MACHINE LEARNING PROJECT

FILM RECOMMENDER SYSTEM

PROJECT REPORT

Sushree S Swain

5th Semester BTech 2023

CV Raman Global University

Odisha, India

Dated: 7th October 2023

CONTENTS

Introduction to Recommender Systems

According to Wikipedia, a recommender system, or a recommendation system (sometimes replacing 'system' with a synonym such as a platform or engine), is a subclass of information filtering systems that provide suggestions for items most pertinent to a particular user.

They are designed to help users discover relevant items or content from a lot of choices, such as movies, products, articles, music, or even potential friends on social media. These systems usually use machine learning algorithms to analyze user behavior, preferences, and historical interactions with the platform to make personalized recommendations.

There are various types of recommender systems, with the two main categories being:

1. **Content-Based Recommender Systems**: These systems recommend items based on their attributes and the user's historical preferences.
2. **Collaborative Filtering Recommender Systems**: Collaborative filtering techniques recommend items by identifying patterns and similarities in user behaviour. There are two subtypes within collaborative filtering:
    - **User-Based Collaborative Filtering**: Recommends items to a user based on the behaviour of similar users. If two users have liked similar items in the past, the system may suggest items liked by one user to the other.
    - **Item-Based Collaborative Filtering**: Recommends items to a user based on the similarity between items they have interacted with.

How do Recommender Systems Work?

In a Machine Learning environment, recommender systems usually work in the following structured manner:

Data Collection: Recommender systems begin by collecting data on users and items. We chose datasets that contain all the information that we require.

Data Preprocessing: The collected data often requires preprocessing to handle missing values, noisy data, and outliers.

Algorithm Selection: Recommender systems use various algorithms to make recommendations. We can choose Collaborating Filtering, Content Filtering, and even Deep Learning.

Scoring and Ranking: After selecting an algorithm, the recommender system assigns scores to items based on their likelihood of being of interest to the user.

Recommendation Generation: Finally, the recommender system generates a list of recommended items for the user.

There can also be additional steps like scaling data, measuring the accuracy of the model, or even further preprocessing.

Problem Statement

We are building a Machine Learning model that can prompt the user for the name of a film, the name of any member of the cast such as an actor or a director, and even a particular genre, and the model with suggest to the user similar films based on their response. This is a very rudimentary step when it comes to building a very large-scale model as of Netflix and Amazon Prime. Our model will simply work on film recommendations when we prompt the name of a film, actor, director, or genre, we are not creating a full-fledged web application with user profiles and history, though they can be added further in a given instance.

Platform and Modules Required

We will be using Jupyter Notebook for our Machine Learning Code and we will be using Python for the creation of a Streamlit virtual environment view.

We will be needing the following modules in Jupyter Notebook and Python:

- Pandas
- NumPy
- Sklearn
- Pickle

Streamlit (for creating an app)

We will be using the TMDB Database as our dataset as it contains information about approximately 5000 films.

Coding

We are using VSCode to run Jupyter Notebook and we have created a new file named as TMDBRecommender.

1. We will import the dataset by using NumPy and Pandas and specifying the path of the dataset stored on our local device.

2. The dataset has two files, one for the names of the films and other information and one for the names of the crew.

3. We will check our data for all the columns and other information such as the number of rows, and names of the columns if there are any missing values or any type of Alpha-numeric values.

4. We will join the two datasets, the film dataset and the crew dataset on the basis of the common column or the ID column.

5. After creating a single dataset out of the given two, we will select the columns that we have to remove in order to reduce the dimensionality of the data.

6. In our code, we have removed the columns that indicate the budget, revenue, production company, and original title of the film as they are not heavily required for a recommender.

7. We have removed certain rows that contain missing values as they were very small compared to the actual number of rows. (If the number of missing values would have been very high, it is not recommended to drop the rows.)

8. We will now try to create only 3 columns for our model, the film ID, name, and other information like the name of the actors, directors, genre, and overview.

9. We will now merge all columns other than film name and ID into one column and it has been named “tags” specifying any other tags for our film rather than the name and ID.

10. After creating only 3 columns, we will now combine all the names into one entity so that there will be no trouble for our model while searching for the names of any actor or director.

11. The next step is to convert all the text into lowercase.

12. All the information in the tag column about any particular film has been assigned its own array, so that grouping the information becomes easier for our model.

