CSC111 Project Report: Films for You- A Movie Recommendation System

Shih-Hsin Chuang and Juliana Zhang

Monday, April 3, 2023

Introduction

Movie nights are a tradition for many families and friends alike. However, choosing the right movie to watch is often a struggle. In order to help people effectively decide on a film, we became motivated to build a movie recommendation system that addresses the following question: **How can we optimize the movie selection process by making recommendations based on user movie history and community reviews?**

A movie recommendation system is an algorithm that filters and recommends movies based on how likely a user will want to watch them (Agrawal). There are two main filtration strategies: content-based and collaborative filtering. *Films for You* uses collaborative filtering to offer movie recommendations by comparing a user's behaviour to others in the system. This involves finding users with similar movie patterns, and recommending some of the other movies they've liked to the current user (Kniazieva).

Films for you has two configurations. The first configuration allows users to receive movie recommendations. To begin, the user selects three movies they've watched and liked in the past from a list of fifty most popular movies. The system will then compare their movie history with that of existing users and check for similarity. In the context of our system, the notion of a 'similar user' is someone who has written reviews on the same movies selected by the user. The more movies that users have in common, the more similar they are. The current user will also be able to specify their preferred movie genre, which allows the system to further narrow down the options. At the end of the interaction, the system will provide a ranking of the top three films it recommends to the user.

The second configuration allows users to add their own movie review. The system starts by assigning the user a distinct id and prompting them to enter the name of the movie they wish to review. Then, it will ask the user to rate the movie out of 10.0.

Datasets

The data for the movies and users will be drawn from "The Movies Dataset", which is a real-world dataset that contains over 45, 000 movies and 26 million ratings from over 270, 000 users (Rounak Banik). In particular, we will use the movies_metadata.csv and ratings_small.csv files, which contain information on genre, ids, title, country of origin, release year, ratings, etc. The columns we are using for the movies_metadata.csv include id, title, genres, vote_average, and vote_count while userId, movieId, and rating from the ratings_small.csv will be used. Since we want to recommend the movies with high ratings to the users, we only select the movies with an average vote of 5.0/10.0 or above from the movies_metadata.csv and ratings 3.0/5.0 or above from the ratings_small.csv. Lastly, we only consider the movies that are contained in both files to ensure that the movies in the subset of data all have user ratings.

Computational Overview

To represent movies and users, we create two data classes: `class RecommendationSystem` and `class Vertex`, which is an abstract class that can be subclassed to implement different types of vertices (User or Movie). The RecommendationSystem class stores all vertices (movies and users) in a dictionary instance attribute where each movie/user id maps to its corresponding Movie/User vertex. For both User and Movie subclasses, they have attributes for the user/movie id and a set of connected movies/users vertices, but Movie class also includes additional attributes for title, genre, average rating, and number of users, which are useful for filtering recommendations later on.

The computations mainly involve building the recommendation system (the graph) to represent connections between movies and users and selecting movies based on similar past users, ratings, and genres. Before building the initial state of the system, `movie_file_reading` and `rating_file_reading` from `file_reading.py` are used to read and select the needed columns from the `movies_metadata.csv` and `ratings_small.csv`, respectively. The return values for both functions are a list of lists, where each inner list represents a row in the csv files. Another function, `combined_files`, returns a list just like `movie_file_reading` but with an additional column in each inner list that contains a list of linked users, which are determined by the ratings from `rating_file_reading`. We are using the movies and ratings returned by `combined_files` for our system, and the ones that do not have user reviews are excluded from the list so that all the movies we are using are ensured to have user reviews.

The initial state of the system will be built by using `add_movies_users` function in `recommendation_system.py` to add movies and users as vertices and create an edge between each user and each of their rated movies. Since `combined_files` has each movie id and its connected users, we simply run through the list, and for each movie, we loop through its linked users and add an edge between the user and the movie.

Next, if the user selects the first configuration to get movie recommendations, we compute a list of 50 most popular movies from the dataset given, with popularity defined by the movie's average rating and number of votes. Then, the user selects three movies that they enjoyed watching in the past.

Given the graph, the system searches for past users who are linked to at least one of the three liked movies using `return_similar_users`. The users who are connected to all three movies are assigned three points, and the users with two shared movies receive two points, and so on. We give users points by mutating their instant attribute `self.point`.

With a set of similar users, we call the function `return_movies` to obtain a set of at least 50 recommended movies, which are selected from the movies linked to the users with three points first. If the number of movies returned is less than 50, we append more movies to the list based on the users with two points, and the same process repeats for the users with one point if needed. Finally, we randomly select movies from `combined_files` to reach a list of 50 movies if the number of movies obtained from the similar users is still less than 50.

