibilité et la documentation,

— la réutilisabilité des composants.

1.2 Objectifs généraux du projet

L'objectif principal de ce projet est la conception et l'implémentation d'un **moteur de recherche textuel** appliqué à un corpus de documents . Ce moteur permet d'explorer automatiquement un ensemble de textes et de fournir des résultats pertinents en réponse à une requête utilisateur.

1.3 Approche progressive et évolution du projet

Le développement du projet s'est déroulé de manière progressive, en suivant les différentes étapes imposées par les travaux dirigés.

Les premiers TD ont permis de mettre en place les structures de base, telles que la représentation des documents, des auteurs et du corpus. Les TD intermédiaires ont introduit des fonctionnalités de recherche simple, de concordance et de statistiques

textuelles. Enfin, les TD avancés ont conduit à la conception d'un moteur de recherche vectoriel reposant sur la pondération TF-IDF et la similarité cosinus.

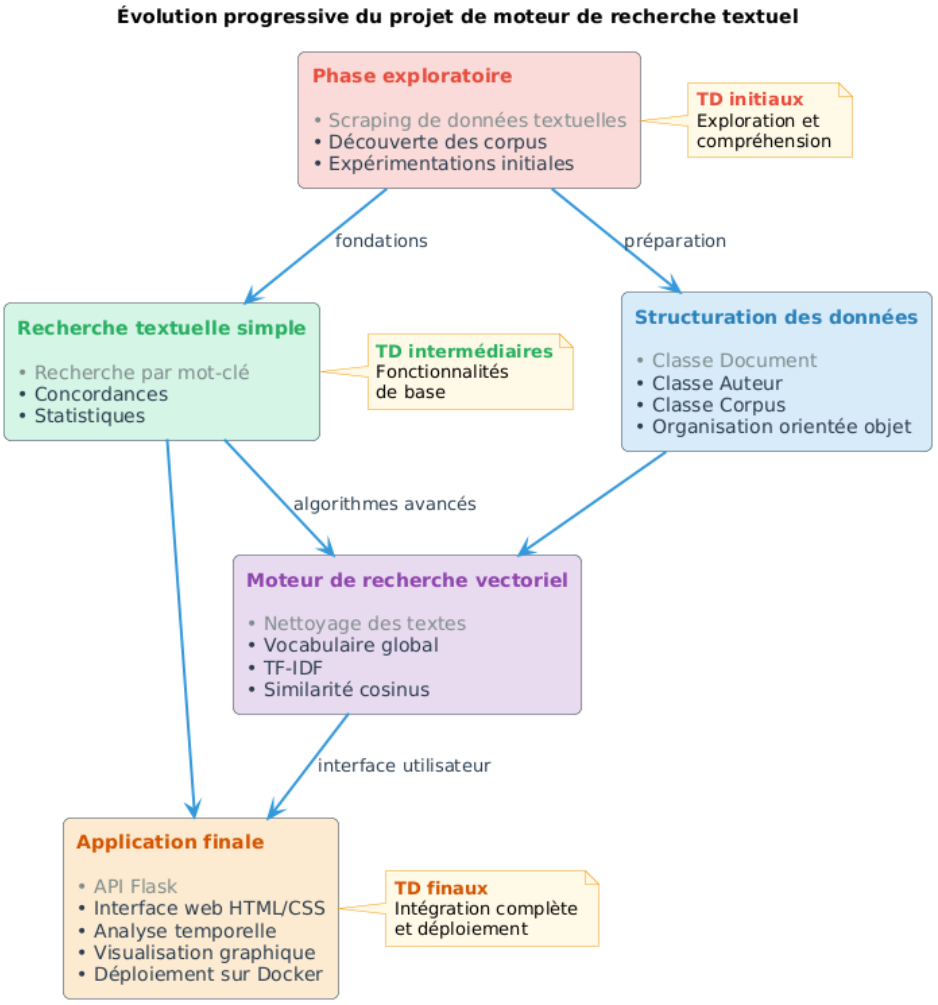

FIGURE 1.1 – Évolution progressive du projet

1.3.1 Phase exploratoire et collecte initiale des données

Avant la mise en place du moteur de recherche textuel, une phase exploratoire a été menée afin de mieux comprendre la nature des données et les problématiques liées à leur exploitation. Cette phase a notamment inclus des travaux de collecte automatique de données textuelles (scraping), ainsi que des expérimentations préliminaires sur le traitement et l'analyse de corpus textuels.

L'objectif de cette étape n'était pas de produire une solution finale, mais de tester différentes approches, d'identifier les limites des données brutes et de mieux

appréhender les enjeux liés au nettoyage, à la structuration et à l'analyse de textes.

Ces travaux exploratoires ont permis d'orienter progressivement le projet vers une approche plus structurée, aboutissant à la conception d'un moteur de recherche textuel basé sur un corpus stabilisé et des méthodes éprouvées de représentation vectorielle.

