Université de
Technologie de
Compiègne

LO17 : Indexation et Recherche d'Information

Rapport de Projet DM

Membre du binôme 1 :
Amen Allah Nticha(Contribution : 95%)

Membre du binôme 2 :
Loïc RECOUPE (Contribution : 5%)

Printemps 2025

Sommaire

1 Introduction **2**
- 1.1 Contexte et Enjeux . . . 2
- 1.2 Objectifs du Rapport . . . 2
- 1.3 Structure du Rapport . . . 2

2 Préparation et Structuration du Corpus **3**
- 2.1 Méthode et Étapes de l'Extraction . . . 3
- 2.2 Automatisation de l'Extraction . . . 3
- 2.3 Commentaires et Points à Noter . . . 3

3 Construction de l'Anti-dictionnaire et Filtrage **5**
- 3.1 Méthode et Étapes du Filtrage . . . 5
- 3.2 Commentaires et Observations . . . 6
- 3.3 Conclusion . . . 6

4 Lemmatisation, Racines et Index Inversés **7**
- 4.1 Lemmatisation et Stemming . . . 7
- 4.2 Substitution et Nettoyage . . . 7
- 4.3 Création des Index Inversés . . . 7
- 4.4 Commentaires et Observations . . . 8
- 4.5 Conclusion . . . 8

5 Correction Orthographique et Lemmatisation **9**
- 5.1 Mesure de Proximité . . . 9
- 5.2 Distance de Levenshtein . . . 9
- 5.3 Correction des Mots-Clés . . . 9
- 5.4 Lemmatisation des Mots-Clés . . . 9
- 5.5 Conclusion . . . 9

6 TD6 : Traitement des requêtes **10**

Partie 1
Introduction

1.1 Contexte et Enjeux

À l'ère de la surcharge informationnelle, la capacité à accéder rapidement et efficacement à des documents pertinents devient un enjeu stratégique, tant pour les entreprises que pour les institutions. Dans le cadre de l'UV LO17 (*Indexation et Recherche d'Information*), ce projet nous a offert l'opportunité de concevoir un moteur de recherche complet, appliqué à un corpus réel : les bulletins électroniques de l'ADIT, riches en contenus technologiques, scientifiques et industriels.

L'objectif était de mettre en œuvre de manière concrète l'ensemble des concepts abordés durant le semestre : nettoyage de corpus, extraction d'information, lemmatisation, indexation inversée, traitement des requêtes en langage naturel, et évaluation de la pertinence des résultats. Ce projet s'est voulu à la fois pédagogique, technique et structurant.

1.2 Objectifs du Rapport

Le présent rapport a pour but de retracer les différentes étapes du développement du moteur de recherche, en mettant en valeur les choix techniques réalisés, les méthodes mises en œuvre, et les résultats obtenus. Il s'agira notamment de :

- Décrire la transformation du corpus initial en un format exploitable, structuré et standardisé (XML).
- Présenter les techniques d'indexation, de traitement des mots-clés, des dates, des rubriques et des titres.
- Expliquer la mise en place du correcteur orthographique et son intégration au système.
- Analyser la manière dont les requêtes en langage naturel ont été traitées et traduites en instructions exploitables.
- Évaluer les performances du système développé, à travers des métriques comme la précision, le rappel et le F1-score.

1.3 Structure du Rapport

Le rapport est structuré de manière progressive, en suivant l'évolution naturelle du projet :

- **Chapitre 2** : Préparation et structuration du corpus XML à partir des fichiers HTML bruts.
- **Chapitre 3** : Calcul des scores TF-IDF et constitution de l'anti-dictionnaire.
- **Chapitre 4** : Lemmatisation et indexation des différentes composantes documentaires.
- **Chapitre 5** : Conception d'un correcteur orthographique fondé sur les distances de Levenshtein et les proximités de surface.
- **Chapitre 6** : Développement du moteur de recherche et évaluation de ses performances sur des requêtes réelles.
- **Chapitre 7** : Bilan personnel, perspectives et réflexion sur le travail accompli.

