

Filière : Data engineer

Anime Recommendation System Text Mining & NLP Application

Réalisé par:

- Rahal jalal

Encadré par:

Mr. Abdelhak MAHMOUDI BENAMER



SOMMAIRE



1

INTRODUCTION

2

DATASET OVERVIEW

3

**DATA UNDERSTANDING
& CHALLENGE**

4

DATA PREPARATION

5

MODEL BUILDING

6

**RESULT & MODEL
COMPARAISON**

7

MODEL DEPLOYMENT

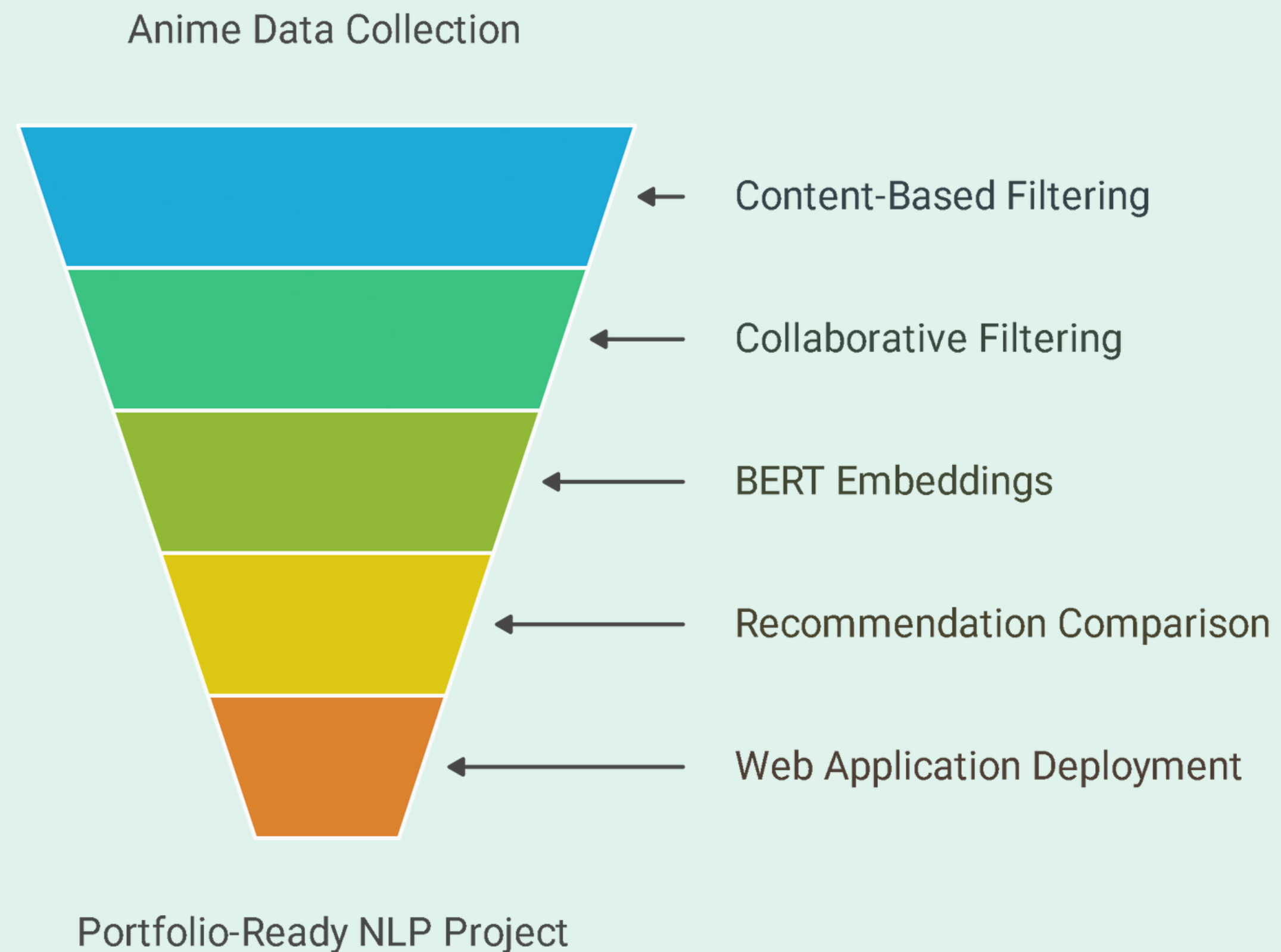
8

CONCLUSION

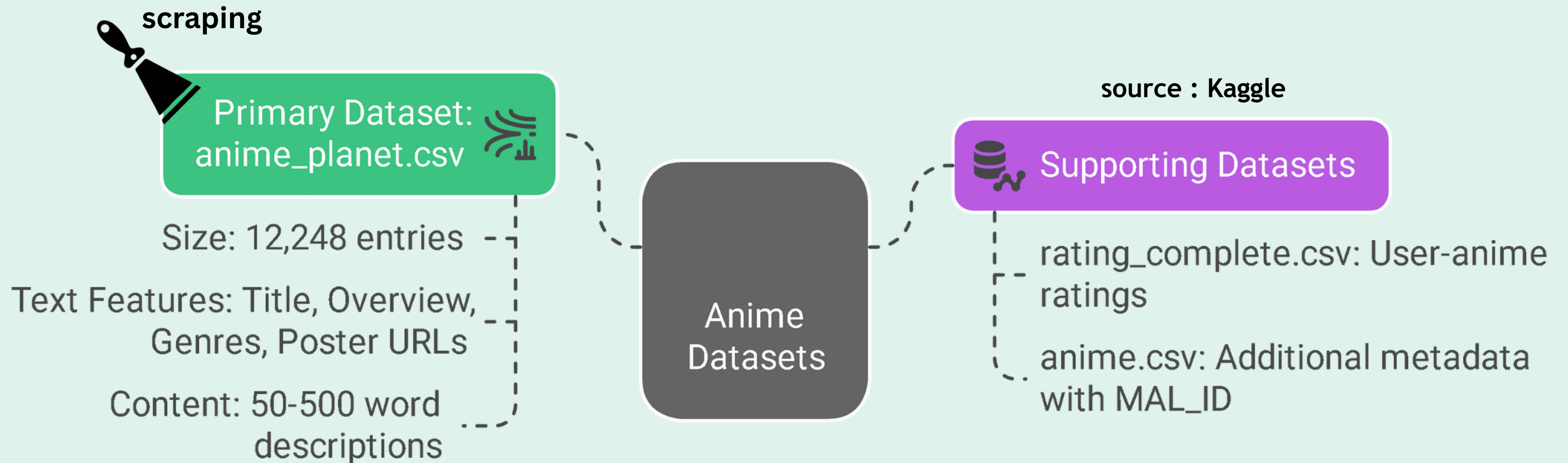




Problem:
With thousands of anime titles, users struggle to discover content matching their preferences