13. We will now convert all these arrays into vectors so that we can calculate the distance of all the films on the basis of similar words and genres.

14. The most frequently used words are used to help our model know if two films are similar or not.

15. We will create a function that finds the angle between all the vectors with each other and return an array.

16. Now finally we will create three functions, one is the recommender function on the basis of the name of a particular film, the other is the recommendations of a film based on the name of the actors or directors and the last function is to create a recommender function to suggest films on the basis of the genre.

17. The final step is to import the Pickle module and create a read binary file that can be saved in our local device.

18. Now we can go to PyCharm and we will import Streamlit to just create a virtual environment that can show how our recommender system works.

19. A drop-down menu and “show more recommendations” button have also been added.

20. This code can be further modified into any other modifications if required.

JUPYTER NOTEBOOK > TMDBRecommender.ipynb > pickle.dump(similarity,open('similarity.pkl','wb'))

+ Code + Markdown | Run All Restart Clear All Outputs | Variables Outline ··· Python 3.10.4

```
import numpy as np
import pandas as pd
```
[288] ✓ 0.0s Python

```
movies = pd.read_csv(r"c:\Users\Sushree S Swain\Desktop\Machine Learning\tmdb_5000_movies.csv")
credits = pd.read_csv(r"c:\Users\Sushree S Swain\Desktop\Machine Learning\tmdb_5000_credits.csv")
```
[289] ✓ 0.5s Python

```
movies.head(1)
```
[290] ✓ 0.0s Python

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_companies	production_countries	release_date	revenue
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ingenious Film Partners", "id": 289...	[{"iso_3166_1": "US", "name": "United States o...	2009-12-10	2787965087

```
credits.head(1)
```
[291] ✓ 0.0s Python

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...

```
#only keeping (genre,id,keywords,title,overview,cast,crew)
```
[297] ✓ 0.0s Python

```
movies = movies[['genres', 'id', 'keywords', 'title', 'overview', 'cast', 'crew']]
```
[298] ✓ 0.0s Python

```
movies.head(1)
```
[299] ✓ 0.0s Python

	genres	id	keywords	title	overview	cast	crew
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...

```
movies.isnull().sum()
```
[300] ✓ 0.0s Python

```
genres      0
id          0
keywords    0
title       0
overview    3
cast        0
crew        0
dtype: int64
```

```
movies.dropna(inplace= True)
```
[301] ✓ 0.0s Python

#only keeping (genre,id,keywords,title,overview,cast,crew)
[297] ✓ 0.0s Python

movies = movies[['genres', 'id', 'keywords', 'title', 'overview', 'cast', 'crew']]
[298] ✓ 0.0s Python

movies.head(1)
[299] ✓ 0.0s Python

	genres	id	keywords	title	overview	cast	crew
0	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...

movies.isnull().sum()
[300] ✓ 0.0s Python

```
genres      0
id          0
keywords    0
title       0
overview    3
cast        0
crew        0
dtype: int64
```

movies.dropna(inplace= True)
[301] ✓ 0.0s Python

```
movies.duplicated().sum()
```
[302] ✓ 0.1s Python

```
0
```

```
movies.iloc[0].genres
```
[303] ✓ 0.0s Python

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

```
import ast
def convert(obj):
    L = []
    for i in ast.literal_eval(obj):
        L.append(i['name'])
    return L
```
[304] ✓ 0.0s Python

```
movies['genres'] = movies['genres'].apply(convert)
```
[305] ✓ 0.1s Python

```
movies.head(1)
```
[306] ✓ 0.0s Python

	genres	id	keywords	title	overview	cast	crew
0	[Action, Adventure, Fantasy, Science Fiction]	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe4809251416c750aca23", "de...

```
movies['keywords'] = movies['keywords'].apply(convert)
```
[307] ✓ 0.3s Python

```
movies.head(1)
```
[308] ✓ 0.0s Python