After obtaining a list of 50 movies, the recommendation system narrows down the list to three according to genre the user specifies and each movie's average rating using `apply_filters`. The system first looks for movies that belong to the same genre by checking the `genre` instance attribute then returns the three with the highest average ratings. If there are less than three movies that match the genre, we simply fill the list with the highest rating movies.

If the user selects the second configuration to add a review, a user vertex with a distinct id is added to modify the graph through the function `add_reviews`. Then, we check whether the movie being reviewed already exists in order to add an edge. If it does, the movie's average rating is recalculated by considering the user's rating as well. If it doesn't, the movie vertex is created first, and the user inputted rating will be its current average rating.

The interactive aspect of our program is a graphical user interface created using the Tkinter library (code in `interface.py`). The interface allows the user to visually interact with each of the system's configurations once.

To navigate between the configurations, our GUI has a tab bar with three tabs: 'Home', 'Movie Recommendations', and 'Write a Review'. This was created using the Notebook widget, which manages a collection of tabs and displays them one at a time ("tkinter.Notebook"). The Frame widget was used to create these tabs. Additionally, we also used the Pathlib library to load the Films for You logo onto our GUI without relying on the specific laptop being used.

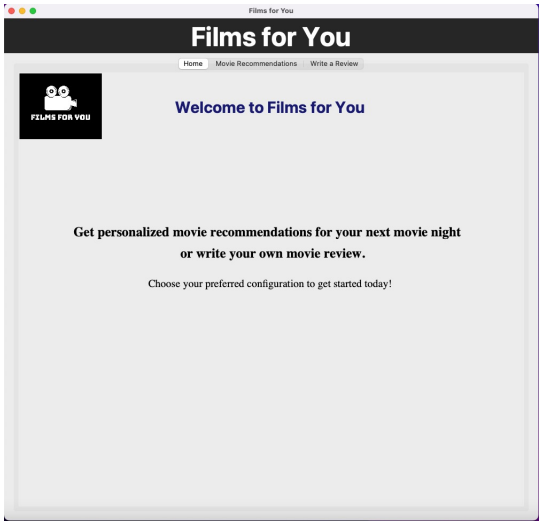
In the 'Movie Recommendation' tab, the user will first be asked to select three movie checkboxes from the 50 most popular movies in the given dataset. This was done by looping through the list returned by the `popular_movies` function in `computation.py`, and making a checkbox for each movie in the list. When a checkbox is selected, it triggers the command to append the corresponding movie to a list containing the user's top three choices. After the user is done selecting their movie history, they can press the "Enter" button, which clears the frame and calls the function `get_filter_frame()`. The `get_filter_frame()` function contains code that prompts the user to select their preferred genre from a drop-down menu. The interface stores the user's selection in an instance of the Tkinter variable class, `StringVar`. There is another "Enter" button, which calls the `get_recommended_movies()` function. This function passes in the user's movie history and preferred genre to compute the three highest scoring movie recommendations. The final movie recommendations are displayed on the interface along with their genres and ratings, ending the interaction.

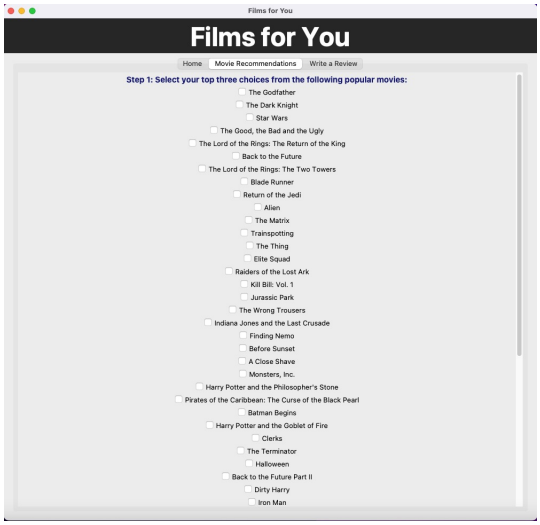
In the 'Write a Review' tab, there are two Entry widgets that allow the user to enter the name of the movie they wish to review and their rating out of 10.0. There is also an "Enter" button that triggers the `add_review()` function when pressed, which adds the reviewed movie into the system's graph and ends the interaction.