Anime Dataset Overview



Introduction

Problématique

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion



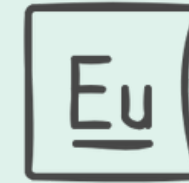
Language
Patterns



Genre
Complexity



Text Variety



Multilingual
Elements



Inconsistent
Formatting



Missing Data



**Balancing Dataset Characteristics and Text
Mining Challenges**

Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

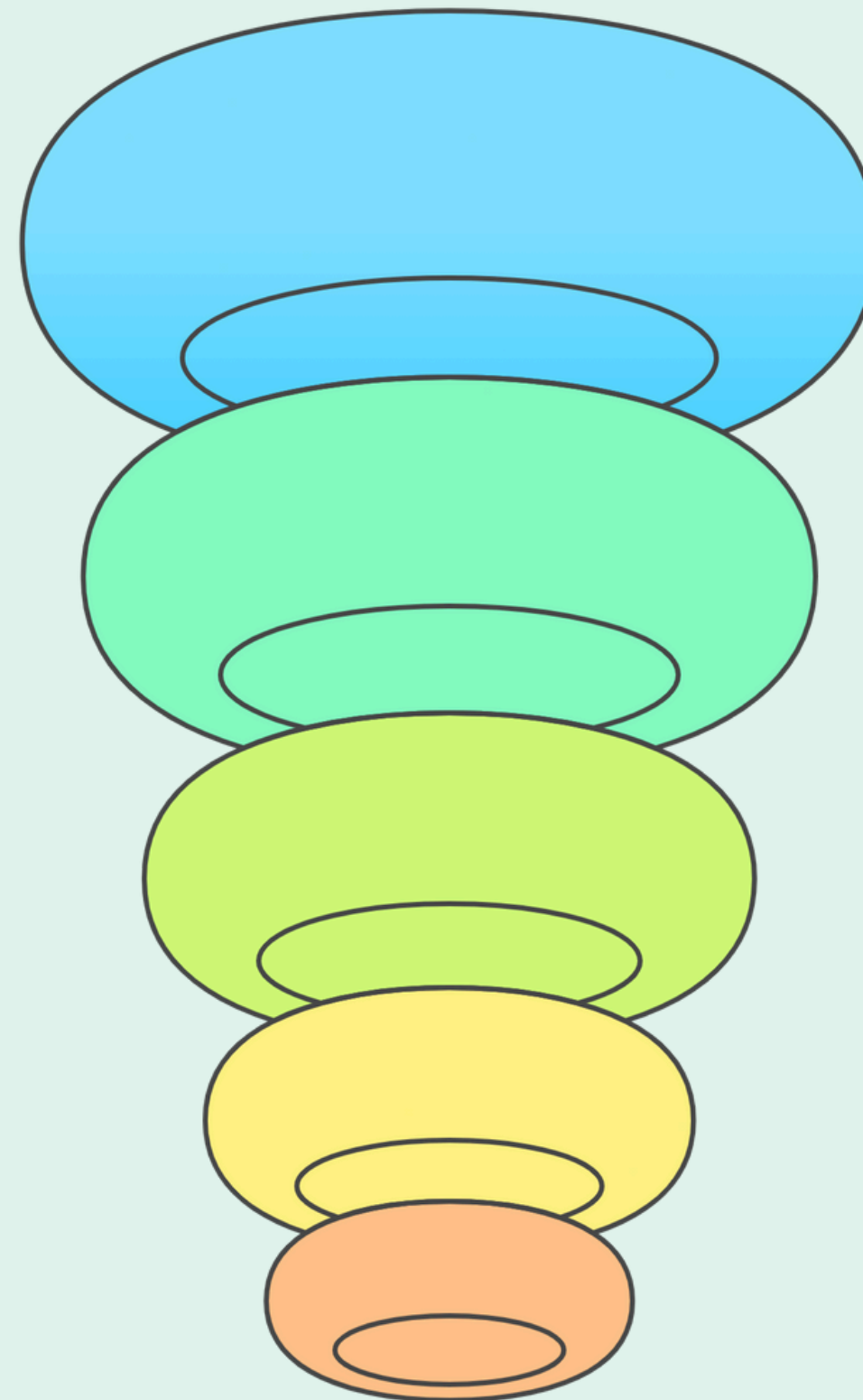
Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

Text Preprocessing Funnel



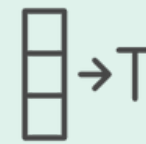
Text Aggregation

Combining text sources for comprehensive features



Data Cleaning

Handling missing values and normalizing text



CountVectorizer

Creating sparse matrix with stop words removed



TF-IDF

Reducing impact of common words with weighting



BERT Preprocessing

Tokenizing and padding for uniform input

Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

```
data=pd.read_csv('anime_planet.csv')
data['tags']=data['Title']+data['overview']+data['genres']
data['tags']=data['tags'].str.strip()
data['tags'][0]
data['title']=data['Title']
new_data=data.drop(columns=['overview','genres','Title'])
new_data=new_data[['title','tags','Poster']]
```

```
from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer(max_features=12248, stop_words='english')
vector=cv.fit_transform(new_data['tags'].values.astype('U')).toarray()
```

```
#Import TfidfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
new_data['tags'] = new_data['tags'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(new_data['tags'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```



Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

Content Based Filtering

Le filtrage basé sur le contenu (Content Based Filtering) est une méthode populaire utilisée dans les systèmes de recommandation, en particulier pour les animes. Cette approche repose sur les caractéristiques des animes eux-mêmes pour suggérer des titres similaires à ceux que l'utilisateur a déjà appréciés

