LI6 : construire un moteur de recherche

Laure Runser
Lily Olivier

May 22, 2022

Abstract

Rapport de projet sur la construction d'un moteur de recherche pour parcourir la base de données de Goodreads.

1 Introduction

Dans le cadre du cours de LI6 sur le TAL, nous avons souhaité construire un moteur de recherche de la base de données de Goodreads, un réseau social orienté sur le thème de la lecture. Nous avons ainsi construit un site qui permet de chercher un livre selon le titre ou son résumé, et qui propose ensuite des recommandations de livres similaires.

2 Stack technique

Nous avons choisi d'utiliser le packet python sk-learn pour le traitement TAL des données parce que c'est le packet que nous avons utilisé en TP. Nous avons aussi utilisé nltk pour pré-traiter les données et numpy pour les trier.

Pour la création du site, nous avons choisi d'utiliser le framework Django. Il présente deux principaux avantages : - c'est un framework python, ce qui nous permet de l'interfacer facilement avec sk-learn - la création de pages se fait avec un système de templates dynamiques, ce qui nous a permis facilement d'injecter des données dans les pages pour affiches les différents livres et index

3 Base de données

Nous avons utilisé un dataset public trouvé sur Kaggle. Les données ont été récupérées et publiées par Austin Reese, à partir de la page "meilleurs livres" de Goodreads. La base de données est très complète, avec le titre et l'auteur du livre, mais aussi son synopsis, les notes données par les utilisateurs du site ou bien le nombre de pages. Pour notre moteur de recherche, nous nous sommes concentrés sur le titre et le synopsis du livre.

Vous pouvez télécharger la base ici si vous le souhaitez (elle est déjà chargée dans notre projet) : https://www.kaggle.com/datasets/austinreese/goodreads-books

4 Compilation et lancement

Pour compiler le projet, il faut : 1) cloner le dépôt disponible ici : https://github.com/laurerunser/goodreads-search-engine 2) installer les dépendances :

```
pip install django nltk sklearn numpy
```

3) se placer dans le dossier goodreads et lancer le server avec

```
python manage.py runserver
```

4) mettre à jour les données

```
python manage.py migrate
```

4) lancer un navigateur et aller à la page http://127.0.0.1:8000/search/ pour avoir l'index de tous les livres, ou à la page http://127.0.0.1:8000/search/search_page pour effectuer une rechercher.

4.1 Si la base de données semble vide

Si la base de données est vide (rien ne s'affiche sur la page index), il faut réimporter les données dans le server : 1) lancer le shell interactif de Django

```
python manage.py shell
```

2) importer la base de données

```
exec(open("searchbooks/import.py").read())
exit()
```

Cette opération peut prendre plusieurs secondes. Vous pouvez interrompre le chargement des données après une dizaine de secondes : tous les livres ne seront pas chargés, mais un petit nombre suffit pour observer des résultats.

Alternativement, vous pouvez modifier le fichier "searchbooks/inmport.py" pour charger un fichier plus léger :

```
with open('searchbooks/data/test.csv') as csvfile:
```

à la place de

```
with open('searchbooks/data/goodreads_books.csv') as csvfile:
```

Ce fichier peut prendre jusqu'à une dizaine de secondes à charger si votre ordinateur est lent.

5 Usage

Les tests de cette section ont été réalisés avec le jeu de données test.cvs (et non pas avec le jeu de données complet) afin que vous puissiez les reproduire facilement.

