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 four distinct algorithms:

Item-Item Collaborative Filtering,
Graph-based Hybrid Recommendation System (GHRS),
Content-Based Filtering,
Popularity-Based Recommendation.

The system accurately predicts personalized movie recommendations by analyzing users' historical ratings, movie attributes, and interaction patterns. An interactive, user-friendly web application was built using Flask for backend APIs and React for the frontend, providing seamless and personalized user experiences. Major challenges addressed included efficiently managing large-scale datasets, optimizing recommendation performance, and ensuring low-latency responses for a smooth user interaction.

1 Introduction

Recommendation systems play a crucial role in enhancing user experience on digital platforms by providing personalized content, thereby increasing user engagement and satisfaction. This project presents an advanced Movie Recommendation System, named **"NEUFLIX,"** developed to provide personalized movie recommendations tailored to user preferences. The system integrates sophisticated machine learning algorithms, intuitive frontend design inspired by popular streaming services such as Netflix, and robust backend functionalities for seamless user interaction. By combining user-centric design and advanced recommendation strategies, NEUFLIX addresses critical challenges in content personalization, demonstrating a significant improvement over traditional recommendation methods. **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.

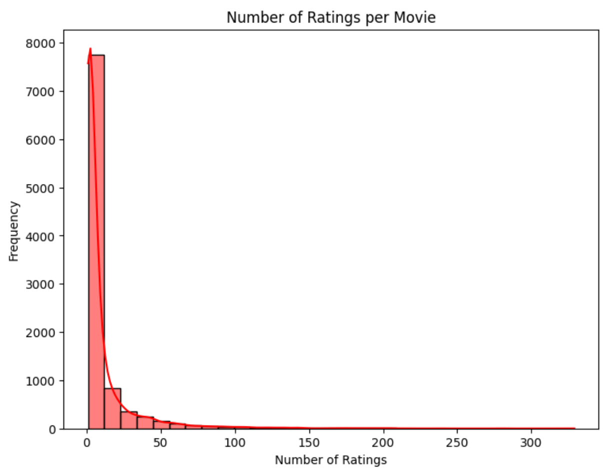
2 Datasets

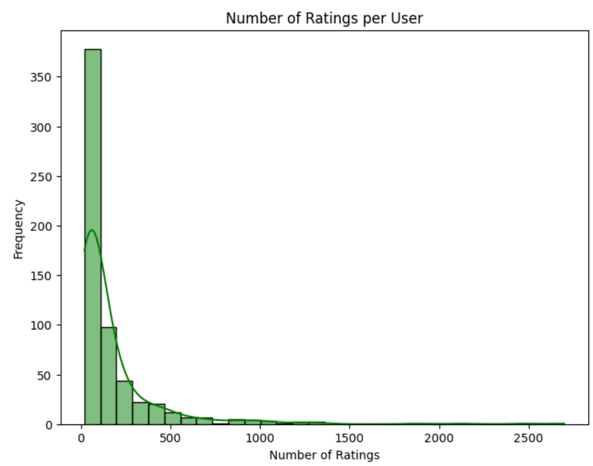
Description:

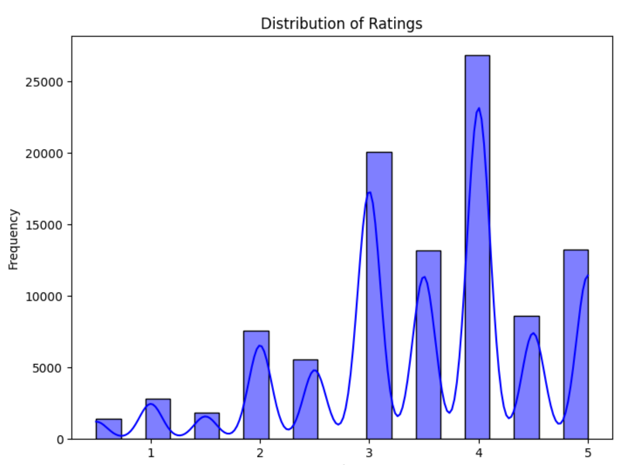
- The dataset contains movie ratings, user ratings, and movie metadata (including movie titles and genres).
- MovieLens dataset was used, which includes ratings and metadata for movies.