Ce type de filtrage est particulièrement utile pour les nouveaux utilisateurs qui n'ont pas encore beaucoup d'historique de visionnage, car il peut rapidement identifier des animes susceptibles de correspondre à leurs goûts.

Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

RECOMMENDATION FUNCTION FOR CONTENT BASED FILTERING

```
1 def get_recommendations(title, cosine_sim=cosine_sim):
2     # Get the index of the movie that matches the title
3     idx = indices[title]
4
5     # Get the pairwise similarity scores of all movies with that movie
6     sim_scores = list(enumerate(cosine_sim[idx]))
7
8     # Sort the movies based on the similarity scores
9     sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
10
11    # Get the scores of the 10 most similar movies
12    sim_scores = sim_scores[1:11]
13
14    # Get the movie indices
15    movie_indices = [i[0] for i in sim_scores]
16
17    # Return the top 10 most similar movies
18    return new_data['title'].iloc[movie_indices]
```



Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

Collaborative Filtering

Le filtrage collaboratif (Collaborative Filtering) est une technique couramment utilisée dans les systèmes de recommandation d'anime pour suggérer des titres aux utilisateurs en se basant sur les préférences et comportements d'autres utilisateurs.

Contrairement au filtrage basé sur le contenu, qui se concentre sur les caractéristiques des animes eux-mêmes, le filtrage collaboratif analyse les interactions des utilisateurs avec divers animes, telles que les évaluations, les notes et les historiques de visionnage. Par exemple, si un utilisateur A a des goûts similaires à ceux d'un utilisateur B, et que l'utilisateur B a apprécié certains animes que l'utilisateur A n'a pas encore vus, ces animes seront recommandés à l'utilisateur A.

Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

RECOMMENDATION FONCTION FOR COLLABORATIVE FILTERING

