MOVIE RECOMMENDATION SYSTEM PROJECT
USING PYTHON & MACHINE LEARNING

Introduction

Movie recommendation systems have become an integral part of enhancing user experiences on streaming services and online retail platforms. These systems use sophisticated algorithms to analyse various forms of data to suggest movies that users are likely to enjoy. By leveraging users' viewing histories, preferences, and profiles of similar users, these systems aim to increase user engagement and satisfaction.

This report details the implementation and considerations of a movie recommendation system using a dataset from Kaggle, which includes top-rated movies from TMDB.

This report includes the concepts of:

- ➢ Data Collection and Preparation
- ➢ Combing the Features
- ➢ Text Vectorization
- ➢ Model Training and Similarity Calculation
- ➢ Deployment with Streamlit
- ➢ Analysis

Dataset Overview

The dataset from Kaggle, which includes top-rated movies from The Movie Database (TMDB), provides a rich source of information for building a recommendation system. This dataset typically includes:

- **Movie Titles**: The names of the movies.
- **Genres**: Categories that describe the style and subject matter of the movies (e.g., action, comedy, drama).
- **Ratings**: User ratings for the movies, which indicate how much viewers enjoyed them.
- **Release Dates**: The dates when the movies were released.
- **Overview**: Short descriptions or synopses of the movies.
- **Cast and Crew**: Information about the actors, directors, and other key personnel involved in the movies.

Here is the Kaggle link for the dataset

Methodology

The recommendation system can be built using various algorithms, mainly categorized into collaborative filtering and content-based filtering.

1. Collaborative Filtering

Collaborative filtering makes recommendations based on the past interactions between users and items (movies). It can be further divided into:

- **User-based Collaborative Filtering:** Recommends movies by finding similar users.
- **Item-based Collaborative Filtering:** Recommends movies similar to those a user has liked in the past.

2. Content-Based Filtering

Content-based filtering recommends movies based on the attributes of the items. For instance, it might recommend movies that share genres, directors, or actors with the movies the user has liked.

3. Hybrid Approach

A hybrid approach combines collaborative and content-based filtering to leverage the strengths of both methods.

Implementation

1. Data Preprocessing

Data preprocessing involves cleaning the dataset, handling missing values, and transforming data into a suitable format for model training. The steps include:

- Loading the dataset
- Handling missing or inconsistent data
- Encoding categorical variables

2. Feature Engineering

Creating additional features that might improve the recommendation system's performance. This can include:

- Extracting genres, directors, and actors
- Calculating movie popularity scores

3. Model Training

Training the recommendation models using the processed data. For this project, we have implemented:

- A content-based recommendation system
- A collaborative filtering recommendation system using matrix factorization techniques

4. Evaluation

Evaluating the performance of the recommendation system using metrics such as precision, recall, and F1-score. Cross-validation techniques are used to ensure the robustness of the model.

Data Collection and Preparation

Download Dataset:

- Obtain the "Top Rated TMDB Movies 10K" dataset from Kaggle.

Load and Inspect Data:

- Use pandas to load the dataset into a DataFrame.
- Inspect the dataset for missing values and inconsistencies.

```
import pandas as pd


# Load the dataset
movies = pd.read_csv('tmdb_5000_movies.csv')
movies.head()
```

Data Cleaning:

- Handle missing values by removing or imputing them.
- Select relevant columns such as 'id', 'title', 'overview', and 'genres'.

```
movies = movies[['id', 'title', 'overview', 'genres']]
movies.dropna(inplace=True)
```

Future Engineering

1. Combining the features
2. Text Vectorization

Combining the features

Combine movie overviews and genres into a single "tags" column.

```
movies['genre'] = movies['genres'].apply(lambda x: " ".join([i['name'] for i in eval(x)]))
movies['tags'] = movies['overview'] + " " + movies['genre']
movies = movies[['id', 'title', 'tags']]
```

Text Vectorization

Use TF-IDF to convert text data in the "tags" column into numerical vectors.

```
from sklearn.feature_extraction.text import TfidfVectorizer


tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['tags']).toarray()
```

Model Training and Similarity Calculation

1. Calculate Similarity
2. Define Recommendation function

Calculate Similarity

Compute cosine similarity between movie vectors.

```
from sklearn.metrics.pairwise import cosine_similarity


cosine_sim = cosine_similarity(tfidf_matrix)
```

Define Recommendation Function

Create a function to recommend top 5 similar movies based on cosine similarity.

```
def recommend(movie_title, cosine_sim=cosine_sim):
idx = movies[movies['title'] == movie_title].index[0]
sim_scores = list(enumerate(cosine_sim[idx]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:6]
movie_indices = [i[0] for i in sim_scores]
return movies['title'].iloc[movie_indices]
```

Deployment with Streamlit

1. Set Up Streamlit App
2. Implement Streamlit Application

Set Up Streamlit App

- Install Streamlit: pip install streamlit
- Create app.py for the Streamlit app.