	genres	id	keywords	title	overview	cast	crew
0	[Action, Adventure, Fantasy, Science Fiction]	19995	[culture clash, future, space war, space colon...	Avatar	In the 22nd century, a paraplegic Marine is di...	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe4809251416c750aca23", "de...

```
import ast
def convert3(obj):
    L = []
    counter = 0
    for i in ast.literal_eval(obj):
        if counter != 3:
            L.append(i['name'])
            counter+=1
        else:
            break
    return L
```
[309] ✓ 0.0s Python

```
movies['cast']=movies['cast'].apply(convert3)
```
[310] ✓ 3.1s Python

```
import ast
def fetch_director(obj):
    L = []
    for i in ast.literal_eval(obj):
        if i['job']=='Director':
            L.append(i['name'])
            break
    return L
```
[312] ✓ 0.0s

```
movies['crew']=movies['crew'].apply(fetch_director)
```
[313] ✓ 2.6s

```
movies.head(1)
```
[314] ✓ 0.0s

```
[317] movies['genres']=movies['genres'].apply(lambda x:[i.replace(" ","") for i in x])
      ✓ 0.0s

[318] movies['cast']=movies['cast'].apply(lambda x:[i.replace(" ","") for i in x])
      ✓ 0.0s

[319] movies['crew']=movies['crew'].apply(lambda x:[i.replace(" ","") for i in x])
      ✓ 0.0s

[320] movies['keywords']=movies['keywords'].apply(lambda x:[i.replace(" ","") for i in x])
      ✓ 0.0s

[321] movies.head(1)
      ✓ 0.0s
```

	genres	id	keywords	title	over
0	[Action, Adventure, Fantasy, ScienceFiction]	19995	[cultureclash, future, spacewar, spacecolony, ...	Avatar	[In, the, 22nd, century,, a, parap M

```
[322] movies['tags'] = movies['overview'] + movies['genres'] + movies['cast'] + movies['keywords'] + movies['crew']
      ✓ 0.2s

[324] new_df = movies[['id','title','tags']]
      ✓ 0.0s

[325] new_df.head(1)
      ✓ 0.0s
```

	id	title	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...

```
[326] new_df['tags']=new_df['tags'].apply(lambda x:" ".join(x))
      ✓ 0.0s

      C:\Users\Sushree S Swain\AppData\Local\Temp\ipykernel_4900\487797088.py:1: SettingWithCopyWarning:
      A value is trying to be set on a copy of a slice from a DataFrame.
      Try using .loc[row_indexer,col_indexer] = value instead

      See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#r
        new_df['tags']=new_df['tags'].apply(lambda x:" ".join(x))

[327] new_df.head(1)
      ✓ 0.0s
```

	id	title	tags
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...

```
      new_df['tags'][0]
```

```
[329]  new_df['tags']=new_df['tags'].apply(lambda x:x.lower())
       ✓ 0.0s
```

```
C:\Users\Sushree S Swain\AppData\Local\Temp\ipykernel_4900\4224080999.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags']=new_df['tags'].apply(lambda x:x.lower())
```

```
[330]  new_df.head(1)
       ✓ 0.0s
```

	id	title	tags
0	19995	Avatar	in the 22nd century, a paraplegic marine is di...

```
[331]  from sklearn.feature_extraction.text import CountVectorizer
       cv = CountVectorizer(max_features=5000,stop_words='english')
       ✓ 0.0s
```

```
[332]  vectors = cv.fit_transform(new_df['tags']).toarray()
       ✓ 0.4s
```

```
       vectors[0]
```

```
[339]  new_df['tags']= new_df['tags'].apply(stem)
       ✓ 3.6s
```

```
C:\Users\Sushree S Swain\AppData\Local\Temp\ipykernel_4900\3021978705.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  new_df['tags']= new_df['tags'].apply(stem)
```

+ Code + Markdown

```
[340]  from sklearn.metrics.pairwise import cosine_similarity
       cosine_similarity(vectors)
       ✓ 2.3s
```

```
array([[1.        , 0.08964215, 0.05976143, ..., 0.02519763, 0.02817181,
        0.        ],
       [0.08964215, 1.        , 0.0625    , ..., 0.02635231, 0.        ,
        0.        ],
       [0.05976143, 0.0625    , 1.        , ..., 0.02635231, 0.        ,
        0.        ],
       ...,
       [0.02519763, 0.02635231, 0.02635231, ..., 1.        , 0.0745356 ,
        0.04836508],
       [0.02817181, 0.        , 0.        , ..., 0.0745356 , 1.        ,
        0.05407381],
       [0.        , 0.        , 0.        , ..., 0.04836508, 0.05407381,
        1.        ]])
```

```
[341]  similarity = cosine_similarity(vectors)
       ✓ 1.5s
```

```
sorted(list(enumerate(similarity[0])),reverse=True,key=lambda x:x[1])[1:6]
```

