Information Retrieval and Web Analytics - IT3041

Final Report

Group Name: Beginner Bugs

Group Members:

Name	Registration Number
B.D.A.D.Hettiarachchi	IT21029868
U.D.K.Navaratne	IT20708276
A.N.Elvitigala	IT21088582
V.L.K.Tennekoon	IT21015212

Table of Contents

Table of Figures

1. Introduction

In this report, we will do a brief discussion about the 'Movie Bug', movie recommendation engine.

A movie-based recommender system is a software tool that suggests movies to users based on their personal preferences. It uses algorithms & machine learning techniques to analyze data points, such as a user's previous movie choices & ratings, to generate personalized recommendations.

According to our research, currently there are 6 main types of recommendation systems available in the technical world, including collaborative, content-based, demographic, utility-based, knowledge-based and hybrid recommender systems. Out of the above-mentioned system types, for this project we have used a content-based recommender system for 'Genre' based recommendation, and a collaborative recommender system to recommend movies from the user's previously rated movies.

A content-based recommender system is a type of recommendation system that suggests items (e.g., movies, products, articles) to users based on the content or features of those items and the user's past preferences or profile, while a collaborative recommender system is a recommender system that operates based on the interactions and behaviors of a group of users. Collaborative system can be either user-based or item-based. User-based collaborative filtering recommends items to a user based on the preferences and behaviors of other similar users. Item-based collaborative filtering recommends items to a user based on the similarity of items they have previously interacted with.

2. About the dataset

We obtained this dataset from the Kaggle website. We have provided the link for the dataset here: [MovieLense Dataset]

This dataset consists of two data frames named 'Movies' and 'Ratings'. The 'Movies' data frame contains the details about the movies while the 'Ratings' data frame contains different user's ratings for different movies.

The 'Movies' data frame has 10329 unique values and 3 columns [movieId, title, genre] and

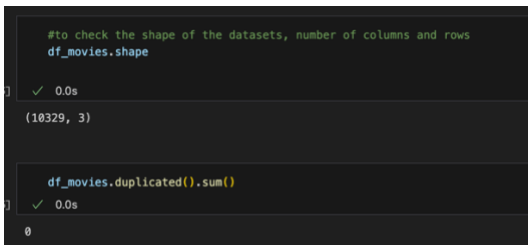

Figure 1: Shape of the 'Movies' data frame

	movieId	title	genres
0	1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji	Adventure|Children|Fantasy
2	3	Grumpier Old Men	Comedy|Romance
3	4	Waiting to Exhale	Comedy|Drama|Romance
4	5	Father of the Bride Part II	Comedy

Figure 2: 'Movies' data frame

'Ratings' data frame has 105344 unique values and 4 columns [userId, movieId, rating, timestamp]. The average rating is 3.5 and minimum and maximum rating is 0.5 and 5 respectively.

Figure 4: Shape of the 'Rating' data frame

	userId	movieId	rating	timestamp	title	genres
0	1	16	4.0	1217897793	Casino	Crime Drama
1	9	16	4.0	842686699	Casino	Crime Drama
2	12	16	1.5	1144396284	Casino	Crime Drama
3	24	16	4.0	963468757	Casino	Crime Drama
4	29	16	3.0	836820223	Casino	Crime Drama

Figure 3: 'Rating' data frame

3. Data Preprocessing

In the 'title' column of the 'movies' data frame it has mentioned the movie's release year as well. We did not consider the year; therefore, we have removed the year from the movie title.

