CSC111 Project Report: **MovieMatch** (Movie Recommendation System Using Graphs)

Christoffer Tan, Faraaz Ahmed, Razan Rifandi

Monday, April 3, 2023

Introduction

Using community reviews of movies, what movies can we recommend people to watch based on their previously watched movies?

One key aspect that people enjoy when using Netflix is its recommendation system. It analyzes the user's watch history and provides similar content to watch, which keeps users hooked on using the platform. Therefore, for Netflix, its movie/show recommendation system is crucial. We have implemented a movie recommendation system that takes into consideration numerous ratings given by various users. Using this information, along with the genre of the movie, the system generates a list of movies that a user can watch next based on the movies they previously watched.

To understand how our recommendation program works, let us consider the following hypothetical situation: Bob likes to watch movies but cannot decide what to watch next. His favorite movies include Harry Potter: Prisoner of Azkaban, The Hobbit, and The Lord of the Rings. Our program, MovieMatch, uses this information to recommend a list of movies for Bob to watch next. It considers the ratings given by other users who watched these movies, as well as the genre of these movies, such as "fantasy". If the recommended movie list contains movies that Bob has already watched, he can opt for another list of recommendations.

Datasets

The dataset we have used is MovieLens (GroupLens, 2019), which includes about 100,000 ratings by about 600 users to about 9,000 movies. The dataset is titled as the small "MovieLens Latest Datasets" which includes ratings of movies released up to 2018. The datasets are in the csv file format and both the movie and user rating datasets that our program uses have been included in our dataset zip file. Our program uses all the columns of the movies.csv file which includes "movieId", "title", "genres" and it also uses the "userId", "movieId", "rating" columns of the rating.csv file.

Computational Plan

Recommender systems are information filtering systems that assist in addressing the issue of information overload by filtering and grouping information and extracting fragments from vast amounts of dynamically generated data in accordance with user preferences, interests, or observed behavior regarding a particular item or items. The two major ways to use recommendation systems are collaborative filtering and content-based filtering. The method in which we are going to implement MovieMatch is through combining these two methods, which uses similarities in content creatures and the past interactions that have been recorded between user and items, in order to produce new recommendations.

Based on the data set produced by GroupLens (GroupLens, 2019), there are about 600 users who gave ratings to about 100,000 movies with the information of their titles and genres. We will be representing the data using graphs, with two types of vertex class:

1. `User`. The `User` vertex represents each user with instance attributes – the `user_id` and the movie(s) he/she has watched with the corresponding ratings. This vertex only connects to the `Movie` vertex with the ratings as the weight of the edge (0.0 to 5.0 **inclusive**).
2. The `Movie` vertex represents each movie with instance attributes – the `movie_id`, the `title`, the `genre`, and the user(s) that have watched this movie with the corresponding ratings. This vertex only connects to the `User` vertex with the ratings as the weight of the edge (0.0 to 5.0 **inclusive**).

To represent the graph of the recommendations, we create a `RatingGraph` class with two private instance attributes, `_users` and `_movies`, which are mapping their id to their corresponding node. This class has several important methods to notice:

1. `add_users`: method used to add a new user with the given `user_id` to this graph.
2. `add_movies`: method used to add a new movie with the given `movie_id`, `title` and `genre` to this graph.
3. `add_edge`: method used to a new edge between a `User` and the `Movie` he/she has watched with the corresponding rating as the weight.

The graph is generated by utilizing the datasets "ratings.csv" and "movies.csv," which are included in the dataset zip folder. The dataset includes about 100,000 ratings by 600 users for about 9,000 movies. The graph is created by using the method `create_graph`, which takes two parameters: the csv file for user ratings and the csv file for movies. This method extensively utilizes the `pandas` (Pandas, 2018) library. The movie file and user file are converted to data frames using `pandas.read_csv`. Then, the data frame is iterated through using the ".index" attribute present in the data frames, and the above methods `add_movies`, `add_users`, and `add_edges` are called for each iteration to generate the graph. Using the `pandas` library has helped us to directly access each of the csv columns such as the "movieId" and "title" columns and efficiently retrieve multiple values for each movie to generate the `Movie` vertex and `User` vertex that we added to the graph. The `create_graph` method returns the `RatingGraph` object, explained below.