Obtaining Datasets and Running the Program

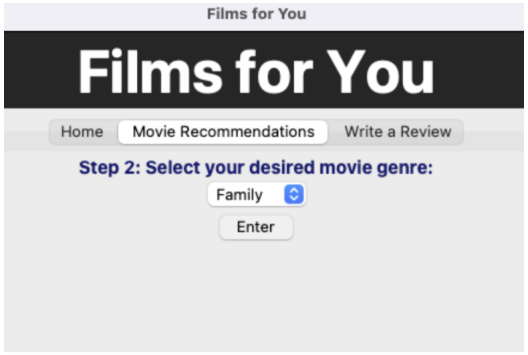
Instructions on how to download the zip file:
We choose the second option of sharing the zip file with the course email address.

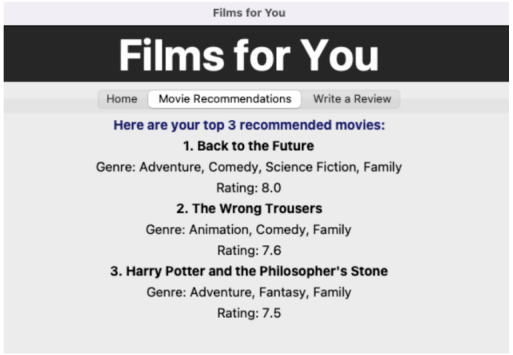
Upon running `main.py`, the following Tkinter GUI should appear. The Logo.ppm image should be located in the same folder as `interface.py` in order for the GUI to load without an error. To access the different interactive features and system configurations, the user can click through the tab bar.



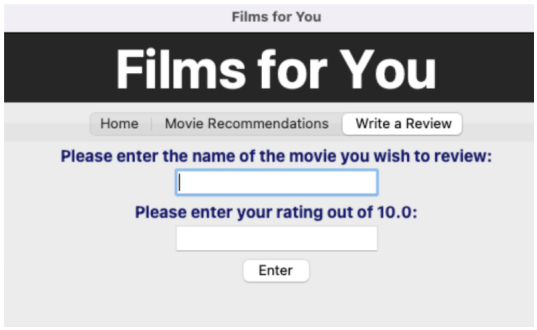


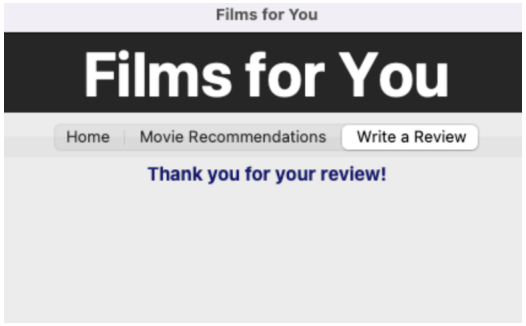
In the 'Movie Recommendation' tab, the user should expect to see the 50 checkboxes of the most popular movies from the dataset. They can also manually drag the scroll bar to access all the checkboxes and the 'Enter' button. After pressing the 'Enter' button, the next screen with the genre drop-down menu will appear. By selecting the desired movie genre and pressing enter, the final top three recommended movies will be displayed along with their associated genres and ratings out of 10.0.





In the 'Write a Review' tab, the user will see two entry boxes prompting them to enter the name of the movie they wish to review and their rating out of 10.0. After pressing 'Enter', they will see a thank you screen that signals the end of the interaction.





Changes to Project Plan

Originally, we allowed the user to manually input the three movies that they enjoyed watching in the past, but we realized that those may not be included in our dataset. Therefore, we now select the 50 most popular movies from the existing movie list, and the user chooses three among them.

With the three chosen movies, we still look for the similar past users and return their connected movies. However, we added a `point` instance attribute to the user vertex to represent the number of movies that overlap with the three chosen ones. The `point` instance attribute allows us to rank and prioritize the past users whose movies are more likely to be liked by the current user. If the number of movies returned is less than 50, rather than taking the previous movie list and look for their users' connected movies, we simply randomly add more movies until the list is 50.

In addition, the filter that the user can apply is now changed to genre only because specifying all three features, the country of production, release year, and genre, makes the selection process too specific; it is very unlikely to obtain movies that satisfy all the criteria.

Finally, based on TA's feedback, we added a testing section to evaluate the accuracy of our movie recommendations using training and testing sets. This way, we can determine whether our algorithm actually recommends users the movies that they are likely to enjoy watching.

Discussion

In order to analyze the extent to which our computational exploration helps answer the research question, we divided the movie dataset from Kaggle into training (80%) and testing (20%) sets. The training dataset was used to run the recommendation system, while the testing data set was used to evaluate its accuracy. The

movies and users in the testing set were kept separate from those in the training set.

