CSC111 Project Proposal: What To Watch? - A Movie Recommendation System Based On User Preferences

Fatimeh Hassan, Shilin Zhang, Dorsa Molaverdikhani, Nimit Bhanshali

April 16, 2021

Introduction

The problem domain we will be exploring for this project is the lack of interactive and personalized movie recommendation services. Time and time again we run into the dilemma of which movie to watch next. We explore various websites ranging from IMDb to Rotten Tomatoes to find a movie that we will like. Yet, we lack a personalized system that takes in user preferences and recommends movies solely based on these preferences. Our goal is to create a program that will provide a range of movies that our users can commit to watching without the concern of having made the wrong choice. Using attributes such as movie genre, release year, the duration and more, we will be able to provide accurate recommendations for the user based on what they like. Additionally, we will provide a recommendation service that respects the privacy of the user and only suggest movies based on the given preferences, unlike certain services that employ search histories to recommend movies.

Given the pandemic, many of our group members took the opportunity to watch new movies during quarantine. However, as stated above, the major issue most of us faced was not knowing what to watch. We took upon different ways to look for movie suggestions whether that be polls on Instagram stories, scrolling through search engines, as well as reading IMDb reviews. However, we weren't able to find any such service that matched our personal preferences. This motivated us to create such a recommendation service that suggests movies that align to our preferences. This way we are able to guarantee an entertaining experience and allow users to commit to a movie that is in line with their preferences.

The above background and context leads us to the following goal for our project: **Our project goal is to provide accurate and helpful movie recommendations based on the user's preferences.**

Datasets

IMDb movies extensive dataset (IMDb movies.csv):
The Kaggle dataset that we used contains over 80,000 movies from IMDb with relevant attributes of the movie. The data is contained within a csv file. Each row represents a movie, and each column represents attributes of the given movie. The data set includes the following columns: unique IMDb movie title id, movie title, original title, release year, date published, genre, duration, country, language, director, writer, production company, actors, description, average vote, votes, budget, USA gross income, worldwide gross income, metascore, reviews from users, and reviews from critics. However, a lot of these factors are redundant and as a result, we are only using a few of the columns in our program. We used the imdb_title_id, original_title, year, genre, duration, country, language and avg_vote columns in our program.

Computational Overview

We created a dataclass called Movie in the `entities.py` file to represent each individual movie in the dataset and store the following movie attributes: IMDb movie id, movie title, release year, genre, duration, languages and the average rating of the movie.

We also created a private vertex class called MovieVertex and a graph class called MovieGraph in the `entities.py` file. MovieVertex represents a vertex in our defined graph and has two instance attributes. One being the item

attribute which is an instance of the Movie dataclass and the other attribute being neighbours which is a dictionary that maps the vertex to a set of all vertices adjacent to it. The MovieGraph represents a graph to represent a movie network that keeps track of common traits between movies, represented by edges between vertices. This class has one private instance attribute, `_vertices` which is a dictionary that maps the movie id to the corresponding MovieVertex. For this MovieGraph, edges are added between two vertices, if the two Movie objects stored in the vertices have at least one common attribute.

For the user interface part of our project we used the tkinter library. We used tkinter to display two windows, one asking the user to rank movie attributes in the order of importance and one asking questions about those rankings using tkinter listboxes. For the user rankings, we implemented the function runner_rankings that creates a window, creates labels using tkinter labels (Amos, 2021) that tells the user what to do, and creates a listbox containing 4 attributes that we ask the user to rank. We also implemented a submit command within this function ("How to get selected value from listbox in tkinter?", 2021). This will do the following: whenever the user clicks the submit button, the chosen attribute will be removed from the listbox. If only one attribute is left in the listbox, when the user submits their preference, the window will be closed using the destroy function in tkinter (Elance, 2019). The order in which the 4 attributes appear in the list is their inputted order of importance for the movie attributes, which implies that the first item in the list is the attribute that the user cares about the most. Finally, the runner_ranking function will return the user_ranking which is a list containing the preferences of the user by ranking and then close the window.

To ask questions regarding the 4 movie attributes, we implemented runner_questions which takes two inputs lists: the list of all available languages and the list of all available genres. This function will create a window,and initialize a dictionary(answers_so_far) which will keep track of the user choice from the listboxes for each question. It will also call on four helper functions: create_genres_listbox, create_duration_listbox,create_year_listbox, create_language_listbox each of which are responsible for creating a frame using the tkinter LabelFrame in the window and a listbox in the created frame.

