CSC111 Project Report: ScreenSelect
Personalized Movie Recommendation System

Dogyu Lee, Narges Movahedian Nezhad, Sidharth Sawhney, Aastha Sharma

April 3rd, 2023

Part 1: Goal

Due to the advent of streaming services, which provide customers immediate access to a vast collection of information, the entertainment industry has undergone a vital change. Finding the right things to watch, for example, might just be more difficult than ever for viewers. This is due to the overflow of available content. As a solution to this issue, recommendation systems have been created; personalizing and recommending material based on users' viewing patterns and interests. With this in mind, **our project intends to develop a movie customization and recommendation app that takes inspiration from Netflix's recommendation system, leveraging watching patterns, user input, and clever algorithms to provide tailored movie suggestions.**

Our project intends to build movie customization that *varies* from that of others regarding algorithm, even though the issue of content overload and the demand for tailored movie recommendations have been thoroughly examined. For instance, we will employ user input data in addition to the users' viewing history to provide more customized recommendations. This goes beyond Netflix's simple recommendations of related content based only on user-watch history. Additionally, our project adds a sense of individualism to every user. Unlike other recommendation platforms, we have deliberately chosen not to take into consideration the location of our users and other users nearby because we understand that everyone has a unique taste. This means that no user will get recommendations based on what their contacts and users in the same area are watching. Having geographically been near someone in no way guarantees similar preferences when it comes to movies.

Moreover, ScreenSelect takes into account only the past 10 movies for every user, as opposed to all viewing history—which is the case with other projects in the market. This is mainly because people's tastes are likely to change over time. By doing so, we are ensuring that our recommendations are updated to appeal to consumers' most recent interests in contrast to the movies they enjoyed in the distant past. This strategy also ensures user happiness and increased engagement with the app. Also, by focusing on the viewer's most recent 10 films rather than their complete viewing history, we prevent recommending identical films, which can cause user boredom and disengagement. Overall, our project provides a fresh and original perspective on the issue of information overload. We strive to deliver a highly customized, enjoyable, and useful movie recommendation app to increase user engagement and commitment.

Our project's objective is to boost user engagement by the means of providing a highly-customized watching experience that changes depending on each user. We will offer a platform that generates the best recommendations for each user every time by taking into account contextual elements—such as the last 10 watched movies, keywords, genre, and language—going beyond Netflix's straightforward suggestions of related material based on user watch history. We want to provide a highly engaging experience that keeps people riveted to the screen and coming back for more by putting a strong emphasis on tailored suggestions.

The app also made several considerations regarding the ethics of ScreenSelect. We put in place several privacy precautions to make sure that our project doesn't divulge any sensitive information. We never request personally identifiable information when gathering user data. The username we take from our users is the only sort of identification verification input we take from them. This username can never be connected, as it can be anything users choose. This will assist in preserving user privacy and preventing the disclosure of private information. We have also put in safeguards to make sure that our program doesn't make any incorrect recommendations to vulnerable populations. Our group now has a dedicated email address on the GUI that we utilize to ask users for feedback, especially from members of vulnerable groups. This will make it easier to spot any offensive ideas or information. The input will help our team improve the app's suggestions and make sure they are appropriate for all users. The movies that cause problems to several users will be taken out of the data set. These precautions will be regularly audited to identify potential privacy and security breaches and ensure that our app complies with privacy regulations.

Overall, We have concentrated on developing a recommendation system that prioritizes user satisfaction and engagement over other metrics like watch time or income to make sure that our project's ideas are valuable to the user as opposed to the Netflix engine. When creating our project, we concentrated on giving user-centered design concepts top priority. Aspects such as reading accessibility (through large fonts) can be seen in all parts of our implementation. This required comprehending the user's requirements and preferences and tailoring the platform to satisfy them. We boosted user engagement and improved the app's suggestions as a result of user testing and feedback. In addition to other helpful measures, we provide users with a wide variety of content, including works that might not be as well-known but are nevertheless well-regarded and pertinent to their interests. This is one of the reasons for our decision to work with a large data set. Users will find fresh material and engage more as a result of this.