1.4 Choix technologiques

Environnement principal de développement		
Logo	**Environnement**	**Utilisation**
jupyter	Jupyter Notebook	Environnement interactif utilisé pour l'expérimentation, le prototypage et la validation progressive

Bibliothèques Python principales		
Logo	**Bibliothèque**	**Utilisation**
	Python 3.x	Langage principal de développement
pandas	pandas	Manipulation des données
np	NumPy	Calculs numériques
mpl	Matplotlib	Visualisation de données

Interface web		
Logo	**Technologie**	**Rôle**
Flask web development, one drop at a time	Flask	API backend
HTML	HTML5/CSS3	Structure et style
JS	JavaScript	Interactivité client

1.5 Organisation du rapport

La suite de ce rapport est structurée comme suit :

— le chapitre 2 présente les spécifications fonctionnelles du projet,

— le chapitre 3 décrit l'analyse et la conception du système,

— le chapitre 4 détaille l'implémentation et les algorithmes principaux,

— le chapitre 5 est consacré à l'nterface web et API applicative

— le chapitre 6 aborde les difficultés rencontrée et les perspectives d'évolution du projet.

Chapitre 2

Spécifications fonctionnelles et objectifs du projet

2.1 Contexte général et motivation

L'essor massif des données textuelles issues de sources hétérogènes (documents institutionnels, discours politiques, articles scientifiques, réseaux sociaux, etc.) rend indispensable le développement d'outils capables de structurer, analyser et interroger efficacement de grands volumes de texte.

Dans ce contexte, le présent projet s'inscrit dans une démarche progressive d'apprentissage et de mise en œuvre des techniques fondamentales du traitement automatique du langage naturel (NLP) à travers le développement d'un moteur de recherche textuel personnalisé en Python.

2.2 Objectifs fonctionnels

L'objectif principal du projet est de concevoir un moteur de recherche textuel capable de :

— stocker et organiser un ensemble de documents textuels au sein d'un corpus ;

— nettoyer et prétraiter les textes afin de les rendre exploitables ;

— construire un vocabulaire global à partir du corpus ;

— représenter les documents sous forme vectorielle ;

— calculer la similarité entre une requête utilisateur et les documents du corpus ;

— restituer les documents les plus pertinents sous forme de résultats classés.

Ces objectifs sont complétés par des fonctionnalités secondaires telles que l'analyse de la fréquence des termes.

2.3 Description des données

Les données exploitées dans le cadre de ce projet proviennent d'un jeu de données textuelles contenant des discours politiques américains. Ces données sont stockées dans un fichier au format CSV et comportent plusieurs champs, notamment :

— le nom de l'orateur (`speaker`) ;

— le contenu textuel du discours (`text`) ;

— la date du discours (`date`) ;

— une description du document (`descr`) ;

— un lien vers la source originale (`link`).

	speaker	text	date	descr	link
0	CLINTON	: I'm getting ready for a lot of things, a lot...	April 12, 2015	Video Remarks Announcing Candidacy for President	http://www.presidency.ucsb.edu/ws/index.php?pi...
1	CLINTON	[ ] : I'll be graduating in May, and on gradua...	April 14, 2015	Remarks in a Question and Answer Session at Ki...	http://www.presidency.ucsb.edu/ws/index.php?pi...
2	CLINTON	: Well, thank you all so much for inviting me ...	April 20, 2015	Remarks in Keene, New Hampshire	http://www.presidency.ucsb.edu/ws/index.php?pi...
3	CLINTON	Thank you so much. I am absolutely delighted t...	April 29, 2015	Address to the David N. Dinkins Leadership & P...	http://www.presidency.ucsb.edu/ws/index.php?pi...
4	CLINTON	Oh, hello. Hi, how are you? Well, it's wonderf...	May 5, 2015	Remarks at a Roundtable with Young Nevada Resi...	http://www.presidency.ucsb.edu/ws/index.php?pi...

FIGURE 2.1 – Aperçu du fichier CSV après chargement dans un notebook

La Figure 2.1 présente un aperçu des premières lignes du jeu de données après son chargement dans un notebook Jupyter.

2.4 Spécifications fonctionnelles

Le système développé doit répondre à un ensemble de besoins fonctionnels clairement identifiés, indépendamment des choix d'implémentation technique.

Les principales spécifications fonctionnelles sont les suivantes :

— le système doit permettre à un utilisateur de soumettre une requête textuelle composée d'un ou plusieurs mots-clés ;

— le système doit être capable de comparer la requête utilisateur à l'ensemble des documents du corpus ;

— le système doit restituer une liste de documents classés par ordre de pertinence ;

— le système doit permettre l'analyse de la fréquence d'apparition d'un mot-clé dans le temps .

Ces spécifications définissent le périmètre fonctionnel du moteur de recherche et servent de base aux choix de conception et d'implémentation présentés dans les chapitres suivants.

Chapitre 3

Conception et modélisation du système

3.1 Choix de conception générale

La conception du système repose sur une approche orientée objet afin de garantir une bonne modularité, une lisibilité accrue du code et une extensibilité future. Chaque entité du système est modélisée sous forme de classe, avec des responsabilités clairement définies.

