OPEN SOURCE SOFTWARE PROJECT

Movie Recommendation System using Python

TEAM:
Anurag Sati 21103153
Paakhi Maheshwari 21103149
Mufti Mohammad Usman 21103165

INTRODUCTION

In the digital age, the abundance of movie content makes it increasingly challenging for users to choose what to watch. Movie recommendation systems have become essential tools to help users discover movies that align with their preferences and interests. This project introduces a Movie Recommendation System implemented in Python and powered by Streamlit, a user-friendly web app framework. The system leverages content-based filtering techniques to provide personalized movie recommendations to users, enhancing their movie-watching experience.

ABSTRACT

The Movie Recommendation System using Streamlit and Python, built on content-based filtering, serves as a user-friendly and interactive platform for delivering movie recommendations tailored to individuals. This system focuses on analyzing movie metadata, such as genres, directors, actors, and other features, to suggest movies that align with a user's specified preferences.

Key features of this Movie Recommendation System include a clean and intuitive user interface created with Streamlit, allowing users to input their movie preferences and receive instant recommendations. Users can explore and discover movies that match their interests based on the content they enjoy the most.

The system is implemented using Python and leverages popular libraries and tools such as Pandas for data manipulation, Scikit-learn for machine learning, and Streamlit for web app development. Movie data is sourced from publicly available datasets and APIs, and recommendations are generated by comparing the content attributes of movies to the user's input preferences.

This project aims to provide a straightforward and customizable solution for movie enthusiasts seeking to explore a world of cinema with ease. With content-based filtering, users can discover movies that resonate with their unique tastes and preferences, enhancing their movie-watching experience through the Movie Recommendation System using Streamlit and Python.

DATASETS

```
movies=pd.read_csv("tmdb_5000_movies.csv")
credits=pd.read_csv("tmdb_5000_credits.csv")
```

These datasets have been imported from Kaggle for further processing and analysis for an accurate ML model.

https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata/data

The movies dataset contains the following details of a movie:

```
['budget', 'genres', 'homepage', 'id', 'keywords',
'original_language',
       'original_title', 'overview', 'popularity',
'production_companies',
       'production_countries', 'release_date',
'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline',
'title', 'vote_average',
       'vote_count']
```

The credits dataset contains the following details for a movie:

```
['movie_id', 'title', 'cast', 'crew']
```

Such details are useful in computing the similarity between a selected movie and the recommended movie.

DATA PROCESSING

Data is processed through Pandas and Numpy libraries to obtain the relevant data required to build an ML model for recommending movies.

Steps in data pre-processing:

1. ***Data Cleaning***
   Any duplicate records are removed and null values are dropped.

2. ***Feature Selection***
   We can observe that from the above information the important data for prediction is:

   ```
   ['movie_id','title','overview','genres','keywords',
   'cast','crew']
   ```

   So, we employ these columns only.

Steps in data transformation:

1. ***Text to list and vice-versa***
   Strings are converted to lists, concatenated then again converted to strings for easier and accurate processing.

2. ***Extracting key information***
   We extract genre names, keywords, the top three actors, and director names from relevant columns for easier recommendation.

3. ***Text Normalisation***
   Spaces are removed and all text is made lower case for uniformity.

4. ***Feature Combination***
   All relevant recommendation features are added to “tags” column for straightforward input in the recommendation system.

Steps in text vectorization:

1. ***Tokenisation***
   Breaking strings into individual words for processing

2. ***Stop Words Removal***
   Removing stop words from the data.

3. ***Limiting Features***
   Limiting top of 5000 words.

At the end of processing the data, we obtain a numerical matrix, where every row represents a movie, and every column is a word.

IMPLEMENTATION

1. ***Cosine Similarity Calculation***

   Cosine similarity is used to measure the similarity between movies based on their content. The cosine similarity matrix is calculated by comparing the vectors of each movie. A higher cosine similarity value indicates that two movies have similar content.

2. ***Movie Recommendation Algorithm***

   A recommendation function is developed to provide movie recommendations. The steps involved in this algorithm are as follows:

- The user inputs the title of a movie for which they want recommendations.
- The system finds the index of the input movie in the dataset.
- Cosine similarity values between the input movie and all other movies are retrieved.
- The values are sorted in descending order, and the top 5 movies with the highest similarity are selected as recommendations.

