Data Analytics, Autumn 2024 Assignment 3

Hardik Pravin Soni[1], **Sake Venkata Vignan Kumar**[2] **and Astitva**[3]

[1] *20CS30023, 5th Year, Computer Science and Engineering, IIT Kharagpur* Mail
[2] *20CS30070, 5th Year, Computer Science and Engineering, IIT Kharagpur* Mail
[3] *20CS30007, 5th Year, Computer Science and Engineering, IIT Kharagpur* Mail

November 15, 2024

1. Problem Statement

Building a Personalized Actor Recommendation System

Objective

To design and implement a recommendation system that suggests **top actors** to users based on their **movie preferences** by leveraging enriched **movie-actor datasets**, user-movie ratings, and recommendation algorithms.

Key Tasks

1. **Data Enrichment: Finding Top Actors for Each Movie**
   - Use the **MovieLens dataset** as the base for movie ratings and enhance it with actor information for each movie.
   - Scrape or retrieve data using **web scraping techniques** or **APIs**.
   - Document and store the **movie-actor relationships** for future use.
2. **User-Actor Rating Matrix Construction**
   - Calculate **user-actor ratings** using the user-movie ratings dataset and the movie-actor associations.
   - Define a user's rating for an actor as the **average rating** given to movies featuring that actor.
   - Normalize and preprocess the data, including handling sparse data by removing **low-activity users** or actors.
3. **Recommendation Algorithm Selection**
   - Identify a suitable algorithm (e.g., **collaborative filtering**, **content-based filtering**) for actor recommendations.
   - Justify the choice of algorithm based on the **dataset characteristics** and the problem's requirements.
4. **Implementation of the Recommendation System**
   - Develop and implement the chosen algorithm to generate **actor recommendations** for users.
   - Provide a ranked list of **top actors** for any given **user ID** based on their preferences.
5. **Evaluation and Performance Analysis**
   - Assess the recommendation system using metrics such as:
     - **Precision@k**: Proportion of relevant actors in the top-k recommendations.
     - **Recall@k**: Proportion of relevant actors retrieved within the top-k recommendations.
     - **NDCG@k**: Quality of ranking, considering the position of relevant actors in the top-k recommendations.
   - Discuss the **performance**, including strengths, limitations, and any enhancements made.
6. **Report Creation**
   - Document the end-to-end process in a structured report including:
     - **Data scraping challenges** and solutions.
     - Steps for constructing the **user-actor rating matrix**.
     - Justification and details of the **recommendation algorithm**.
     - Evaluation metrics, results, and a critical analysis of the system's **performance**.
     - Optional **visualizations** of insights derived from the recommendation system.

Outcome

The system will enable users to receive **personalized actor recommendations**, leveraging a robust evaluation of **ranking-based metrics** to ensure relevance and quality in the recommendations.

2. Methodology

Finding Top Actors for Each Movie

This report describes the methodology and implementation for finding the **top actors** for each movie using **The Movie Database (TMDb) API**. The process involves:

1. Searching movies using their **titles** or **IMDb IDs**.
2. Fetching detailed movie data, including **cast information**.
3. Extracting the **top 10 actors** based on their **popularity**.

Movie Search by Title or IMDb ID

The system uses two approaches to identify a movie when `tmdbId` is unavailable:

- **Search by Title**: This uses the TMDb `/search/movie` endpoint to find a movie by its **title**.
- **Search by IMDb ID**: The TMDb `/find` endpoint is used to locate a movie using its **external IMDb ID**.

Below is the implementation for searching movies:

```python
async def search_movie_by_title(title):
    """Search for a movie by title if tmdbId is missing."""
    url = f'https://api.themoviedb.org/3/search/movie'
    params = {'api_key': API_KEY, 'query': title}
    async with aiohttp.ClientSession() as session:
        async with session.get(url, params=params) as response:
            if response.status == 200:
                data = await response.json()
                results = data.get('results', [])
                if results:
                    return results[0]['id']  # Return the first result's TMDb
      ID
```

Code 1. Search for a Movie by Title

Fetching Cast Information

Once the `tmdbId` of a movie is available, the `/movie` endpoint with the `append_to_response=credits` parameter retrieves **cast details**. If the `tmdbId` is missing, fallback mechanisms use the title or IMDb ID to find the movie.