3.2 Diagramme de classes

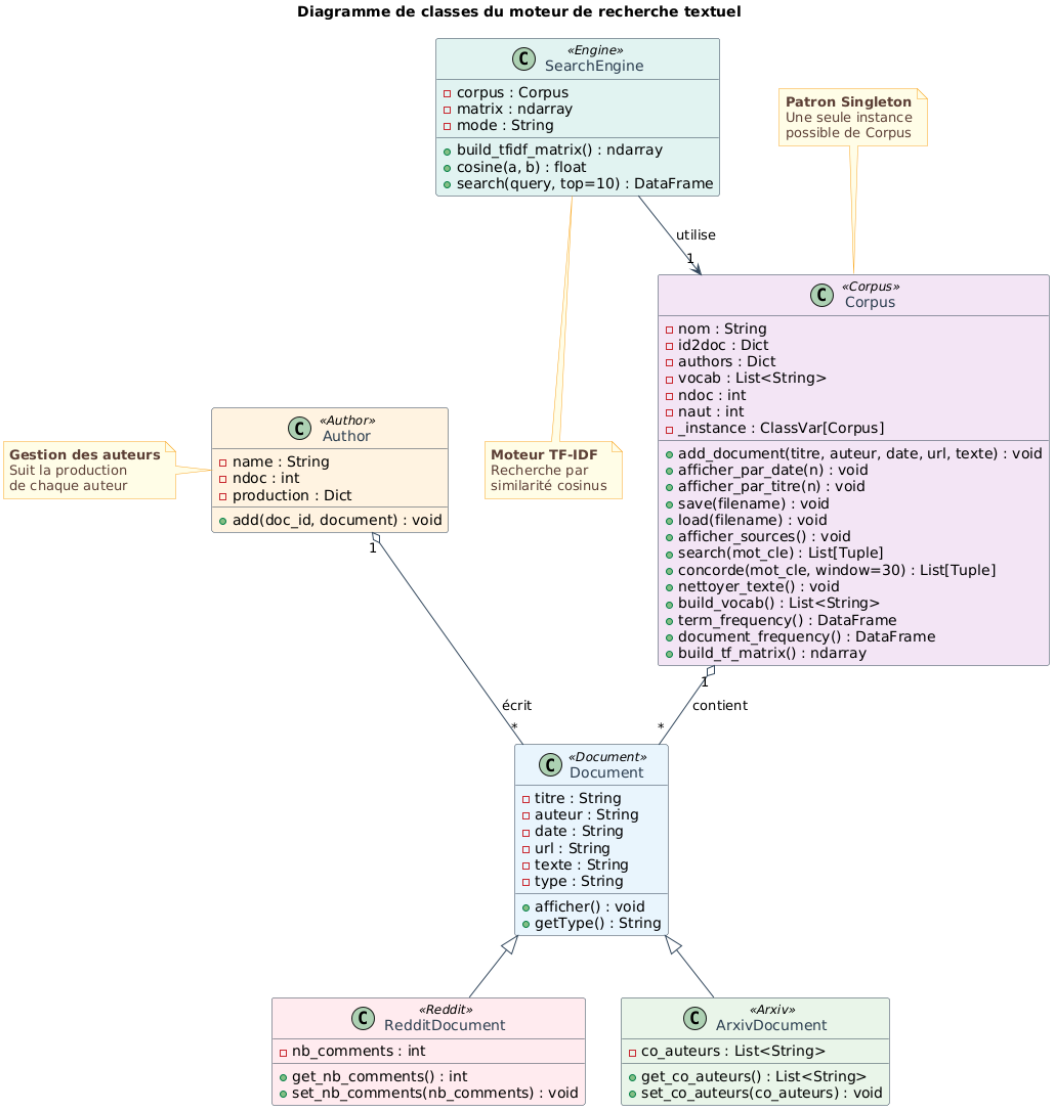

FIGURE 3.1 – Diagramme de classes du système

La Figure 3.1 présente le diagramme de classes du système. La classe `Document` constitue la classe de base représentant une unité textuelle élémentaire. Elle contient les attributs communs à tous les documents, tels que le titre, l'auteur, la date, l'URL et le contenu textuel.

La classe `Corpus` joue un rôle central . Elle est responsable du stockage, du nettoyage, de la construction du vocabulaire et de la gestion des statistiques

textuelles.

La classe `SearchEngine` implémente la logique de recherche avancée en s'appuyant sur des représentations vectorielles des documents et le calcul de la similarité cosinus.

