Machine Learning Report

Akash Bahri
College of Engineering
Northeastern University
Toronto, ON
Bahri.a@northeastern.edu

Abstract

In this project, a **movie recommendation system** was developed using the **Item-Item Collaborative Filtering** algorithm based on **Cosine Similarity**. The system predicts the most relevant movies for users by analyzing their past ratings and movie similarities. An **interactive web application** was built to provide personalized movie recommendations to users. The backend was implemented using **Flask**, while the frontend was created with **React**. Challenges included handling large datasets and ensuring efficient performance for fast recommendations.

1 Introduction

A recommendation system is a software tool that helps suggest relevant items to users based on their preferences or past behavior. In this project, a movie recommendation system is built using Item-Item Collaborative Filtering, which analyzes the similarity between movies based on user ratings. The system predicts and recommends movies that a user may like, improving their experience by suggesting movies they might not have encountered otherwise. The project also includes building an interactive web application to allow users to input their preferences and receive personalized movie recommendations.

Objective: The goal of this project is to develop a recommendation system using Item-Item Collaborative Filtering to suggest movies to users based on their ratings and the similarity between movies. (similar to Netflix)

Scope: This system includes:

- **Jupyter notebook** for data processing and building the model
- A **backend** for processing recommendations using **Flask**.
- A **frontend** developed with **React** to display recommendations in an interactive manner.

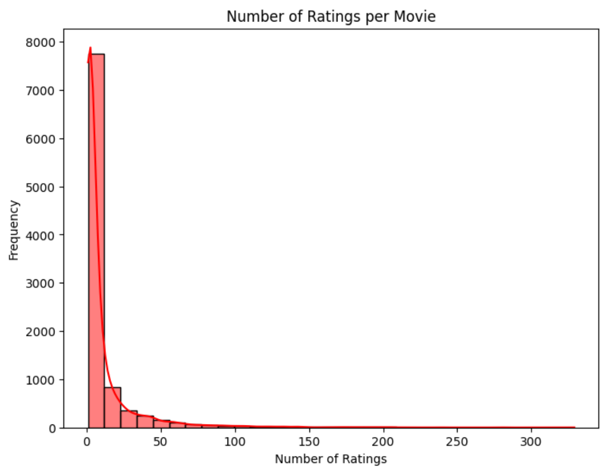
2 Datasets

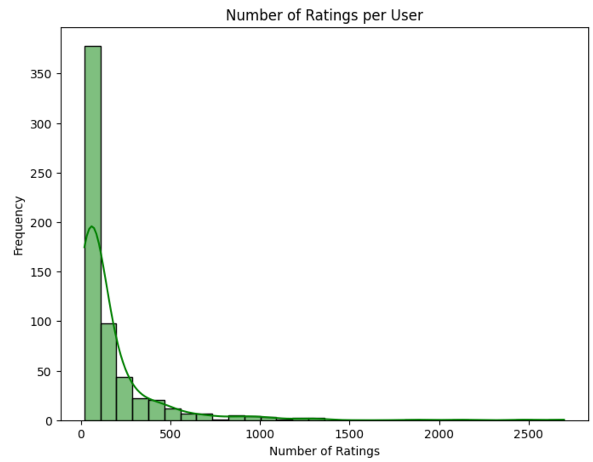
Description:

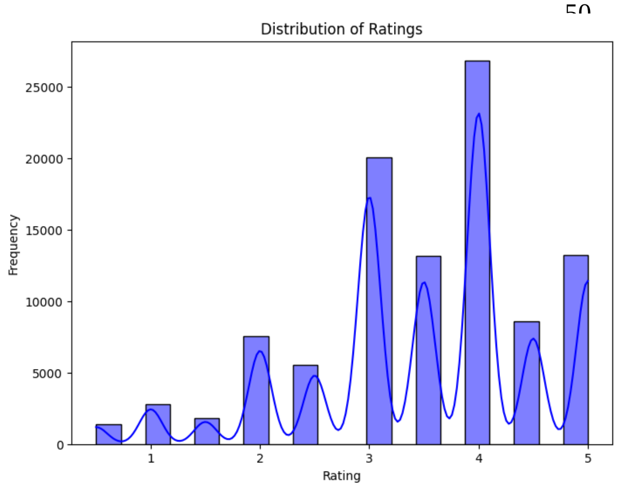
- The dataset contains movie ratings, user ratings, and movie metadata (including movie titles and genres).
- MovieLens dataset was used, which includes ratings and metadata for movies.