Consider this following graph of four movies evaluated by four users:

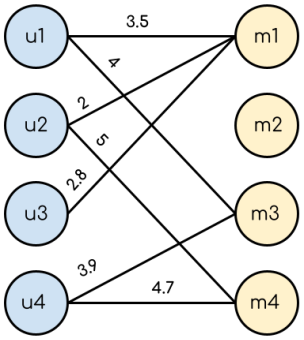

Figure 1: Graph connecting users and movies with ratings

Based on this graph, the first user gives a rating of 3.5 and 4 to the first movie and the third movie. Meanwhile, the third user only gives a rating to the first movie with a value of 2.8. After representing the data set in a graph, we will be determining the recommended movie for each user in it using cosine similarity to determine the most similar movies.

We implemented the function `compute_cosine_similarity` to determine the cosine similarity value of two `Movie` objects based on the following formula:

$$cos\,\theta = \frac{M_1 \dot{M}_2}{||M_1||||M_2||} = \frac{\Sigma_{i-1}^{n} M_{1i} M_{2i}}{\Sigma_{i-1}^{n} M_{1i}^2 \Sigma_{i-1}^{n} M_{2i}^2}$$

The function works by creating two lists for each movie containing the ratings of all users who have watched both movies. These lists represent the ratings vector, in which the value at index `i` is the rating given for the movie by a user `i`. The function then computes the dot product using a list comprehension that multiplies the corresponding elements of each list and summed them and the norms of each vector by taking the square root of the sum of squares of the ratings. After that, the cosine similarity is calculated as the dot product divided by the product of the norms of the two vectors and returned. If fewer than two users have watched both movies, or if both movie nodes refers to the same movie, the function returns `None` as an output.

The `arrange_cosine_similarity` function computes the cosine similarity value between the given movie and all other movies in the graph using the `compute_cosine_similarity` function. The cosine similarities are stored in a list of tuples, where the first element of each tuple is the cosine similarity value and the second element is the name of the corresponding movie. If the cosine similarity value is `None`, we do not add it to the list, taking into account cases when the movie being compared is the same as the original movie or when there are fewer than two users who have watched both movies. This list is then sorted in ascending order based on the cosine similarity value. Additionally, the function also takes in a boolean variable `compare_genres`, which specifies whether to add only movies with the same genre into the list if it is `True`.

Based on the two previous functions, the function `recommendations` is used to determine the movies that we will recommend to the users. The function calls `arrange_cosine_similarities` for three movies that are inputted by the user to accumulate similar movies, prioritizing movies with the same genre. If there are not enough similar movies, the function will recompute the similar movies without taking the genre into account. Then, the function will recommend five movies according to the steps explained below.

In order to recommend new movies, a random user needs to input the movies they liked through an interactive window. The user needs to enter three movies in the entry boxes of the window opened when the code is run. The user can input the movie by searching the name of the movie by typing it in the entry box and then selecting the name of the movie from the list box below. Then the user will press the button labeled as "Get Recommendations!". The GUI is created using the python `tkinter` module (Python Software Foundation, 2019).

The GUI includes 2 buttons to get recommendations and delete recommendations, 3 entry boxes to input the watched movies, a list box to show the list of movies that are available in the dataset and a text box to output the recommendations. When a user enters characters in one of the entry boxes, the list box of movies updates to show the movies present in the dataset that has those characters. This is done by the `search` function defined in the code.

Given up to three movies that a random user likes, we will arrange the five most similar movies for each of the movies according to the genre and the computed cosine similarity. Based on these variables, the recommendation system will recommend new movies with these following steps:

1. For every movie that a random user likes, the system will select the top five movies that are most similar. These movies should have the same genre and the highest cosine similarity with the movie that the user likes.
2. If the system does not have enough movies to recommend, it will select the top five most similar movies based only on cosine similarity for each watched movie.
3. Nevertheless, the recommended movies should not have been watched by the user, and they should not contain any duplicates.