3.3 Diagramme de séquence de la recherche

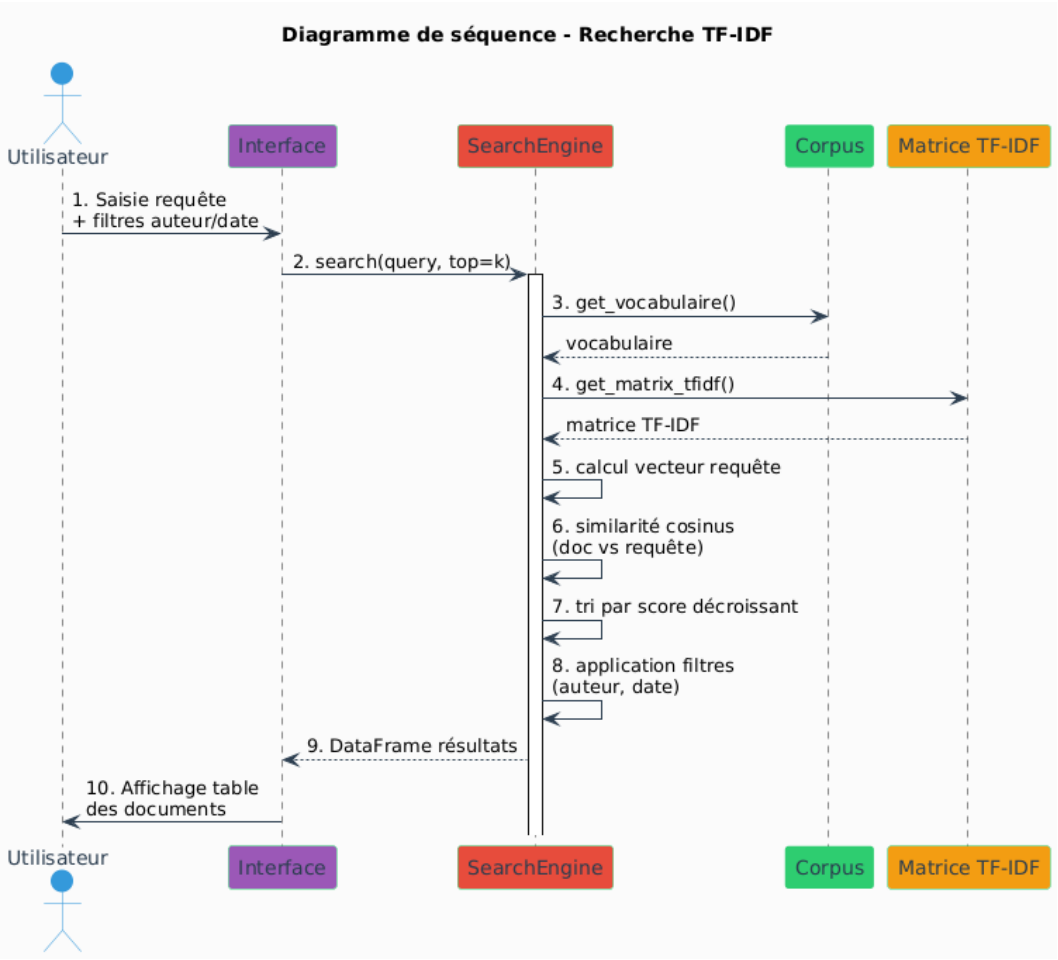

FIGURE 3.2 – Diagramme de séquence du processus de recherche

Ce diagramme de séquence illustre le déroulement d'une recherche depuis la saisie d'une requête par l'utilisateur jusqu'à l'affichage des résultats. Le moteur de recherche interroge le corpus, calcule les scores de similarité et retourne une liste de documents classés par pertinence.