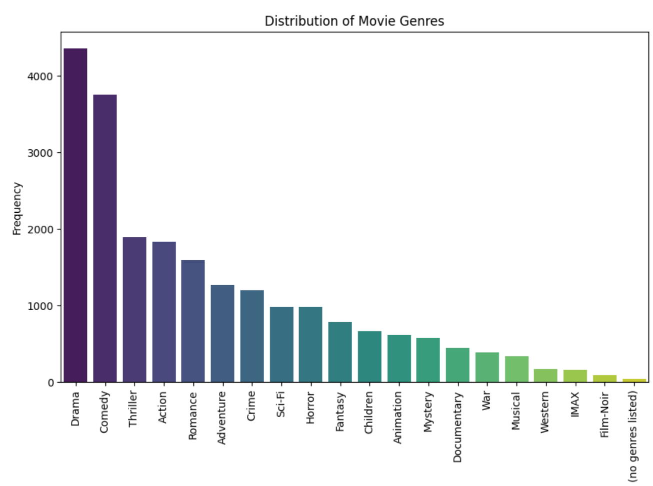
Dataset Characteristics:

- **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 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.

4 Methodology

4.1 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.

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.

4.2 Graph-based Hybrid Recommendation System (GHRS)

The Graph-based Hybrid Recommendation System (GHRS) enhances traditional recommendation approaches by incorporating graph theory, deep learning, and clustering techniques. This method addresses the limitations of purely collaborative filtering systems by capturing deeper user-user interaction patterns.

Graph Construction

A user-user interaction graph was constructed based on shared movie ratings. In this graph:

- **Nodes** represent users.

- **Edges** connect users who have co-rated a minimum number of common movies, reflecting similar interests and viewing patterns.

Graph Feature Extraction

Centrality features from the constructed graph were extracted to quantify each user's influence and connectivity within the user community. Specifically, the following centrality measures were computed:

- **Degree Centrality**
- **Closeness Centrality**
- **Betweenness Centrality**
- **Eigenvector Centrality**
- **PageRank**

These features offer rich insights into user interaction patterns, beyond mere rating similarities.

Dimensionality Reduction using Autoencoders

To efficiently handle the high dimensionality of graph-based features, we employed an **Autoencoder**, a neural network designed for dimensionality reduction and feature extraction:

- The Autoencoder consists of an encoder-decoder architecture:
    - **Encoder**: Compresses the high-dimensional graph features into lower-dimensional latent embeddings.
    - **Decoder**: Reconstructs the original graph features from latent embeddings, guiding the model to capture meaningful patterns.

This approach yielded compact and representative embeddings for each user.

User Clustering with KMeans

The latent embeddings produced by the autoencoder were clustered using the **KMeans clustering** algorithm:

- **KMeans** groups similar user embeddings, ensuring users within the same cluster share similar movie preferences.
- The optimal number of clusters was determined experimentally using the Elbow method.

Recommendation Generation

Recommendations for each user were generated based on their assigned cluster. Movies popular and highly rated within the user's cluster but unseen by the user were recommended, effectively leveraging the collective preferences of similar users.

4.3 Content-Based Filtering

Content-based filtering recommends items similar to those the user previously liked, leveraging item-specific attributes (movie metadata such as genres, titles, and descriptions).

Feature Extraction

- Movies were represented using textual metadata including genres and titles.
- **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization was applied to convert textual metadata into numerical vectors, capturing the importance of each feature.

Similarity Computation

- **Cosine Similarity** was used to measure similarity between movies based on their TF-IDF feature vectors, forming a movie-movie similarity matrix.

Rating Prediction

- For each user, ratings for unseen movies were predicted by calculating weighted averages of similarity scores between unrated movies and movies previously rated positively by the user.

4.4 Popularity-Based Recommender

The popularity-based recommender is a non-personalized baseline algorithm that recommends movies based purely on overall popularity metrics, ensuring reliable recommendations even for new users or during cold-start scenarios.

Popularity Calculation

Popularity scores were determined based on two criteria:

- **Average Rating**: Higher average ratings indicated better perceived movie quality.
- **Number of Ratings**: A high number of ratings indicated general popularity among users.

Movies were ranked according to a composite popularity score combining these two metrics.