The key function for fetching cast details is as follows:

```python
async def fetch_cast(tmdb_id, title=None, imdb_id=None):
    """Fetch top 10 cast members for a movie by TMDb ID, IMDb ID, or title.
      """
    if not pd.isna(tmdb_id):
        url = f'https://api.themoviedb.org/3/movie/{int(tmdb_id)}?api_key={
      API_KEY}&append_to_response=credits'
        async with aiohttp.ClientSession() as session:
            async with session.get(url) as response:
                if response.status == 200:
                    data = await response.json()
                    return extract_cast(data)
    # Fallback mechanisms using title or IMDb ID follow similar logic.
```

Code 2. Fetch Top 10 Cast Members

Extracting Top Actors

The **cast data** is processed to extract the **top 10 actors** based on their **popularity**. The actors are sorted in descending order of popularity, as shown below:

```python
def extract_cast(data):
    """Extract top 10 cast members from the movie data."""
    cast_data = data.get('credits', {}).get('cast', [])
    sorted_cast = sorted(cast_data, key=lambda x: x['popularity'], reverse=
      True)
    top_actors = [actor['name'] for actor in sorted_cast[:10]]
    return top_actors
```

Code 3. Extract Top Actors

*Parallel Processing To handle large datasets efficiently, the system uses **asynchronous processing** and splits the dataset into chunks, distributing the workload across multiple processes. The **ProcessPoolExecutor** and `asyncio.gather()` are used for concurrent execution.

```python
async def process_movie_chunk(movie_chunk):
    tasks = []
    for _, row in movie_chunk.iterrows():
        tasks.append(fetch_cast(row['tmdbId'], row['title'], row.get('imdbId'
      )))
    results = await asyncio.gather(*tasks)
    return results

def run_async_task(movie_chunk):
    return asyncio.run(process_movie_chunk(movie_chunk))
```

Code 4. Process Movie Chunks

Results and Discussion

The pipeline assigns the **top 10 actors** for each movie in the dataset. The results are integrated back into the dataset for further analysis or recommendation system development.

Conclusion

This system effectively retrieves and processes **cast data** for movies using the TMDb API, ensuring robust fallback mechanisms and efficient parallel processing. The extracted top actors provide valuable insights for applications such as **personalized recommendations**.

Constructing the User-Actor Rating Matrix

In this section, we construct the User-Actor Rating Matrix by analyzing user ratings and their association with top actors in movies. The steps involved are as follows:

- **Loading Data:** The ratings data is loaded using `pd.read_csv()` from the file `ratings_set1.csv` which contains movie ratings from users.
- **Merging Data:** The ratings are merged with the movie-actor associations using the `merge()` function on the `movieId` column, resulting in the `user_ratings` DataFrame.
- **Calculating Average Rating:** We calculate the average rating for each `userId` and `top_actor` pair by grouping the data using `groupby()` and calculating the mean with `agg()` function.
- **Descriptive Analysis:** We generate descriptive statistics for the ratings using `describe()` to get a sense of the rating distribution.
- **Visualization:** Several plots are generated to visualize the ratings data:
  - **Histogram:** A histogram of the ratings distribution is plotted to observe the frequency of different ratings.
  - **Boxplot:** A boxplot is used to visualize the variability and spread of ratings for a sample user.
  - **Density Plot:** A Kernel Density Estimate (KDE) plot is generated to visualize the continuous distribution of ratings.
  - **Top Users:** The number of ratings per user is analyzed, and the top 10 users by rating count are visualized in a bar plot.
- **Top-Rated Actors:** We calculate the average ratings for each actor and display the top 20 actors with the highest average ratings using a horizontal bar plot.

By analyzing the **user-actor ratings**, we gain insights into user preferences for actors and the distribution of ratings. This helps identify the most popular actors for users and explore the variability in ratings across different users.