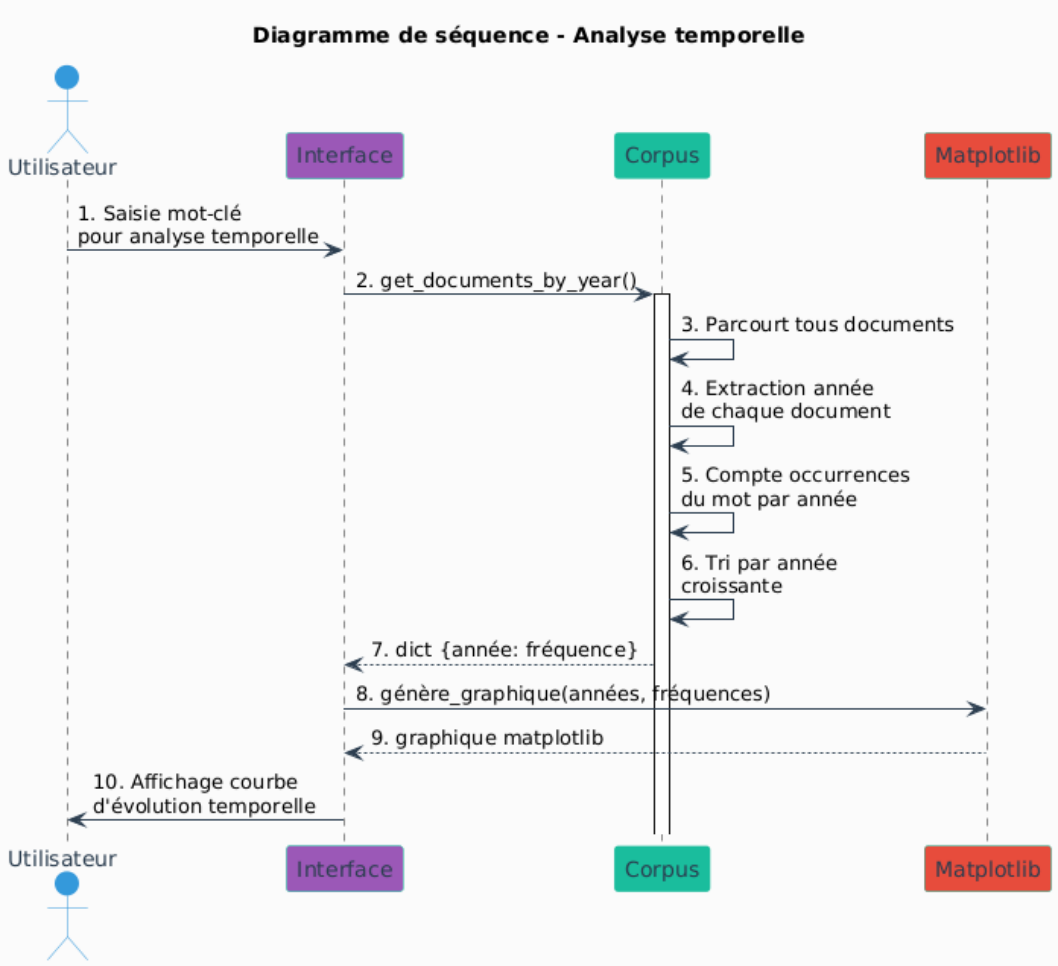

FIGURE 3.3 – Diagramme de séquence de l'analyse temporelle d'un mot-clé

Ce diagramme de séquence décrit le processus d'analyse temporelle d'un mot-clé. À partir de la requête de l'utilisateur, le système parcourt l'ensemble des documents du corpus afin d'extraire les années associées aux textes et de compter les occurrences du terme recherché pour chaque période. Les résultats sont ensuite transmis à un module de visualisation basé sur la bibliothèque `Matplotlib`, qui génère un graphique illustrant l'évolution temporelle du mot-clé.

Chapitre 4

Implémentation du moteur de recherche textuel

4.1 Architecture du moteur de recherche

Le moteur s'appuie principalement sur deux classes centrales : `Corpus`, chargée de stocker et préparer les documents, et `SearchEngine`, responsable de la construction des représentations vectorielles et de l'exécution des requêtes.

4.2 Chargement et structuration des données

La première étape de l'implémentation consiste à charger les données textuelles depuis un fichier CSV contenant les discours politiques. Chaque ligne du fichier est transformée en un objet `Document`, qui encapsule à la fois le texte et ses métadonnées associées.

Ces documents sont ensuite ajoutés à un objet `Corpus`, chargé de centraliser l'ensemble des données et de fournir des méthodes d'analyse et de traitement.

```python
import pandas as pd
import sys
sys.path.append("../src")

from Corpus import Corpus

df = pd.read_csv("discours_US.csv", sep="\t")

corpus = Corpus("discours_US")
for i, row in df.iterrows():
    corpus.add_document(
        titre=row["descr"],
        auteur=row["speaker"],
        date=row["date"],
        url=row["link"],
        texte=row["text"]
    )
```

✓ 0.0s

FIGURE 4.1 – Chargement des données CSV et création du corpus

4.3 Nettoyage et prétraitement des textes

Avant toute analyse, les textes doivent être nettoyés afin d'éliminer les éléments parasites susceptibles de perturber les calculs. Le prétraitement mis en place comprend notamment :

— la conversion de l'ensemble des caractères en minuscules ;

— la suppression des retours à la ligne ;

— l'élimination des caractères spéciaux et des chiffres ;

— la normalisation des espaces.

Cette étape est essentielle pour garantir l'homogénéité du vocabulaire et améliorer la qualité des représentations vectorielles.

```python
def nettoyer_texte(self):
        for doc_id, doc in self.id2doc.items():
            texte = doc.texte.lower()
            texte = texte.replace('\n', ' ').replace('\r', ' ')
            texte = re.sub(r'[^\w\s]', ' ', texte)
            texte = re.sub(r'\d+', ' ', texte)
            texte = re.sub(r'\s+', ' ', texte).strip()
            doc.texte = texte
```

FIGURE 4.2 – Fonction de nettoyage

4.4 Construction du vocabulaire

Une fois les textes nettoyés, un vocabulaire global est construit à partir de l'ensemble des documents du corpus.

```python
def build_vocab(self):
    vocab = set()

    for doc in self.id2doc.values():
        vocab.update(doc.texte.split())

    self.vocab = sorted(vocab)
    return self.vocab
```

FIGURE 4.3 – Construction du vocabulaire

4.5 Représentation vectorielle et calcul TF–IDF

Les documents du corpus sont représentés à l'aide du modèle TF–IDF (Term Frequency – Inverse Document Frequency). Ce modèle permet de pondérer l'importance d'un mot dans un document en fonction de sa fréquence locale et de sa rareté dans l'ensemble du corpus.

```python
def build_tfidf_matrix(self):
    tf = self.corpus.build_tf_matrix()
    df = self.corpus.document_frequency()["document_frequency"].to_dict()
    N = len(self.corpus.id2doc)

    idf = np.array([log(N / (df[word] + 1)) for word in self.corpus.vocab])
    return tf * idf
```

FIGURE 4.4 – Construction de la matrice TF–IDF

4.6 Calcul de la similarité cosinus

Pour évaluer la pertinence d'un document par rapport à une requête, la similarité cosinus est utilisée. Cette mesure permet d'évaluer l'angle entre deux vecteurs dans l'espace vectoriel.