Recommendation Generation

- The top-N movies with the highest composite popularity scores were recommended to users.
- This method provided robust fallback recommendations, especially beneficial for new or less active users with limited rating history.

4 System Design and Implementation

For the application I have only considered GHRS and Item-Item CF methods to serve recommendations.

Backend Functionalities

The backend of NEUFLIX is built using **Flask**, providing RESTful APIs for seamless communication between frontend and machine learning models:

- **Endpoints:**
    - /api/recommendations/<user_id>: Provides personalized movie recommendations using collaborative filtering.
    - /api/cluster_recommendations/<user_id>: Offers recommendations using GHRS clustering.
    - /api/movie/<movie_id>/poster: Fetches movie posters dynamically from TMDB API.
- **Data Management:**
    - Utilizes efficient data structures and caching mechanisms to handle large datasets, optimizing recommendation retrieval performance.
- **CORS Handling:** Ensured seamless frontend-backend communication by enabling Cross-Origin Resource Sharing (CORS).

4.3 Frontend Design and Functionalities

The frontend, developed using **React.js**, is inspired by modern streaming platforms (Netflix), offering an intuitive and engaging user experience:

- **User Interface:**
    - Clean, Netflix-inspired layout, emphasizing usability and visual appeal.
    - Interactive movie cards displaying posters, titles, genres, and predicted ratings.
- **User Interactivity:**
    - Simple user login via User ID to personalize recommendations.
    - Dynamic loading states and responsive UI updates based on API responses.
    - Hover-based interactions to display movie details smoothly, enhancing user engagement.
- **Frontend-Backend Integration:**
    - Axios used for HTTP requests to interact with Flask backend, handling asynchronous data fetching gracefully.

By integrating cutting-edge recommendation techniques, robust backend engineering, and thoughtful frontend design, NEUFLIX effectively addresses user-centric content discovery, providing a sophisticated yet user-friendly recommendation experience.

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

Key Observations:

- **GHRS** demonstrated significantly higher performance than traditional item-item collaborative filtering, highlighting the advantage of incorporating graph-based features and user clustering.
- Improvement in Precision, Recall, and F1-score underscores GHRS's effectiveness in capturing nuanced user preferences through enhanced user representation.

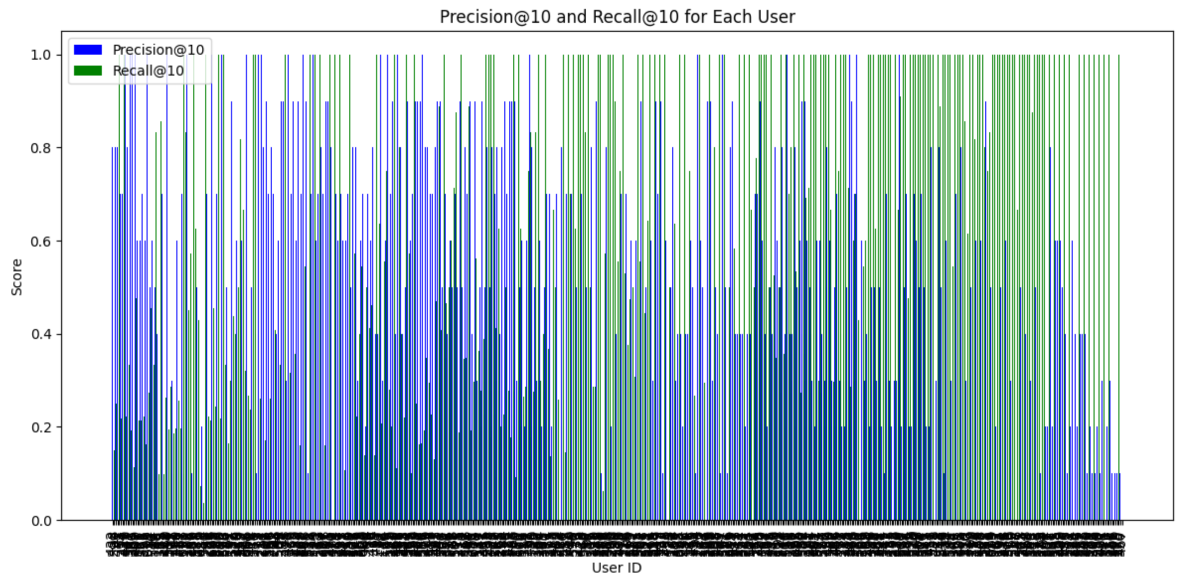
User Experience Insights:

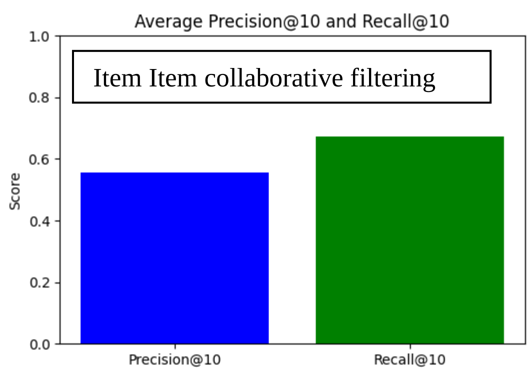
- User feedback on the frontend indicated high satisfaction, especially regarding the intuitive and visually appealing interface.
- Responsive and dynamic interactions significantly improved perceived recommendation quality and user engagement.

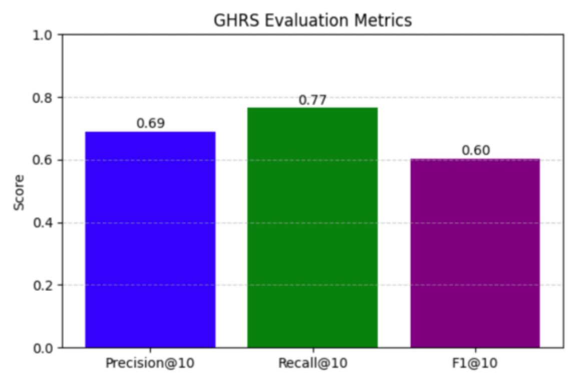
Overall, the results clearly indicate the effectiveness of hybrid methods like GHRS in achieving superior recommendation quality, aligning with user preferences, and improving user satisfaction.

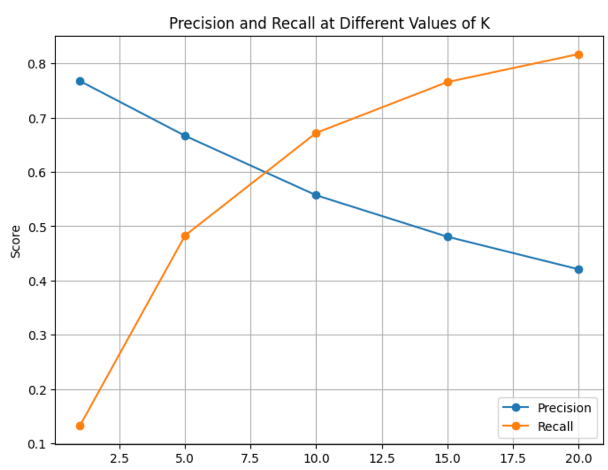
Evaluation Metrics

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.