Choosing and Deploying a Recommendation Model for Actor Suggestions

To recommend actors to users based on their preferences, we employ a collaborative filtering approach using Singular Value Decomposition (SVD). The SVD algorithm is chosen for its ability to factorize the user-actor rating matrix, capturing latent factors that influence user preferences. This method works well for tasks involving implicit feedback, like recommending actors based on past ratings.

The following steps describe the implementation:

- **Data Loading**: The user-actor ratings are loaded into the Surprise library using the `Dataset.load_from_df` method, with the rating scale set from **0.5** to **5.0**.
- **Data Splitting**: The dataset is split into training and testing sets using an **80-20 split**, where 80% of the data is used for training and 20% for testing.
- **Model Initialization**: The `SVD` model is initialized, which is a matrix factorization technique that is effective for collaborative filtering. It captures underlying patterns in the data, such as preferences for certain actors.
- **Model Training**: The model is trained on the training set using the `model.fit(trainset)` method.
- **Model Evaluation**: The trained model is evaluated on the test set using `model.test(testset)`. Performance metrics like **RMSE** (Root Mean Squared Error) and **MAE** (Mean Absolute Error) are computed to assess the quality of recommendations.

The SVD algorithm is appropriate for this task because it efficiently identifies hidden patterns in user preferences based on historical ratings. It is scalable and works well when users have sparse ratings, which is typical in real-world recommendation tasks. This makes it a suitable choice for recommending actors to users based on their previous ratings and preferences.

3. Evaluation and Analysis

In this section, we evaluate the performance of the recommendation system using various metrics and provide a brief analysis of the model's effectiveness. The recommendation system has been evaluated using the following metrics:

Evaluation Metrics

- **RMSE (Root Mean Squared Error):** Measures the average magnitude of the error between predicted and actual ratings. The lower the RMSE, the better the model's accuracy in predicting ratings.
- **MAE (Mean Absolute Error):** Represents the average of the absolute differences between predicted and actual ratings. It provides an intuitive measure of model accuracy.
- **Precision@5:** Indicates the proportion of relevant actors in the top-5 recommended actors. A higher Precision@5 signifies that the top recommendations are highly relevant to the user.
- **Recall@5:** Measures the proportion of relevant actors retrieved within the top-5 recommendations. A higher Recall@5 implies that the model is able to retrieve most of the relevant actors.
- **NDCG@5 (Normalized Discounted Cumulative Gain):** Evaluates the quality of the ranking by considering the position of relevant actors. NDCG accounts for the order in which

actors are recommended, penalizing late appearances of relevant actors.

Performance Metrics

The following performance metrics were observed during the evaluation of the recommendation system:

RMSE: 0.4636

MAE: 0.2990

Precision@5: 0.6984

Recall@5: 0.8366

NDCG@5: 0.7114

Analysis of Results

RMSE and MAE: Both RMSE and MAE are indicative of the model's prediction accuracy. With an RMSE value of 0.4636 and an MAE of 0.2990, the model shows a reasonable degree of accuracy in predicting ratings, with relatively small prediction errors.

Precision@5 and Recall@5: The Precision@5 score of 0.6984 indicates that approximately 70% of the top-5 recommended actors are relevant to the user, suggesting that the model is generally successful in recommending relevant actors. The Recall@5 value of 0.8366 highlights the model's ability to retrieve most of the relevant actors within the top-5 recommendations, achieving a high retrieval rate.

NDCG@5: The NDCG@5 score of 0.7114 shows that the model ranks relevant actors relatively well in the top-5 list. This metric emphasizes the importance of both retrieving relevant actors and placing them higher in the ranked list, and the model's score indicates that it achieves a good balance in this respect.

Strengths and Limitations

Strengths:

- The recommendation system provides high Recall@5, meaning it retrieves a significant proportion of relevant actors.
- The NDCG@5 value indicates that the system ranks actors effectively, placing relevant actors in the top recommendations.
- The model demonstrates acceptable accuracy with low RMSE and MAE, making it reliable for predicting user ratings.