4.7 Implémentation de la recherche

La méthode de recherche permet à l'utilisateur de saisir une requête composée d'un ou plusieurs mots-clés. Cette requête est transformée en vecteur, puis comparée à l'ensemble des documents du corpus.

Les documents sont ensuite classés par ordre décroissant de score de similarité, et les résultats les plus pertinents sont retournés à l'utilisateur.

```python
def search(self, query, top=10):
    query = query.lower().split()

    query_vec = np.zeros(len(self.corpus.vocab))
    for w in query:
        if w in self.corpus.vocab:
            query_vec[self.corpus.vocab.index(w)] += 1

    scores = []
    for i, (doc_id, doc) in tqdm(enumerate(self.corpus.id2doc.items()), total=len(self.corpus.id2doc)):
        score = self.cosine(query_vec, self.matrix[i])
        scores.append((doc_id, doc.titre, score))

    df = pd.DataFrame(scores, columns=["id", "titre", "score"])
    return df.sort_values(by="score", ascending=False).head(top)
```

FIGURE 4.5 – Implémentation de la méthode de recherche par similarité cosinus

4.8 Analyse temporelle et visualisation

En complément de la recherche textuelle classique, une fonctionnalité d'analyse temporelle a été implémentée. Elle permet d'étudier l'évolution de l'occurrence d'un mot-clé au fil des années présentes dans le corpus.

Les résultats de cette analyse sont visualisés à l'aide de la bibliothèque `Matplotlib`, sous forme de graphiques facilitant l'interprétation des tendances observées.

```
def evolution_temporelle(b):
    with output_evol:
        clear_output(wait=True)

        mot = mot_input.value.lower()
        freq_par_an = {}

        # Calcul : occurrences du mot par année
        for doc_id, doc in corpus.id2doc.items():
            year = pd.to_datetime(doc.date, errors='coerce').year
            if year is None:
                continue
            texte = doc.texte.lower().split()
            freq = texte.count(mot)
            freq_par_an[year] = freq_par_an.get(year, 0) + freq

        # Données triées
        annees = sorted(freq_par_an.keys())
        freqs = [freq_par_an[a] for a in annees]

        # Graphique
        plt.figure(figsize=(8,4))
        plt.plot(annees, freqs, marker="o")
        plt.title(f"Évolution du mot '{mot}' dans le temps")
        plt.xlabel("Année")
        plt.ylabel("Fréquence")
        plt.grid(True)
        plt.show()

        display(query_input, k_slider, author_filter, date_filter, btn, output)
```

FIGURE 4.6 – Analyse temporelle et visualisation

Chapitre 5

Interface web et API applicative

Ce chapitre présente la mise en place d'une interface web permettant d'exploiter le moteur de recherche textuel développé précédemment. L'objectif est de proposer une interaction utilisateur simple et efficace, reposant sur une architecture client–serveur et une API applicative dédiée.

5.1 Architecture générale

L'application repose sur une architecture de type client–serveur, séparant clairement la logique de traitement des données de l'interface utilisateur.

Le système est composé de deux parties principales :

— un **front-end web**, accessible via un navigateur, développé à l'aide de HTML, CSS et JavaScript ;

— un **back-end applicatif**, implémenté en Python avec le framework `Flask`, chargé d'exposer les fonctionnalités du moteur de recherche.

5.2 Implémentation du back-end avec Flask

Le back-end de l'application est implémenté à l'aide du micro-framework.

Au démarrage de l'application, les données textuelles sont chargées depuis un fichier CSV contenant les discours politiques. Chaque ligne du fichier est transformée en un objet `Document` et ajoutée à un objet `Corpus`, centralisant l'ensemble des documents.

