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 [1]. 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 using libraries such as pandas and tkinter, 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 [2]. 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. While we originally planned to use external APIs to fetch additional information about the movies, we determined that the time cost far outweighed its benefits for this project.

Specifically, our system processes:

- **ratings.csv:** Contains individual user reviews for movies. Each line contains a **userId**, **movieId**, and **rating**, alongside a timestamp we have not utilized.
- **movies.csv:** Maps the **movieId**'s found in *ratings.csv* to tangible titles. Also provides each movies' genre.

Only relevant columns (e.g., `movieId`, `rating`, `genres`) are used to build the graph. We also utilize a smaller provided dataset for faster recommendations at the potential expense of accuracy and movie breadth.

3 Computational Overview

Our project contains several modules, but can generally be separated into 3 categories:

1. **Backend:** The `graph_building.py` module is responsible solely for building a ReviewGraph from the dataset. It contains just 2 functions, `load_movies_from_file` and `load_reviews_from_file`, which are fairly self-explanatory. The rest of the action occurs in the `data_types.py` module, which contains the Graph and Vertex classes. We've implemented distinct `_Movie` and `_User` classes, both of which inherit from the parent `_Vertex` class. All vertices contain a set of `Review` objects, which simply hold a User, Movie, and a rating (float). Movie vertices additionally hold a title, year of release, and a set of genres. They also implement many additional methods, such as `avg_review_score` and `bayesian_weighted_score`. The only extra piece of information User vertices hold is a user_id. Finally, the `ReviewGraph` class works similarly to the graphs we used in Exercises 3 and 4 but with many additional methods added. The most notable additions are `recommend_by_genre`, which recommends highly-rated movies which match a set of provided genres and `recommend_by_similarity`, which recommends movies similar to a given set of movies.
2. **API:** To make everything described above available to the user, `api.py` defines a `BackendInstance` class, which allows the UI to easily create a graph from a provided dataset, and query the graph as necessary. Most of this class simply calls the relevant functions from `ReviewGraph`, but it also ensures data is formatted in a way the frontend can easily work with.
3. **Frontend:** 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. Utilizing the API when necessary, the GUI allows the user to generate recommendations

with ease, only requiring traversing through a few menus. Furthermore, the frontend allows users to save their watched movies to a file, so when they look to generate new recommendations, they need not refill information.

On the backend, we used the *pandas* library to efficiently load data from the file into our graph. Both functions in `graph_building.py` make use of `pandas.read_csv()`, which provides a "dataframe" that can be iterated over to read each row. On the frontend, we made frequent used of *tkinter* to implement the GUI. Tkinter acts as an interface to **Tcl/Tk**, which is a widely-used GUI toolkit. Across the GUI, we used components like Frames, Labels, and Listboxes to provide data to the user in intuitive ways. For example, in the `preferences()` window function, a listbox is used to display all movie genres represented in the dataset (which is grabbed through the API) and allow the user to select their preferences.

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:** Please download and extract the zip file located here. (Note: If you cannot access the file, please ensure you are logged in with your UofT credentials. The file is accessible only to those within UofT). It should be placed in the root folder of the project (that is, the folder named "data" should sit in the same directory as `main.py`).
3. **Execution:** From the command line or within your IDE, run the `main.py` file. In main.py, you can specify which dataset to use, selecting either *ml-latest-small* or *ml-32m*. (Note: when the 32m dataset is selected, `graph_building.py` is configured to only parse the first 1 million reviews, as the full dataset takes upwards of 8 GB of RAM and multiple minutes to digest)
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 Implementation:** We created two methods for generating recommendations, one based solely on a user's preferred movie genres and another based on movies they've previously enjoyed. To achieve the first, movies are ranked based on their *bayesian_weighted_ratings*, which helps to prevent movies with a small number of highly rated reviews from being overrated in recommendations. It does so by effectively adding "fake" reviews to movies. By adding extra reviews which are weighted to the average movie rating in the dataset, movies with few ratings tend towards the average instead of standing alone at the top or bottom of the rankings. The second algorithm takes a different approach than that showcased in Exercise 4. Instead of comparing the user to others in the dataset, their preferred movies themselves are compared to others in the dataset (that is, similar by genre), and those which are both rated highly and similar to multiple movies liked by the user are rated highest. We chose to take this approach as we found that it was fairly likely to make recommendations we viewed as "good" and avoided recommending obscure movies we didn't recognize.