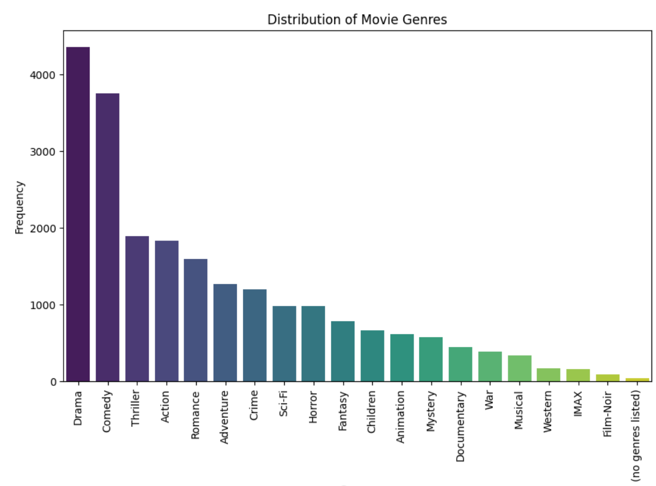
Dataset Characteristics:

- **Size**: 100,000 ratings, 9,000 movies, 600 users.
- **Attributes**: Movie ID, title, genres, user ratings.
- **Challenges**: Sparse data (most users have rated a small portion of movies), handling missing values.









3 Methodology

Item-Item Collaborative Filtering

The recommendation system implemented in this project uses **Item-Item Collaborative Filtering**, a popular algorithm in recommendation systems. This approach works by recommending items that are similar to those the user has previously rated or interacted with. The main idea is to find the **similarity** between items (movies, in this case) and suggest those that are most similar to the movies the user has rated highly.

Cosine Similarity

To determine the similarity between items, we use **Cosine Similarity**, which measures the cosine of the angle between two vectors in a multidimensional space. The formula for **Cosine Similarity** between two items AAA and BBB is given by:

$$\text{Cosine Similarity} = \frac{A \cdot B}{\|A\|\|B\|}$$

Where:

- $A \cdot B$ is the dot product of vectors A and B,
- $\|A\|$ and $\|B\|$ are the magnitudes (norms) of the vectors.

This metric produces values between **0** and **1**:

- A **value of 1** means the two items are identical (i.e., their vectors are pointing in the same direction).
- A **value of 0** means the items are completely dissimilar.

In our system, **item-item similarity** is calculated for all movie pairs using their **ratings** from all users. Once the similarity between items is computed, the next step is to predict ratings for items that a user hasn't rated yet.

Data Preprocessing

The dataset used in this project is the **MovieLens dataset**, which includes ratings from users for various movies. The data preprocessing steps include:

1. **Handling Missing Data**:
   - The **user-item interaction matrix** contains missing values where users haven't rated some movies. These missing values are handled by using **zeros** in the matrix, which means unrated movies are considered as "not rated yet."
2. **Data Transformation**:
   - The **MovieLens dataset** is in CSV format. We use **pandas** to load the dataset and transform it into the required **user-item matrix** format, where each row represents a user, and each column represents a movie. The entries in the matrix are the ratings given by users to movies, with **zeros** indicating unrated movies.
3. **Matrix Creation**:
   - We create two main matrices:
     1. **User-Item Matrix**: This matrix represents user preferences. Each entry MijM_{ij}Mij in the matrix represents the rating given by user iii to movie jjj.
     2. **Item-Item Similarity Matrix**: This matrix stores the **cosine similarity** between each pair of movies. It helps in identifying which movies are similar to each other, so we can recommend

similar movies to users.

Item-Item Similarity Calculation

The similarity between items is calculated using **Cosine Similarity**. For each movie jjj, the system computes the similarity between it and all other movies based on user ratings. The resulting similarity values are stored in the **item-item similarity matrix**, where each entry SijS_{ij}Sij represents the similarity between movie iii and movie jjj.