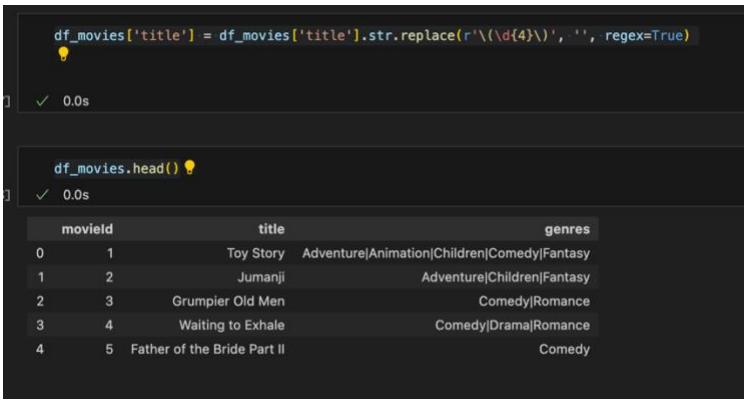

Figure 5: Removal of unnecessary data from the movie title

As you can see in the above picture, the 'genre' column, the different genres get separated from a symbol '|', to get a string of keywords, we have removed that symbol as well.

Figure 6: Removal of unnecessary data from the 'genre' column

To avoid the model complexity, we have merged the two data frames into one data frame named ‘ movie_data’ based on the ‘movieId’ column.

```
# Merge the ratings and movies data
movie_data = pd.merge(df_ratings, df_movies, on='movieId')

#merging had done on the basis of movieId
```

✓ 0.0s

```
movie_data.head()
```

✓ 0.0s

	userId	movieId	rating	timestamp	title	genres
0	1	16	4.0	1217897793	Casino	Crime Drama
1	9	16	4.0	842686699	Casino	Crime Drama
2	12	16	1.5	1144396284	Casino	Crime Drama
3	24	16	4.0	963468757	Casino	Crime Drama
4	29	16	3.0	836820223	Casino	Crime Drama

Figure 7: New merged data frame, 'movie_data'

Data types of the columns in the ‘movie_data’ data frame;

```
result= movie_data.dtypes
print(result)
```

✓ 0.0s

```
userId        int64
movieId       int64
rating      float64
timestamp     int64
title        object
genres       object
dtype: object
```

Figure 8: Data types of each column

In this data frame, there is a column named ‘timestamp’ which will not be used in our next few steps. Therefore, we have dropped this column from our data frame.

```
movie_data = movie_data.drop('timestamp', axis=1)
```

✓ 0.0s

```
movie_data.head()
```

✓ 0.0s

	userId	movieId	rating	title	genres
0	1	16	4.0	Casino	Crime Drama
1	9	16	4.0	Casino	Crime Drama
2	12	16	1.5	Casino	Crime Drama
3	24	16	4.0	Casino	Crime Drama
4	29	16	3.0	Casino	Crime Drama

Figure 9: Removal of 'timestamp' column

We have also used the movie ratings column for our recommendations. In order to use that we have done some additional preprocessing,
We have created a pivot table to fill the movies that a user hasn't rated earlier.

```
# Create a user-movie rating pivot table
user_movie_rating = movie_data.pivot_table(index='userId', columns='title', values='rating')

# Fill missing values with zeros (unrated movies)
user_movie_rating = user_movie_rating.fillna(0)

user_movie_rating.head()
```

✓ 0.1s

title	'71	'Hellboy': The Seeds of Creation	'Round Midnight	'Til There Was You	'burbs, The	'night Mother	(500) Days of Summer	*batteries not included	...And Justice for All	10	...	[REC]	[REC]²	[REC]³ 3 Génesis	a/k/a Tommy Chong	eXistenZ
userId																
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0

5 rows × 10007 columns

Figure 10: Pivot Table Creation

Standardizing user ratings to have zero mean and unit variance, using the StandardScalar function we have brought the rating values to a similar level.

```
# Standardize user ratings to have zero mean and unit variance
scaler = StandardScaler()
user_movie_rating_scaled = scaler.fit_transform(user_movie_rating)
print(user_movie_rating_scaled)
```