Partie 2
Préparation et Structuration du Corpus

Nous avons commencé par une étape fondamentale : l'extraction et la structuration des données à partir des bulletins électroniques (BE) au format HTML. Cette première phase a pour objectif de créer un corpus XML structuré qui servira de base aux traitements ultérieurs (antidictionnaire, lemmatisation, index inversés, etc.).

2.1 Méthode et Étapes de l'Extraction

Le script Python mis en place applique les étapes suivantes pour chaque bulletin HTML :

1. **Identification du fichier** : L'ID du bulletin est extrait du nom du fichier et écrit dans la balise `<fichier>` du fichier XML.
2. **Extraction des métadonnées** : Le titre, la date (format inversé `j/m/a`) et le numéro de bulletin sont extraits à partir de la balise `<title>` du HTML. Ces informations sont encapsulées dans les balises `<titre>`, `<date>` et `<numero>` respectivement.
3. **Extraction de l'auteur** : Les métadonnées HTML (balises `<meta>` et `<span class="style95">`) sont parcourues pour identifier le nom de l'auteur, généralement présent sous forme de chaîne contenant un email.
4. **Extraction de la rubrique** : Les rubriques sont localisées dans les balises `<span class="style42">` et intégrées dans la balise `<rubrique>` du fichier XML.
5. **Extraction du texte principal** : Le contenu textuel principal est extrait à partir des balises `<span class="style95">` et regroupé dans une unique balise `<texte>`, en veillant à ne conserver que les phrases complètes (finissant par un point).
6. **Extraction des images et légendes** : Les balises `<img>` (pour l'URL de l'image) et `<span class="style21">` (pour la légende) sont traitées. Ces informations sont structurées dans des balises `<image>` imbriquées dans `<images>` pour chaque bulletin.
7. **Extraction des contacts** : Les balises `<span class="style85">` contiennent les contacts à extraire. Ils sont ajoutés dans les balises `<contact>` dans le XML final.

2.2 Automatisation de l'Extraction

Les fonctions décrites précédemment sont regroupées dans une fonction principale `extract_file` qui applique toutes les étapes d'extraction pour un fichier HTML donné. Pour traiter l'ensemble du corpus (répertoire des bulletins HTML), la fonction `extract_all_files` parcourt tous les fichiers du dossier et applique `extract_file` sur chacun d'eux.

Le résultat final est un fichier `corpus.XML`, encapsulé dans une balise `<corpus>` globale, qui contient les balises `<bulletin>` pour chaque fichier traité.

2.3 Commentaires et Points à Noter

— **Lisibilité du XML** : Les balises XML sont écrites de manière lisible et structurée pour faciliter l'exploitation ultérieure par les scripts des TD suivants.
— **Nettoyage des données** : Certains caractères indésirables (espaces, guillemets, etc.) sont nettoyés lors de l'extraction (notamment pour les titres).

Cette étape a permis de préparer un corpus homogène et structuré, facilitant la suite du projet (construction des index inversés, recherche et correction orthographique, etc.). Le code Python, grâce à ses fonctions modulaires, assure une extraction robuste et facilement adaptable à des évolutions ultérieures.

Partie 3
Construction de l'Anti-dictionnaire et Filtrage

L'étape suivante consiste à nettoyer le corpus et à préparer un anti-dictionnaire. Cette étape vise à éliminer les mots « parasites » qui n'apportent que peu ou pas d'informations pertinentes à la recherche. Pour ce faire, nous avons appliqué les techniques classiques de pondération TF (Term Frequency) et IDF (Inverse Document Frequency).

3.1 Méthode et Étapes du Filtrage

3.1.1 1. Nettoyage et Segmentation

Tout d'abord, le corpus XML est segmenté en mots (tokens) pour chaque bulletin. Les étapes sont les suivantes :

- **Nettoyage** : Suppression de la ponctuation et mise en minuscules des textes, à l'aide de la fonction `nettoyer_texte`.
- **Segmentation** : Les mots sont extraits ligne par ligne et sauvegardés dans un fichier CSV (`tokens.csv`), où chaque mot est associé à un `doc_id`.