[342] ✓ 0.0s

```
[(539, 0.26089696604360174),
 (1194, 0.2581988897471611),
 (507, 0.25302403842552984),
 (260, 0.25110592822973776),
 (1216, 0.24944382578492943)]
```

+ Code + Markdown

```
def recommend(movie):
    movie_index = new_df[new_df['title'] == movie].index[0]
    distances = similarity[movie_index]
    movie_scores = [(i, score) for i, score in enumerate(distances) if i != movie_index]
    movie_scores.sort(key=lambda x: x[1], reverse=True)
    top_recommendations = movie_scores[:5]
    for i, score in top_recommendations:
        print(new_df.iloc[i].title)
```

[347] ✓ 0.0s

```
recommend('Superman')
```

[349] ✓ 0.0s

```
Superman II
Superman Returns
Superman III
Superman IV: The Quest for Peace
Man of Steel
```

```
def recommend_by_crew(name):
    relevant_movies = new_df[new_df['tags'].str.contains(name, case=False)]

    if relevant_movies.empty:
        print(f"No movies found for {name}.")
        return

    movie_indices = relevant_movies.index
    similarity_scores = similarity[movie_indices].sum(axis=0)
    movie_scores = [(i, score) for i, score in enumerate(similarity_scores)]
    movie_scores.sort(key=lambda x: x[1], reverse=True)
    top_recommendations = movie_scores[:5]

    for i, score in top_recommendations:
        print(new_df.iloc[i].title)
```

[97] ✓ 0.0s

```
recommend_by_crew('AlPacino')
```

[98] ✓ 0.0s

```
Donnie Brasco
City Hall
People I Know
Sea of Love
The Godfather: Part III
```

```python
def recommend_by_genre(name):
    relevant_movies = new_df[new_df['tags'].str.contains(name, case=False)]

    if relevant_movies.empty:
        print(f"No movies found for {name}.")
        return

    movie_indices = relevant_movies.index
    similarity_scores = similarity[movie_indices].sum(axis=0)
    movie_scores = [(i, score) for i, score in enumerate(similarity_scores)]
    movie_scores.sort(key=lambda x: x[1], reverse=True)
    top_recommendations = movie_scores[:3]

    for i, score in top_recommendations:
        print(new_df.iloc[i].title)
```

[106] ✓ 0.0s

```python
recommend_by_genre('Comedy')
```

[109] ✓ 0.0s

```
My Big Fat Greek Wedding 2
Four Single Fathers
Boynton Beach Club
```