The helper create_genres_listbox takes in the following inputs: the window, the user_answer dictionary, and the genres list. This helper creates a frame with related questions in it, a listbox within the frame, and the items in the listbox are the elements of the genres list which is all the genres in the dataset. We also implemented a submit command within this function ("How to get selected value from listbox in tkinter?", 2021). This will add what the user chooses from the listbox to the user_answer dictionary which was given to it as an input, after the user click the submit button corresponding to this listbox, the listbox and the frame will disappear from the window using the Labelframe.destroy function in tkinter (Elance, 2019). The other 3 helpers do the same thing except the elements in the listbox are the choices that we create for duration (we categorized it into 3 categories, short being the movie is less than 60 minutes, medium being between 60 to 180, and long being more than 180 minutes), year (contains all decades from 1890 which is the oldest release year in the dataset to 2020 which is the earliest release year in the dataset as intervals (Ex. 1890-1900) which is the output of the helper function create_decade_options with 1890 as the start year and 2020 as the end year) and languages. Using the helpers, the function runner_questions will create all 4 listboxes and after the user selects their preferences, it returns a dictionary which the keys are the 4 attributes and the values are the user choices for each one.

Finally, we implemented the main_runner function which will call load_genres_and_rankings with our dataset to obtain the list of all genres and languages from the dataset which are the genres and languages options for the user when they are asked the question. Then, we call the runner_ranking function asking the user to rank the 4 movie attributes. After the user ranks them, we use the genres and languages lists to call the runner_questions functions in order to ask the user to answer related questions. Finally, this function will return a tuple where the first element is the return value of the runner_ranking function and the second element is the output of the runner_questions function.

The load_dataset function is responsible for generating a MovieGraph based on the inputted dataset and user Movie object. This function initializes a MovieGraph and first creates a MovieVertex using the user Movie object, which we will refer to as the user vertex in this report. Additionally, we utilize the Python library, pandas (specifically pd.read_csv), to filter columns of the dataset and we iterate through the rows of the dataset using a for loop. At each iteration, we generate a Movie object for the movie described in the row of the dataset. We then add a MovieVertex to the initialized MovieGraph and also add an edge between the newly added MovieVertex and the existing user vertex. Once we iterate through every row in the dataset we are left with a MovieGraph that consists of vertices and edges, which is returned by the function.

Another important function is the similarity_score function which calculates the similarity score between a MovieVertex representing the user and a MovieVertex representing a movie in the MovieGraph. Given the order of the user's preferences, the similarity score will be calculated by adding points depending on the traits that the movies has in common with the user's preferences. If the user's first preference is the same this will add 10 to the score, 5 for the second preference, 3 for the third preference, and 1 for their last preference. The similarity score function is a helper for the recommend_movies function which will calculate the similarity scores of the user vertex's neighbours and return a list based on the highest similarity scores, sorted by IMDb ratings in the event that the similarity scores of two recommended movies are equal. Finally, we return the first 10 movies of the list to recommend the 10 movies best aligns with the user preferences.

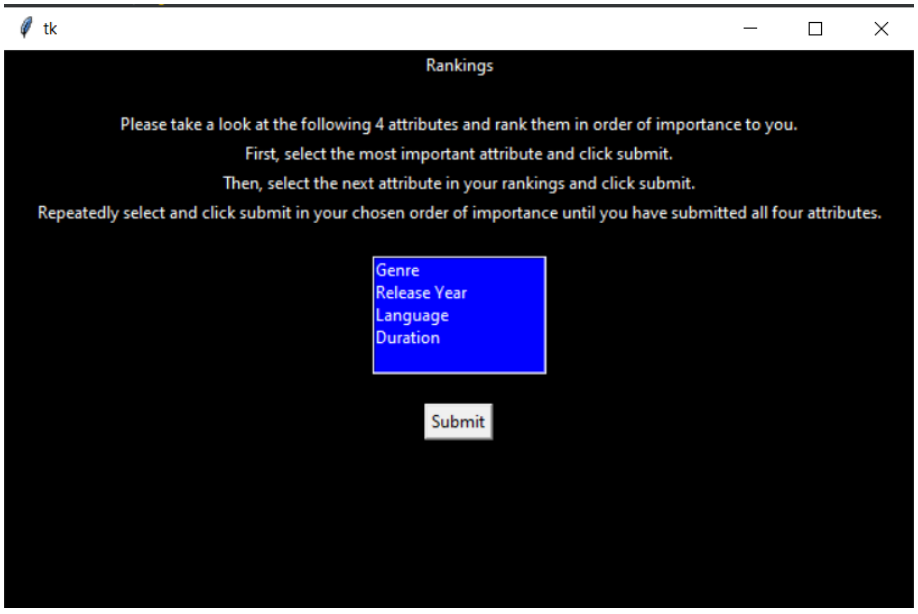
Instructions for Running the Program

Upon installing the Python libraries and downloading the .py files from MarkUs, you will need to download the dataset. To access the dataset, please use the following OneDrive access link (you may need to login with your UofT email/UTORid):

Click here to load the access link.