3.1.2 2. Calcul des coefficients TF et IDF

- **TF (Term Frequency)** : Le nombre d'occurrences de chaque mot dans chaque document est calculé et sauvegardé dans un fichier `tf.csv`.
- **IDF (Inverse Document Frequency)** : Le nombre de documents contenant chaque mot est déterminé, et le coefficient IDF est calculé selon la formule suivante :

$$idf(mot) = \log_{10}\left(\frac{N}{dft(mot)}\right)$$

où N est le nombre total de documents et $dft(mot)$ le nombre de documents contenant le mot. Les résultats sont enregistrés dans `idf_coefficients.csv`.

3.1.3 3. Calcul final TF*IDF

Les deux jeux de coefficients (TF et IDF) sont fusionnés pour obtenir un fichier final `output.csv` contenant les valeurs de TF*IDF pour chaque mot et document.

3.1.4 4. Analyse et Visualisation

Un histogramme des valeurs TF*IDF (limitées à 10 pour lisibilité) est généré afin de mieux visualiser la distribution de ces valeurs et aider à identifier les seuils pertinents pour la création de l'anti-dictionnaire.

3.1.5 5. Création de l'Anti-dictionnaire

Sur la base des observations des histogrammes et des quantiles calculés, les mots « parasites » sont identifiés :

- Les mots dont le TF*IDF est inférieur au premier quantile (environ 17 % des données).
- Les mots dont le TF*IDF est supérieur à un seuil (environ 15) : souvent des mots spécifiques, mais peu pertinents globalement.

Ces mots sont rassemblés dans un fichier `anti_dict3.csv`.

3.1.6 6. Substitution et nettoyage final du corpus

Enfin, un script de substitution (`substitue`) remplace ces mots parasites par un espace vide dans le corpus XML, créant un fichier `corpus_nettoyé3.XML`. Les lignes vides restantes sont ensuite supprimées grâce à la fonction `delete_empty_lines`, pour obtenir un corpus final propre (`corpus_final.XML`).

3.2 Commentaires et Observations

- **Robustesse** : L'approche fondée sur les quantiles permet d'adapter le seuil de filtrage en fonction des distributions observées (visualisation par histogramme).
- **Adaptabilité** : Les seuils utilisés pour l'anti-dictionnaire (0.17 pour le premier quartile et 15 pour les valeurs élevées) ont été choisis empiriquement, mais ils peuvent être ajustés facilement selon les résultats des visualisations.
- **Préparation aux étapes suivantes** : Le corpus final obtenu est prêt pour les phases de lemmatisation, d'indexation inversée et d'implémentation du correcteur orthographique.

3.3 Conclusion

La création de l'anti-dictionnaire et le nettoyage du corpus constituent une étape clé pour garantir la pertinence des résultats de recherche ultérieurs. Grâce à ces étapes, les « bruits » et les mots peu informatifs ont été filtrés, rendant les recherches plus précises et plus significatives.

Partie 4
Lemmatisation, Racines et Index Inversés

L'objectif principal de cette étape est d'améliorer la pertinence des recherches en normalisant les mots du corpus. Nous avons mis en place :

— La **lemmatisation** (formes canoniques des mots),
— Le **stemming** (racines des mots),
— La **substitution** pour normaliser les mots parasites,
— La **création des index inversés** pour faciliter la recherche par différents champs (titre, texte, date, rubrique, images).

4.1 Lemmatisation et Stemming

4.1.1 1. Extraction des lemmes

À l'aide de la bibliothèque `spaCy` et son modèle `fr_core_news_sm`, les lemmes ont été extraits pour chaque mot dans les balises `<texte>` et `<titre>` du corpus XML. Ces lemmes sont sauvegardés dans le fichier `lemmes.csv`.

4.1.2 2. Extraction des racines (stems)

Pour comparaison, les racines des mots ont été extraites avec l'algorithme de `SnowballStemmer` de NLTK. Les résultats sont enregistrés dans `stems.csv`. Cependant, les racines peuvent dénaturer certains mots, notamment les noms propres : la lemmatisation est donc privilégiée pour la suite du projet.

