CSC111 Project Report: Creating Meaningful Movie Recommendations with IMDB reviews

Mojan Majid, Kimiya Raminrad, Michael Galloro, Rafee Rahman

Friday, April 16, 2021

Problem Description and Research Question

Recommendation systems have become a great way for people to discover new things. There are recommendation systems for all variety of subjects: music on Spotify, video games on Steam, books on Good Reads, and movies on IMDB. These systems employ a variety of different methods, but the goal is the same: to help people discover things they will enjoy. The method employed in each case depends on what data is available. For movies, there are genres for example, and a simple algorithm could take the genre of some movies the user enjoyed and recommend new movies within that genre. However, this simple system would not create satisfying recommendations. A person's tastes are often more complicated and specific than just watching one genre. The question then becomes, how can we come up with a system that recommends movies that the user has a good chance of enjoying, and accounts for a person's complicated tastes?

With the advent of user reviews, there is no shortage of user data for movies. We will be using a huge database of over five million movie reviews to form the foundation of our recommendation system. Our aim is to connect reviewers together, by recommending movies that other users with shared interests have enjoyed. For example, if two users Jim and Alex have reviewed movies A, B, and C highly, and Jim has also reviewed movie D highly, our recommendation system will recommend movie D for Alex. In this way, instead of depending on simplistic attributes of a movie to recommend new ones, we can leverage the power of a massive review database to help connect people with others that have similar taste to them. Thus, our project question is as follows: **How can we use movie reviews to make meaningful new movie recommendations for people?**

During implementation we found that our original game plan worked well and didn't need revising. Recommending movies in the way that we described above went as planned, as we will describe further in the next sections. We also found that the dataset we chose lent itself to answering our project question. Due to its massive size we were able to find many reviewers of varying and distinguished movie tastes with many reviews, some as high as a thousand. This means that there is a great range of possible user movie inputs for which we can provide meaningful movie recommendations.

Dataset Description

- Dataset: IMDb Largest Review Dataset
  Source: Kaggle(link)
  Format: JSON files
  Description: The IMDb largest Review Dataset is a comprehensive dataset of more than 5 million movie reviews collected from IMDb. The dataset contains 7 JSON files, sample.json, part-01.json, part-02.json, etc. Each file has data on nine variables including `reviewer_id`, `movie`, `spoiler_tag`, etc (see figure 1).

- Dataset: imdb_reviews.json (download zip file)
  Format: JSON file
  Description: This file is created by calling the functions in cleaning_data.py. To create this file, we first construct a DataFrame from the 6 JSON files, part-01.json, part-02.json, etc. Then we create a DataFrame that only contains the `reviewer`, `movie`, and `rating` columns, and has no null ratings. Afterwards, we remove TV show reviews from the DataFrame. Lastly, we generate a JSON file from the DataFrame.

```
{'helpful': ['42', '69'],
 'movie': 'Portrait of a Lady on Fire (2019)',
 'rating': '10',
 'review_date': '24 February 2020',
 'review_detail': 'This just blew me away. I watch films at home mostly now, '
                  'which can lead to bad habits: file hopping in place of '
                  'channel hopping, never settling in for the duration. But '
                  "this grabbed me and wouldn't let me go like nothing else "
                  'has for a couple of years (I think Toni Erdmann was the '
                  'last one that just grabbed me by the scruff of my neck). '
                  "I'll be watching it again tomorrow night.\n"
                  'It really is a masterpiece.',
 'review_id': 'rw5507595',
 'review_summary': "Thought I'd watch 5 minutes before bed, watched to the end",
 'reviewer': 'meleftheriou-2',
 'spoiler_tag': 0}
```


Figure 1: Sample data for one review in the dataset

Computational Overview

- **Filtering the Dataset:** As described in the Dataset Description, we filter out the `None` ratings, select the columns of `reviewer`, `movie`, and `rating`, and remove TV show reviews to generate a new dataset. We call the functions `load_dataframe()`, `clean_dataframe`, and `remove_shows` before generating imdb_reviews.json. We used the pandas library, the DataFrame data structure, and its methods to select and filter data. Additionally, we use regular expressions to create a year pattern that we use to remove TV show reviews.