PyCharm Code

```python
import streamlit as st
import pickle
import pandas as pd

# Load movie data and similarity matrix
movies_list = pickle.load(open("movies.pkl", 'rb'))
movies_list = movies_list['title'].values
similarity = pickle.load(open('similarity.pkl', 'rb'))

# Define the background image URL
background_image = "https://media.istockphoto.com/id/1336937059/photo/film-reels-on-black-background-movie-video-and-cinema-prodaction-and-edition-concept.jpg?

# Add custom CSS to set the background image and style
st.markdown(
    f"""
    <style>
        body {{
            background: url("{background_image}");
            background-size: cover;
            font-family: 'Times New Roman', serif; /* Change the font to Times New Roman */
            color: #ffffff; /* Text color */
        }}
        .reportview-container .main .block-container {{
            max-width: 800px; /* Adjust the maximum width of content */
            padding: 20px; /* Add some padding */
            border-radius: 10px; /* Add rounded corners */
            background-color: rgba(0, 0, 0, 0.7); /* Add a semi-transparent background */
        }}
        .center-align {{
```

```
            text-align: center; /* Center align the text */
        }}
        .uppercase-title {{
            text-transform: uppercase; /* Make the text uppercase */
            font-size: 40px; /* Increase the title font size */
            font-weight: bold; /* Make the title bold */
        }}
        .faded-line {{
            border-top: 1px dashed rgba(255, 255, 255, 0.5); /* Add a faded dashed line */
            margin-top: 10px; /* Adjust the margin for separation */
            margin-bottom: 10px;
        }}
    </style>
    """,
    unsafe_allow_html=True,
)

# Title with custom styling and center alignment
st.markdown("<p class='center-align uppercase-title'>FILM RECOMMENDER SYSTEM</p>",
            unsafe_allow_html=True)  # Center-aligned title

# Subtitle
st.markdown("<p class='center-align' style='font-size: 30px;'>Explore the world of cinema</p>", unsafe_allow_html=True)

# Create a search box with a dropdown menu
selected_movie = st.selectbox("Search for a movie:", movies_list)

# Display the selected movie
if selected_movie:
    st.write("You selected:", selected_movie)
```

```
# Function to recommend similar films
def recommend(movie, num_recommendations=5):
    movie_index = list(movies_list).index(movie)
    distances = similarity[movie_index]
    movie_scores = [(i, score) for i, score in enumerate(distances) if i != movie_index]
    movie_scores.sort(key=lambda x: x[1], reverse=True)
    recommended_films = [movies_list[i[0]] for i in movie_scores[:num_recommendations]]
    return recommended_films


# Display the first 5 recommendations without a toggle button
num_recommendations = 5
recommended_films = recommend(selected_movie, num_recommendations)

if recommended_films:
    st.markdown(f"<p class='center-align uppercase-title'>{num_recommendations} RECOMMENDED FILMS</p>",
                unsafe_allow_html=True)
    for film in recommended_films:
        st.write(film)

# Create a faded line after the initial recommendations
st.markdown("<hr class='faded-line'>", unsafe_allow_html=True)

# Create a button to show more recommendations
show_more_recommendations = st.button("Show More Recommendations")

# Display additional recommendations when the button is clicked
if show_more_recommendations:
    num_more_recommendations = num_recommendations + 10
    additional_recommendations = recommend(selected_movie, num_more_recommendations)
```

```python
# Create a faded line after the initial recommendations
st.markdown("<hr class='faded-line'>", unsafe_allow_html=True)

# Create a button to show more recommendations
show_more_recommendations = st.button("Show More Recommendations")

# Display additional recommendations when the button is clicked
if show_more_recommendations:
    num_more_recommendations = num_recommendations + 10
    additional_recommendations = recommend(selected_movie, num_more_recommendations)

    st.markdown(f"<p class='center-align uppercase-title'>10 MORE RECOMMENDATIONS</p>", unsafe_allow_html=True)
    for film in additional_recommendations[num_recommendations:num_more_recommendations]:
        st.write(film)
```

Procfile | setup.sh | .gitignore | requirements.txt | requirements1.txt | requirements2.txt | App.py

```
venv
```

Procfile | setup.sh | .gitignore | requirements.txt | requirements1.txt | requirements2.txt | App.py