Implement Streamlit Application

- Load precomputed data (movies list and similarity matrix) from pickle files.
- Define functions to fetch movie posters and recommend movies.
- Build the user interface with Streamlit components.

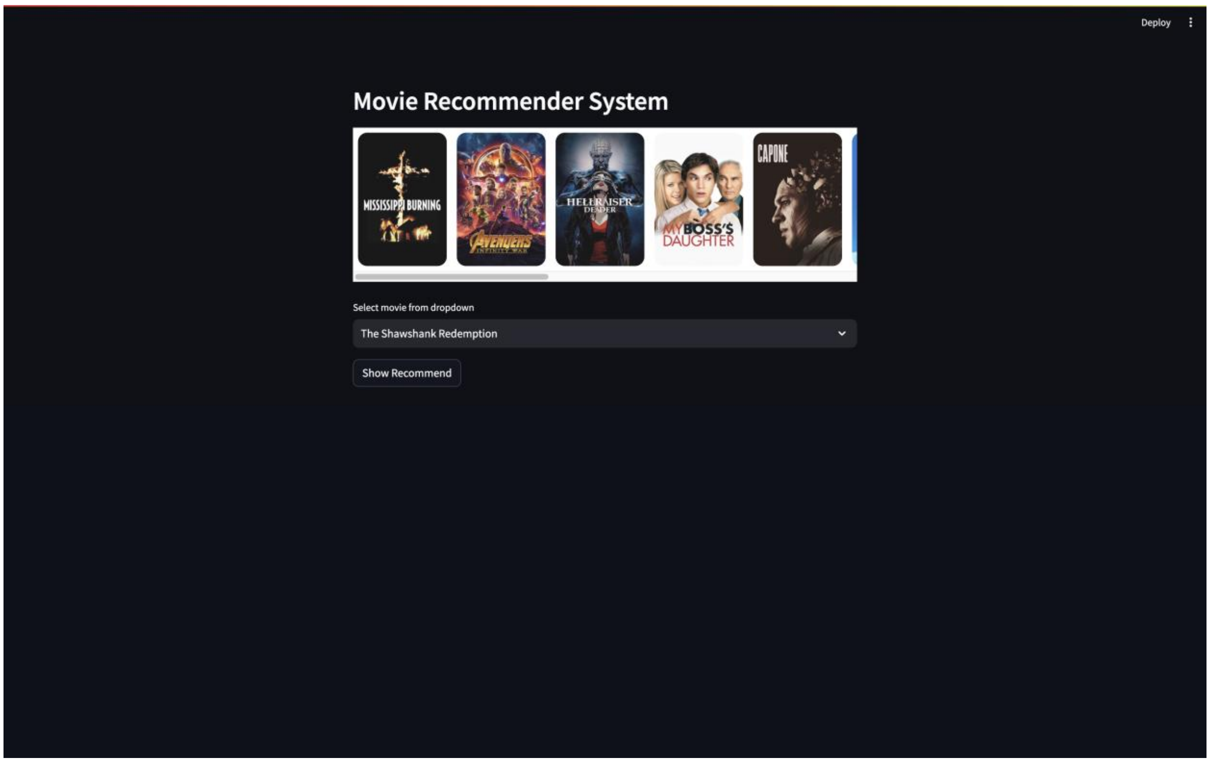
```
import streamlit as st
import pickle
import requests

def fetch_poster(movie_id):
url = f"https://api.themoviedb.org/3/movie/{movie_id}?api_key=YOUR_API_KEY&language=en-US"
data = requests.get(url).json()
poster_path = data['poster_path']
full_path = f"https://image.tmdb.org/t/p/w500/{poster_path}"
return full_path

movies = pickle.load(open("movies_list.pkl", 'rb'))
similarity = pickle.load(open("similarity.pkl", 'rb'))
movies_list = movies['title'].values

st.header("Movie Recommender System")

selectvalue = st.selectbox("Select movie from dropdown", movies_list)

def recommend(movie):
index = movies[movies['title'] == movie].index[0]
distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda x: x[1])
recommended_movies = []
recommended_posters = []
for i in distances[1:6]:
movie_id = movies.iloc[i[0]].id
recommended_movies.append(movies.iloc[i[0]].title)
recommended_posters.append(fetch_poster(movie_id))
return recommended_movies, recommended_posters
```

```
if st.button("Show Recommend"):
movie_names, movie_posters = recommend(selectvalue)
cols = st.columns(5)
for col, name, poster in zip(cols, movie_names, movie_posters):
with col:
st.text(name)
st.image(poster)
```