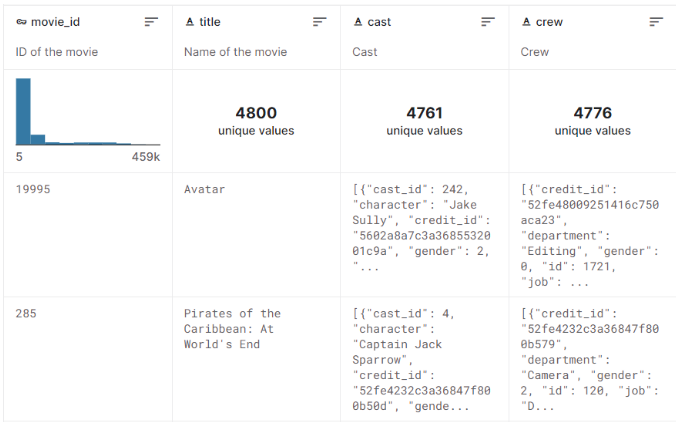
Part 2: Datasets

ScreenSelect used two datasets in the form of CSV files: tmdb_5000_movies.csv (Movies Dataset) and tmdb_5000_credits.csv (Credits Dataset). Both files were downloaded from Kaggle. The Movies Dataset consists of 20 columns with the information and statistics of 4803 movies. Out of the 20 columns in the Movies Dataset, ScreenSelect utilized a subset of nine columns: id, title, genres, keywords, runtime, original_language, overview, vote_average, and release_date. The Credits Dataset consists of four columns with the id, title, cast, and crew for each of 4803 movies. Out of the four columns in the Credits Dataset, ScreenSelect utilized a subset of two columns: title and crew.

We have extracted approximately 400 movies in a smaller version of the original Movies data set CSV file called tmbd_400_movies for regular, consistent testing purposes in a separate CSV file. The larger data set is already loaded in a graph object in the file.pkl. This will make it easier to test and run the program. The smaller subset is provided to see if the movies get uploaded into the graph object properly and whether it continues to run. We have done this because, upon testing, we found that it takes a long time to load the entire data for 4803 movies into the system. We found that everything on the smaller data set was much more time-effective as it provided desired results faster. For comparison, loading the entire graph with the original Movies data set takes around 15 minutes while loading the smaller version takes approximately 2 minutes. We will provide both the smaller subset of 400 movies along with the original Movies Dataset CSV file in our submissions with the file.pkl which contains the pre-saved graph of 4803 movies.

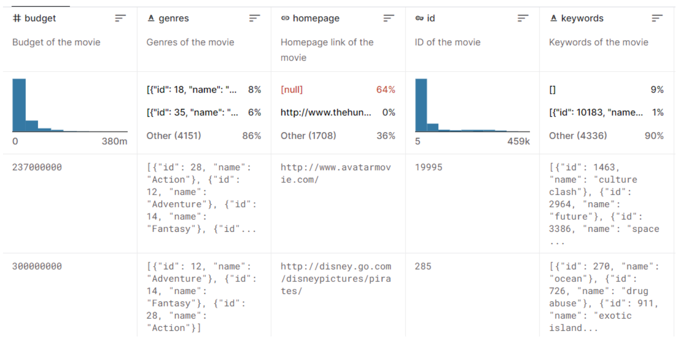
Note: The csv file with the crew is used in both the original data set and the subset of 400 movies. Also, the references for these are provided in the references section.

