Graph-Based Movie Recommendation System: Final Project Report

Chettleburgh, Aiden
Bar-Tur, Judah
Kim, Riho
Abu Shaban, Hasan

March 31, 2025

Abstract

This report describes the development of a graph-based movie recommendation system designed to improve personalized movie suggestions by leveraging movie attributes, user ratings, and their inter-relationships. Building upon our original proposal, we have implemented a modular Python program that constructs a graph from the MovieLens dataset and employs graph algorithms to recommend movies that best match user preferences. This document details the problem motivation, datasets used, computational methods, instructions for running the program, changes made since the proposal, and an in-depth discussion of our findings.

1 Introduction

With the ever-increasing number of movies available across multiple streaming platforms, users face the daunting challenge of deciding what to watch. Traditional recommendation systems on platforms like Netflix or Hulu often favor in-house productions and may not fully capture a user's diverse preferences. Our project addresses this issue by developing a graph-based recommendation system. The core research question is:

How can we use graph-based models to improve personalized movie recommendations by comparing the relationships between user preferences and movie attributes?

By modeling movies and users as vertices in a graph and user ratings as weighted edges, our approach leverages structural similarities to identify movies that align well with a user's taste. This method aims to reduce the time users spend searching and increase the accuracy of recommendations. Our system is implemented in Python, with key modules responsible for data ingestion, graph construction, recommendation computation, and a user-friendly GUI for interaction.

2 Dataset Description

Our project primarily utilizes the MovieLens dataset provided by GroupLens [3]. The dataset contains information on movies (titles, genres, and basic metadata) and user ratings, which are used to create vertices and weighted edges in the recommendation graph. Additional information such as movie descriptions and external tags has been incorporated by querying supplementary APIs (e.g., TheMovieDB) to enhance movie profiles.

Specifically, our system processes:

- **ratings.csv:** Contains user ratings with fields such as `userId`, `movieId`, `rating`, and `timestamp`.
- **movies.csv:** Includes movie details like `movieId`, `title`, and `genres`.
- **tags.csv & links.csv:** Provide additional metadata for movies.

Only relevant columns (e.g., `movieId`, `rating`, `genres`) are used to build the graph. We have also pre-processed the dataset to extract a manageable subset suitable for rapid testing and demonstration purposes.

3 Computational Overview

Our system is built in Python and is organized into several modules:

1. **Data Processing and Graph Construction:** A dedicated module reads CSV files from the MovieLens dataset, creates `Movie` and `User` objects, and builds a graph using the `networkx` library. In this graph, nodes represent movies and users, while edges capture the rating relationships.
2. **Recommendation Engine:** Based on the constructed graph, our recommendation engine uses similarity measures and weighted edge computations to identify movies that are similar to those a user has positively rated. This engine considers both direct ratings and indirect relationships via common user nodes.
3. **User Interface:** The GUI, built with Tkinter, prompts users for their preferences (e.g., favorite genres or a list of movies they liked) and displays the recommended movies. The interface is designed to be intuitive and requires minimal technical knowledge.
4. **Main Driver:** The `main.py` file serves as the entry point. It integrates the modules, loads the data, constructs the graph, and launches the GUI.

Throughout the code, best practices have been followed: modularization, clear function and variable naming, and comprehensive documentation (including inline comments and docstrings). The computational pipeline has been optimized for both clarity and performance.

4 Instructions for Obtaining Datasets and Running the Program

To run our project, please follow these steps:

1. **Installation:** Ensure that Python 3.13 is installed. Use the provided `requirements.txt` to install necessary libraries. Run:

   ```
   pip install -r requirements.txt
   ```

2. **Datasets:** Download the MovieLens dataset (included in the submission zip file) and any pre-processed data files.
3. **Execution:** From the command line or within your IDE, run the `main.py` file:

   ```
   python main.py
   ```

4. **Using the GUI:** Follow the on-screen instructions to enter your preferences. The program will then display a list of recommended movies along with brief descriptions and ratings.