5.1 Systeme de recommandation

Pour obtetnir les recommandation par rapport à un titre, il suffit de cliquer sur celui-ci a la page http://127.0.0.1:8000/search. Les titres indiqués sont parfois associés à un numéro. Vous pouvez accéder à la page de recommandation de ces titres en vous rendant sur la page http://127.0.0.1:8000/search/numero

Par exemple, lorsque l'on est sur la page de "Princess in Waiting" (93729), on peut voir que la première et la troisième recommandation sont aussi des histoires tournant autour du même thème (les princesses). De la même façon, pour la BD Asterix le Gaulois (71292), la seconde recommendation est le livre Caesar de Colleen McCullough, tout deux situant leurs récits dans le même contexte spatio temporel

On remarque aussi que le livre "The Flame Trees of Thika: Memories of an African Childhood" reviens dans beaucoup de suggestions (il est présent dans le top deux des deux ci-dessus dessus, mais dans beaucoup d'autres aussi). Quelques exemples: Princess in Waiting, Collected Stories, Dangerous Lady, etc. Le résumé de ce livre est assez long et contient un certain nombre de mots référant à des thèmes très récurent dans la littérature (comme young girl, adventure, parents, etc). Cela peut expliquer pourquoi ce livre a souvent un bon classement.

5.2 Recherche par mot clés

A partir de la page http://127.0.0.1:8000/search/search_page, vous pouvez faire des recherches concernant les titres des livres ou bien leurs descriptions.

Par exemple, tapez girl et vous serez redirigé vers une page contenant quatres résultats, contenant tous le mot girl dans leur titres, comme par exemple The Girl at Midnight (20345202). De la même façon, on peut rechercher dans la description. Par exemple, le mot clé monster renvoie entre autres une réécriture du célèbre conte La Belle et la Bête (Beauty and the Beast, 13570639) ou un livre à propos des loups-garous : Cycle of the Werewolf (550844).

Les résultats seraient plus nombreux et plus fiables avec la base donnée entière, plutôt qu'un simple extrait de 500 livres arbitraires.

6 Implémentation : TAL

6.1 Pré-traitement des données

Afin d'avoir des recherches cohérentes, nous avons fait un pré-traitement des données pour qu'elles soient utilisables. Ce pré-traitement est appliqué soit au titre, soit au synopsis, en fonction de ce dans quoi l'utilisateur cherche. Le même traitement est appliqué aux mots de recherche donnés par l'utilisateur. Ce code est situé dans le fichier "searchbooks/views.py" (dans la fonction "clean-up bas de la page).

Nous avons tout d'abord retiré la ponctuation et mis tous les mots en minuscule :

```
str = str.translate(str.maketrans('', '', string.punctuation))
str = str.lower()
```

Nous avons ensuite découpé le texte en mots individuels :

```
words = str.split(' ')
```

Nous n'avons pas remarqué de problèmes dans la séparation en mots, même si nous avions vu en TP à quel point cela pouvait être une opération compliquée à gérer.

Nous avons ensuite utilisé le PorterStemmer de nltk pour faire de la stematization. Cela permet de ramener tous les mots à des racines communes qui peuvent ensuite être comparées :

```
stemmed_words = []
for w in words:
    stemmed_words.append(STEMMER.stem(w))
```

Enfin, nous avons enlevé les mots les plus communs de la langue anglaise (la majorité des informations sont en anglais dans la base).

```
stop_words = {'the', 'be', 'to', 'of', 'and', 'a', 'in', 'that', 'have', 'I', 'it', 'for
              'as', 'you', 'do', 'at', 'this', 'but', 'his', 'by', 'from', 'wikipedia'}
words = [w for w in words if w not in stop_words]
```

Nous avons réutilisé la liste de mots que nous avions vue en TP, même si elle n'est pas la plus pertinente. Par exemple, le mot 'wikipedia' n'apparaît pas dans la base de données, mais nous aurions sûrement pu enlever les mots 'livre' ou 'auteur' qui apparaissent souvent dans les synopsis.

6.2 Recherche par title et synopis

Nous avons procédé d'une manière assez similaire à celle vue en TP, mais adaptée pour qu'elle soit compatible avec Django. Le code suivant est disponible dans le fichier "searchbooks/views.py", dans la fonction "search_results".