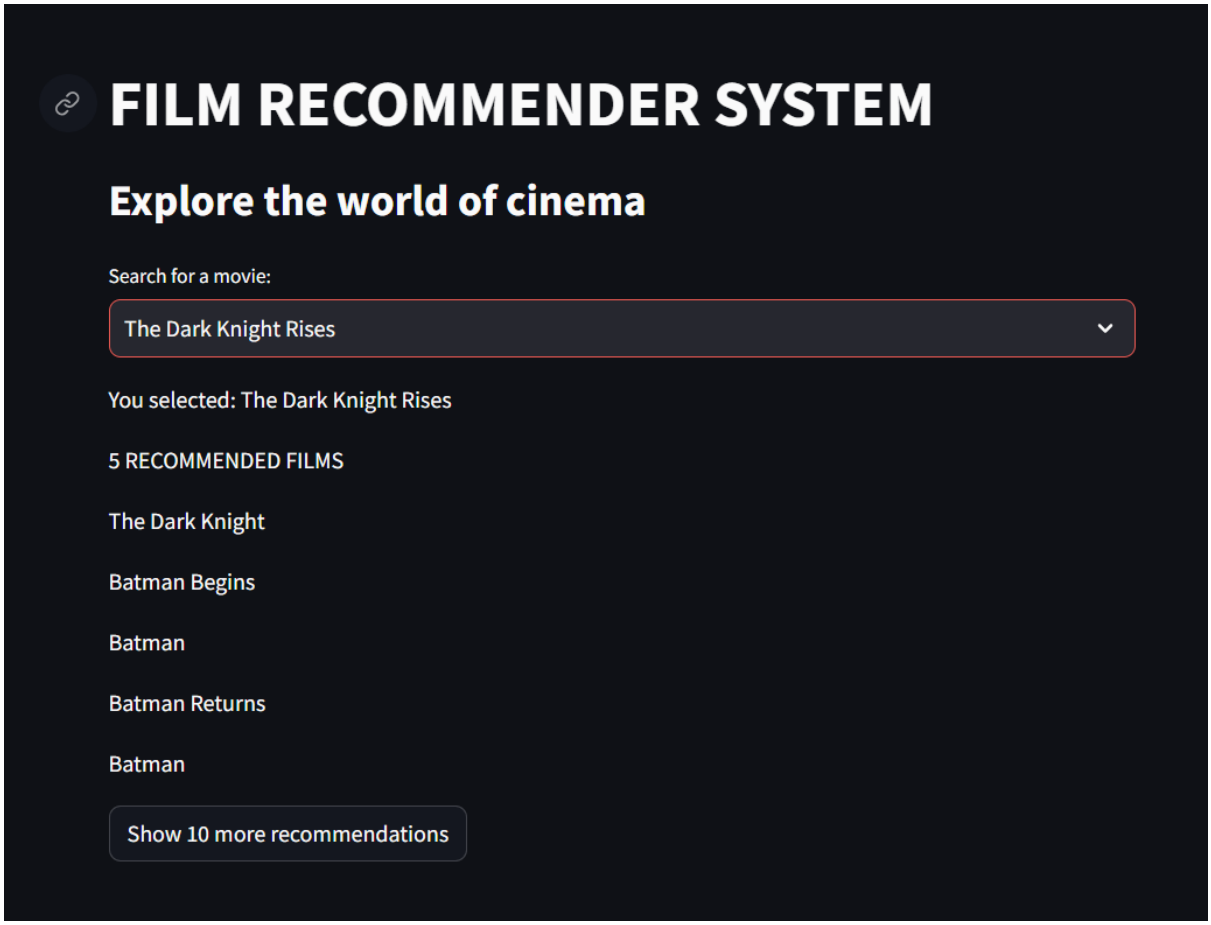
```
web: sh setup.sh && streamlit run app.py
```

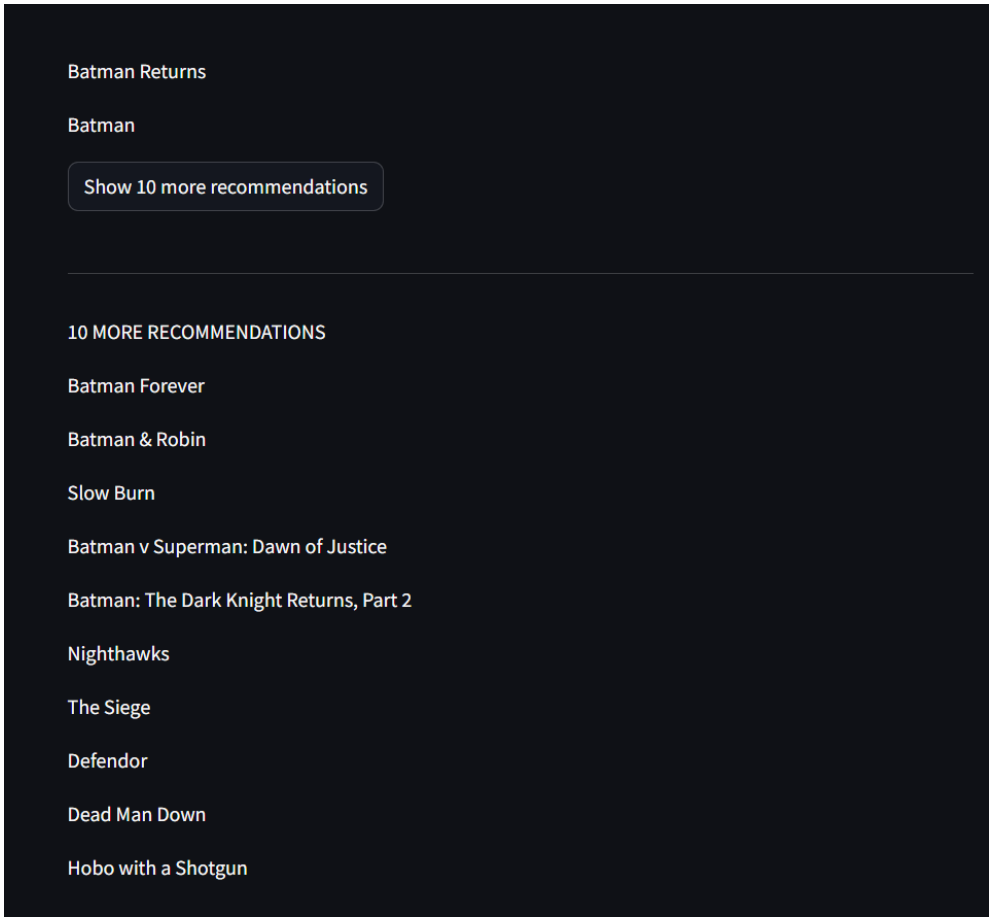
Procfile | setup.sh | .gitignore | requirements.txt | requirements1.txt | requirements2.txt | App.py