Afin d'optimiser les performances, le moteur de recherche est préalablement entraîné puis sérialisé à l'aide du module `pickle`. Il est ensuite rechargé en mémoire lors du lancement du serveur, évitant ainsi des recalculs coûteux.

```python
# ---------------------------
# Initialiser Flask
# ---------------------------
app = Flask(
    __name__,
    static_folder=os.path.join(BASE_DIR, "web"),
    static_url_path="/static"
)

# ---------------------------
# Charger les données
# ---------------------------
DATA_PATH = os.path.join(BASE_DIR, "notebooks", "discours_US.csv")
ENGINE_PATH = os.path.join(BASE_DIR, "notebooks", "engine.pkl")

df = pd.read_csv(DATA_PATH, sep="\t")

corpus = Corpus("discours_US")
for _, row in df.iterrows():
    corpus.add_document(
        titre=row["descr"],
        auteur=row["speaker"],
        date=row["date"],
        url=row["link"],
        texte=row["text"]
    )

with open(ENGINE_PATH, "rb") as f:
    engine = pickle.load(f)
```

FIGURE 5.1 – Initialisation de l'application Flask et chargement du moteur

5.3 API de recherche textuelle

L'API expose un point d'entrée dédié à la recherche textuelle, accessible via une requête HTTP de type `POST`.

```python
# --------------------------
# API : SEARCH
# --------------------------

@app.route("/search", methods=["POST"])
def search():
    data = request.json
    query = data.get("query", "")
    k = int(data.get("k", 5))
    author = data.get("author", "Tous")
    date = data.get("date", "Toutes")

    results = engine.search(query, top=9999)

    if author != "Tous":
        results = results[results["id"].apply(
            lambda doc_id: corpus.id2doc[doc_id].auteur == author
        )]

    if date != "Toutes":
        results = results[results["id"].apply(
            lambda doc_id: corpus.id2doc[doc_id].date == date
        )]

    return jsonify(results.head(k).to_dict(orient="records"))
```

FIGURE 5.2 – Point d'entrée de l'API de recherche textuelle

5.4 Analyse temporelle des mots-clés

En complément de la recherche textuelle, une fonctionnalité d'analyse temporelle a été intégrée à l'API. Elle permet d'étudier l'évolution de l'occurrence d'un mot-clé au fil des années présentes dans le corpus.

```python
@app.route("/evolution", methods=["POST"])
def evolution():
    data = request.get_json(force=True)
    mot = data.get("mot", "").lower().strip()

    freq = {}

    for doc_id, doc in corpus.id2doc.items():
        # convertir proprement la date en année
        year = pd.to_datetime(doc.date, errors="coerce").year
        if pd.isna(year):
            continue  # si date pourrie, on saute

        year = int(year)

        # compter le mot dans le texte
        tokens = doc.texte.lower().split()
        count = tokens.count(mot)

        if count > 0:
            freq[year] = freq.get(year, 0) + count

    # transformer en liste triée pour le JSON
    out = [{"year": y, "freq": freq[y]} for y in sorted(freq.keys())]
    return jsonify(out)
```

FIGURE 5.3 – Point d'entrée de l'analyse temporelle

5.5 Aperçu de l'interface utilisateur

La Figure 5.4 présente l'interface principale de recherche. Elle permet à l'utilisateur de saisir une requête textuelle, de définir des filtres (auteur, date, nombre de résultats) et de consulter les documents retournés par le moteur, classés par pertinence.

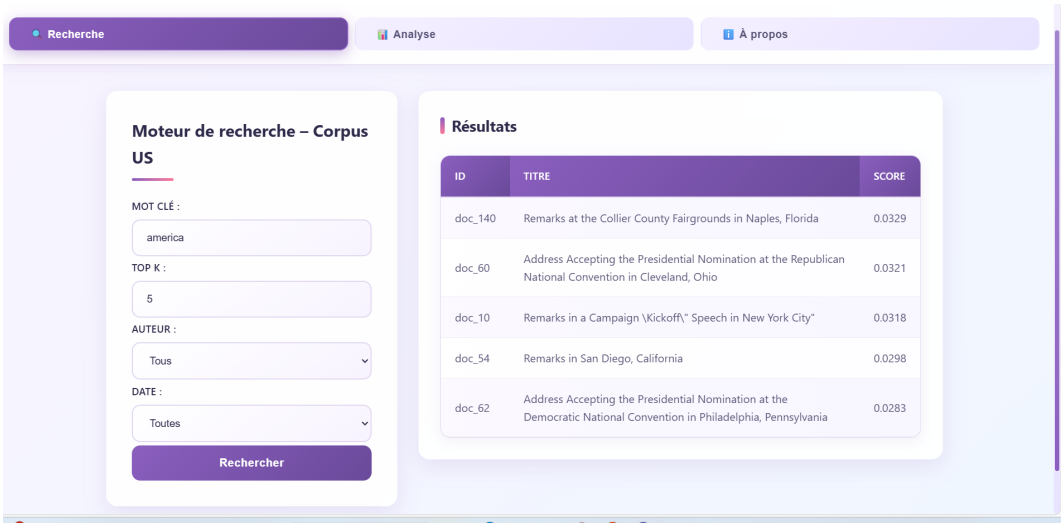

FIGURE 5.4 – Interface web de recherche textuelle

La Figure 5.5 illustre l'interface dédiée à l'analyse temporelle d'un mot-clé. L'utilisateur peut visualiser l'évolution de la fréquence d'un terme au fil des années sous forme graphique, facilitant l'interprétation des tendances observées dans le corpus.