Nous commençons par récupérer les données dans la base de données. Puis nous leur appliquons le pré-traitement mentionné ci-dessus.

```
all_content = []
all_ids = []
for data in all_books:
    if request.POST['where_search'] == 'title':
        all_content.append(cleanup(data.title))
    else:
        all_content.append(cleanup(data.description))
    all_ids.append(data.id)
```

Puis nous appliquons TF-IDF aux données et à la requête de l'utilisateur.

```
# apply TFIDF on books
vectorizer = TfidfVectorizer()
matrix_tfidf = vectorizer.fit_transform(all_content)

# prep the search query
query_tfidf = vectorizer.transform([query])
```

Enfin, nous appliquons la fonction de distance cosine, et nous gardons les 20 meilleurs résultats.

```
# get the cosine distance and then keep the 20 best
distances = cosine_similarity(query_tfidf, matrix_tfidf).flatten()
best = np.argsort(distances)
```

Il ne reste alors plus qu'à envoyer les livres choisis à la vue qui s'occupe de les afficher ! Cette vue fait une dernière vérification : si la distance cosine est trop petite, le livre est ignoré.

Nous avons trouvé dans nos tests que la conjonction de ces deux mesures (seulement les 20 meilleurs et ignorer les trop petites distances) permet de ne garder que des résultats pertinents.

6.3 Recommendations

Le système de recommandations fonctionne exactement de la même manière que le système de recherche. A la place de la requête de l'utilisateur, on utilise la description du livre comme clé de recherche.

Cela donne des résultats assez peu satisfaisants cependant, parce que les descriptions sont souvent assez pauvres en informations, et que nos tests ont été faits sur le jeu de données le plus petit. Il n'y avait donc pas beaucoup de livre qui avaient des mots ou des thèmes en commun.

7 Implémentation : web

7.1 Mise en place de Django

C'était la première fois que nous utilisions le framework, et nous avons été ravies de sa simplicité. La documentation officielle est très bien faite, et les tutoriels nous ont rapidement permis de nous lancer et de créer le site.

Django était très adapté à ce projet, car il nous a permis d'avoir une base de données sous la main, très bien intégrée au système de template. Cela permet de faire circuler l'information très facilement entre les différents composants, sans avoir à parser du JSON par example.

De plus, la génération des pages server-side permet une grande flexibilité pour ajouter un nombre arbitraire de résultats, tout en gardant une interface assez agréable.

7.2 Les templates

Nous avons utilisé un certain nombre de templates pour notre site :

1. book_details.html : la page d'un livre. Elle contient quelques informations sur le livre, et ensuite une liste des livres recommendés
2. index.html : l'index avec tous les livres de la base de données
3. search_results.html : les résultats d'une recherche
4. search_page.html : la page avec le formulaire de recherche

Vous pouvez les retrouver dans le dossier "searchbooks/templates/search".

7.3 Ajout de la base de données

Nous avons eu un peu de mal à ajouter la base de données sur le server. Vous avez pu voir dans les instructions d'installation qu'il faut plusieurs étapes, et nous en sommes désolées.

7.4 Style

Nous avons ajouté quelques lignes de CSS pour rendre les pages plus jolies, mais elles restent assez illisibles, surtout lorsque le synopsis des livres est affiché.

8 Améliorations

Nous pourrions améliorer les recommendations, notamment en utilisant certaines des informations qui sont données dans la base (par example les "étagères" sur lesquelles les utilisateurs ont placés les livres pourraient aider à catégoriser).

Certains livres sont dans une langue autre que l'anglais, mais nous traitons toutes les données à partir de la langue anglaise. Nous pourrions ajouter un système pour rechercher par langue, mais aussi pour traiter les données des autres langues en fonctions de leurs particularités (par example avec une liste de stop-words adaptée).

Le style du site pourrait être amélioré : les longues descriptions sont assez difficiles à lire en l'état. Nous disposons dans la base de l'ufr de la photo de couverture, on aurait pu l'ajouter pour embellir les pages.