To begin, we computed each testing user's connected/reviewed movies with ratings above 5.0. Then, we narrowed down the testing users to those whose connected movie list contains at least 3 popular movies from the system's list of top 50 popular ones. From here, we selected three options from the overlapping movies. This was done to simulate the first step of the recommendation process, where the user selects their movie history. Then, we passed the three movies into the `get_recommended_movies()` function to obtain a list of all the recommended movies. Finally, we calculated the percentage of overlapping movies between the user's reviewed movies—minus 3 to account for the previously selected ones—and the recommended movies. This yielded the following results:

```
>>> testing_recommendation_accuracy('data/movies_metadata.csv', 'data/ratings_small_without_testing.csv', 'data/testing_ratings.csv')
...
[0.667, 1.0, 0.72, 0.476, 0.87, 1.0, 0.875, 1.0, 1.0, 1.0, 0.875, 1.0, 0.833, 0.875, 0.667, 0.8, 0.833, 0.714, 0.588, 1.0, 1.0]
```

As seen, most of the testing users had an accuracy percentage above 0.5 (50%), and some had an accuracy percentage of 1.0 (100%). This means that most of the recommended movies were already among the testing user's reviewed movies that received a rating above 5.0. As a result, the user would've hypothetically enjoyed watching the suggested movie since they gave a relatively positive review on it. Therefore, our system has the ability to filter and recommend movies based on how likely a user will want to watch them.

One of the limitations with our algorithm is its long running time due to the large datasets. The function `combined_files` takes a long time to run because it gets connected users for all the movies returned from `movie_file_reading`. Therefore, for each movie, we loop through all the user ratings and add the ones for the movie into the set. This results in long running time for other functions that call `combined_files` in their body as well.

Furthermore, it was challenging to determine how exactly our algorithm should evaluate the "similarity" between users. Our initial computational strategy returned very random movie selections, which suggested that our algorithm was not giving the best recommendations. Eventually, we took the movies' average ratings into consideration, and it resulted in higher accuracy based on the testing algorithm.

Moreover, we encountered limitations with the Tkinter library when connecting the frontend to the backend of the program. We found it particularly challenging to get and store the value of the user's input in order to pass it to the computational functions. For instance, the widgets in the interface had to be coordinated with commands that trigger different events, such as the `.get()` function that gets the value of the user's input. We often had to convert the retrieved user input into strings or a list of strings to match the type contract of the computational functions as well. Due to this unfamiliarity with the Tkinter library, we were only able to interactively display one run-through of the recommendation or review process instead of allowing the user to go back (console allows for repeated actions).

One area for further exploration is to change the interactive user interface so that it can accept more than one run-through each time. Although our graph methods allow the user to add reviews and receive recommendations repeatedly under the same system, the user interface finishes after one interaction. Therefore, a next step would be to enable the user to go back on the GUI.

Another step for further exploration is to display the graph network visually in a Plotly graph. Doing so will allow the user to see the path through which their recommended movies came from. In addition, when a user adds a review to the current system, they can only use the `return_avg_rating()` method to verify that the average rating of the movie has changed. However, it would be meaningful to display the users' contributions in a visual way as well.

Together, *Films for You* gives people the opportunity to obtain recommendations, or contribute to the movie community by adding reviews and playing a part in future recommendation processes.

References

Agrawal, Shubham Kumar. "Recommendation System -Understanding The Basic Concepts." *Analytics Vidhya*, 13 July 2021, https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/. Accessed 6 March 2023.

Amos, David. "Python GUI Programming With Tkinter – Real Python." *Real Python*, 30 March 2022, https://realpython.com/python-gui-tkinter/. Accessed 6 March 2023. Banik, Rounak. "The Movies Dataset." *Kaggle*, https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset?select=movies_metadata.csv. Accessed 6 March 2023.

Kniazieva, Yuliia. "Guide to Movie Recommendation Systems Using Machine Learning." *Label Your Data*, 14 April 2022, https://labelyourdata.com/articles/movie-recommendation-with-machine-learning. Accessed 6 March 2023.

"Pandas - Analyzing DataFrames." *W3Schools*, https://www.w3schools.com/python/pandas/pandas_analyzing.asp. Accessed 6 March 2023.

"pandas.DataFrame.loc — pandas 1.5.3 documentation." *Pandas*, https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html. Accessed 6 March 2023.

Selvaraj, Natassha. "How to Build a Recommendation System in Python?" *365 Data Science*, 16 September 2022, https://365datascience.com/tutorials/how-to-build-recommendation-system-in-python/#2. Accessed 6 March 2023.

"tkinter — Python interface to Tcl/Tk — Python 3.11.2 documentation." *Python Docs*, https://docs.python.org/3/library/tkinter.html. Accessed 6 March 2023.

"tkinter.Notebook." *the Tcler's Wiki!*, https://wiki.tcl-lang.org/page/tkinter.Notebook. Accessed 2 April 2023.