- **Transforming, Aggregating, and creating algorithms to make the data meaningful:** We transformed our data into a weighted graph with `load_review_graph_json()`, with movies and reviewers as vertices, and the weight of each edge as the given review scores. We took our user input of favorite movies and created a new vertex within this graph with `add_reviewer()`, as a reviewer who gave all of their favorite movies a ten rating. We created an algorithm called `suggest_movies()`, which takes two reviewer vertices, and suggests movies for one based on the neighbours of the other which have a high weight. We created an algorithm called `reviewer_similarity_score()` which calculates a score of how similar two reviewer vertices are, based on which neighbours they share, and how similarly they are weighted. Finally we created an algorithm called `get_suggestions()` which takes the vertex made by the user's input and aggregates all of the neighbours of all of their favorite movies. Then it calls `reviewer_similarity_score()` on each of these neighbours to aggregate their similarity scores into a ranking of the most similar other reviewers. Finally, it calls `suggest_movies()` to create a list of movies that the most similar other reviewers have seen and enjoyed, but which the user has not, and returns that list.

- **Creating the UI:** As we said, we used the Tkinter library and its methods to create our functions. We use it to take inputs from the user in our algorithm and show the results to the user. This library completely satisfies our purposes because it provides us with methods to creates different windows in our program.

  We use this library to create three pages. To generate the first page, we use the `first_page()` function in the app.py file. This function includes code that creates and shows the first page to the user and contains details such as background, title, font, a button for entering the movie selection page, and a button for visualization. On the first page, the user sees a black page with the title "Start," a label, which says the name of our algorithm, "movie recommender," and two buttons: ENTER and VISUALIZE. When the user clicks the enter button, our program calls the function `new_window()` and displays the second window. When visualize is clicked, the plotly figure opens in the user's browser. When the GUI goes from the first page to the second one, it uses a function that closes the first page and opens the second one.

  If the user clicks the ENTER button, the second page will contain two entry boxes, a suggestion scrollbar, which includes the names of all the movies in our dataset, and two buttons. In one of the entry boxes, the user can select a movie name, and in the other, the user can choose the number of recommendations they want. We

use one of the buttons to add a movie name and the other to view the results. On this page, we show the user if the selected movie name was successfully received and added to the list of chosen movies. We also display if the chosen movie name is not in our dataset and is not a valid input.

When the user adds all their selected movies and clicks the GENERATE button, our algorithm opens a new window and deletes the current window. In this step, the `page_three()` function is called. On the third page, the `page_three()` function shows our recommendation system's results to the user, which are a list of suggested movie names. If the number of recommended movies is high and the movies cannot be shown on one page, the list of movies will be displayed in a box with a scroll bar.

- **Visualization:** Our visualization uses networkx and plotly to illustrate a portion of our graph network. It uses a subset of our vertices, 1000 as default, and showcases the vertices and their edges. As seen in the picture below, the green dots represent movie titles, and the purple dots are the usernames of reviewers. If a user reviewed a movie, an edge is created. By comparing these edges and other reviewers who reviewed the same movie, based on their ratings of our recommendations are made (more information in discussion).

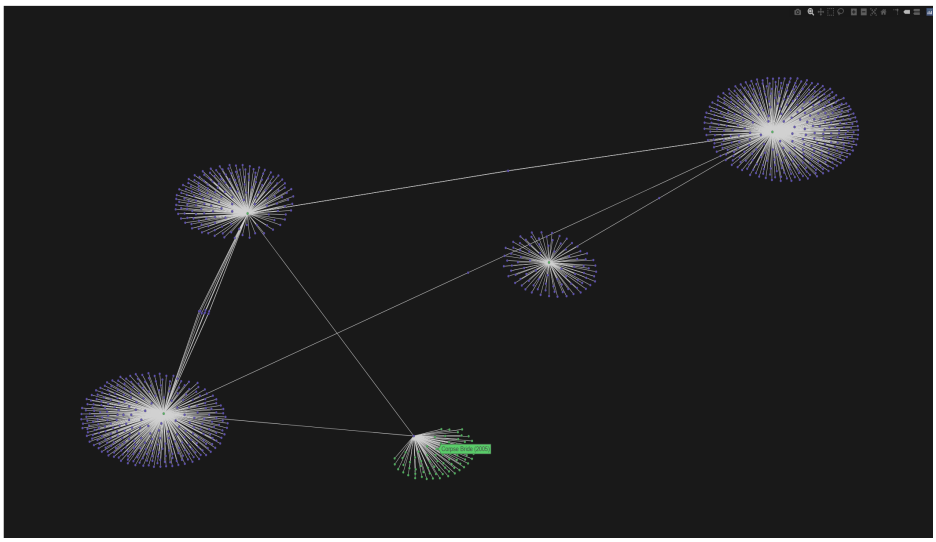

Figure 2: Graph of reviewers and users (subset)