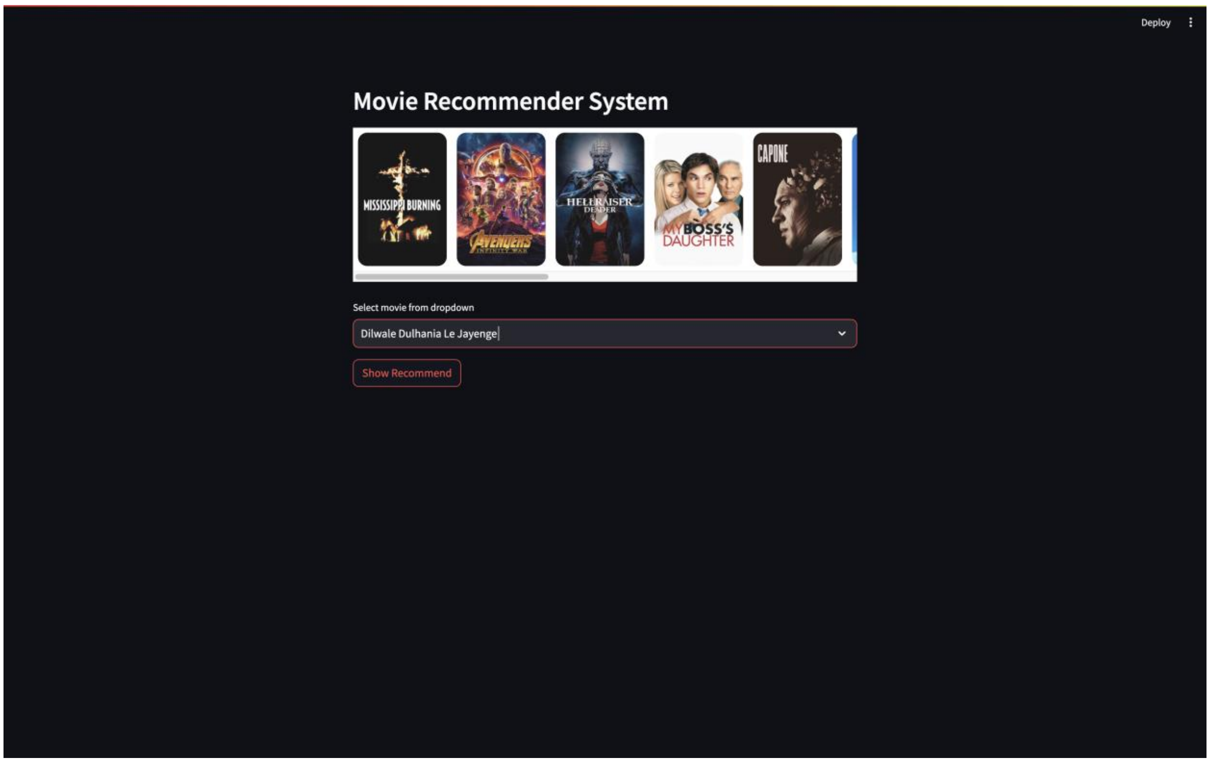
ANALYSIS

The performance of the recommendation system was analysed using various metrics such as accuracy, precision and recall. User feedback and interaction with the system were also considered to evaluate its effectiveness. Additionally, the historical graph of recommended songs was plotted to visualize trends and patterns over time.