Loading this link will give you access to the .zip file which has our dataset. Download the zip file and extract that IMDb movies.csv file, which is our dataset. Please save the IMDb movies.csv file in the same sub-folder where you saved the .py files. Once you have downloaded both the .py files and our dataset, mark the subfolder in which these files are saved in as a source root. Upon doing this you will have a subfolder marked as the source root with the following files: entities.py, main.py, visualization.py, requirements.txt, IMDb movies.csv. It is essential that the names of the files all remain unchanged.

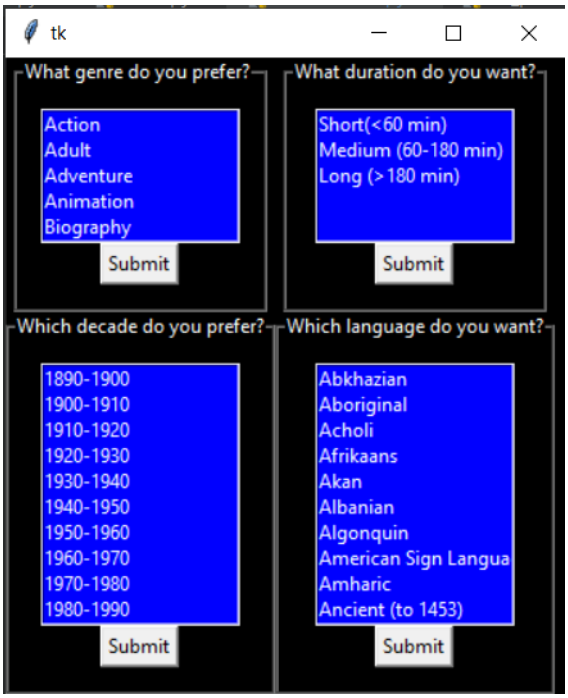
To run our program, you will need to run main.py. Note: We have commented out all the code for PyTA across all the files, as checking the preconditions greatly increases the running time of the functions/methods. Hence, if you want to run PyTA you will need to uncomment the lines of code manually. Once you run main.py, a new window will open with our user interface display (may not pop-up ahead of PyCharm, check your taskbar). Over here you will have to select and submit movie attributes one at a time, in the order of importance to you (detailed instructions will be displayed on the user interface display). Make sure to not click the submit button twice for the same attribute as that will give an error.



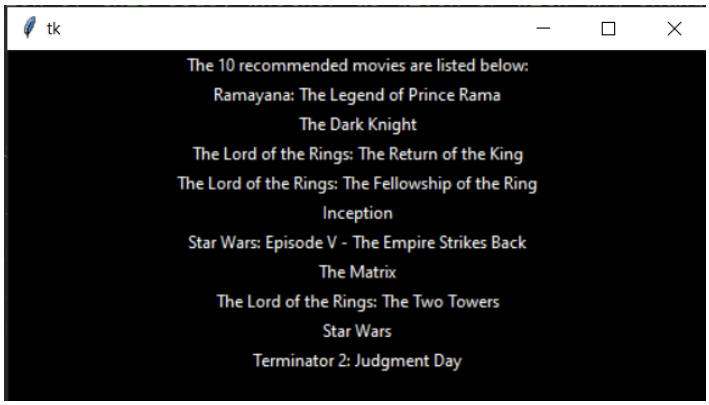
A screenshot of our user interface display for ranking the movie attributes.

Once, you have ranked your movie attributes by selecting and submitting them one at a time, the current window will close, and a new window will open up. This window will have four listboxes with genre, duration, language and

release year options. You can scroll through the options in the listboxes by hovering your cursor over the options and using the scroller on your mouse or by sliding up and down your trackpad. Select and submit your preference for each listbox. Your preference for each listbox is limited to once. For instance, for the genre listbox, you may only choose 1 genre, and the same goes for the other listboxes. Once you have submitted all your preferences, the current window will close. After a few seconds (10 - 20 seconds), a new window will open displaying the movies that our program will have recommended for you.



A screenshot of our user interface display that asks for user preferences for movie attributes.



A screenshot of our user interface display that displays the recommended movies.