```
# Fonction pour faire des recommandations pour un utilisateur donné
def get_recommendations(user_id, algo, ratings, anime_data, num_recommendations=10):
    # Obtenir tous les films que l'utilisateur n'a pas encore notés
    all_anime_ids = ratings['anime_id'].unique()
    rated_anime_ids = ratings[ratings['user_id'] == user_id]['anime_id']
    unrated_anime_ids = [anime_id for anime_id in all_anime_ids if anime_id not in rated_anime_ids.values]

    # Prédire les notes pour les films non notés
    predictions = [algo.predict(user_id, anime_id) for anime_id in unrated_anime_ids]

    # Trier les prédictions par note estimée
    predictions.sort(key=lambda x: x.est, reverse=True)

    # Retourner les meilleurs films recommandés
    top_predictions = predictions[:num_recommendations]
    top_anime_ids = [pred.iid for pred in top_predictions]

    # Obtenir les titres des films recommandés
    recommended_animes = anime_data[anime_data['MAL_ID'].isin(top_anime_ids)][['MAL_ID', 'Name']]
    return list(recommended_animes['Name'])

# Exemple d'utilisation de la fonction de recommandation
user_id = 137
recommendations = get_recommendations(user_id, algo, ratings, anime_data)
print("Anime recommandés pour l'utilisateur {}".format(user_id))
print(recommendations)
```



Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

BERT

**au place d'utiliser le bag of words on va encoder les mots à des
language compréhensible par BERT**

Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison




Model
Deployment

Conclusion

RECOMMENDATION FONCTION FOR BERT

```
def get_recommendations(title, anime_data, top_k=5):  
    # Obtenir l'index de l'anime  
    idx = anime_data[anime_data['Title'] == title].index[0]  
    # Calculer la similarité cosinus entre l'anime et tous les autres animes  
    cosine_sim = cosine_similarity(anime_data['encoded'].tolist()[idx], np.vstack(anime_data['encoded'].tolist()))  
    # Obtenir les indices des animes les plus similaires  
    sim_scores = list(enumerate(cosine_sim[0]))  
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)  
    sim_scores = sim_scores[1:top_k + 1] # Exclure l'anime lui-même  
    # Obtenir les titres des animes les plus similaires  
    anime_indices = [i[0] for i in sim_scores]  
    return anime_data['Title'].iloc[anime_indices]  
  
# Exemple d'utilisation de la fonction de recommandation  
title = 'Naruto'  
recommendations = get_recommendations(title, anime_data)  
print("Animes recommandés pour '{}': {}".format(title, recommendations.tolist()))
```



		<div><div></div><div>TF-IDF</div></div>	<div><div></div><div>Collaborative Filtering</div></div>	<div><div></div><div>BERT Embeddings</div></div>
Speed		1 second	N/A	2.5 seconds
	Precision/Quality	78% relevance	RMSE Score: 0.94	Superior thematic matching
Coverage		Works for all anime	Limited by rating data	N/A
Strengths		Fast, explainable	Discovers hidden preferences	Best semantic understanding
Weaknesses		Limited to content	Cold start problem	Computational overhead
Other		Example Quality	N/A	Memory Usage: 2.1GB

Anime Recommender System

Select anime from dropdown

Demon Slayer: Kimetsu no Yaiba - Entertainment District Arc

Show Recommend

Demon Slayer:
Kimetsu no Yaiba
Movie - Mugen
Train



Demon Slayer:
Kimetsu no Yaiba



Demon Slayer:
Kimetsu no Yaiba -
Swordsmith
Village Arc



Kimetsu Gakuen:
Valentine-hen



Yashahime:
Princess Half-
Demon - The
Second Act



Introduction

Dataset
Overview

Data
understanding &
challenges

Data
Preparation

Model Building

Results & Model
Comparison

Model
Deployment

Conclusion

Conclusion

Merci de votre attention !