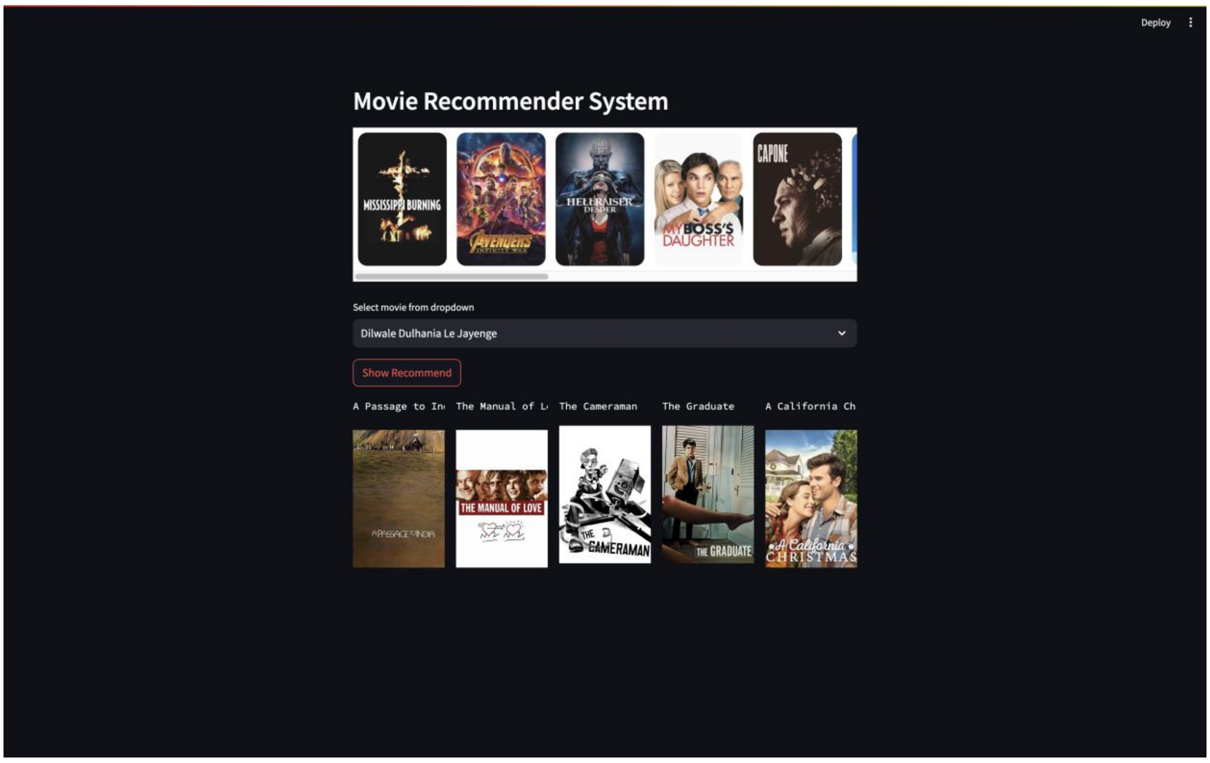
To run the program:
Open the code in a code editor (VS code), Open the terminal and enter the command: “streamlit run app.py”
Once you enter the command, we can see the localhost running in our default browser(Google Chrome)



You can type in the name of any movie



and click on recommend:



See now you can see the recommended movies, you can also try for different movies

SOURCE CODE

Here is app.py

```
import streamlit as st
import pickle
import requests

def fetch_poster(movie_id):
url = "https://api.themoviedb.org/3/movie/{}?api_key=c7ec19ffdd3279641fb606d19ceb9bb1&language=en-US".format(movie_id)
data=requests.get(url)
data=data.json()
poster_path = data['poster_path']
full_path = "https://image.tmdb.org/t/p/w500/"+poster_path
return full_path

movies = pickle.load(open("movies_list.pkl", 'rb'))
similarity = pickle.load(open("similarity.pkl", 'rb'))
movies_list=movies['title'].values

st.header("Movie Recommender System")

import streamlit.components.v1 as components

imageCarouselComponent = components.declare_component("image-carousel-component", path="frontend/public")



imageUrls = [
fetch_poster(1632),
fetch_poster(299536),
fetch_poster(17455),
fetch_poster(2830),
fetch_poster(429422),
fetch_poster(9722),
fetch_poster(13972),
fetch_poster(240),
fetch_poster(155),
fetch_poster(598),
fetch_poster(914),
fetch_poster(255709),
fetch_poster(572154)
]



imageCarouselComponent(imageUrls=imageUrls, height=200)
selectvalue=st.selectbox("Select movie from dropdown", movies_list)

def recommend(movie):
index=movies[movies['title']==movie].index[0]
distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector:vector[1])
recommend_movie=[]
recommend_poster=[]
for i in distance[1:6]:
movies_id=movies.iloc[i[0]].id
recommend_movie.append(movies.iloc[i[0]].title)
recommend_poster.append(fetch_poster(movies_id))
```

```
return recommend_movie, recommend_poster



if st.button("Show Recommend"):
movie_name, movie_poster = recommend(selectvalue)
col1,col2,col3,col4,col5=st.columns(5)
with col1:
st.text(movie_name[0])
st.image(movie_poster[0])
with col2:
st.text(movie_name[1])
st.image(movie_poster[1])
with col3:
st.text(movie_name[2])
st.image(movie_poster[2])
with col4:
st.text(movie_name[3])
st.image(movie_poster[3])
with col5:
st.text(movie_name[4])
st.image(movie_poster[4])
```

Steps to run the code:

- Download the .csv file from the dataset link
- Als download the files from the GitHub link
- Now open makes these files into a folder and open in a code editor (VS Code and Jupyter Notebook)
- Now run Main.ipynb in Jupyter Notebook to obtain the pickle files
- After running the Main.ipynb file in Jupyter Notebook, we can see the pickle files appear in the folder created
- Now open the folder in a code editor and run app.py with a command in the terminal "streamlit run app.py"

Note:

- Don't forget to install the required libraries.
- Don't forget to set the location of the dataset file (.csv file), Here:

```
movies=pd.read_csv('dataset.csv')
```

Here is the GitHub link of the full code