3. ***Recommendations and Serialization***

   To demonstrate the recommendation system, an example is provided where the movie "Pirates of the Caribbean: At World's End" is used as input, and the top 5 recommended movies are displayed. Additionally, the preprocessed movie data and similarity matrix are serialized using the pickle module for easy access and reuse in the future.

INTERFACE

The interface of the recommendation system is hosted through the Streamlit web-app development framework. This is a popular framework used to host ML models on the web. The code to build the interface is in the *app.py* file.

CODE

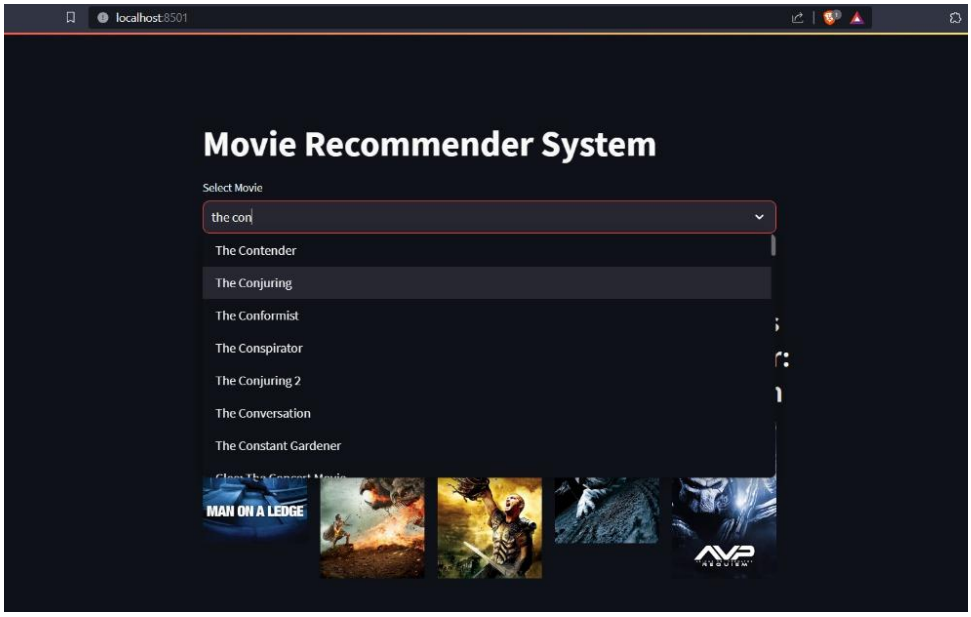
INTERFACE:

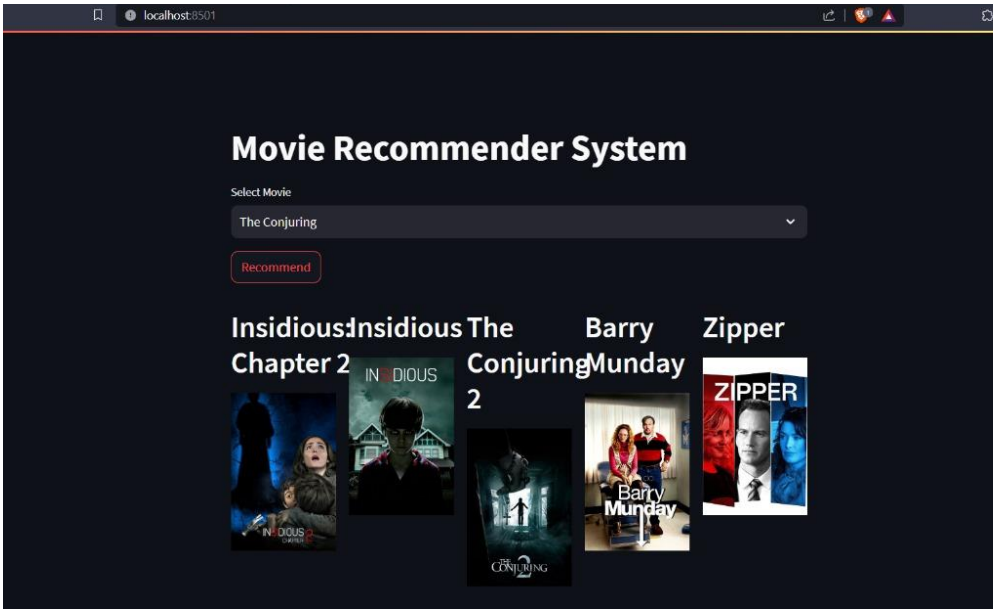
app.py

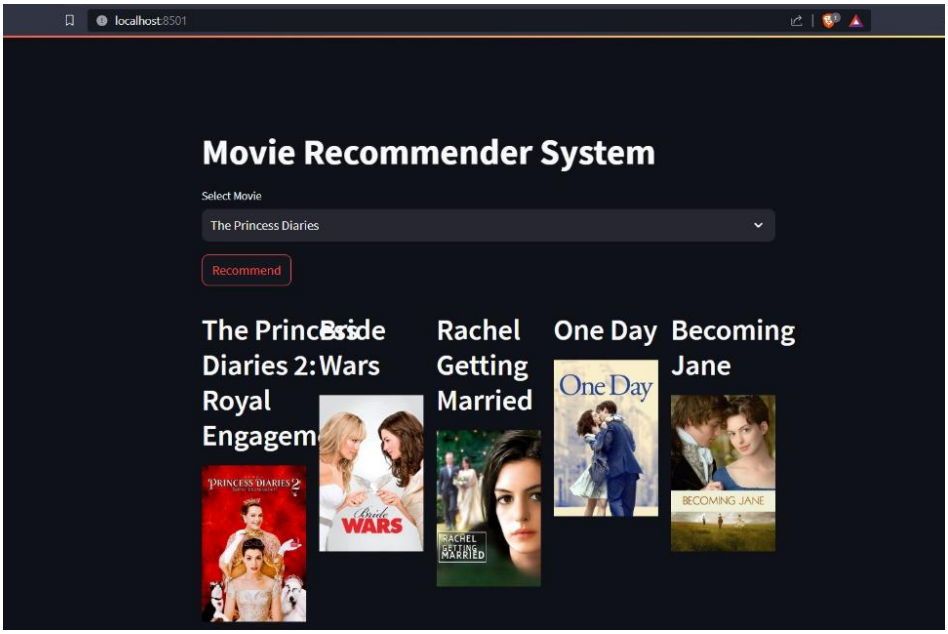
```
import streamlit as st

import pickle
import pandas as pd
import requests

movies_dict=pickle.load(open('/Users/paakhim10/Desktop/movie_recommender/movie_dict.pkl','rb'))
movies=pd.DataFrame(movies_dict)#dataframe with id,title,tags

similarity=pickle.load(open('/Users/paakhim10/Desktop/movie_recommender/similarity.pkl','rb'))

st.title('Movie Recommender System')

selected_movie_name=st.selectbox(
    'Select Movie',movies['title'].values
)

def fetch_poster(movie_id):
    data =
requests.get('https://api.themoviedb.org/3/movie/{}?api_key=9f0ac80516edd242622d9ea2223bb9a27&language=en-US'.format(movie_id))
    data=data.json()
    return "https://image.tmdb.org/t/p/w500/"+ data['poster_path']
def recommend(movie):
 movie_index=movies[movies['title'] ==movie].index[0]
 distances=similarity[movie_index]#gives a list of all distances for a movie

movies_list=sorted(list(enumerate(distances)),reverse=True,key=lambda x:x[1])[1:6]#since we want top 5 recommendations
```

