Rapport de projet - Traitement automatique du langage

Émilien LINSART & Hugo TUFFIN

1 Introduction

Dans le cadre du projet de traitement automatique du langage, ce rapport détaille le développement d'une application interactive en Python, destinée à fournir des recommandations de films. Nous souhaitons ici développer une application qui fournit aux utilisateurs des idées de films à regarder lorsqu'ils ne sont pas inspirés. Afin d'innover par rapport aux plateformes qui fournissent déjà ce genre de recommandations, nous avons souhaité utiliser des techniques de recommandations différentes. L'objectif principal de ce projet est donc de transcender les approches conventionnelles de recommandation basées sur les campagnes publicitaires ou la popularité mainstream pour mettre en lumière des œuvres cinématographiques qui, autrement, pourraient passer inaperçues par les amateurs de films.

Il est en effet fréquent que les utilisateurs soient souvent submergés par un vaste choix de films. Les algorithmes de recommandation actuels tendent à privilégier les films dotés de budgets marketing importants ou ceux produits par des studios renommés, laissant ainsi de côté des perles moins connues. Notre projet vise à combler cette lacune en utilisant une approche centrée sur l'utilisateur, exploitant les avis réels des spectateurs pour recommander des films. Pour cela, nous utiliserons des analyses de sentiments sur les avis des films laissés sur le site web The Movie Database (TMDB)[1]. L'objectif est donc de recommander, pour un genre de film choisi par l'utilisateur, une liste de film qui semblent avoir le plus plu. Cette liste devra être ordonnée pour faire apparître les films à voir en priorité.

Ce document vise donc à présenter les différentes étapes qui ont été nécessaire à la réalisation du projet; de la collecte des données à la création de l'application qui présente les résultats.

2 Collecte des données

Afin de disposer exactement des informations dont nous voullions, nous avons décidé de construire nous même notre propre base de données. Dans un premier temps, nous avons tenté de récolter les données du site https://www.imdb.com/. Deux méthodes ont été testées pour recueillir les données de ce site :

- Récupération des données à partir de scraping du site internet
- Récupération des données en passant par l'API du site internet
- Utilisation d'une librairie Python adaptée à ce site : imdbpy (cette approche n'a pas été approfondie)

Quelques difficultés dans la collecte des données pour ce site nous ont amené à changer de stratégie. Face à ces contraintes, notre choix s'est finalement porté sur le site The Movie Database (TMDB). Cette plateforme offre une API robuste, nous permettant ainsi d'accéder de manière structurée à une étendue importante d'informations sur les films. C'est donc avec les données issues de l'API de TMDB que nous avons continué nos traitements. Un trace de la collecte des données sur le site IMDb, avec les 2 méthodes, reste tout de même disponible dans le fichier `collecte_IMDb.ipynb`.

Afin de limiter nos requêtes à l'API pour réduire les temps d'exécution, nous avons choisi de restreindre la période d'analyse pour ne pas avoir à récupérer l'ensemble des films présents sur le site. Nous avons de plus ajouté une deuxième condition avant de stocker les données. Il fallait qu'un film dispose d'au minimum un avis pour pouvoir faire partie de notre base. Cela était nécessaire puisque le suite du travail repose justement sur ces commentaires. C'est donc à cela que servent les fonctions `get_movies_between_dates`

[1] https://www.themoviedb.org/

et `filter_movies_with_reviews`. Nous ne gardons ensuite que les informations qui nous intéressent, en les stockant dans un format qui nous arrange. La fonction `get_movie_details_and_reviews` remplie cette tâche. Puisque cette étape peut prendre un certain temps, nous avons stocké les résultats dès la première exécution dans le fichier `movies_details_3mois.json` pour éviter de refaire cette étape à chaque fois.

Au final, pour chaque film remplissant les conditions, nous disposons d'une grande diversité de données. Toutes ne seront pas utilisées par la suite, il s'agit là d'un axe d'amélioration de l'application. Celles dont nous allons le plus nous servir concernent les avis laissés. Nous disposons pour chacun d'entre eux de :

- Informations sur l'auteur
- Date de rédaction du commentaire
- Contenu de l'avis rédigé
- Une note attribuée au film (nous nous en servirons pour comparer la qualité de nos recommandations). Il arrive que cette note n'ait pas été renseignée

3 Traitement des données

Dans le cadre de notre projet d'analyse de sentiments des avis de films, une étape cruciale a été le nettoyage et la préparation des données textuelles. Cette étape est essentielle pour garantir que les analyses ultérieures soient à la fois précises et pertinentes. Pour ce faire, nous avons mis en œuvre une fonction spécifique de traitement du texte, nommée `clean_avis`, qui effectue plusieurs opérations de nettoyage et de filtrage sur chaque avis.