Once the similarity matrix is computed, the **top-N most similar movies** are identified for each movie. These similar movies are then used to predict the rating a user would give to unrated movies based on their ratings for the similar movies.

Rating Prediction

For each user, the system predicts the rating for movies that the user has not yet rated. The predicted rating RuiR_{ui}Rui for a user uuu and an unrated movie iii is computed as the **weighted sum** of the ratings for similar movies, weighted by their **similarity** to movie iii. Specifically, the prediction is calculated as:

$$R_{ui} = \frac{\sum_{j \in \text{similar to } i} S_{ij} \cdot R_{uj}}{\sum_{j \in \text{similar to } i} |S_{ij}|}$$

Where:

- R_{uj} is the rating given by user u to movie j,
- S_{ij} is the similarity between movies i and j,
- The sum is taken over all movies j that are similar to movie i.

4 System Design and Implementation

Backend Implementation (Flask)

The backend of the system is built using **Flask**, a lightweight web framework for Python. The backend handles the following:

1. **API for Movie Recommendations**:
    - This API receives a **user ID** as input and returns a list of **top-N recommended movies**. The recommendation is based on the **item-item similarity** and the **predicted ratings**.
2. **API for Movie Posters**:
    - The system fetches **movie posters** from the **TMDB API** using the **movie ID**. It retrieves the **poster URL** and returns it along with the movie title and genres.
3. **Data Storage**:
    - The **user-item matrix** and **item similarity matrix** are stored in **Pickle files**

for fast retrieval during the recommendation process. This minimizes the need for recalculating the similarity matrix every time a recommendation request is made.

Frontend Development (React)

The frontend is built using **React** to create an interactive user interface:

1. **User Input**:
   - Users can input their **user ID**, which is sent to the backend to retrieve personalized movie recommendations.
2. **Movie Display**:
   - The frontend displays the movie title, genres, predicted rating, and the movie poster for each recommended movie. The UI is styled using **CSS** to ensure that it is responsive and aesthetically pleasing, with a 3-column grid layout for movie cards.
3. **Interaction**:
   - The user can interact with the app by clicking the **"Get Recommendations"** button to fetch movie suggestions based on their user ID. The recommended movies are displayed dynamically, along with relevant metadata.

5 Challenges and Solutions

Data Challenges

- **Sparsity**: The user-item matrix was sparse, with many missing ratings. This made it difficult to find enough similar movies for accurate recommendations.
- **Cold Start Problem**: New users or movies that have few ratings had no data to base predictions on. A fallback strategy was used for handling new users and movies.

Performance Optimization

- **Memory Usage**: The large size of the dataset required optimization to prevent memory issues. The matrices were precomputed to speed up the recommendation process.
- **Speed of Recommendations**: The recommendation process was optimized by storing precomputed item similarities, reducing the time needed to make predictions.