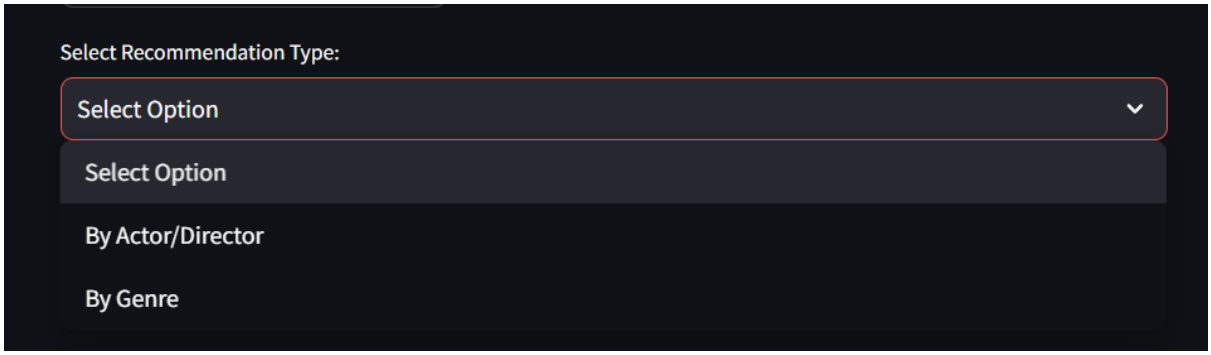
```
mkdir -p ~/.streamlit/

echo "\
[server]\n\
port = $PORT\n\
enableCORS = false\n\
headless = true\n\
" > ~/.streamlit/config.toml
```