✓ 0.1s

```
[[-0.03872015 -0.03872015 -0.03872015 ... -0.09358784 -0.23547702
  -0.03872015]
 [-0.03872015 -0.03872015 -0.03872015 ... -0.09358784 -0.23547702
  -0.03872015]
 [-0.03872015 -0.03872015 -0.03872015 ... -0.09358784 -0.23547702
  -0.03872015]
 ...
 [-0.03872015 -0.03872015 -0.03872015 ... -0.09358784 -0.23547702
  -0.03872015]
 [-0.03872015 -0.03872015 -0.03872015 ... -0.09358784 -0.23547702
  -0.03872015]
 [-0.03872015 -0.03872015 25.82634314 ... -0.09358784  2.37528892
  25.82634314]]
```

Figure 11: Standardizing Ratings Feature

With the above step, we have completed the data preprocessing and now our merged dataset is ready to be used in the development of movie recommendation engine.

4. Development of Movie Recommendation Engine

To find the similarity between the ratings we have used the cosine similarity method, to find the most popular movies in our dataset, then we can suggest them to our users.

```
# Calculate the similarity matrix using cosine similarity
cosine_sim = cosine_similarity(user_movie_rating_scaled)
```

Figure 12: Finding the cosine similarity between the ratings

```
58  # Create a TF-IDF vectorizer for movie descriptions
59  tfidf_vectorizer = TfidfVectorizer(stop_words=stopwords.words('english'))
60  df_movies['genres'] = df_movies['genres'].fillna('')  # Fill missing genres
61  tfidf_matrix = tfidf_vectorizer.fit_transform(df_movies['genres'])
62
```

Figure 13: Creating TF-IDF Vector for the 'Genre' column

In the code snippet, we utilize the TF-IDF vectorization technique, available in the scikit-learn library, to transform a collection of raw movie genre descriptions into a structured numerical format. To enhance the quality of the TF-IDF representation, we have included a preprocessing step to eliminate common English stop words (e.g., "a," "the," "and"). We accomplish this by employing a list of English stop words from the Natural Language Toolkit (NLTK) library. It's essential to ensure data completeness. Therefore, we handle any potential missing values (NaN) in the 'genres' column of our movie dataset ('df_movies') by replacing them with empty strings. This action guarantees that no gaps exist in the 'genres' information.

Following data preprocessing, we proceed to convert the 'genres' column in our movie dataset ('df_movies') into a TF-IDF matrix using the 'tfidf_vectorizer' that we configured earlier. The TF-IDF (Term Frequency-Inverse Document Frequency) values are numerical representations that convey the importance of each term within a movie's genre description relative to the entire dataset.

Higher TF-IDF values are assigned to terms that are distinctive to a particular movie's genre, while common terms shared across genres receive lower TF-IDF weights.
We have used the above Tf-idf matrix to compute the cosine similarity between each movie in the df_movies Data Frame. Then the similarities are sorted and the top 5 movies with the highest cosine similarity are returned from the above function.

When displaying the recommended movies to the users, we had provided them a movie poster as well, to get the movie poster, we had created another function as below. For that we have used the libraries below as well.

- Tmdbv3api library

```
# TMDb setup
tmdb = TMDb()
tmdb.api_key = '8265bd1679663a7ea12ac168da84d2e8&language=en-US'  # Replace with your TMDb API key
# Function to get movie poster from TMDb
def get_movie_poster(movie_title):
    movie = Movie()
    search = movie.search(movie_title)
    if search:
        movie_id = search[0]['id']  # Assuming the first search result is the correct movie
        movie_info = movie.details(movie_id)
        poster_path = movie_info['poster_path']
        return f"https://image.tmdb.org/t/p/w500{poster_path}"
    else:
        return None
```