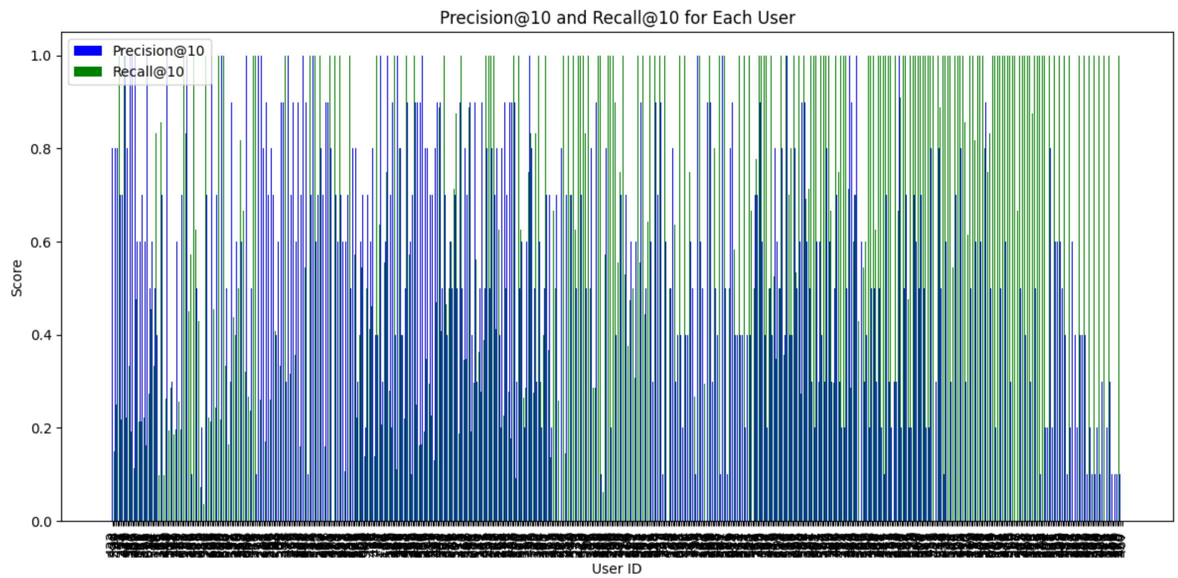
6 Results and Evaluation

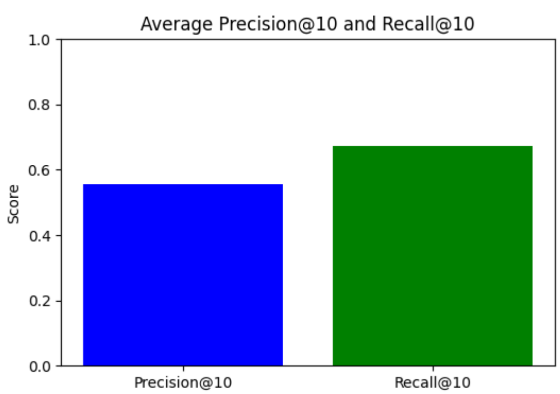
Performance Evaluation

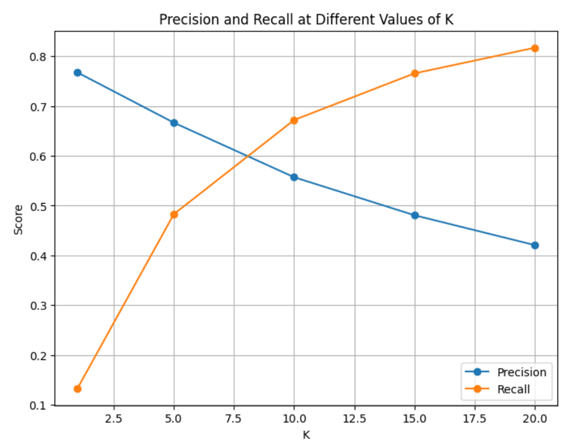
The recommendation system was evaluated using metrics such as **Precision** and **Recall**. For instance, **Precision@10** was used to measure how accurate the top-10 recommendations were for each user. The system performed well, providing relevant movie recommendations based on user preferences.

Evaluation Metrics

The system's performance was evaluated using **Precision@10** and **Recall@10** metrics. The results showed that the recommendation system was able to suggest relevant movies to users effectively.