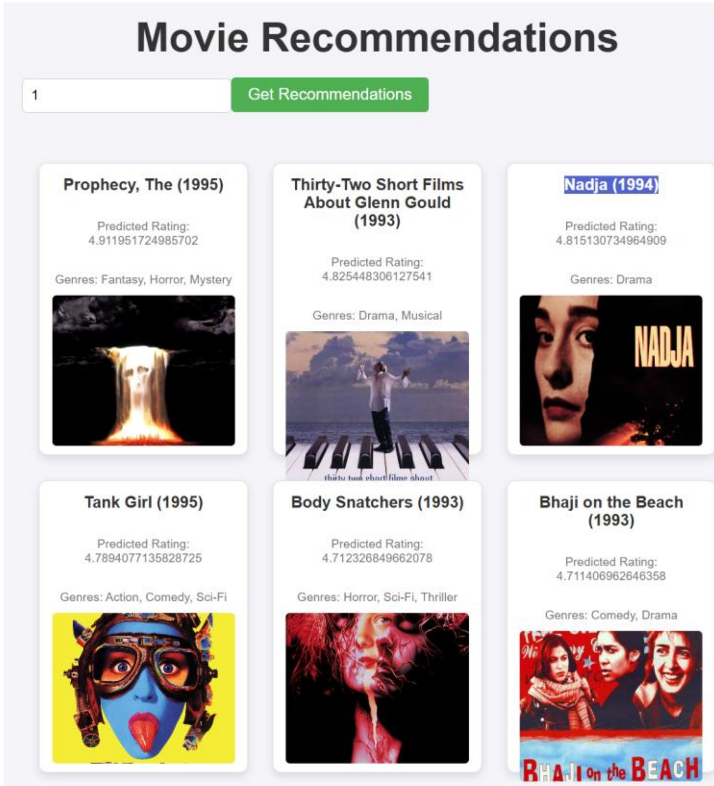




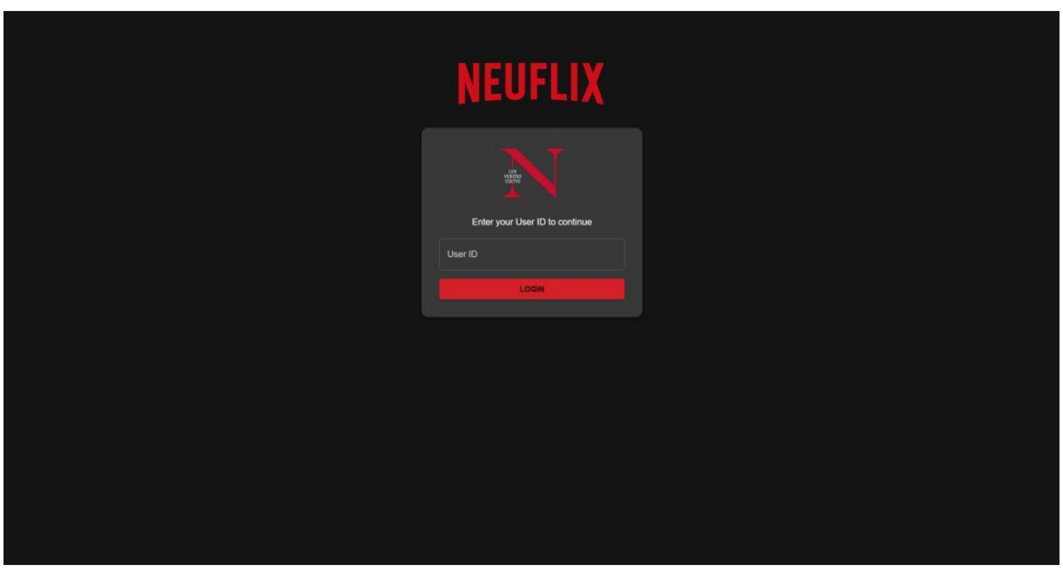