CSV File 1

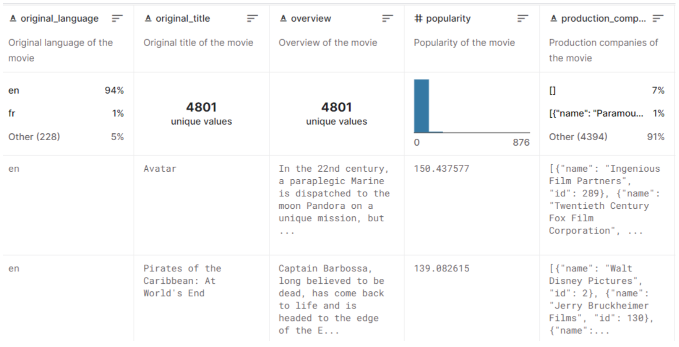


movie_id	title	cast	crew
ID of the movie	Name of the movie	Cast	Crew
5 – 459k	4800 unique values	4761 unique values	4776 unique values
19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a36855320 01c9a", "gender": 2, "...	[{"credit_id": "52fe48009251416c750 aca23", "department": "Editing", "gender": 0, "id": 1721, "job": ...
285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Sparrow", "credit_id": "52fe4232c3a36847f80 0b50d", "gende...	[{"credit_id": "52fe4232c3a36847f80 0b579", "department": "Camera", "gender": 2, "id": 120, "job": "D...

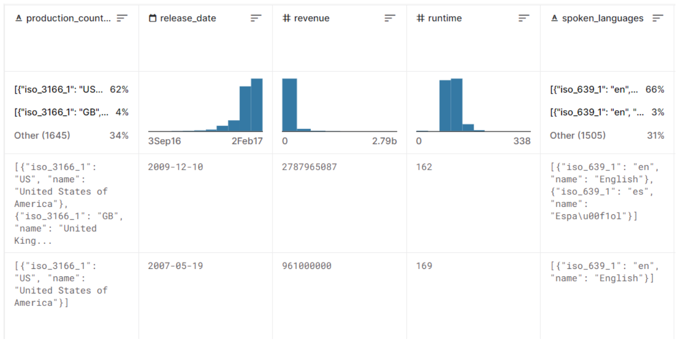
CSV File 2



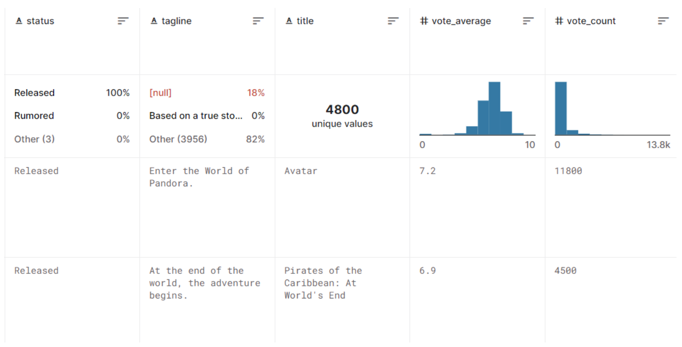
budget	genres	homepage	id	keywords
Budget of the movie	Genres of the movie	Homepage link of the movie	ID of the movie	Keywords of the movie
0 – 380m	[{"id": 18, "name": "... 8% [{"id": 35, "name": "... 6% Other (4151) 86%	[null] 64% http://www.thehun... 0% Other (1708) 36%	5 – 459k	[] 9% [{"id": 10183, "name... 1% Other (4336) 90%
237000000	[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id"...	http://www.avatarmov ie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"}, {"id": 3386, "name": "space ...
300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 28, "name": "Action"}]	http://disney.go.com /disneypictures/pira tes/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "name": "drug abuse"}, {"id": 911, "name": "exotic island...



original_language	original_title	overview	popularity	production_comp...
Original language of the movie	Original title of the movie	Overview of the movie	Popularity of the movie	Production companies of the movie
en 94% fr 1% Other (228) 5%	4801 unique values	4801 unique values	0 – 876	[] 7% [{"name": "Paramou... 1% Other (4394) 91%
en	Avatar	In the 22nd century, a paraplegic Marine is dispatched to the moon Pandora on a unique mission, but ...	150.437577	[{"name": "Ingenious Film Partners", "id": 289}, {"name": "Twentieth Century Fox Film Corporation", ...
en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, has come back to life and is headed to the edge of the E...	139.082615	[{"name": "Walt Disney Pictures", "id": 2}, {"name": "Jerry Bruckheimer Films", "id": 130}, {"name":...