4.2 Substitution et Nettoyage

Après l'extraction des lemmes, les mots parasites identifiés sont remplacés par leurs lemmes ou par un vide (selon l'anti-dictionnaire créé dans le TD3). Cette étape de **substitution** est effectuée dans le fichier `corpus_post_final.XML`. Un nettoyage supplémentaire supprime la ponctuation et les articles français inutiles.

4.3 Création des Index Inversés

Des index inversés sont créés pour accélérer les requêtes sur différents champs :

— **Index inversé pour les titres** : `reverse_index_titre.csv`
— **Index inversé pour le texte principal** : `reverse_index_texte.csv`
— **Index inversé pour les dates** : `reverse_index_date.csv`
— **Index inversé pour les rubriques** : `reverse_index_rubrique.csv`
— **Index inversé pour les images** : `reverse_index_image.csv`

Chaque fichier contient deux colonnes :

— `mot` : le mot, la date ou l'attribut.
— `docs` : la liste des identifiants de documents contenant ce mot ou attribut.

4.4 Commentaires et Observations

— La **lemmatisation** est plus pertinente que le stemming, notamment pour conserver le sens des entités nommées.
— Les index inversés facilitent les requêtes ultérieures, car ils permettent d'associer directement un mot à une liste de documents.
— Les étapes de nettoyage (ponctuation, articles) garantissent la cohérence des index inversés.

4.5 Conclusion

L'indexation inversée et la normalisation par lemmatisation constituent une base solide pour l'implémentation du moteur de recherche final. Ces étapes garantissent à la fois la pertinence des résultats et la rapidité des requêtes dans les phases suivantes (correcteur orthographique et moteur de recherche complet).

Partie 5
Correction Orthographique et Lemmatisation

Cette partie vise à améliorer la qualité des requêtes et du corpus en normalisant les formes des mots et en corrigeant les erreurs orthographiques. En théorie, la correction orthographique pourrait être appliquée à l'ensemble de la requête. Toutefois, pour **faciliter les traitements ultérieurs** et conserver l'authenticité de la requête d'origine, j'ai choisi de l'appliquer uniquement aux mots-clés explicitement recherchés.

5.1 Mesure de Proximité

La première étape consiste à évaluer la **proximité de surface** entre un mot de la requête et les mots du corpus. Pour cela :

— On mesure la longueur du préfixe commun entre les deux mots.

— Les mots trop courts (moins de 3 caractères) ou avec une différence de longueur trop importante (plus de 4 caractères) sont écartés.

Cette proximité est mesurée par la fonction `recherche_proximite`.

5.2 Distance de Levenshtein

La **distance de Levenshtein** affine ensuite cette comparaison :

— La fonction `levenshtein` détermine le mot du corpus qui ressemble le plus au mot de la requête, en comptant le nombre minimal d'opérations pour passer de l'un à l'autre.

5.3 Correction des Mots-Clés

La fonction `correction_orthographique` applique ces principes :

1. Si le mot-clé existe déjà dans le corpus, il est conservé tel quel.
2. Sinon, elle identifie les mots proches à l'aide de la mesure de proximité.
3. Elle applique la distance de Levenshtein pour trouver le mot correct et remplace le mot-clé initial par ce lemme.

Ainsi, **seuls les mots-clés de la requête** sont corrigés pour garantir la cohérence des recherches, sans altérer les autres parties de la requête.

5.4 Lemmatisation des Mots-Clés

Une étape supplémentaire de **lemmatisation** est appliquée aux mots-clés corrigés pour harmoniser leurs formes. La fonction `lemmatize` réalise ce traitement, en utilisant un fichier de lemmes pour chaque mot-clé.

5.5 Conclusion

En limitant la correction orthographique aux seuls mots-clés recherchés, nous préservons le sens global de la requête tout en améliorant la précision des résultats. Cette approche garantit une meilleure robustesse du moteur de recherche final, tout en restant fidèle à l'intention de l'utilisateur.

Partie 6
TD6 : Traitement des requêtes

Dans ce TD, j'ai travaillé sur l'analyse et l'extraction d'informations à partir de requêtes en langage naturel formulées en français. L'objectif principal était de structurer ces requêtes afin de pouvoir les interpréter et les utiliser efficacement dans un moteur de recherche ou une base de données.