4. Then, from the pool of the recommended movies, the system will take five movies randomly.

The recommended movies will be displayed in the Text Box. The user can opt for more recommendations by pressing the "Get Recommendations!" button again or can delete the recommendations by pressing the "Delete" button.

Instructions for Obtaining Datasets and Running the Program

To obtain the dataset please extract the datasets zip file we have provided and copy the datasets folder in the same location as `main.py, rating_graph.py, gui.py and recommender.py`:

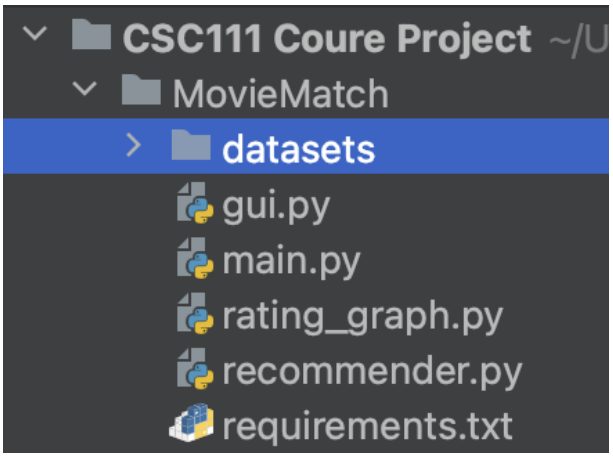

Figure 2: The datasets folder should be in the same location as `main.py, rating_graph.py, gui.py and recommender.py`

To run the program simply run the `main.py` file and a `tkinter` window should pop up as shown below:

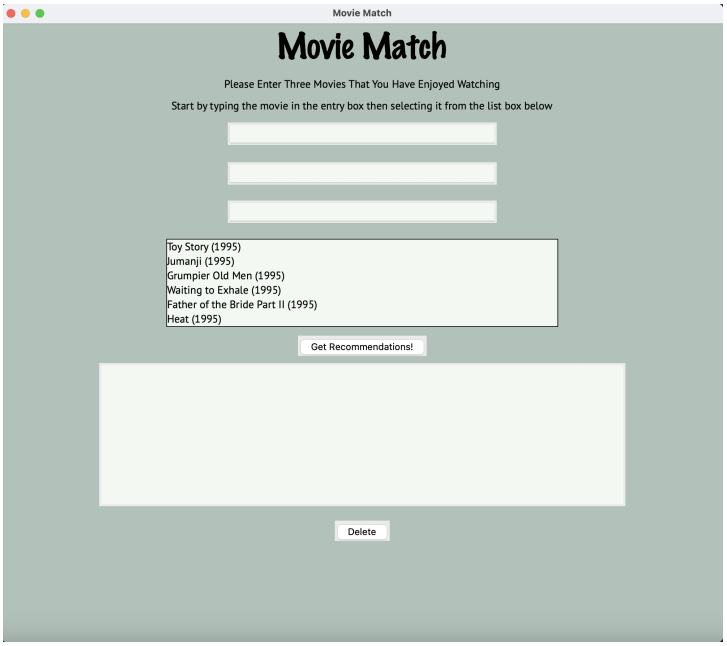

Figure 3: The startup window of Movie Match

To use the program, enter three movies in the entry boxes by typing in the entry boxes and then selecting the respective movie in the list box. Then press the "Get Recommendations!" button to display the recommendations:

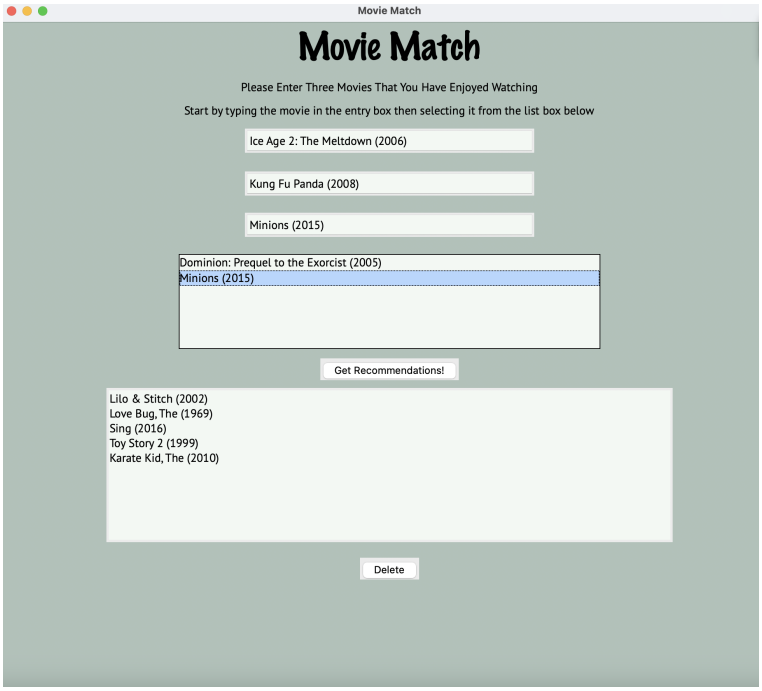

Figure 4: The results of Movie Match displayed in the text box

Please note that the recommendations may vary each time as we are providing a random sample of 5 movies with the highest cosine similarity with the same genre every time.

Changes to the Project Plan

After completing the project proposal, the project faced several challenges that required us to make changes to our project plan. To provide a summary, the following are the problems that arose in our project and the changes we took to address them:

1. In the project proposal, the result of the cosine similarities is stored into a new graph that connects `Movie` vertices to each other with the value of the cosine similarity as the weight of the edge. However, we realize that this representation will be too complex since the recommendation system only needs the five most similar movies of every watched movie. Therefore, the result of the cosine similarities is stored in a `list` of `tuples` – that consists of the `Movie` it is compared with and the corresponding cosine similarity. Then, in order to get the five most similar movies, the system only needs to sort the list based on the value of the cosine similarity.

2. Initially, based on our project proposal, the recommendation system was implemented solely using the ratings recorded between users and movies. However, due to the small data set size and the potential bias in the way we calculated cosine similarity, the recommended movies were not accurate. Therefore, we decided to combine content-based filtering and take the genre of movies into consideration in the process of recommending new movies.

Discussions

From our results, we have successfully addressed our initial question of how to create a movie recommender system using a dataset of community-based reviews. We achieved this by utilizing a graph data structure to store the dataset and applying the cosine similarity algorithm to determine similar movies for recommendation. However, we observed that our results were not very accurate, even after considering movie genres. Ultimately, we attributed this to the small size of the dataset we used, which is only a subset of the complete dataset. To improve our system's accuracy, we could switch to a larger dataset. However, we determined that it is infeasible to do so for this project due to the longer computation time required. Furthermore, our recommender system only takes into account the genre and rating (based on the cosine similarity value) of the movies, which we assumed to be correlated with how "similar" the movies actually are. Therefore, we could explore further ways to improve our recommender system by taking additional factors into account, such as user preferences and movie features, and potentially implementing other algorithms, such as Machine Learning.

References

- Li, C., Chen, J., & Wan, Y. Building Music Recommender Systems with Graph-based Method. University of California San Diego (UCSD). Retrieved March 6, 2023, from https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/008.pdf.
- (2019, April 26). GroupLens. https://grouplens.org/datasets/movielens/ [Dataset]
- Arefyev, G. (2019, April 3). Shortest Path Similarity: A Fresh Breath to Item-based Recommendations. Towards Data Science. https://towardsdatascience.com/shortest-path-similarity-a-fresh-breath-to-item-based-recommendations-9ac3d6ba7240.
- Pandas. (2018). Python Data Analysis Library — pandas: Python Data Analysis Library. Pydata.org. https://pandas.pydata.org/.
- Python Software Foundation. (2019). tkinter — Python interface to Tcl/Tk — Python 3.7.2 documentation. Python.org. https://docs.python.org/3/library/tkinter.html.