- **Supplemental Data:** While we originally hoped to have advanced information accessible through the GUI, such as movie trailers, actors, etc., we realized as the project continued that implementing these features was not feasible with our current skills and in the necessary timeframe. We've still made full use of the information provided by the dataset, as we've relied heavily on the included genres for recommendation purposes.
- **Interface Development:** As originally planned, we used Tkinter to implement our GUI. While we originally hoped to have more information available to the user than we were able to provide, we did build upon our original plans by adding a login system which saves a user's watched movies to disk for future use.
- **Performance Optimizations:** As recommended by our TA, we shifted to using pandas to manipulate our data for use in our graph.

6 Discussion

Our original goal with this project was to evaluate how graph-based models can be used to improve the personalization of recommendation algorithms. Through our exploration, we have learned both the pros and cons of this approach.

For starters, Graph-based algorithms are indeed incredibly powerful, especially when built upon a large dataset. With great care and tuning, you can use graphs to generate very accurate recommendations for individual users. By comparing the similarity of movies and users, you can estimate what other users are *like* your target, then use other movies they like as recommendations for your user. In this way, graphs are great!

That said, the "great care and tuning" required is very real. Because the question of "What movies would this user enjoy?" is so open-ended, there are countless ways to generate recommendations from the "magic box" that is an incredibly large graph. Some ways will quickly reveal themselves as poor solutions, while others will appear to be very good until the last minute. This was largely the case with our implementation, as the idea of using genres and weighted ratings seemed to provide great results until we compared it with the alternative we learned in Exercise 4.

Truly, the greatest takeaway from this project is the importance of trying a variety of outside-the-box solutions. Devoid of comparison, any solution can appear to be a good one. Only by trying multiple methods and comparing can you truly evaluate your implementations.

Our first major speed bump came very early as we looked to decide how to format the data into a graph. Ultimately, we decided to create a Review class to serve as the edges of the graph, holding references to both the user and movie vertices, as well as the review rating.

With this decision made, next came designing the recommendation algorithms in the graph. While we at first tried a system similar to that used in Exercise 3, we ran into problems with incorrect recommendations and inconsistent results. Thus, we pivoted to our current approach, making greater use of the average rating of each movie alongside its genres to make recommendations.

However, this came with its own set of issues, as we quickly realized that all of the recommended movies were very obscure. This was because these movies, given their obscurity, had only one or two reviews in the dataset. If these reviews were 4-5 stars, the movies would have an incredibly high average rating, placing them above more established movies, which have a variety of ratings. After some research, we came across the Bayesian average method, which incorporates a pre-existing belief to normalize ratings. In essence, the method "fibs" a bunch of average ratings onto each movie, shifting them towards the mean. This affects movies with a small number of ratings the most, as their average ends up being very close to the given average. Conversely, movies with a multitude of ratings aren't affected much, as the fibbed ratings make up only a small portion of the total.

When it comes to the GUI, we ran into a variety of mostly small issues throughout our implementation.

Firstly, we realized we had vastly overestimated the power of tkinter. Implementing things such as video playback and collapsing menus requires significant work or the use of external packages, which often lack proper documentation.

Next, tkinter has many pedantic limitations, which caused our implementation to feel fairly "chunky". Because everything is tied to a root window, we were forced to have subsequent menus open on top of the original menu instead of replacing it. While there are likely ways to circumvent this limitation, we didn't find it detrimental enough to the user experience to worry about it.

Finally, we also spent a long time figuring out how to get the back end to communicate with the front end. Because the two modules were completed separately, when it came to combining them, significant changes were required. Due to some miscommunication, much of the front-end implementation didn't work with the API developed by the backend. As a result, we needed to redesign many functions to make use of the single backend instance and to allow the results to be displayed back to the user.

Ultimately, this project was a great introduction to the complexities of not just recommendation algorithms but also full-stack development. We tackled algorithmic problems, implementation incompatibilities, and programming in a group environment. In the future, we could explore additional algorithms for generating recommendations, allowing us to gain an even deeper understanding of what makes recommendation algorithms "good". We could also spend more time redesigning the GUI to provide additional functionality and improve aesthetics. Finally, we could develop a better strategy for tackling large projects in groups, utilizing better communication and delegation skills to get more done faster.

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