CSC111 Project 2: Movie Recommendation System

Toryn Chua, Jacob Keluthara, Yeonjun Kim, Daehyeon Kim

April 1, 2024

Introduction

- **Background and Overview**
  IMDb is a prominent platform for aggregating reviews of movies, distinguishing content with ratings based on both critic and audience reviews, as well as overall show popularity. IMDb uses a traditional 10-star rating system, where users can rate movies and TV shows with a score out of 10, as well as a written review. This score out of 10 is also weighted, which ensures that the score accurately captures public and critic perception of the movie or TV show. While the site does provide recommendations, for these recommendations to be accurate, users need to maintain an account, which requires the user to rate a large number of movies they have watched for the recommendations to be accurate. This project aims to enhance user experience by developing a recommendation system that considers individual preferences, offering a simpler way to find content to view.

- **Motivation**
  Throughout history, movies have been a primary source of entertainment for diverse audiences. However, as the volume of available content has grown, so has the challenge of choosing what to watch, especially given the limited leisure time most people have. While search engines and platforms offer ways to discover new movies, they often fall short in providing personalized recommendations. IMDb, for instance, categorizes content based on selected criteria while also considering individual viewing histories, which does lead to usually useful and accurate recommendations, but at the expense of the user requiring to actively maintain an IMDb account. Recognizing these limitations, we aim to develop an interactive program that tailors movie recommendations to users' preferences and previously watched or highly-rated content, making the discovery process more targeted and enjoyable. This will be achieved using datasets from Rotten tomatoes and user information.

- **Project Goal: To recommend new movies to people based on their personal preferences and previously watched movies.**

Datasets

For this project, there is one dataset we have used, 'IMDB Movie Dataset'. This dataset contains information about the top 1000 movies from IMDb, which our project will use to make recommendations from the most popular movies on IMDb. The dataset has 16 columns:

1. Poster_Link: This contains a link to the poster for the movie.
2. Series_Tite: This column contains the title of the movie.
3. Released_Year: The year the movie was released
4. Certificate: This is the age rating of the movie, if the movie has one.
5. Runtime: The running time of the movie, in minutes.
6. Genre: A list of genres that pertains to the movie, this column contains more than just a single element.
7. IMDB_Rating: This column contains the rating out of 10 from IMDb for the movie, aggregated from user reviews.
8. Overview: A brief description for the movie.

9. Meta_score: This score is out of 100, and is formulated by IMDb using purely critic rating.
10. Director: The director for the movie.
11. Star1, Star2, Star3, Star4: These four columns contain the main stars for the movie.
12. No_of_Votes: The number of users that have rated the movie
13. Gross: The Box Office total for the movie.

Not all of these columns are used, the main ones being used for the recommendations being 'Released_Year', 'Runtime', 'Genre', 'IMDB_Rating', 'Director' and 'Star1', 'Star2', 'Star3', 'Star4'. An example row of the dataset with some columns removed is displayed below:

Series_Title	Released_Year	Runtime	Genre	IMDB_Rating	Director	Star1
Apollo 13	1995	140 min	Adventure, Drama, History	7.6	Ron Howard	Tom Hanks

Table 1: Sample Row with Columns Removed

This dataset can be obtained here:
`https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows`

Computational Overview

- **Functions and Data Types in data_computations.py**
  There is one class in this file, which is the `Movie` class, which is a data type used to represent a singular movie and the attributes of that movie that we require, such as the runtime and genre. This class is used to make accessing attributes of the movie easier and allows us to more easily represent the movies throughout the code through the use of this data type. The function in this file, `csv_to_object`, takes the dataset as an input and process the data and outputs a list of `Movie` objects, representing all the movies that exist in the dataset.

- **Functions and Data Types in weighted_decision.py**
  There is one class that is used in `weighted_decision.py`, which is used to make the recommendations. This is the `Tree` class, which is similar to the implementation of this data type from lecture. It has two instance attributes, `_root` and `_subtrees`, which are the same as the implementation of this data type from lecture. There are two methods of this data type in our implementation, `insert_sequence` and `convert_to_subtree`. `insert_sequence` is similar to the implementation of this function from Exercise 2, and is used in the construction of the decision tree, later in this file. `convert_to_subtree` is a method that turns a tree into one of its subtrees with a matching root, and if there is no subtree with a matching root, it turns it into an empty tree instead.