Problème rencontré

Initialement, j'ai tenté de résoudre le problème en utilisant une **grammaire formelle** pour représenter toutes les requêtes possibles. J'ai envisagé d'utiliser la bibliothèque `nltk` en Python pour cette implémentation. Cependant, cette approche s'est révélée trop complexe pour les requêtes naturelles en français, notamment en raison de la variété des formulations et des nuances linguistiques. J'ai donc décidé de changer de stratégie.

Méthodologie adoptée

J'ai opté pour une approche plus pragmatique, fondée sur la **reconnaissance de motifs** et la **manipulation de chaînes de caractères**. Le script que j'ai conçu procède par étapes pour identifier les différents composants de la requête, en utilisant des listes de mots-clés spécifiques.

Les étapes principales sont les suivantes :

1. **Prétraitement de la requête** : la requête est mise en minuscules, les ponctuations inutiles sont supprimées et les articles ou mots de remplissage sont retirés afin de simplifier l'analyse.
2. **Identification du résultat attendu** (*return*) : on repère si la requête demande des articles, des rubriques, des bulletins ou des recherches.
3. **Détection des images** : certains mots-clés indiquent que la requête porte sur des éléments contenant des images.
4. **Extraction de la rubrique** : le script vérifie la présence de rubriques spécifiques ou de thèmes mentionnés.
5. **Analyse des titres** : s'il existe des indications sur le titre (par exemple, « contient », « est », « traite »), ces éléments sont extraits.
6. **Détermination des dates** : le script cherche des indications temporelles comme des dates précises, des intervalles ou des références à un mois ou une année.
7. **Extraction des mots-clés** : il reste ensuite à identifier les mots-clés principaux (souvent des sujets ou des thématiques) qui n'appartiennent pas aux catégories précédentes.

Résultats obtenus

À la fin de ce processus, la requête est transformée en un **dictionnaire structuré** qui contient tous les éléments pertinents : la nature de la recherche, les images, les rubriques, les dates, les titres et les mots-clés. Cette représentation structurée est facilement exploitable pour une indexation ou une recherche ultérieure :

- **return** : indique le type de résultat attendu par la requête (par exemple, « article », « rubrique », etc.).
- **mots_cles** : un dictionnaire qui contient :
  - **yes** : la liste des mots-clés présents dans la requête.

  - **no** : une valeur indiquant si certains mots-clés sont explicitement exclus.
- **operateurs_mots_cles** : opérateurs logiques (comme "ou") appliqués aux mots-clés, s'ils sont mentionnés.
- **rubrique** : la rubrique ou le thème principal de la recherche.
- **operateurs_rubrique** : opérateurs logiques appliqués aux rubriques, le cas échéant.
- **dates** : un dictionnaire qui regroupe :
  - **début** : date de début d'une période spécifiée.
  - **fin** : date de fin d'une période spécifiée.
  - **précis** : une date unique s'il y a une référence temporelle spécifique.
  - **not** : si une période est explicitement exclue.
- **titre** : le titre recherché ou une indication sur celui-ci.
- **operateurs_titre** : opérateurs logiques appliqués au titre, si nécessaire.
- **images** : mentionne la présence ou l'absence d'images dans la requête.

Cette structure de sortie facilite l'exploitation des informations contenues dans la requête pour des recherches ultérieures ou pour l'indexation des résultats.

Bilan

Bien que j'aie perdu du temps avec l'approche initiale basée sur une grammaire formelle, ce contretemps m'a permis de mieux comprendre les limites de ce type de méthode, notamment pour des requêtes en langage naturel. En adoptant une approche modulaire et orientée sur l'identification progressive des composants, j'ai pu finaliser le TD de manière satisfaisante, même si cela a entraîné un léger retard dans la livraison.