Figure 14: Function to import movie poster

After that we have created the movie recommendation engine as below;

```
# Create a function to get movie recommendations and posters
def get_recommendations_with_posters(title, num_recommendations=5):
    idx = df_movies[df_movies['title'] == title].index[0]
    cosine_similarities = cosine_similarity(tfidf_matrix[idx], tfidf_matrix)
    similar_movies = list(enumerate(cosine_similarities[0]))
    similar_movies = sorted(similar_movies, key=lambda x: x[1], reverse=True)
    similar_movies = similar_movies[1:num_recommendations + 1]
    recommended_movies = [df_movies['title'].iloc[i[0]] for i in similar_movies]

    recommended_movies_with_posters = []
    for movie_title in recommended_movies:
        poster_url = get_movie_poster(movie_title)
        recommended_movies_with_posters.append((movie_title, poster_url))

    return recommended_movies_with_posters
```

Figure 15: Movie Recommendation Engine Creation

By this function, it will recommend the most similar movies to the movie that user has provided to the system using the cosine similarity. Then this function will retrieve both the movie name and poster for the movie.

Then we have created another recommendation engine which suggests the most popular movies from our dataset to the user, so they can check for those movies as well.
That function is as below;

```
def get_recommendations(title, num_recommendations=10):
    # Create a DataFrame for movie titles and their corresponding indices
    movie_indices = pd.Series(df_movies.index, index=df_movies['title'])
    idx = movie_indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    movie_indices = [i[0] for i in sim_scores]
    return df_movies['title'].iloc[movie_indices[1:num_recommendations + 1]]
```

Figure 16: Most Popular Movie Suggestion System Creation

5. Development of Movie Recommendation Application

Here, the code begins by checking if the 'movie_name' has been provided, which is the movie for which we want to receive recommendations. If a 'movie_name' is provided, the code proceeds to fetch movie recommendations along with their associated poster images.
A header is displayed to indicate the purpose of the following content, which is to provide recommended movies based on the input movie.
An example of how the system work is included in the images below;

```
# Show movie recommendations with posters if a movie name is provided
if movie_name:
    try:
        recommendations = get_recommendations_with_posters(movie_name)
        st.write(f'Recommended Movies for {movie_name}:')

        # Create five columns to display posters in rows of 5
        columns = st.columns(3)

        for i, (recommended_movie, poster_url) in enumerate(recommendations):
            col = columns[i % 3]  # Alternate between the 5 columns

            col.write(recommended_movie)
            if poster_url:
                col.image(poster_url, caption=recommended_movie, width=150)
    except KeyError:
        st.error('Movie not found in the dataset. Please try a different movie name.')
```

Figure 17: Code snippet to display the recommended movies

Then for each movie suggested above we have allowed the user to input a rating as feedback, so that as recommendation engine developers we can get a good idea about the accuracy of our recommendation engine.

```
126 # Allow users to rate genre-based recommended movies and store ratings
127 if recommendations:  # Check if recommendations exist
128     st.subheader('Rate Genre-Based Recommended Movies')
129     ratings = {}
130     for recommended_movie, poster_url in recommendations:
131         rating = st.slider(f'Rate {recommended_movie}', min_value=0, max_value=5, value=0, key=recommended_movie)
132         ratings[recommended_movie] = rating
133
134     if st.button('Submit Genre-Based Ratings'):
135         # Append new genre-based ratings to the ratings.csv file
136         new_ratings = []
137         for movie, rating in ratings.items():
138             movie_id = df_movies[df_movies['title'] == movie]['movieId'].values[0]
139             new_ratings.append({'userId': user_id, 'movieId': movie_id, 'rating': rating, 'timestamp': 0})
140
141         df_new_ratings = pd.DataFrame(new_ratings)
142         print(df_new_ratings)
143         df_ratings = pd.concat([df_ratings, df_new_ratings], ignore_index=True)
144         print(df_ratings)
145         df_ratings.to_csv(ratings_file, sep=',', index=False, encoding='utf-8')
146
147         st.success('Genre-based ratings submitted and stored in ratings.csv')
148
149
150
```

Figure 18: Code snippet to create the ratings slider

In here, the system first checks if there are any recommendations available. Recommendations are required to allow users to rate movies.