Limitations:

- The system could be improved by incorporating more personalized features such as user demographics or additional actor-related attributes.
- While the Precision@5 is decent, there is still room for improvement, particularly in ensuring that the top-5 actors are even more relevant to the users.
- Further tuning of hyper-parameters and exploring other recommendation algorithms might improve the overall performance.

Visualization

(Optional: Include a visualization that showcases the performance of the recommendation system, such as the top-rated actors for a sample user or the distribution of actor ratings.)

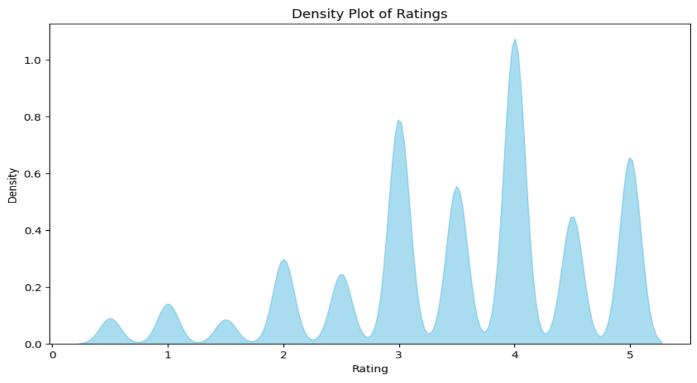


Figure 1. Density of Ratings given by Users to Movies

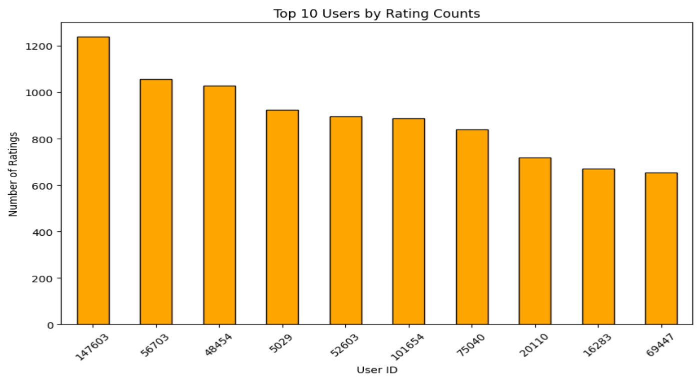
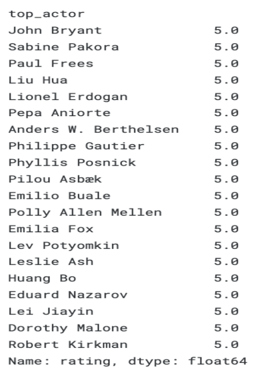
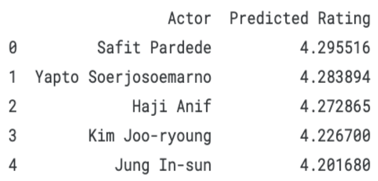


Figure 2. Top Users with maximum Ratings Given



```
top_actor
John Bryant            5.0
Sabine Pakora          5.0
Paul Frees             5.0
Liu Hua                5.0
Lionel Erdogan         5.0
Pepa Aniorte           5.0
Anders W. Berthelsen   5.0
Philippe Gautier       5.0
Phyllis Posnick        5.0
Pilou Asbæk            5.0
Emilio Buale           5.0
Polly Allen Mellen     5.0
Emilia Fox             5.0
Lev Potyomkin          5.0
Leslie Ash             5.0
Huang Bo               5.0
Eduard Nazarov         5.0
Lei Jiayin             5.0
Dorothy Malone         5.0
Robert Kirkman         5.0
Name: rating, dtype: float64
```

Figure 3. Top Actors with thier Ratings



```
                  Actor  Predicted Rating
0          Safit Pardede          4.295516
1  Yapto Soerjosoemarno          4.283894
2             Haji Anif          4.272865
3         Kim Joo-ryoung          4.226700
4           Jung In-sun          4.201680
```

Figure 4. Recommendations for User with id: 1