Movie Recommendation System

Satya Deepu Mandapati, Chandi Priya Katta, Hemanth Yalamanchili

1 Implementation steps

1.1 Load Movies Dataset:

The code begins by loading the movies dataset from the "movies.csv" file using the pandas library's `read_csv()` function. This dataset contains information about movies, including their titles and genres.

1.2 Clean the Data

Define a function `clean_title(title)` that takes a movie title as input and returns a cleaned version of the title. Inside the function, it utilizes the `re.sub()` function from the `re` module to perform a regular expression-based substitution. The regular expression [a-zA-Z0-9] matches any character that is not an alphanumeric character or a space. The `re.sub()` function replaces any matched characters with an empty string, effectively removing them from the title. This cleaning process ensures that only alphanumeric characters and spaces remain in the title, eliminating any special characters or symbols. Finally, the `apply()` function is used to apply the `clean_title` function to each title in the "movies" DataFrame, and the cleaned titles are stored in a new column named "clean_title".

1.3 Define Search Function:

Use a TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer which converts text data (movie titles) into numerical vectors. Next, create a search function which calculates the cosine similarity between the query vector and the TF-IDF matrix (`tfidf`) using the `cosine_similarity()` function from scikit-learn. The resulting similarity scores are flattened into a 1-dimensional array. Then, the indices of the top 5 most similar movies are extracted using `np.argpartition()`. Finally, the function retrieves the top 5 most similar movies from the "movies" DataFrame based on the calculated indices, reverses the order to display the most similar movie first, and returns these results.

1.4 Define Function to Find Similar Movies:

The ratings data is loaded from a CSV file named "ratings.csv", and users who rated the selected movie highly (rating ¿ 4) are identified. Subsequently, movies that these similar users have also rated highly (rating ¿ 4) are retrieved, and the percentage of similar users who rated each of these movies is calculated. Movies with a high percentage of ratings from similar users (greater than 10 Percent) are considered as potential recommendations from users who liked the given movie.

1.5 Calculate the Similarity Score:

Considering all users who rated the movies highly (rating ¿ 4), the ratings DataFrame is filtered to include only the ratings for movies that were highly rated by similar users. Then, it calculates the number of times each potential recommendation movie was rated highly by all users and divides it by the total number of unique users who rated any of these movies highly. This calculation yields the percentage of all users who rated each potential recommendation movie highly. These percentages are then combined into a DataFrame named `rec_percentages`, where each row represents a potential recommendation movie, and columns "similar" and "all" represent the percentage of ratings from similar users and all users, respectively. The DataFrame is then sorted in descending order based on the "score" column to prioritize movies with a higher score, indicating greater popularity among similar users relative to all users. Finally, the top 10 movies with the highest scores are selected from `rec_percentages`, merged with the "movies" DataFrame based on the movie IDs, and displayed, providing a list of potential movie recommendations ranked by their similarity to the given movie and their popularity among users.

1.6 Create Interactive Movie Recommendation Widget:

Create an Interactive widget using Jupyter IPython widgets which takes the user's input as a movie title and recommends similar movies with the highest similarity score.

2 Goals

2.1 Calculate the percentage of similar users who rated other movies positively

The recommendation system loads ratings data from a CSV file named "ratings.csv" and identifies users who rated the selected movie highly (rating ¿ 4). It then retrieves movies that these similar users have also rated highly (rating ¿ 4) and calculates the percentage of similar users who rated each of these movies. Movies with a high percentage of ratings from similar users (greater than 10

percent) are considered as potential recommendations from users who liked the given movie.

```
similar_users = ratings[(ratings["movieId"] == movie_id) & (ratings["rating"] > 4)]["userId"].unique()
```

```
similar_users
```

```
array([    21,    187,    208, ..., 162469, 162485, 162532], dtype=int64)
```

```
similar_user_recs = ratings[(ratings["userId"].isin(similar_users)) & (ratings["rating"] > 4)]["movieId"]
```

```
similar_user_recs = similar_user_recs.value_counts() / len(similar_users)

similar_user_recs = similar_user_recs[similar_user_recs > .10]
```

```
all_users = ratings[(ratings["movieId"].isin(similar_user_recs.index)) & (ratings["rating"] > 4)]
```

```
all_user_recs = all_users["movieId"].value_counts() / len(all_users["userId"].unique())
```