Changes to Project

Although we used the same dataset as we originally planned and our project goal did not change, we made a few adjustments to our computational plan. Originally, we planned to create edges between every movie vertex with traits in common. However, as the dataset contains over 80,000 movies, we soon discovered it was not computationally feasible to create such a high number of edges between movies. It would take an extremely long time to run and create the graph itself. As a result, we decided to create a user vertex containing the user's preferences and add edges between any movie vertices that had traits in common with the user's preferences. This way, the user's preferences would be contained inside the graph and we could connect movies directly to the user.

Due to the large number of movies in our dataset, we also set a benchmark rating so that only movies with a rating above 8.0 would be returned. We set this benchmark rating since the running time to calculate the similarity score between the user vertex and all of its neighbours turned out to be very large. Therefore, we created a global constant to represent the benchmark rating. This benchmark can be changed, however, it does come with the cost of the program taking a very long time to recommend movies. Other than these changes, our project goal, dataset and overall plan remained the same.

Discussion

Our initial project goal was to provide accurate and helpful movie recommendations based on the user's preferences, and we believe we succeeded despite a few limitations and improvements that could be made to our program. Due to the computational limits of using such a large dataset, we weren't able to create edges between the movie vertices themselves. If we were able to do this, we could have asked the user what movies they enjoy or have already watched and provided even more personalized recommendations using this information. Other movie recommendation systems may consider similar movies to ones that the user has already seen, and if it was computationally feasible to create edges between movie vertices we would have been able to include this approach in our system as well. However, we believe that asking the user questions about their preferences is also a good way to recommend movies to them and by ranking which attribute matters to them the most we are able to provide better results as well.

Based on the dataset we used, we chose the movie attributes we considered the most important and asked the user questions related to these attributes. However, one thing we noticed was that because the movie dataset has movies in a variety of languages, unless language was ranked as the user's first preference, movies from other languages would almost always appear in the movie recommendations. If there were more movie attributes to choose from and edges between the movies, we could have ensured that the user receives recommendations in the language they chose if that's what they wanted as we would have asked them more questions as well. However, a lot of people do enjoy foreign films and we wanted to create a recommendation system that applies to all kinds of people.

Our similarity score system is also based on if the user's ranked preferences are met, with movies that meet all four preferences being returned first, movies that meet the top three preferences being returned next, movies that meet the top two preferences being returned after that and movies that meet at least the first preference being returned last. This way, the user's ranking of their preferences actually plays a factor in the movies we recommend to them. Instead of relying on just one movie attribute, we want to return movies that meet all of the user's preferences if possible.

There are certain times where no movies are returned because not enough of the user's preferences are met and the similarity scores for the movies are not high enough based on their choices. This may be because although our dataset is quite large and contains over 80 000 movies, there are approximately 500 000 movies in existence (Velikovsky 2012). Therefore, our dataset is limited and we can only return the movies that exist in our dataset. This generally occurs if the user chooses a language that may not be as widely spoken, an older decade, the "short" duration, or a movie genre with limited movies. Based on a combination of these, there may not be any movies that completely matches the user's preferences.

We also set a benchmark rating for the movies so we returned only highly rated movies on IMDb, and the recommended movies we return are in order of highest to lowest rating, or they may all share the same rating if enough movies that fit the user's preferences exist. We did this to narrow down the returned movies, but maybe in order to improve the program we could have asked the user their preferences regarding IMDb movie ratings, if any.

Ultimately, we created a movie recommendation system based on user preferences using the dataset we discovered. The system works correctly and recommends movies according to our algorithm. Although some improvements or changes could be made in the future, our original goal was definitely met. It is up to personal choices whether the movies we recommended are something the user would actually watch but we designed our program in a way that maximizes user preference above anything else. As a result, we believe that our recommendations are accurate and helpful, just as we originally intended.

References

Amos, D. (2021). *Python GUI Programming With Tkinter*. Real Python. https://realpython.com/python-gui-tkinter/#building-your-first-python-gui-application-with-tkinter

Elance, P. (2019). *Destroy() method in TKINTER - Python.* https://www.tutorialspoint.com/destroy-method-in-tkinter-

How to get selected value from listbox in tkinter? *GeeksforGeeks.* (2021). https://www.geeksforgeeks.org/how-to-get-selected-value-from-listbox-in-tkinter/

Leone, Stefano. (2020). IMDb movies extensive dataset. *Kaggle.* https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset?select=IMDb+movies.csv

More Widgets. TkDocs Tutorial - More Widgets. (n.d.). https://tkdocs.com/tutorial/morewidgets.html

pandas. *Python Data Analysis.* https://pandas.pydata.org/

tkinter. *The Python Standard Library.* https://docs.python.org/3/library/tkinter.html

Velikovsky, JT. (2012). How Many Movies Are There? *StoryAlity.* https://storyality.wordpress.com/2012/12/17/storyality-19-how-many-movies-are-there/