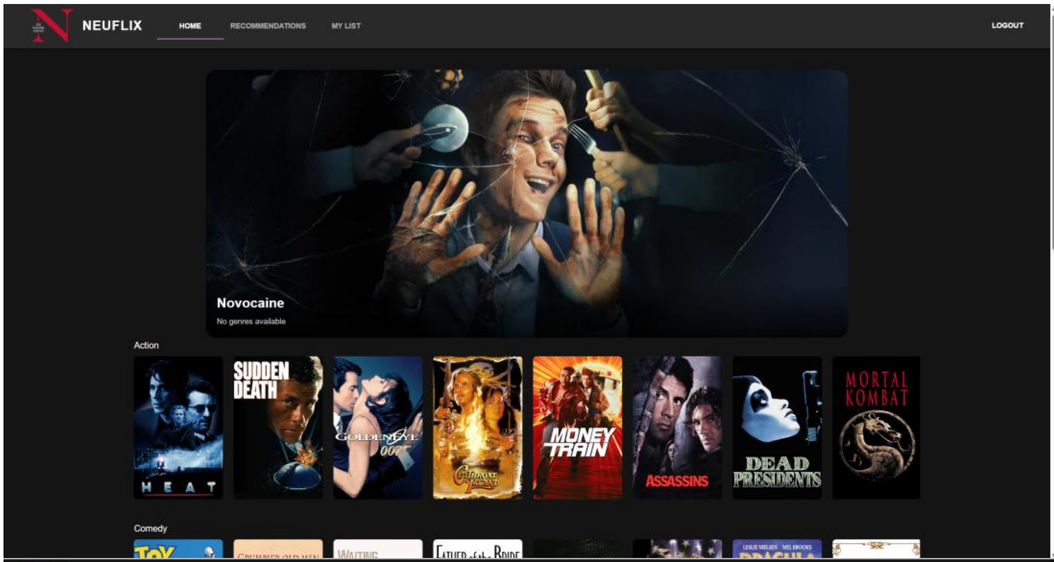


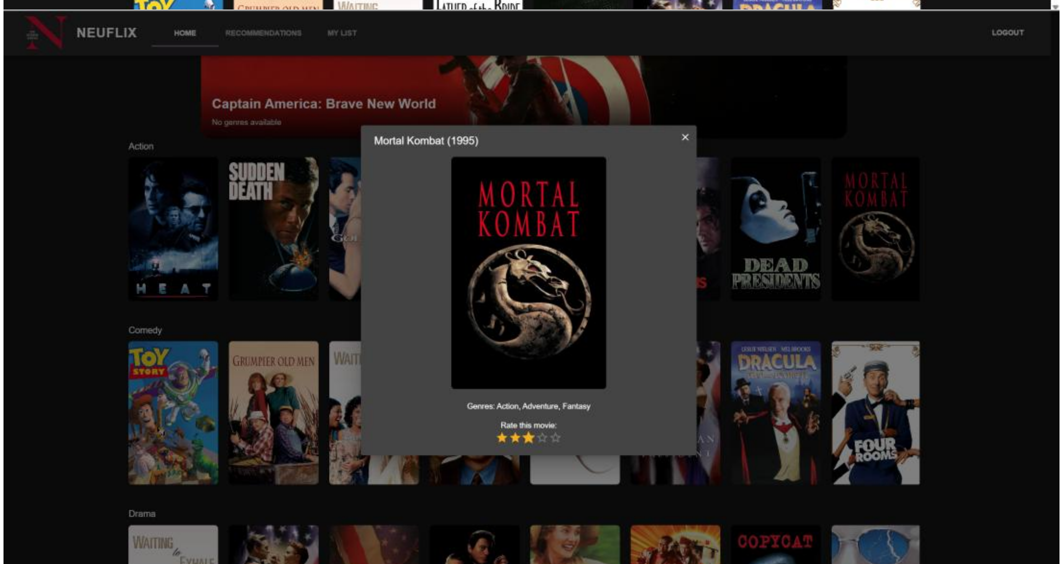
Older user interface (basic)