Partie 7
TD7 : Système de Recherche d'Information

Ce dernier TD avait pour objectif de finaliser l'intégration du moteur de recherche en évaluant ses performances. Cela inclut le traitement automatique des requêtes en langage naturel, l'extraction des critères (mots-clés, dates, rubriques, titres, images), la recherche documentaire sur la base d'index inversés et l'évaluation quantitative de la qualité des résultats retournés.

Pipeline d'exécution

1. Une requête est saisie par l'utilisateur.
2. Elle est traitée via `traiter_requete()`.
3. Une correction orthographique est appliquée uniquement aux mots-clés pour ne pas altérer la structure syntaxique de la requête.
4. La recherche documentaire est effectuée par `recherche_documents()` en consultant les index inversés.

Modules implémentés

— `compare_dates()` : permet de filtrer les documents selon les dates (avec contraintes sur le début, la fin, des dates précises ou des exclusions).
— `recherche_documents()` : agrège les résultats des recherches sur mots-clés, rubriques, titres, images et dates en tenant compte des opérateurs logiques.
— `traiter_et_rechercher()` : pipeline principal pour traiter la requête et retourner les documents.

Évaluation des performances

Métriques utilisées

Les performances du moteur ont été mesurées à l'aide des indicateurs classiques :
— **Précision** : proportion de documents retournés qui sont réellement pertinents.
— **Rappel** : proportion de documents pertinents qui ont été effectivement retournés.
— **F1-score** : moyenne harmonique entre précision et rappel.

Méthodologie

— 10 requêtes ont été choisies.
— Pour chaque requête, la liste des documents pertinents a été établie manuellement en consultant les index inversés du TD4.
— Les fonctions `get_precision_recall()` et `calculate_precision_rappel()` ont permis de mesurer les performances pour chaque requête.
— Un graphique a été généré pour visualiser l'évolution des scores.

Extrait des résultats

Requête	Précision	Rappel	F1-score
1	1.00	1.00	1.00
2	1.00	1.00	1.00
3	1.00	0.00	0.00
4	0.04	1.00	0.08
5	1.00	1.00	1.00
6	1.00	1.00	1.00
7	1.00	0.05	0.09
8	1.00	0.50	0.67
9	1.00	1.00	1.00
10	1.00	0.40	0.57

TABLE 7.1 – Scores pour 10 requêtes test

Temps de réponse moyen : 0.1773 secondes (sur 100 exécutions pour chaque requête).

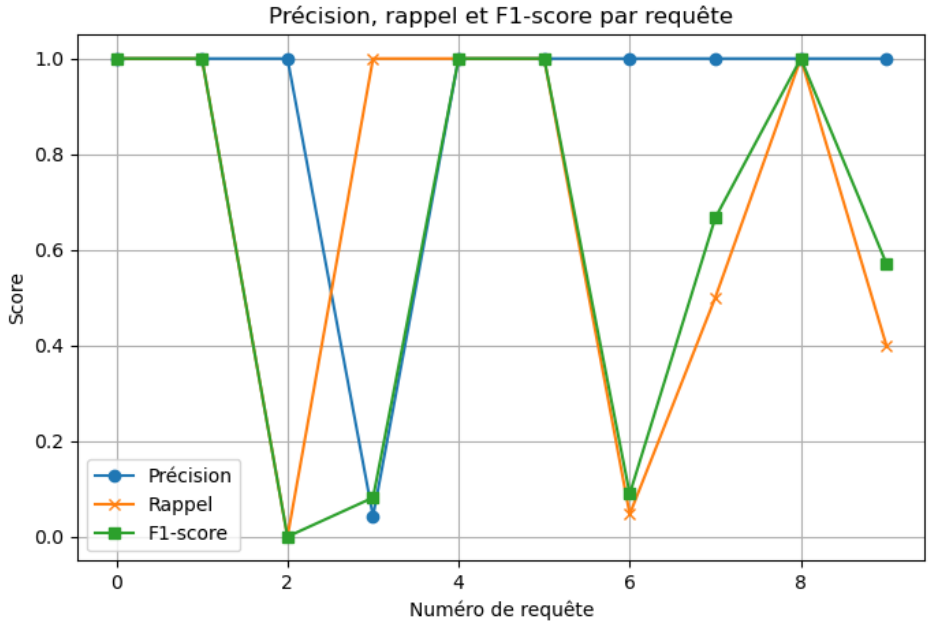

FIGURE 7.1 – Évolution de la précision, du rappel et du F1-score selon les requêtes