5 Changes from the Project Proposal

Since our initial proposal, several significant changes have been made:

- **Algorithm Enhancements:** We refined our recommendation algorithm by incorporating weighted edges and additional similarity metrics. This has improved the accuracy of our recommendations compared to our initial plan.
- **Data Enrichment:** While the proposal focused solely on the basic MovieLens data, we have now integrated supplementary movie metadata from external APIs. This extra information helps enhance the quality of recommendations.
- **Interface Development:** Based on TA feedback, we developed a full-featured Tkinter GUI, making the system interactive and more user-friendly.
- **Code Organization:** Our codebase has been restructured into multiple modules to better separate concerns, improve readability, and facilitate easier debugging and testing.
- **Performance Optimizations:** We optimized data parsing and graph construction to support a larger subset of the dataset, ensuring that the system runs efficiently even with increased data volume.

6 Discussion

Our graph-based movie recommendation system represents a significant evolution from the initial proposal. The decision to use a graph model allowed us to capture the nuanced relationships between users and movies that traditional recommendation systems often overlook. By treating movies and users as nodes and ratings as weighted edges, we were able to design an algorithm that computes similarities not only based on explicit user ratings but also via indirect connections through shared preferences.

One of the primary advantages of this approach is the ability to integrate multiple sources of data. The incorporation of external metadata from TheMovieDB has enhanced the descriptive power of each movie node. This enrichment makes it possible to draw more meaningful connections between movies, even if the user ratings alone would not suffice. For instance, movies that might seem unrelated based solely on user ratings can be linked through common genres or similar cast members.

In practice, the recommendation engine first builds the graph by reading and processing the MovieLens data. The use of `networkx` facilitated the creation and management of complex graph structures, and its built-in algorithms helped in calculating centrality and similarity measures. One challenge we encountered was ensuring that the graph remained manageable in size, as the full MovieLens dataset is quite large. To address this, we implemented data filtering techniques that focus on a representative subset of the data, thereby striking a balance between computational efficiency and recommendation accuracy.

Our Tkinter-based GUI was designed with usability in mind. Early user testing revealed that clear instructions and a straightforward input interface were critical for non-technical users. As a result, we made several iterations on the design to ensure that the process—from entering preferences to viewing recommendations—was both intuitive and responsive. The feedback from these tests was invaluable in refining the final product.

Despite the improvements, there are limitations to our current implementation. For example, the system's performance can be affected by the size of the input dataset, and the reliance on static pre-processed data might not reflect real-time changes in user behavior. Moreover, while our similarity metrics work well for a controlled dataset, the model may require further calibration when exposed to more diverse and dynamic movie data.

Looking forward, there are several potential enhancements. Future work could include integrating real-time user feedback to continuously update and refine the graph, implementing more sophisticated machine learning techniques to predict user preferences, and extending the GUI to provide additional interactive visualizations of the recommendation network. Additionally, experimenting with different graph-based algorithms might yield even better personalization results.

Overall, this project has not only provided a robust system for movie recommendations but also offered significant insights into the benefits and challenges of graph-based data modeling. The iterative improvements made throughout the project demonstrate the importance of both user feedback and computational efficiency in developing an effective recommendation system.

7 References

References

[1] Center, Netflix Help. (n.d.). *How Netflix's Recommendations System Works*. Accessed March 4, 2025. `https://help.netflix.com/en/node/100639/`.

[2] Fitzgerald, T. (2019). *How Many Streaming Video Services Does The Average Person Subscribe To?* Accessed March 4, 2025. `https://www.forbes.com/sites/tonifitzgerald/2019/03/29/`.

[3] GroupLens. (n.d.). *MovieLens*. Accessed March 4, 2025. `https://grouplens.org/datasets/movielens/`.

[4] IMDb. (n.d.). *IMDb*. Retrieved from `https://www.imdb.com/`.