A sub header is displayed to specify the purpose of the section, which is to rate genre-based recommended movies. For each recommended movie, a slider input is provided to allow users to rate the movie. The slider ranges from 0 to 5, providing a range of possible ratings.
The user's ratings are stored in the 'ratings' dictionary, with the movie title as the key and the assigned rating as the value.
When the "Submit Genre-Based Ratings" button is clicked, the code does the following:

- Create an empty list called 'new_ratings' to store the new ratings.
- Iterate through the user's ratings for each movie and appends the relevant information to 'new_ratings'. This information includes 'userId', 'movieId', 'rating', and 'timestamp'.
- Construct a new DataFrame ('df_new_ratings') from 'new_ratings'.
- Append the new ratings to the existing 'df_ratings' DataFrame using the 'pd.concat' method. This allows the ratings to be stored along with any existing ratings.
- Finally, the code saves the updated ratings DataFrame to the 'ratings.csv' file.

This feature allows users to rate the recommended movies based on their preferences and stores these ratings for future reference and analysis.

Then we use K-nearest neighbors model to suggest the most popular movies on the website as we had mentioned earlier.

```
# Use the Nearest Neighbors algorithm for user-specific recommendations
if movie_name and user_id:
    movie_id = df_movies[df_movies['title'] == movie_name]['movieId'].values[0]
    user_ratings = user_movie_rating_scaled[user_id - 1].reshape(1, -1)

    # Fit a Nearest Neighbors model using the user-movie rating matrix
    model = NearestNeighbors(n_neighbors=10, metric='cosine', algorithm='brute')
    model.fit(user_movie_rating_scaled)

    distances, indices = model.kneighbors(user_ratings, 10)
    recommended_movie_indices = indices[0]
    recommended_movies = [df_movies['title'].iloc[idx] for idx in recommended_movie_indices]
    st.write(f'Most Popular Movies ')
    st.write(recommended_movies[:5])
```

Figure 19: KNN Model Development

6. Evaluation of The Recommended Movies

Using the tf-idf matrix we have built, we calculated the relevance for the recommended movies based on their cosine similarities. This way it is easier for the user to get an idea about the recommendations.

```
    # Calculate relevance for each recommended movie
    cosine_similarities = cosine_similarity(tfidf_matrix[i], tfidf_matrix)
    relevance = (cosine_similarities[0][0] * 100).round(2)
    col.write(f'Relevance: {relevance}%')
```