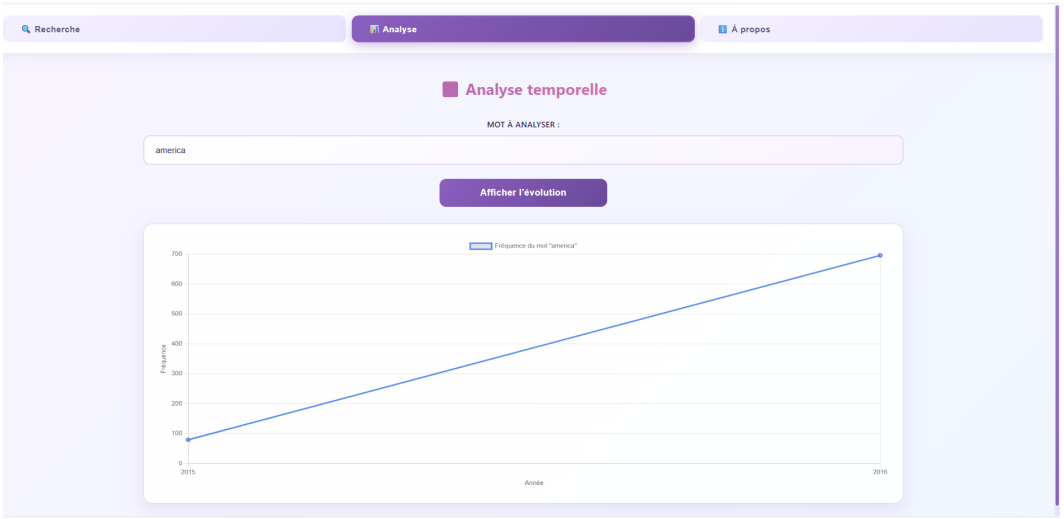

FIGURE 5.5 – Interface web d'analyse temporelle des mots-clés

Chapitre 6

Bilan, difficultés rencontrées et perspectives

Ce projet a permis de mettre en pratique de manière progressive les notions abordées tout au long des travaux dirigés, allant de la manipulation de données textuelles à la conception d'un moteur de recherche complet intégrant une interface web et une API applicative.

6.1 Organisation du travail

Le projet a été réalisé intégralement en autonomie. Chaque séance de travaux dirigés, d'une durée de 3 heures, était consacrée à l'avancement d'au moins un travail dirigé. Le travail personnel effectué en dehors des séances a permis de consolider les acquis et de compléter les fonctionnalités, ce qui a conduit à un rythme moyen d'environ deux travaux dirigés par semaine.

La dernière phase du projet a été dédiée à la structuration du code, à la stabilisation de l'application et à la rédaction du rapport final.

6.2 Difficultés rencontrées

Plusieurs difficultés ont été rencontrées au cours du développement. La principale concernait l'adoption d'une approche orientée objet, notamment la structuration des classes `Corpus`, `Document` et du moteur de recherche. N'étant pas initialement habituée à concevoir des architectures orientées objet complètes dans un temps limité, la compréhension et l'organisation du code ont nécessité un effort d'adaptation important.

Des difficultés ont également émergé lors de la gestion des performances, en

particulier lors des premières tentatives de construction de matrices de représentation vectorielle, ainsi que lors de l'intégration du moteur de recherche dans une application web fonctionnelle.

6.3 Apports et compétences acquises

Ce projet a permis de renforcer plusieurs compétences essentielles, notamment :

— la manipulation et l'analyse de données textuelles ;

— la conception d'applications orientées objet ;

— la mise en œuvre de modèles de recherche d'information ;

6.4 Perspectives d'amélioration

Plusieurs pistes d'amélioration peuvent être envisagées. Le moteur de recherche pourrait être enrichi par l'intégration de techniques de traitement du langage naturel plus avancées, telles que la lemmatisation, la suppression des mots vides ou l'utilisation de modèles sémantiques.

D'un point de vue applicatif, l'interface web pourrait être améliorée par l'ajout de nouvelles fonctionnalités de filtrage, de visualisations interactives supplémentaires, ou par le déploiement de l'application sur une infrastructure distante.

En conclusion, ce projet constitue une expérience formatrice, ayant permis de consolider à la fois des compétences techniques et une capacité à mener un projet informatique complet en autonomie.

Conclusion générale

Ce projet a permis de mettre en œuvre de manière concrète les concepts fondamentaux liés à la recherche d'information et au traitement automatique de données textuelles. À partir d'un corpus de discours politiques, un moteur de recherche textuel fonctionnel a été conçu et intégré dans une application web complète.

Au-delà des aspects techniques, ce travail a également favorisé le développement d'une méthodologie de projet rigoureuse, fondée sur une progression par étapes, la résolution autonome de problèmes et la structuration orientée objet du code.

Enfin, ce projet constitue une base solide pour des évolutions futures, tant sur le plan algorithmique que sur le plan applicatif, ouvrant la voie à l'intégration de techniques plus avancées de traitement du langage naturel et à l'amélioration de l'expérience utilisateur.