Conclusion

Le moteur est désormais capable de :
— Interpréter une requête en langage naturel.
— Corriger automatiquement les erreurs orthographiques sur les mots-clés.
— Gérer des critères variés comme les rubriques, les dates, les titres et les images.
— Fournir des résultats pertinents avec une bonne précision et un temps de réponse acceptable.
Cependant, on note des limitations :
— Les expressions multi-mots ne sont pas toujours bien prises en compte dans les index.
— L'approche manuelle pour constituer les résultats de référence peut introduire un biais si elle n'est pas rigoureuse.

Des améliorations pourraient inclure :
— Le traitement des expressions composées dans les index.
— Un système de pondération ou de classement des documents (scoring).
— Une interface utilisateur visuelle pour une meilleure expérience.

Partie 8
Conclusion

Ce projet m'a permis de découvrir, comprendre et manipuler des notions fondamentales de traitement automatique des langues (TAL), d'analyse d'index inversés et d'évaluation de systèmes d'information. À travers les différents TDs, j'ai progressivement construit un moteur de recherche rudimentaire mais fonctionnel, capable d'interpréter une requête en langage naturel, d'en extraire les éléments clés, de les corriger le cas échéant, puis d'interroger efficacement un corpus documentaire structuré.

J'ai particulièrement apprécié ce projet pour son aspect à la fois technique et conceptuel : il m'a permis de mettre en œuvre des compétences en Python, en manipulation de données avec `pandas`, mais aussi de me familiariser avec les enjeux liés à la qualité des données, à la désambiguïsation, et aux méthodes d'évaluation de performance comme la précision, le rappel ou le F1-score.

Au-delà des compétences techniques acquises, ce projet m'a également donné une meilleure intuition des défis concrets que posent les systèmes de recherche d'information, même dans des contextes limités ou simulés.

Contribution du travail

Tout au long de ce projet, je me suis retrouvé à assumer seul la totalité des tâches à réaliser, du début jusqu'à la fin du semestre. Mon binôme n'a été présent qu'à une seule séance de TD, et malgré mes multiples relances par message, il a systématiquement ignoré ou feint d'ignorer mes sollicitations. À aucun moment, je n'ai pu compter sur une répartition du travail réelle ni sur un engagement de sa part.

La seule initiative prise de son côté a été une tentative de rédaction du rapport, dans laquelle il évoquait des bibliothèques ou méthodes que nous n'avons jamais utilisées, ce qui témoigne d'un manque de suivi concret du projet. Lorsqu'il m'a assuré avoir « compris les TDs », cela intervenait souvent une à deux semaines après que j'aie moi-même finalisé ces mêmes travaux. Je lui avais demandé — dans un dernier effort pour le faire participer — de simplement recopier proprement les codes des TD2 à TD5, tâche qu'il a réalisée très tardivement, en me transmettant ses fichiers le jour même de la date limite. Pire encore, les noms de fonctions avaient été modifiés, cassant ainsi la cohérence et la logique de tout le projet.

Ayant perdu espoir d'obtenir une contribution constructive, j'ai finalement assumé toutes les dimensions du projet : analyse, implémentation, tests, corrections, évaluation et rédaction. J'ai essayé au début du semestre de ne pas faire remonter ces problèmes dans l'espoir d'un redressement de la situation, mais aujourd'hui je ne vois plus d'autre choix que d'exposer les faits en toute transparence.

Je transmets donc avec ce rapport :

— l'intégralité du travail que j'ai mené et documenté ;

— en annexe, le code que mon binôme m'a envoyé le jour du rendu, pour que vous puissiez, si nécessaire, l'évaluer de manière distincte selon sa propre contribution.

Je tiens à rester juste dans mon propos : je n'ai pas souhaité accuser hâtivement, mais plutôt relater fidèlement les circonstances dans lesquelles s'est déroulé ce projet.