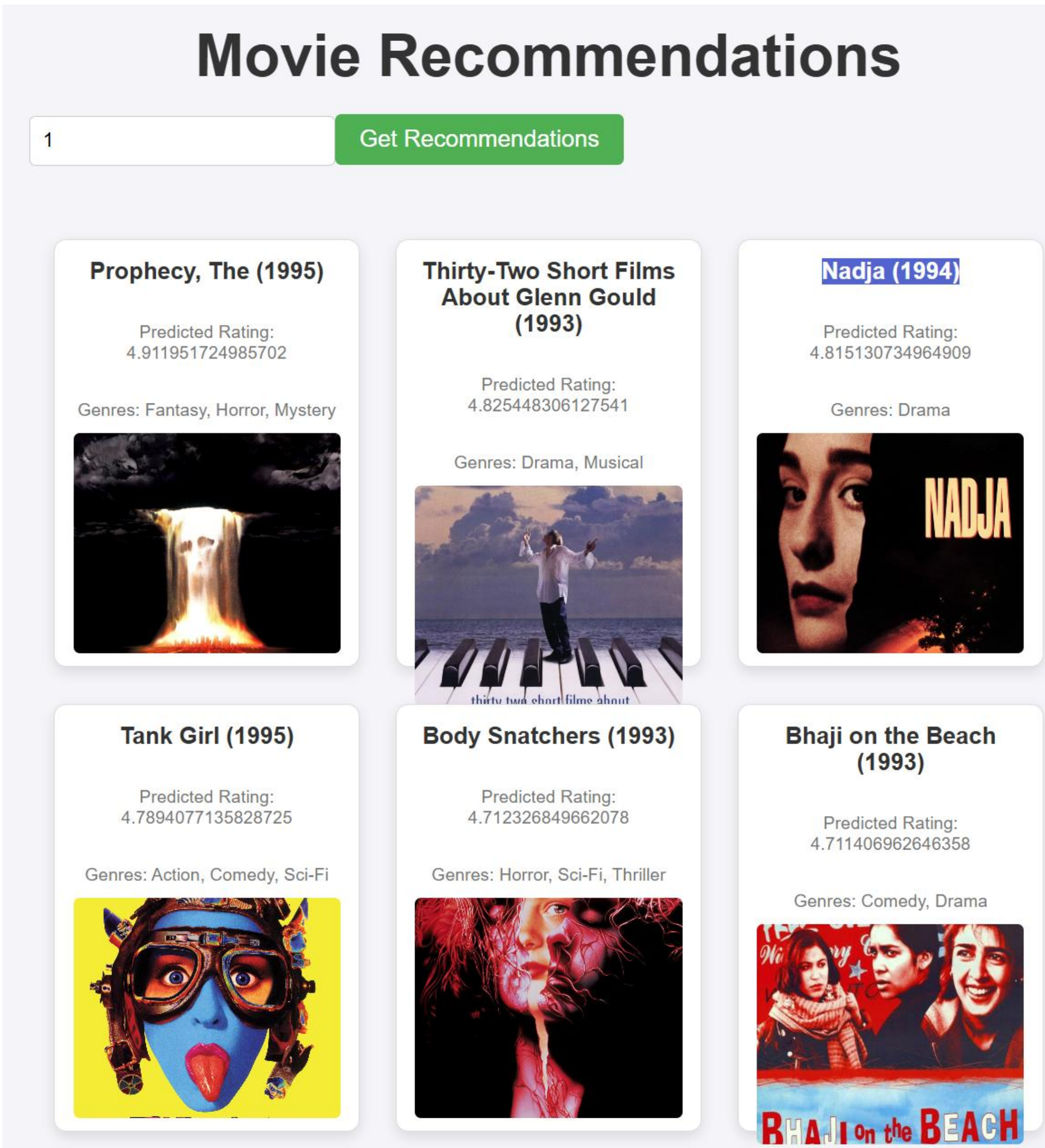




Sample Recommendations

Here are some examples of movie recommendations for user 1:

- **Movie 1**: **Prophecy, The (1995)** (Predicted Rating: 4.9)
- **Movie 2**: **Thirty-Two Short Films About Glenn Gould (1993) (**Predicted Rating: 4.8)
- **Movie 3**: **Nadja (1994)** (Predicted Rating: 4.8)



7 Conclusion

This project demonstrates the development of a movie recommendation system using **Item-Item Collaborative Filtering** and **Cosine Similarity**. The backend was built using **Flask**, and the frontend was developed with **React** to provide an interactive user interface. The system was able to generate personalized movie recommendations based on user ratings and item similarities. Future work could include addressing the **cold start problem**, improving the **speed of recommendations**, and exploring hybrid models that combine multiple recommendation techniques.

8 References

1. MovieLens Dataset - https://grouplens.org/datasets/movielens/
2. Cosine Similarity Explanation - https://en.wikipedia.org/wiki/Cosine_similarity
3. TMDB API Documentation - https://www.themoviedb.org/documentation/api
4. Flask Documentation - https://flask.palletsprojects.com/
5. React Documentation - https://reactjs.org/docs/
6. OpenAI. ChatGPT: Language Model 4o for Assistance: https://www.openai.com/chatgpt