  Then, this file has the `build_decision_tree` function, which is outside the `Tree` class. This function takes in the dataset file as an input, and from this file creates a decision tree, which has the movies at the end of each branch of the tree. The layers of the tree, which are the attributes of the movies that will be used to make recommendations are the genre, year, runtime, director and actors. Furthermore, this function also does the processing of the `.csv` file, removing the columns that are not needed, such as the poster link and number of user votes. This function also does some processing of the data before adding it as part of the tree. Since one specific runtime and year would be too specific, before inserting the movies and their branches into the tree, it groups the runtime based on every 60 minutes, and the year by the decade. This means that for the decision tree, if the user input is 175 minutes for the runtime and the year 1982, the tree will consider all the movies that have a runtime of 120-180 minutes, and all the movies released from 1980-1990. Without this processing, the recommendation system would not be able to make meaningful recommendations because the user inputs would be too specific. Towards building the tree, this function calls the `insert_sequence` method of the `Tree` class, similar to how this function was called in Exercise 2 to build a decision tree.

  There are two more functions within this file, `recommendation_system` and `get_top_5_movies`. The latter of these finds the top 5 movies based on the user rating from the file, and returns the titles of these movies as an ordered list. This is used in the case there are not enough movies that match the user's recommendations, or in the case there are no movies that match the user's recommendations. The `recommendation_system` function does the overall recommending based on the user inputs and the dataset. This function takes in the dataset and

the list of user inputs. First, this function creates the decision tree using the `build_decision_tree` function, and then also removes all the subtrees that do not match the user's inputted year using the `convert_to_subtree` function. Then, this takes the user's inputted runtime and fits it into the group it belongs to. Then, it uses the rest of the users inputs to traverse the tree, until it reaches the end. If there are no movies here, it uses calls the `get_top_5_movies` function, and returns the top 5 movies instead. If there are movies that match the user's choices, then it takes those movies, and uses a dictionary that maps them to their ratings, then uses their ratings to sort them before returning the list of the titles of the recommended movies, sorted based on their ratings. Furthermore, if there are more than 5 movies that matches the user's input, it only returns the top 5 based on their ratings, and if there are less than 5, the `get_top_5_movies` function is used to supplement the recommendations until there are 5 movies.

- **Visualization in recommendation_system.py**
  To create the user interface for the project, we have used the `pygame` library, this being the primary external library we have used for the project. Additionally, there is another library we have used, the `pygame-gui` library, which is an extension of the `pygame` library, used to more efficiently make a user interface for the program. There are two functions in this program, `draw_text`, which is used to display the text on the pygame screen, and `is_integer`, which is used to check whether the users input is an integer, a function that is required to ensure that the runtime and year match what is required.

  Within this file, there are multiple classes which are created and utilized, for the purpose of creating the user interface. The first of these classes is the `StateManager` class, which is used to represent the current state of the screen, which is required to switch between the different pages of the interface, the home page, the questionnaire page, and the final page with the recommendations. There is also an abstract class, `Pages`, which represents an arbirtary page of the interface. The `Pages` class has subclasses, which have concrete implementations. The first of these is the `Results` class, which represents the final results page. This class uses the `draw_text` function as well as the `recommendation_system` function from `weighted_decision.py`, and displays the titles of the recommendations on the screen. Similarly, the `Start` class represents the home page, and is also a concrete subclass of the `Pages` class. Finally, there is the `Questionnaire` class, which is a concrete subclass of `Pages`, and is used to represent the page that takes in the user's inputs. This page has an error message through the `current_error_message` instance attribute, displaying the error message stored in this instance attribute through the `errormessage` method. This page also has the input boxes, which are made using the `pygame-gui` library.

  Lastly, there is the `Main` class, which represents the main loop of the screen, used to initialize the screen and the pages, state and questions, and this class is used to run the overall program in `main.py`.

User Instructions

- **Required Python Libraries**
  For the required Python libraries, look to the `requirements.txt` file. The main external library used is `pygame-gui`, an extension to the `pygame` library that makes making a user interface more efficient.