Instructions

- Install the libraries listed in `requirements.txt`.
- Download the python files, and extract the filtered json dataset provided in the send.utoronto.ca link. The claim id is 'RGZrBit6tdcG53NX', and the claim passcode is 'DJyApoPkKhiceVNn'. Alternatively, you can download the zip file from the link in the Dataset Description. The dataset is titled 'imdb_reviews.json'. **Ensure the dataset is in a folder labelled data**.
- For all the .png files that do not contain 'latex' in their title, download them and add them to a folder **titled 'images'**. This is necessary for loading our UI.
- Execute main!
- The UI will take a moment to load as it creates the graph. When selecting movies, the search bar can take a moment to load depending on your computer, but it will register your keystrokes even if they do not show right away. This is because the list of movies is very long, and the Tkinter widget may not be optimized for our update function. You can click the movie you want, then click again for the text entry to update with that movie. We recommend adding at least three of your favorite movies for meaningful results.
- Here are two inputs you can try:

  - **Horror/Thriller Package:** Add `Midsommar, It Follows, The Cabin in the Woods`, any number of recommendations.
  - **Quentin Tarantino Package:** Add `Pulp Fiction, The Hateful Eight, Inglourious Basterds`, any number of recommendations.
- After clicking 'add' for each of these movies and setting a threshold for the number of recommendations you want, click generate.
- A list of movie recommendations will then show based on our graph network.