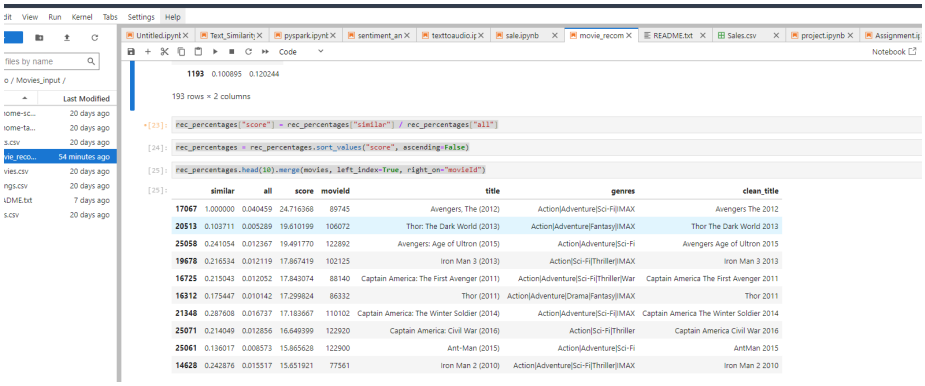
```
rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis=1)
rec_percentages.columns = ["similar", "all"]
```

```
rec_percentages
```

movieId	similar	all
89745	1.000000	0.040459
58559	0.573393	0.148256
59315	0.530649	0.054931
79132	0.519715	0.132987
2571	0.496687	0.247010
...	...	...
47610	0.103545	0.022770

2.2 Find the similarity score for each recommended movie based on the ratio of similar users' ratings to all users' ratings

[**?**] The recommendation system considers all users who rated movies highly (rating ¿ 4). It filters the ratings DataFrame to include only the ratings for movies that were highly rated by similar users. Then, it calculates the percentage of all users who rated each potential recommendation movie highly by dividing the number of times each movie was rated highly by all users by the total number of unique users who rated any of these movies highly. These percentages are combined into a DataFrame named `rec_percentages`, where each row represents a potential recommendation movie, and columns "similar" and "all" represent the percentage of ratings from similar users and all users, respectively. The DataFrame is then sorted in descending order based on the "score" column to prioritize movies with a higher score, indicating greater popularity among similar users relative to all users. Finally, the top 10 movies with the highest scores are selected from `rec_percentages`, merged with the "movies" DataFrame based on the movie IDs, and displayed, providing a list of potential movie recommendations ranked by their similarity to the given movie and their popularity



193 rows × 2 columns

```
rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"]
rec_percentages = rec_percentages.sort_values("score", ascending=False)
rec_percentages.head(10).merge(movies, left_index=True, right_on="movieId")
```

	similar	all	score	movieId	title	genres	clean_title
17067	1.000000	0.040459	24.716368	89745	Avengers, The (2012)	Action|Adventure|Sci-Fi|IMAX	Avengers The 2012
20513	0.103711	0.005289	19.610199	106072	Thor: The Dark World (2013)	Action|Adventure|Fantasy|IMAX	Thor The Dark World 2013
25058	0.241054	0.012367	19.491770	122892	Avengers: Age of Ultron (2015)	Action|Adventure|Sci-Fi	Avengers Age of Ultron 2015
19678	0.216534	0.012119	17.867419	102125	Iron Man 3 (2013)	Action|Sci-Fi|Thriller|IMAX	Iron Man 3 2013
16725	0.215043	0.012052	17.843074	88140	Captain America: The First Avenger (2011)	Action|Adventure|Sci-Fi|Thriller|War	Captain America The First Avenger 2011
16312	0.175447	0.010142	17.299824	86332	Thor (2011)	Action|Adventure|Drama|Fantasy|IMAX	Thor 2011
21348	0.287608	0.016737	17.183667	110102	Captain America: The Winter Soldier (2014)	Action|Adventure|Sci-Fi|IMAX	Captain America The Winter Soldier 2014
25071	0.214049	0.012856	16.649399	122920	Captain America: Civil War (2016)	Action|Sci-Fi|Thriller	Captain America Civil War 2016
25061	0.136017	0.008573	15.865628	122900	Ant-Man (2015)	Action|Adventure|Sci-Fi	AntMan 2015
14628	0.242876	0.015517	15.651921	77561	Iron Man 2 (2010)	Action|Adventure|Sci-Fi|Thriller|IMAX	Iron Man 2 2010