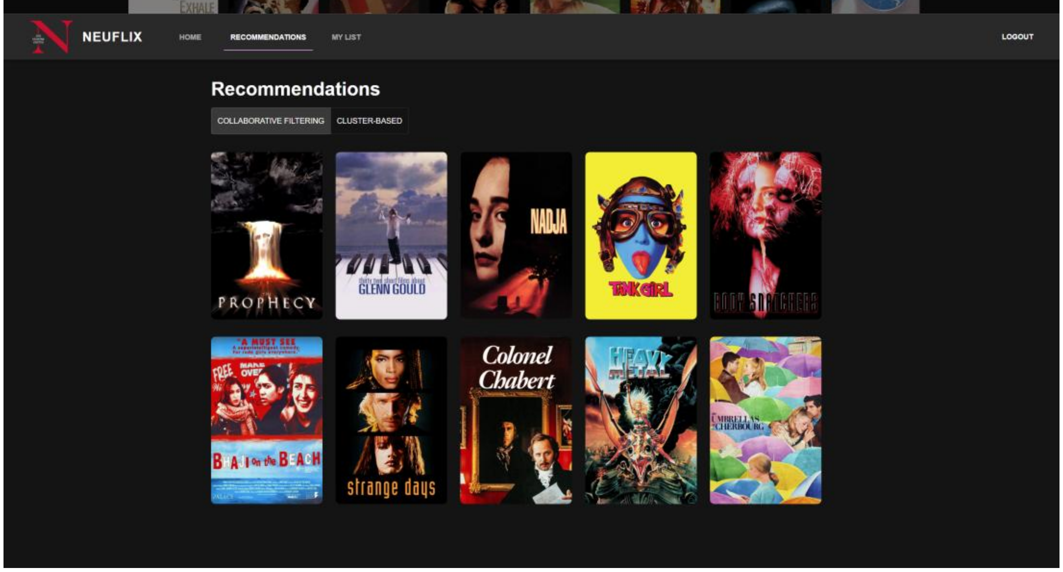


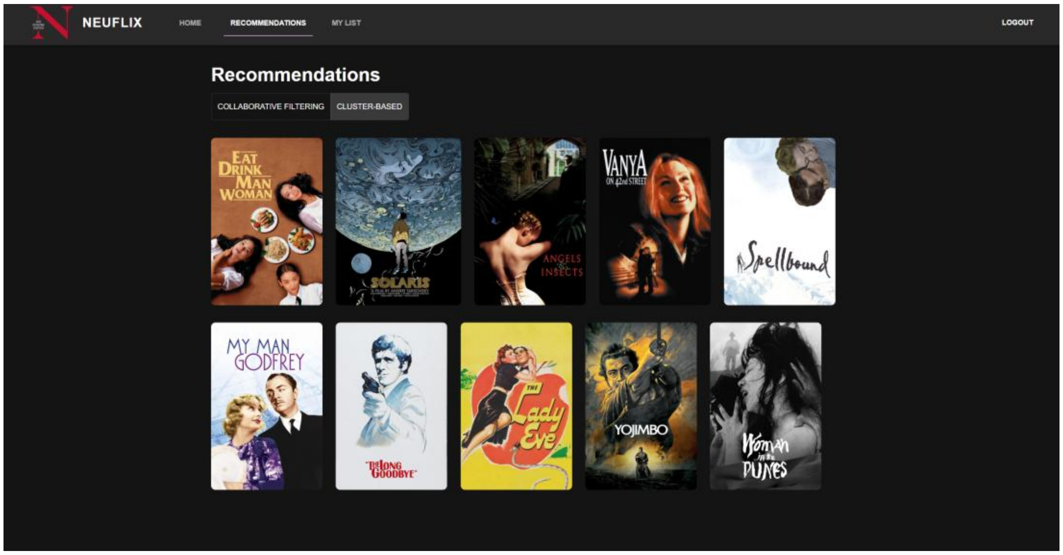
Updated User Interface (advanced)