- **Obtaining Datasets**
  Within the `.zip` file for the project, the `data` folder contains the dataset that is needed for the project. Keep the `data` folder in its current place, and do not move the files inside the folder. Below are some screenshots of what the folders should look like.

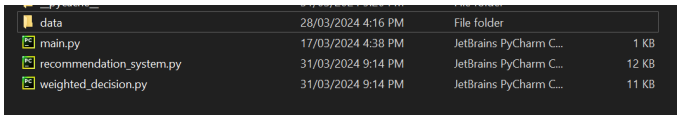

Figure 1: Overall Project Folder

Figure 2: Inside of the data Folder

- **Running the Program**
  - To start with, open the `main.py` file, and run the file. Running this file will open the user interface for the program.
  - The first screen the user sees will be the introduction screen, giving a brief introduction of the program. From here, the user needs to press SPACE to continue, moving them to the page that allows the user to input the preferences they want for the movies. When all the fields are filled, the user can press TAB to have the recommendations show, or if they inputted something incorrectly, they can have press ESC to restart.
  - Now, the user will be at the page that will show the names of the 5 movies that most closely match the inputs of the user. From here, the user can press BACKSPACE, which will allow them to exit the program.
  - Below are some images of the interface for reference.

(a) Home Page of the Interface

(b) Input Page of the Interface

Figure 3

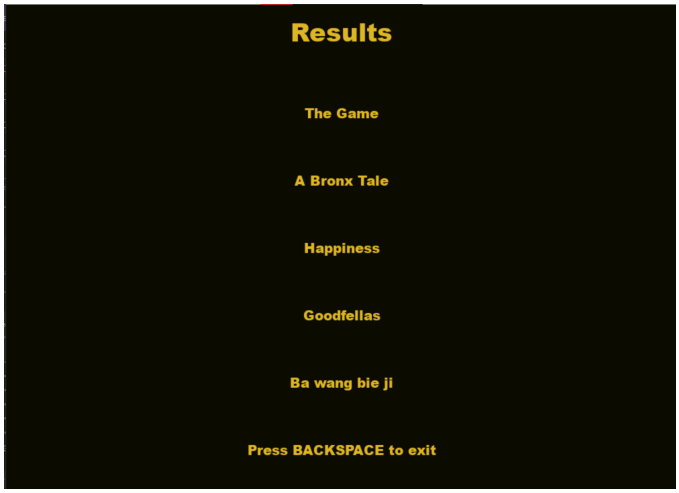

Figure 4: Final Recommendations

Discussion

- **Conclusion**
  Overall, the program works as expected. Entering values into the interface did output results that we expected, comparing the values from the dataset to the outputs of the program. Using a decision tree, we take in the user's preferences and provide some recommendations based on the user's inputs. Considering that in most

scenarios, the recommendations that the system provides matches what the expected output would be, we can say that the system does its intended purpose. Furthermore, the interface we made is interactive and intuitive enough, allowing the user to easily use the program to find recommended movies. However, despite this, there is still room for improvement, as well as some limitations, which will be discussed below. Additionally, we had to compromise in some areas from our original goals, due to factors out of our control.

- **Changes From Proposal**
  Throughout the development of the project, we encountered many obstacles that required us to change the idea for how we would execute the project. The first change we made was a change in the dataset we used. Originally, we planned to use a dataset that had data from the movie review website Rotten Tomatoes, but upon further inspection of the dataset, we realised that many of the columns for each movie were empty, making it extremely difficult to use that dataset to make recommendations. Therefore, we had to look for a different dataset, and we used a dataset containing data from the website IMDb instead. Furthermore, one of our original ideas was to use the user's Rotten Tomatoes account to access the movies they have watched and rated, and then use those movies to find similar movies that the user has not watched but may like. However, this was not possible to accomplish due to the fact that, similar to the IMDb API, the Rotten Tomatoes API was also only available for commercial use. Additionally, there we were unable to find a complete and usable dataset for Rotten Tomatoes, making this idea more unfeasible. Therefore, we had to compromise and use the IMDb dataset and scrap the idea of accessing the user's account.