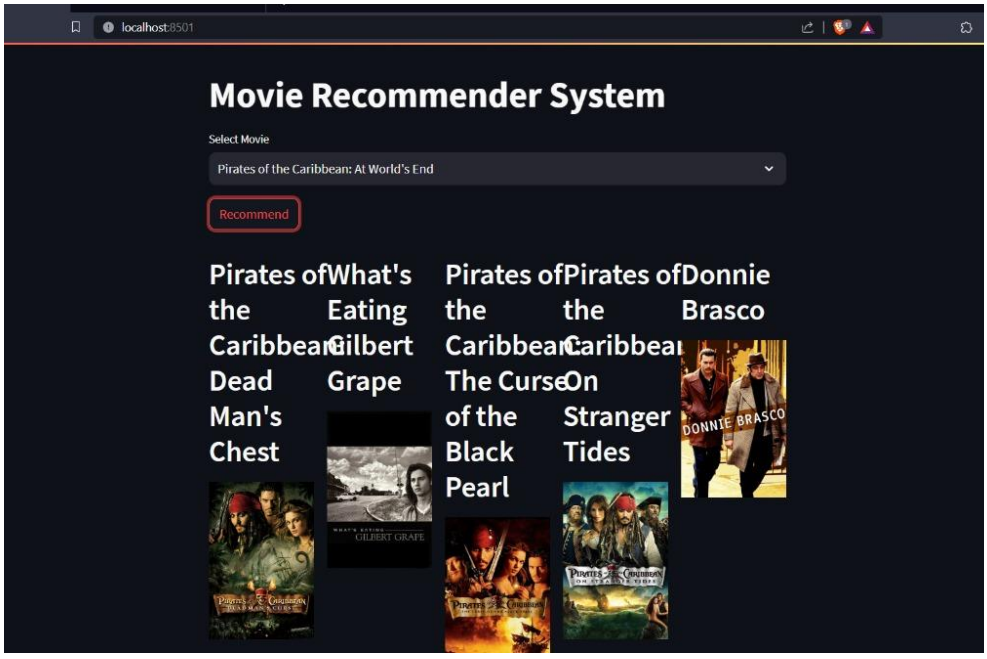
```
 m=[]
 p=[]
 #movies_list will return tuple with index of movie and dist corres
 for i in movies_list:
   m_id=movies.iloc[i[0]].movie_id
   #fetch poster from api
   p.append(fetch_poster(m_id))
   m.append((movies.iloc[i[0]].title))
   #print(i[0])
 return m,p


if st.button('Recommend'):
   names,posters=recommend(selected_movie_name)
   col1, col2, col3, col4, col5 = st.columns(5)
   with col1:
       st.header(names[0])
       st.image(posters[0])
   with col2:
       st.header(names[1])
       st.image(posters[1])


   with col3:
       st.header(names[2])
       st.image(posters[2])
   with col4:
       st.header(names[3])
       st.image(posters[3])
   with col5:
       st.header(names[4])
       st.image(posters[4])
```

OUTPUT


localhost:8501
Movie Recommender System
Select Movie
the con
The Contender
The Conjuring
The Conformist
The Conspirator
The Conjuring 2
The Conversation
The Constant Gardener
MAN ON A LEDGE


localhost:8501
Movie Recommender System
Select Movie
The Conjuring
Recommend
Insidious: Chapter 2
Insidious
The Conjuring 2
Barry Munday
Zipper
INSIDIOUS
ZIPPER
Barry Munday


localhost:8501
Movie Recommender System
Select Movie
The Princess Diaries
Recommend
The Princess Diaries 2: Royal Engagem
Bride Wars
Rachel Getting Married
One Day
Becoming Jane
PRINCESS DIARIES 2
Bride WARS
RACHEL GETTING MARRIED
One Day
BECOMING JANE


localhost:8501
Movie Recommender System
Select Movie
Pirates of the Caribbean: At World's End
Recommend
Pirates of the Caribbean Dead Man's Chest
What's Eating Gilbert Grape
Pirates of the Caribbean The Curse of the Black Pearl
Pirates of the Caribbean On Stranger Tides
Donnie Brasco
GILBERT GRAPE
DONNIE BRASCO

CONCLUSION

In this project, we have successfully developed a content-based movie recommendation system using natural language processing and cosine similarity. The system processes movie data to provide personalized movie recommendations to users based on content, genres, keywords, and cast/crew information. Here are the key takeaways from our project:

Data Preparation and Transformation: We began by cleaning and merging two datasets, "tmdb_5000_movies.csv" and "tmdb_5000_credits.csv." After refining the dataset to relevant columns, we applied data transformation techniques to extract essential information, including genres, keywords, actors, and directors. The 'tags' column was created by combining these elements, providing a holistic representation of each movie's content.

Text Processing and Vectorization: To enable similarity calculations, we used the CountVectorizer to transform the textual data into a numerical format. This process included tokenization, stop word removal, and limiting features to the top 5000 words. The resulting vectors represented each movie in a vector space.

Cosine Similarity Calculation: Cosine similarity was employed to measure the likeness between movies based on their content. A higher cosine similarity score indicated greater content similarity, which served as the foundation for our recommendation system.

Recommendation Algorithm: The recommendation algorithm was designed to accept a movie title as input and identify similar movies based on cosine similarity scores. The top 5 recommendations were presented to the user. This algorithm provides a straightforward yet effective means of suggesting movies that align with a user's preferences.