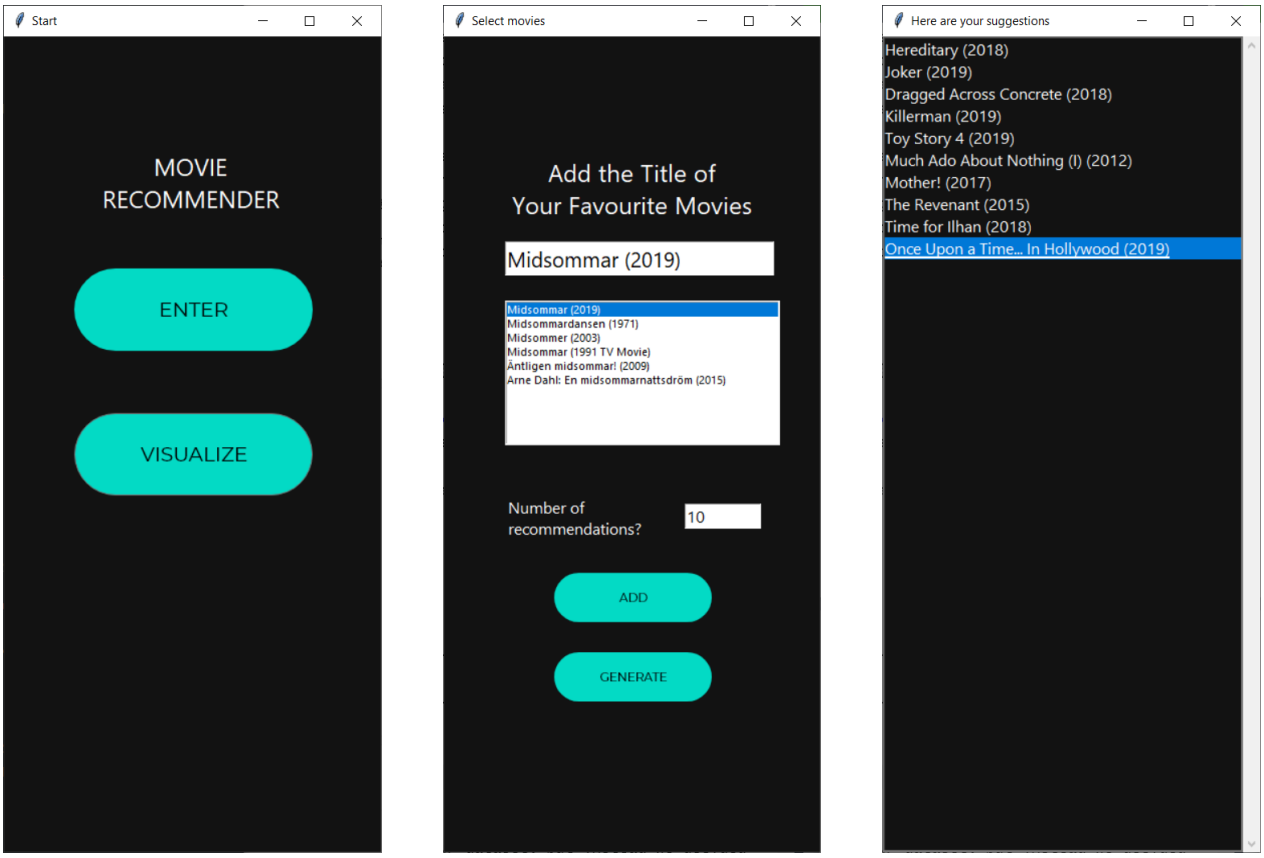

Figure 3: Horror/Thriller Example

Changes to the Project Plan

Initially, we were planning to use a small sample from our dataset, but instead we decided to create a new dataset by filtering our original datset, which contains a large amount of reviews.

For the user interface, instead of recommending at most ten movies with an input of at most five, we made it so that the user can have any number of movie inputs, and any number of suggestions.

Discussion Section

- **Results of Computational Exploration:** We found an algorithm using a dataset that includes reviewers, movie names, and ratings reviewers gave to movies. Our algorithm makes a bipartite graph with two bipartitions, one for reviewers and another for movie names. It creates a weighted edge between these groups based on the rates reviewers gave to each movie. The base of our recommendation algorithm is finding similarities

between two people by analyzing the movies they have watched and their ratings to them.

For a new user, we assume they would rate their favorite selected movies a score of 10. We create a new vertex for the user and create an edge to the favorite movies that they selected. We assign the connecting edge, the movie, with a weight of 10. After that, we find other reviewers who had the same favorites movies (people who also watched the same films and gave high scores to them).

As a result, we recommend movies to the user that are highly rated by the people who have similar favorites to the user. In conclusion, we can say that our algorithm is a very good answer to how to make meaningful movie recommendations to people using a dataset of movie names, reviewers, and ratings.

This method is commonly known as 'user-based collaborative filtering'. It assumes that people with similar characteristics share a similar taste in movies. An article I found online has an example that puts it best, "If you are interested in recommending a movie to our friend Bob, suppose Bob and I have seen many movies together and we rated them almost identically. It makes sense to think that in future we would continue to like similar movies and use a similarity metric to recommend movies" (Pathak 2019).

- **Limitations:** One of the limitations of our dataset is that there may be movies that the user wants to select or that would be good recommendations for the user that do not appear in our dataset. For example, newly made movies or reviews will not be present in our dataset. Currently, our program does not have a way of updating the dataset as new reviews are made on IMDb. A limitation of the algorithms we use is that our dataset contains the data for many reviews and causes loading the graph to take a considerable amount of time. For this reason, actions in the GUI, such as selecting a movie can be slow.

  In addition, for a system that takes in any movies which is not limited to the dataset, can also have an issue of movies having no reviews. If this were the case, then generating recommendations would be hard since there are no reviewers to create edges between. For further exploration, one idea could be to explore different recommendations algorithm that doesn't use reviewers.

- **Further Exploration:** Currently, our program relies on reviewer ratings and a list of movies the user likes to make suggestions. In further exploration, we can expand our algorithm and consider other factors decided by the user. For example, we can modify our suggestions to only include movies from a given genre selected by the user.

  Additionally, in further exploration, we can ask the user to rate their selected movies instead of asking them to give a list of movies they like. We can make our recommendations based on these ratings. Moreover, with each new user that uses our program, we can add to a new reviewer and their rating to a movie to our dataset to improve our future movie suggestions. To see how accurate our algorithm is, we can also ask the users to evaluate the quality of our suggestions. Based on those evaluations we can make changes to our algorithm to see if it will perform better.

References

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

Liu, David, and Mario Badr. "CSC110/111 Course Notes," n.d. https://www.teach.cs.toronto.edu/ csc110y/fall/notes/. Shuvayan, S. D. (2015, September 24). Beginners guide to learn about content based recommender engine. Retrieved March 15, 2021, from https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/

Pathak, Ashay. "Recommendation Systemsnbsp;: User-Based Collaborative Filtering Using N Nearest Neighbors." Medium. SFU Professional Master's Program in Computer Science, March 9, 2019. https://medium.com/sfu-cspmp/recommendation-systems-user-based-collaborative-filtering-using-n-nearest-neighbors-bf7361dc24e0.

Tkinter - python interface to tcl/tk. (n.d.). Retrieved March 15, 2021, from https://docs.python.org/3/library/tkinter.html#module-tkinter

IMDB reviews EDA Starter. (2021, February 19). Retrieved April 14, 2021, from https://www.kaggle.com/spoons/imdb-reviews-eda-starter