  Another change that was made to the proposal was the data structures we used. Originally, we planned to use a combination of both decision trees and graphs in order to make effective recommendations. However, with the removal of the aspect that relies on the users account, we were unable to use graphs in a meaningful way that would help our recommendations, which meant that we did not use graphs to make recommendations.

- **Limitations**
  While the program does work as intended, there are a range of limitations faced that prevent the program from completing the original goal as well as possible.

  1. Dataset Used: One of the major limitations with the program lays with the dataset we used. The number of movies from the dataset is only 1000, which limits the recommendations we can make. To get around this, we would require either access to the IMDb API, allowing us a more official way of accessing the data for a large number of movies, or finding a larger dataset. Sadly, larger datasets were not publicly available, and IMDb's API is only available for commercial use and is quite expensive, making it out of the range for this project. The other option would be to create a program that can web scrape the information of movies from IMDb, which is most likely the most effective way to circumvent the issue. Additionally, the dataset we used was 3 years old, meaning that the dataset does not contain information about new, popular movies that could have been recommended that were released since the dataset was made.
  2. Use of Trees: For the program, the method through which we made recommendations was through the use of decision trees. This approach has some limitations, as if the inputs the user uses do not have results in our dataset, that means the end of that branch of the tree is empty, and the program cannot make useful recommendations. Through the use of trees alone, we cannot make meaningful, personalised recommendations this way, and this is one of the major limitations of our program.
  3. Attributes Represented: For the use of the decision tree, we need to focus on a few primary attributes of the movies. However, this means that other attributes that users might care about are under-represented in our recommendations, limiting the accuracy of the recommendations.

- **Next Steps**
  To improve this project, there are some further changes we could make. One of these changes would be to try and web scrape data from IMDb, Rotten Tomatoes, or other movie review sites. Scraping the data ourselves from these websites would allow us to get a more complete dataset that matches our requirements, allowing us to enhance the recommendations made. Another change we could make would be to implement graphs into our system. We can do this by making graphs that have weights based on the similarity between movies. This would not be done through finding similar users, like in Exercise 3 and 4, but rather through comparing the attributes of movies. This could be done through consider each movie as a vector in an n-dimensional space, with each attribute of the movie being a dimension, and then finding the distance between these vectors and using this as a form of a 'similarity' score between movies. This could then be used to find movies similar to the movies already likes, which can be used to make accurate recommendations. We were unable to implement this in the current iteration due to the fact that this would require us to quantify the different attributes and

assign weights to each attribute, which would require a larger amount of data and more testing, which would not be possible in the current time frame. Another change that could be made is the implementation of more of the user's preferences, extending the decision tree and allowing for recommendations that are more specific to the user. The last change that we could make towards improving this project would be to streamline the user interface further. While the current interface is usable and functions properly, there is further room for improvement, making the interface cleaner and more user friendly, while also adding more options.

References

[1] Cravit, Rachel. "What is a Decision Tree and How to Make One [Template + Examples]" Venngage, 14 Nov. 2023, https://venngage.com/blog/what-is-a-decision-tree/. Accessed 1 Mar. 2024.

[2] Jaimadan3. "What is Weighted Graph with Applications, Advantages and Disadvantages" GeeksforGeeks, 7 Jun. 2023, https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-weighted-graph/. Accessed 1 Mar. 2024.

[3] Fincher, Jon. "Pygame: A Primer on Game Programming in Python" Real Python, 2 Jul. 2021, https://realpython.com/pygame-a-primer/. Accessed 1 Mar. 2024.

[4] Amos, David. "Python GUI programming with Tkinter" Real Python, 10 Jan. 2021, https://realpython.com/python-gui-tkinter/. Accessed 1 Mar. 2024.

[5] Wilkinson, Alissa. "Rotten Tomatoes, explained" Vox, 14 Jun. 2018, https://www.vox.com/culture/2017/8/31/16107948/ /andrezaza/clapper-massive-rotten-tomatoes-movies-and-reviews?select=rotten tomatoes movies.csv. Accessed 2 Mar. 2024.

[6] Lewis, Robert. "IMDb" Britannica, 1 Apr. 2024, https://www.britannica.com/topic/IMDb. Accessed 1 Apr. 2024.