System Design du Recommandeur de critiques de films

Réalisée par : BEN ABDELATIF Maya

Lien vers GitHub : https://github.com/tiredoof/Movie-critique-recommender

Lien vers Docker : mayq1/data_critique:latest

Introduction

Ce document présente le système de recommandation et d'analyse de critiques de films développé pour le test technique de SensCritique. L'objectif est de fournir pour une critique introduite dans une application web de recommandation de film, une liste paginée de critiques similaires pour le même film.

Afin de réaliser ce système, nous avons intégré un traitement automatique du langage naturel basé sur des modèles de type *Sentence-BERT* afin de calculer la similarité sémantique entre les critiques. Cela permet de regrouper et d'analyser les avis en fonction de leur contenu et sémantiques, au-delà des simples mots-clés.

Dans ce rapport, nous présentons :

- Le schéma d'architecture logicielle/infrastructure expliquant comment la solution fonctionne.
- Le fonctionnement du site de traitement des critiques (flux online/offline).
- Justifications des choix techniques utilisés pour l'implémentation.

Donc Ce document constitue un guide pour comprendre le design, le fonctionnement et l'utilisation de notre application.

1. Schéma du Fonctionnement de l'Application

La Figure 1 représente une simulation du fonctionnement de l'application web **« Recommandeur de critiques de films »** en temps réel.

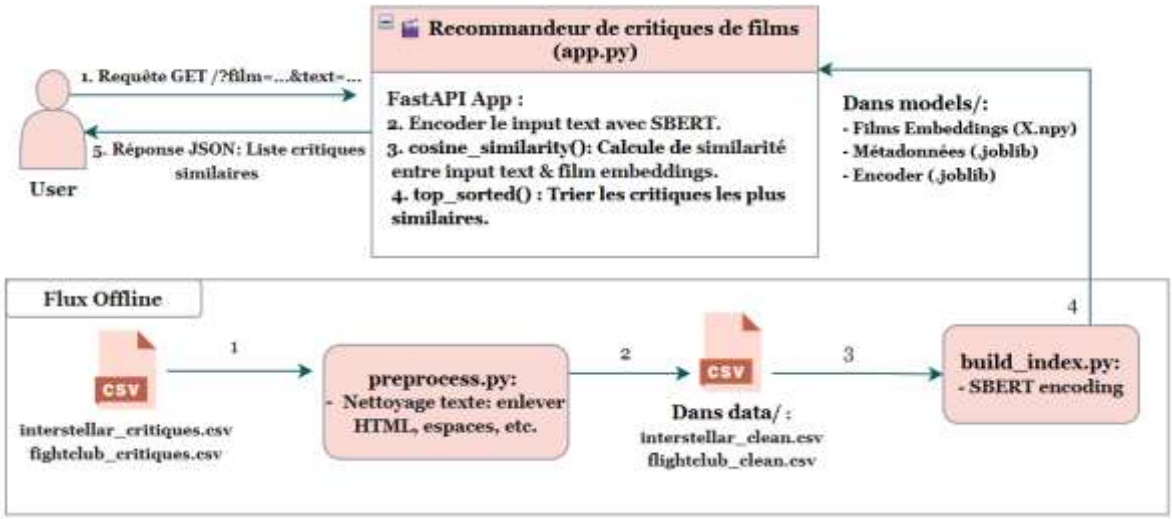

Figure 1 : Schéma d'Architecture des flux de l'application

2. Flux du Système

Nous pouvons diviser ces flux en deux parties :

2.1 Flux Online

1. L'utilisateur envoie une requête GET avec un texte et un film cible.
2. L'API (sur *app.py* avec FastAPI) encode la critique entrée avec le modèle SBERT.
3. Nous appelons la fonction **cosine_similarity( )** au niveau de *app.py*, pour calcule la similarité cosinus entre l'input et les embeddings précalculés.
4. Les résultats sont triés par ordre de similarité décroissante **(top_sorted( )).**
5. L'API retourne une réponse JSON contenant les critiques les plus proches pour le même film.

2.2 Flux Offline

1. Les fichiers bruts de critiques interstellar_critiques.csv & fightclub_critiques.csv sont nettoyés via *preprocess.py* (suppression HTML, espaces, etc).
2. Les critiques nettoyées interstellar_clean.csv & fightclub_clean.csv sont sauvegardées dans le chemin data/.
3. *build_index.py* encode les critiques avec SBERT méthode et puis retourne les embeddings (.npy), les métadonnées (.joblib), l'encodeur des deux films.
4. Les résultats derniers sont stockés dans le chemin models/ et sont utilisées durant l'appel de la fct **cosine_similarity( )** dans l'inférence online.

Pour connaitre plus sur la conception / workflow interne de l'application, veuillez vous rediriger vers README.md sur mon lien GitHub, où vous trouverez une explication plus détaillée + un schéma récapitulatif du développement et déploiement de l'application web.

3. Justification des Choix Techniques

- **SBERT (Sentence-BERT)**: est choisi car il produit des embeddings contextuels de phrases et fichiers **optimisés pour la similarité sémantique**. Contrairement à TF-IDF qui repose uniquement sur la cooccurrence de mots, ou Word2Vec/GloVe qui ont des représentations mot à mot nécessitant une agrégation perdant souvent le contexte, **SBERT** encode directement la phrase entière en tenant compte de l'ordre des mots et de leur interaction, ce qui **capture le sens global de la critique** (ton, ironie, contexte), permettant ainsi de comparer les critiques sur leur sens global et pas seulement sur leurs vocabulaires.
- **Cosine Similarity** : c'est métrique standard et robuste en NLP pour comparer les vecteurs de même dimension ; elle est indépendante de la norme (nous comparons directement les directions) et donc adaptée parfaitement aux embeddings SBERT.
- **Numpy (.npy) & Joblib (.joblib):** utilisés pour stocker rapidement les matrices d'embeddings et les métadonnées associées. Ce format est léger, très rapide au chargement et bien adapté aux jeux de données modestes comme nos fichiers _critiques.csv. Pour les grands volumes, j'utilise généralement FAISS afin d'accélérer les recherches de voisinage et réduire la latence.
- **FastAPI:** c'est un Framework web très connu et asynchrone, il est idéal pour exposer facilement une API en réseau externe. Il combine both la performance, simplicité de prototypage et la documentation automatique (Swagger/OpenAPI), permettant ainsi à la fois un déploiement rapide et une mise en production allégée, ce qui est idéal pour ce projet.
- **Docker:** l'outil le plus facile et rapide pour le dépoilement du système et le rendre portable et facile à intégrer sur n'importe quel environnement.

Conclusion

L'application web « **Recommandeur de critiques de films »** développée illustre l'apport du IA (NLP) dans l'analyse de critiques de films. Grâce à l'utilisation de **Sentence-BERT**, il devient possible de dépasser la simple recherche par mots-clés et de proposer une analyse plus fine et pertinente des avis des utilisateurs.

En perspective, plusieurs améliorations pourraient être envisagées : élargir le nombre de films disponibles, affiner les modèles utilisés en utilisant des LLMs et enrichir l'interface avec de nouvelles fonctionnalités interactives.

Donc ce travail constitue une première étape vers un système de recommandation plus intelligent et évolutif, combinant **l'IA** et **expérience utilisateur**.