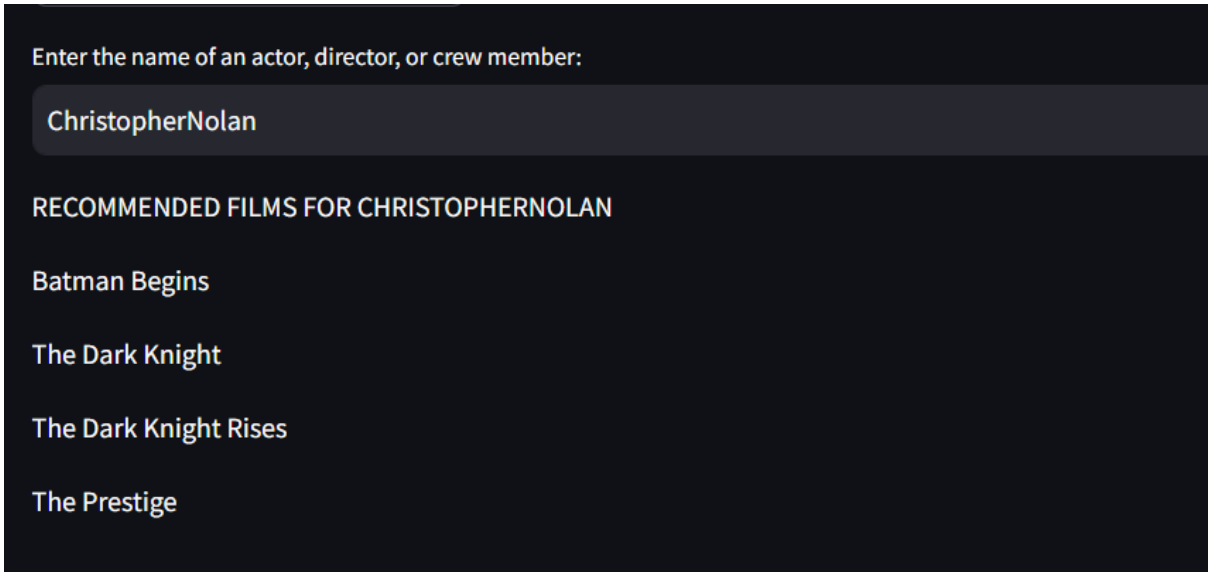
Terminal: Local | Command Prompt

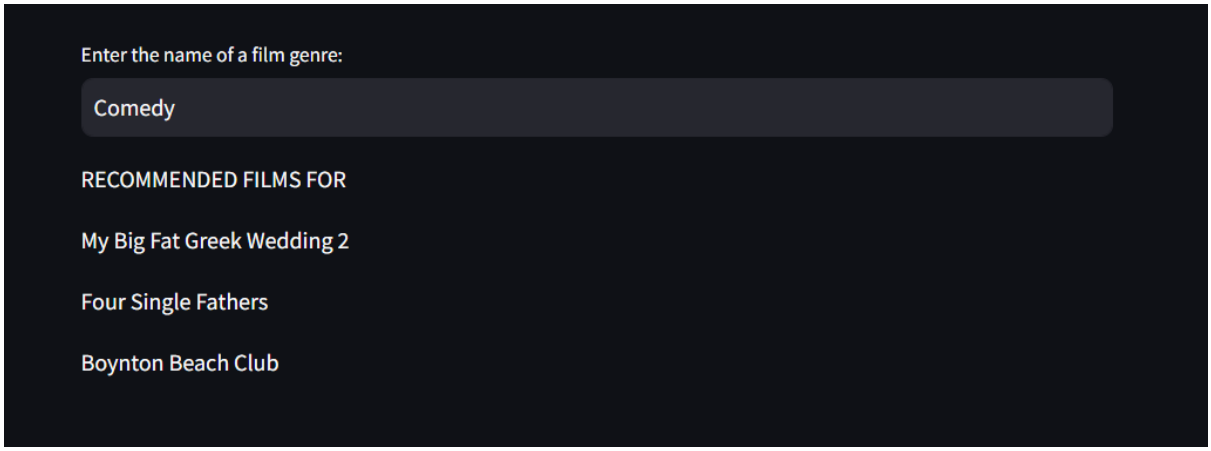
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Sushree S Swain\PycharmProjects\FilmRecommenderSystem> streamlit run app.py
```


FILM RECOMMENDER SYSTEM
Explore the world of cinema
Search for a movie:
The Dark Knight Rises
You selected: The Dark Knight Rises
5 RECOMMENDED FILMS
The Dark Knight
Batman Begins
Batman
Batman Returns
Batman
Show 10 more recommendations


Batman Returns
Batman
Show 10 more recommendations
10 MORE RECOMMENDATIONS
Batman Forever
Batman & Robin
Slow Burn
Batman v Superman: Dawn of Justice
Batman: The Dark Knight Returns, Part 2
Nighthawks
The Siege
Defendor
Dead Man Down
Hobo with a Shotgun


Select Recommendation Type:
Select Option
Select Option
By Actor/Director
By Genre


Enter the name of an actor, director, or crew member:
ChristopherNolan
RECOMMENDED FILMS FOR CHRISTOPHERNOLAN
Batman Begins
The Dark Knight
The Dark Knight Rises
The Prestige


Enter the name of a film genre:
Comedy
RECOMMENDED FILMS FOR
My Big Fat Greek Wedding 2
Four Single Fathers
Boynton Beach Club

Advantages & Limitations of the Software

This software is created using Jupyter Notebook and we have hardly used four to five modules. The code is very short and simple and it gives such a beautiful result.

The limitations of this software are that it cannot recommend films based on user ratings or IMDb ratings which many users might prefer. This software is very simple and hasn't been connected to the internet so we cannot access any other data, plus it cannot store user information and history.

Conclusion

Recommender systems can be simple yet complex to build, depending on what we need. We can create recommender systems for almost all services, this was an example of a Film recommendation system. These systems help the user spend less time browsing and we can help them get their interests resolved easier which in turn also keeps them happy as we provide an excellent user experience for them.

Bibliography & References

http://www.google.com/
https://www.wikipedia.org/
http://github.com/
https://stackoverflow.com/

Find these project files on
https://github.com/SushreeSwain/TMDBRecommender