Figure 20: To evaluate the relevance of the recommended movies

7. User Interfaces

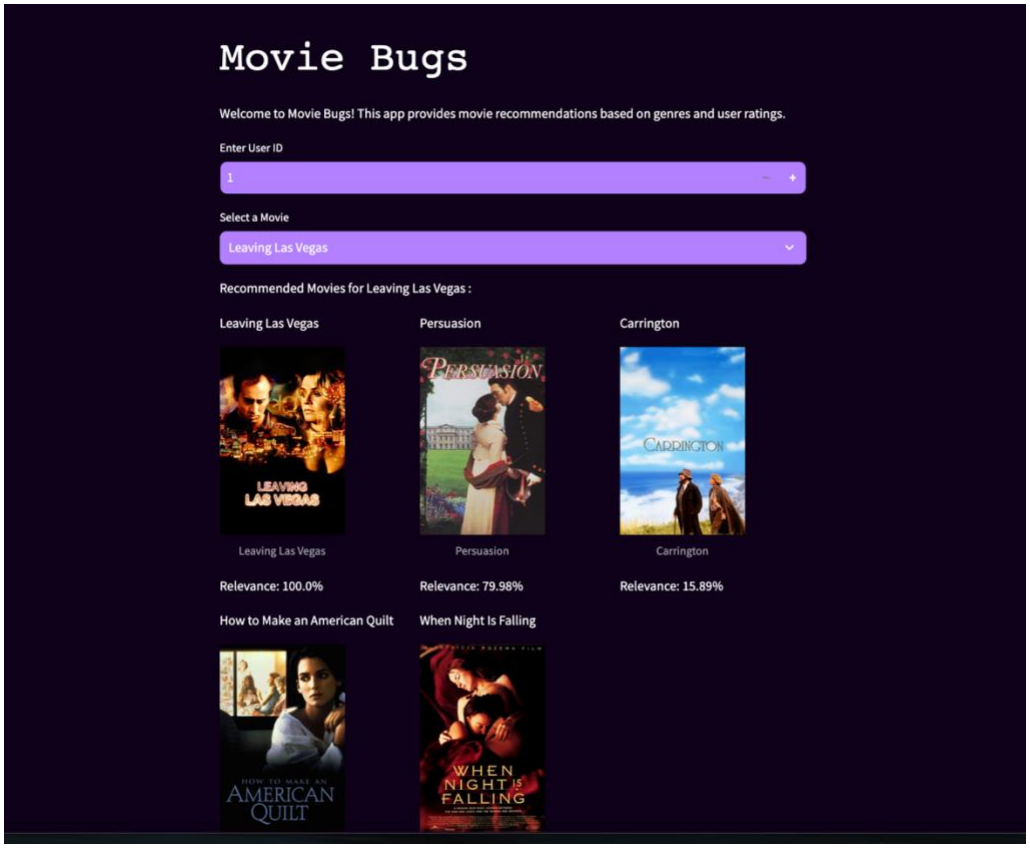

Figure 21: Recommendation UI

Figure 22: Recommendation Feedback Slider UI

Figure 23: Most Popular Movie Suggestion

8. Project Structure

We had used the version controlling to build this recommendation engine collaboratively with all the team members.

Here we have provided the github link for our repository:
[Beginner_Bugs- Movie Recommendation System]

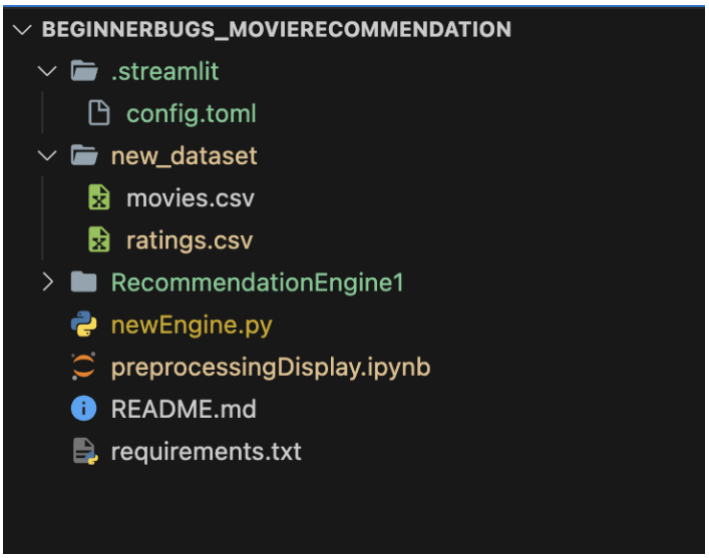

Figure 24: Project Structure

In this project structure, as you can see it has another folder named 'Recommendation Engine1'. We had tried out another content-based recommendation system using another movie dataset.

9.References

[1] "Recommender Systems — A Complete Guide to Machine Learning Models," [Online]. Available: https://towardsdatascience.com/recommender-systems-a-complete-guide-to-machine-learning-models-96d3f94ea748 .

[2] "Beginners Guide to Content Based Recommender Systems," [Online]. Available: https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/.

[3] "Content-based Filtering," [Online]. Available: https://developers.google.com/machine-learning/recommendation/content-based/basics.

[4] "What Is Collaborative Filtering: A Simple Introduction," [Online]. Available: https://builtin.com/data-science/collaborative-filtering-recommender-system.

[5] "Movie Recommendation System using Machine Learning," [Online]. Available: https://www.shiksha.com/online-courses/articles/movie-recommendation-system-using-machine-learning/.