2.3 Display the top recommended movies and merges their details using similarity score.

An interactive widget is created using ipywidgets, allowing users to input a movie title. As the user types the movie title, the system dynamically updates and displays a list of top 10 similar movies based on user ratings.

```
[35]: import ipywidgets as widgets
from IPython.display import display

movie_name_input = widgets.Text(
    value='Toy Story',
    description='Movie Title:',
    disabled=False
)
recommendation_list = widgets.Output()

def on_type(data):
    with recommendation_list:
        recommendation_list.clear_output()
        title = data["new"]
        if len(title) > 1:
            results = search(title)
            movie_id = results.iloc[0]["movieId"]
            display(find_similar_movies(movie_id))

movie_name_input.observe(on_type, names='value')

display(movie_name_input, recommendation_list)
```

Movie Title: Hulk

	score	title	genres
12425	306.384000	Incredible Hulk, The (2008)	Action|Sci-Fi
11642	93.389775	Fantastic Four: Rise of the Silver Surfer (2007)	Action|Adventure|Sci-Fi
6411	82.806486	Hulk (2003)	Action|Adventure|Sci-Fi
10070	81.266488	Fantastic Four (2005)	Action|Adventure|Sci-Fi
20018	67.388808	Wolverine, The (2013)	Action|Adventure|Fantasy|Sci-Fi
16595	66.695837	Transformers: Dark of the Moon (2011)	Action|Adventure|Sci-Fi|War|IMAX
21454	65.800591	The Amazing Spider-Man 2 (2014)	Action|Sci-Fi|IMAX
13460	59.126737	Transformers: Revenge of the Fallen (2009)	Action|Adventure|Sci-Fi|IMAX
11561	56.570902	Spider-Man 3 (2007)	Action|Adventure|Sci-Fi|Thriller|IMAX
12146	56.420364	Jumper (2008)	Action|Adventure|Drama|Sci-Fi|Thriller

3 Conclusion

In conclusion, leveraging technologies such as Pandas, NumPy, regular expressions (re), and TF-IDF vectorization (TfidfVectorizer), movie recommendation systems have revolutionized the way users discover and consume movies. By harnessing the power of data analysis and machine learning, these systems provide personalized recommendations based on users' preferences and viewing history.

Using Pandas, data about movies, ratings, and user interactions can be efficiently processed and analyzed. NumPy facilitates mathematical operations and data manipulation, enabling the extraction of meaningful insights from large datasets. Regular expressions (re) offer powerful tools for text processing and cleaning, ensuring that movie titles and other textual data are appropriately formatted for analysis.

Moreover, TF-IDF vectorization with TfidfVectorizer converts textual movie data into numerical vectors, allowing for efficient calculation of similarities between movies. This technology plays a crucial role in determining relevant recommendations by identifying movies with similar characteristics or themes.

By integrating these technologies, movie recommendation systems deliver several key benefits. They enhance user satisfaction by offering personalized movie suggestions tailored to individual preferences, leading to increased user engagement and retention. Additionally, these systems drive revenue growth for businesses in the entertainment industry by promoting content that aligns with users' interests, ultimately contributing to a more enjoyable and rewarding movie-watching experience for audiences worldwide.

4 References

- 1.Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions. Sensors (Basel). 2022 Jun 29;22(13):4904. doi: 10.3390/s22134904. PMID: 35808398; PMCID: PMC9269752.
- 2.Movie Recommendation System With Python And Pandas: Data Project https://www.youtube.com/watch?v=eyEabQRBMQAt=159s
- BIG DATA RECOMMENDATION SYSTEMS by Jahnavi Redrouthu. https://www.linkedin.com/pulse/big-data-recommendation-systems-jahnavi-redrouthu/
- Making Your Data Analytics Come to Life using ipywidgets by Wei-Meng Lee published in Towards data science