La première opération est de retirer tous les caractères non alphabétiques du texte. Cela inclut les chiffres, les symboles et les ponctuations, qui sont remplacés par des espaces. Cette étape est réalisée à l'aide d'une expression régulière `re.sub('[^a-zA-Z]', '␣', text)`, qui garantit que seuls les caractères alphabétiques sont conservés. Par la suite, nous convertissons tout le texte en minuscules pour uniformiser les données `text.lower()`.

Après le nettoyage initial, nous procédons à la segmentation du texte en mots individuels et appliquons un filtrage pour retirer les mots non pertinents. Ce filtrage repose sur une liste personnalisée de stopwords, qui inclut les mots courants en anglais ainsi que des termes spécifiques à notre contexte, comme 'film' et 'movie'. Ces termes sont retirés car, bien que fréquemment présents dans les avis de films, ils n'apportent que peu d'informations utiles pour l'analyse de sentiment.

4 Fouille de texte

Pour donner un aperçu du contenu et des avis sur un film, nous avons mis en place des WordClouds affichés lorsque l'on clique sur un film. Nous avons pour cela créé un WordCloud par film, qui prend en compte l'ensemble des avis laissés pour ce film. Chaque WordCloud est stocké dans un répertoire et nommé à partir de son ID pour pouvoir y accéder facilement ensuite.

Comme évoqué dans l'introduction, nous avons ensuite regroupé les films par genre pour que l'utilisateur tombe plus facilement sur les films qui l'intéressent. L'analyse de sentiments qui va être détaillée par la suite a donc été réalisée pour comparer les différents films d'une même catégorie. Dans la partie consacrée à l'analyse de sentiments, nous utilisons la bibliothèque Python TextBlob, une interface simplifiée pour le traitement du langage naturel. Une des fonctionnalités de TextBlob est le fait de pouvoir analyser le sentiment d'un texte avec la commande `TextBlob(text).sentiment.polarity`. Cela nous rend, après avoir déduit un objet Sentiment qui extrait le sentiment global du texte, un nombre compris entre -1 et 1. -1 représente un sentiment extrêmement négatif, 0 indique une neutralité tandis que 1 indique un sentiment extrêmement positif. Cette analyse permet donc de transformer des données textuelles qualitatives en valeurs quantitatives exploitables.

Après avoir mis en œuvre l'analyse de sentiments pour évaluer individuellement les avis, il était nécessaire de concevoir une méthode pour agréger les scores pour classer les films. Pour obtenir cet indicateur, nous avons procédé à une simple agrégation en calculant la moyenne des notes de sentiments obtenues pour chaque

film. Pour chaque catégorie de genre, nous avons ainsi pu sélectionner les films ayant la moyenne de sentiment la plus élevée. Cette sélection a été réalisée dans le but de recommander les films les plus appréciés dans chaque catégorie. Les films sont ainsi classés du meilleur au moins bon, selon leur score final.

5 Conclusion

Pour un meilleur confort de lecture, nous avons présenté nos résultats dans une application. Elle donne accès à 3 recommandations par genre. Pour avoir plus d'informations sur un film si nécessaire, il est possible de cliquer dessus. On peut ainsi voir le détail du film, son WordCloud généré, et 5 avis déposés par les utilisateurs (ou moins si nous disposons de moins d'avis). Un dernier traitement a été nécessaire pour afficher les notes. Pour une meilleur clarté et pour pouvoir ensuite comparer avec les notes originales des utilisateurs, il a fallu transformer l'échelle des notes pour la passer d'un intervale de -1 à 1 à un intervale de 0 à 10. Pour accéder à l'application, il faut lancer les chunks avant celui qui démarre par `app = Flask(__name__)`. Puis, en lançant ce dernier, un lien apparait dans l'output. Ce lien redirige vers l'application.

Au final, pour ce qui est de nos notes estimées, nous voyons que nos résultats ne sont pas très bons. Plusieurs aspects peuvent expliquer cette tendance. Tout d'abord, il est important de noter que la quantité d'avis disponibles pour chaque film varie considérablement. Dans les cas où le nombre d'avis est limité, notre analyse de sentiment basée sur ces avis peut ne pas refléter fidèlement l'opinion générale des spectateurs. Cette limitation des données peut conduire à une polarité moyenne qui ne capture pas la diversité des opinions ou qui est trop influencée par des avis individuels. En outre, il est courant de constater que les notes attribuées par les utilisateurs peuvent rapidement devenir extrêmes par rapport à leur ressenti réel. Les utilisateurs ont tendance à exprimer des opinions très fortes, soit très positives soit très négatives, ce qui peut fausser l'évaluation moyenne. Par exemple, un utilisateur peut attribuer la note maximale à un film simplement parce qu'il est fan d'un acteur particulier, ou à l'inverse, donner une note très basse en raison d'un désaccord avec certains éléments du film, indépendamment de sa qualité globale. Les recommandations qui découlent de ces estimations ne sont donc peut-être pas optimales.