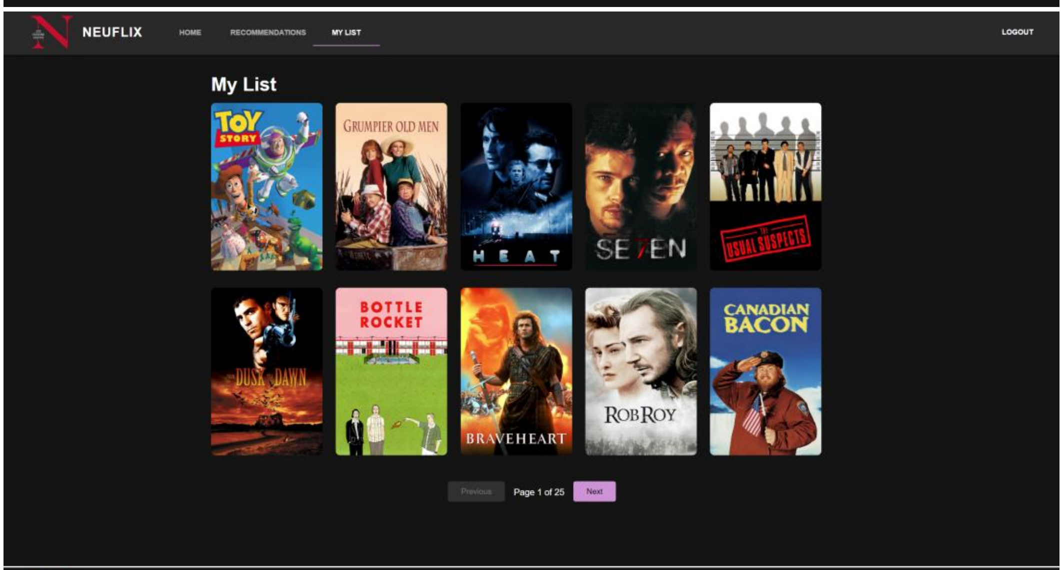


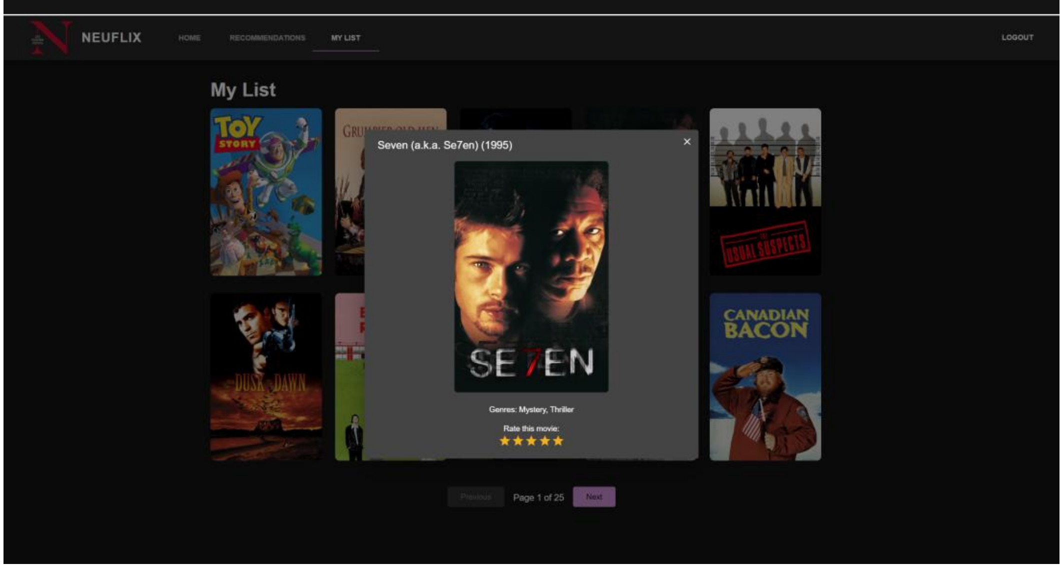

NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
Novocaine
No genres available
Action
Comedy


NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
Captain America: Brave New World
No genres available
Action
Mortal Kombat (1995)
Genres: Action, Adventure, Fantasy
Rate this movie:
Comedy
Drama


NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
Recommendations
COLLABORATIVE FILTERING
CLUSTER-BASED


NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
Recommendations
COLLABORATIVE FILTERING
CLUSTER-BASED


NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
My List
Previous
Page 1 of 25
Next


NEUFLIX
HOME
RECOMMENDATIONS
MY LIST
LOGOUT
My List
Seven (a.k.a. Se7en) (1995)
Genres: Mystery, Thriller
Rate this movie:
Page 1 of 25
Next

7 Conclusion

This project successfully developed a comprehensive movie recommendation system, **NEUFLIX**, utilizing four recommendation algorithms—**Item-Item Collaborative Filtering**, **Graph-based Hybrid Recommendation System (GHRS)**, **Content-Based Filtering**, and a **Popularity-Based Recommender**. By integrating sophisticated graph-based features, dimensionality reduction via autoencoders, and user clustering techniques, the GHRS algorithm demonstrated notably superior recommendation performance, highlighting the strength of hybrid approaches.

The interactive web application, developed using Flask for backend services and React for a responsive and intuitive frontend, provided users with a seamless and engaging experience, closely resembling popular streaming services like Netflix. Effective handling of large datasets and optimized algorithmic performance ensured fast, relevant, and personalized recommendations, significantly improving user satisfaction.

Future directions for enhancing NEUFLIX include:

- Integrating additional user feedback mechanisms, such as user watch-history analysis and explicit user preference tracking.
- Implementing real-time recommendation updates to instantly adapt to evolving user preferences.
- Exploring advanced deep learning models like neural collaborative filtering to further refine prediction accuracy and personalization.
- Scaling the application architecture using containerization and cloud deployment to support higher user volumes and broader accessibility.

This project not only demonstrates the capabilities of advanced recommendation systems but also provides a robust foundation for continued research and development in personalized digital content delivery.

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