production_count...	release_date	revenue	runtime	spoken_languages
[{"iso_3166_1": "US... 62% [{"iso_3166_1": "GB",... 4% Other (1645) 34%	3Sep16 – 2Feb17	0 – 2.79b	0 – 338	[{"iso_639_1": "en",... 66% [{"iso_639_1": "en", "... 3% Other (1505) 31%
[{"iso_3166_1": "US", "name": "United States of America"}, {"iso_3166_1": "GB", "name": "United King...	2009-12-10	2787965087	162	[{"iso_639_1": "en", "name": "English"}, {"iso_639_1": "es", "name": "Espa\u00f1ol"}]
[{"iso_3166_1": "US", "name": "United States of America"}]	2007-05-19	961000000	169	[{"iso_639_1": "en", "name": "English"}]



status		tagline		title	vote_average	vote_count
Released	100%	[null]	18%	4800 unique values	0 – 10	0 – 13.8k
Rumored	0%	Based on a true sto...	0%			
Other (3)	0%	Other (3956)	82%			
Released		Enter the World of Pandora.		Avatar	7.2	11800
Released		At the end of the world, the adventure begins.		Pirates of the Caribbean: At World's End	6.9	4500

Part 3: Computational Overview

1. ScreenSelect's data is represented using a Graph class which consists of a dictionary containing all the vertices(_Movie and _User instances). An edge in the graph is made between the user vertex and the movie vertex when the user selects one movie from the given five movie recommendations. The selected movie becomes a neighbour of the user and vice versa. A movie will be recommended to the user based on the user's previous and current desires and the 10 most recent movies watched. The same movie will not be recommended again if the user has watched it. The existing edge also prevents the same vertices, a user and a movie, from being connected twice, avoiding duplicate suggestions.

2. **read_csv_and_create_data** in *main_functions.py* processes movie data from csv files. This function adds the movies in the data as a vertex (_Movie) in the given graph. Data filtering is done using private helper functions and additional type conversions to pass correct arguments to our _Movie initializer. We also imported the literal_eval() method of the **ast** module to assist with the filtering process. To load the graph with our larger data set (4803 movies) quickly, we created a .pkl file (pickle file) using the **pickle** module. The file already stores all 5000 movies; therefore, we just need to open the file and assign it to our empty graph to load it without any iteration to access and filter accessing each movie.

   Given a graph and a username, the **compute** function in $main_functions.py$ uses _Movie class's method **score** to estimate the movies to be recommended using comparison of preferences and previously watched movies (*past_10_neighbours*) with other movies in the dataset and returns the top 5 movies. With the given user vertex, the **score** method aggregates the points on the variable *score* by adding 5 or 10 points based on a match by comparing the user and the movie's *genre*, *lang*, *keywords*, and *director*. The first 5 movie vertices, with their scores, are appended in a tuple form to the list *_top_scores* of the user vertex. However, when there are already 5 movies in *_top_scores*, only movies with scores greater than the minimum of the vertices' scores are appended after removing the tuple with the minimum score. Thus, by the end of the computation, 5 movies with the highest preference scores remain in *_top_scores* which is returned by the compute function.

3. This part contains both how the PyQt6 library was used to accomplish its task and how it is responsible for the visual report of the resulting computations in ScreenSelect.

   Screenselect's user interface is created using the PyQt6 library. Its QtWidgets, QtCore, and QtGui modules are imported and used in our program. In *screen_select_gui.py*, three subclasses, RecommendationScreen, PrefenceScreen, and LogInScreen, of QtWidgets exist. Each subclass represents a window the program displays for the user and can use show() or close() to appear or disappear while the program is running. The QtWidgets module's QWidget, QPushButton, QLabel, and QLineEdit classes are used to create graphical user interface (GUI) components such as buttons, labels, and textboxes, for each screen. QLineEdit's displayText() is used multiple times to return the QLineEdit() input as a string for using it with our functions in *classes.py* and *main_functions.py*. QtWidgets module's QGridLayout class and QtCore module's Qt class are employed to set a layout for each screen using setLayout(), to divide the screens into sections using setSpacing() and setContentMargins(), and add each widget to a layout and set their location using addWidget(). The QtGui module's QFont and QIcon classes are used to design the screens and the style of the components to be more pleasing to the eye and fit the cinematic theme of our project. Particularly, functions such as setFont() and setStyleSheet() are used to adjust the fonts and colors of label figures.

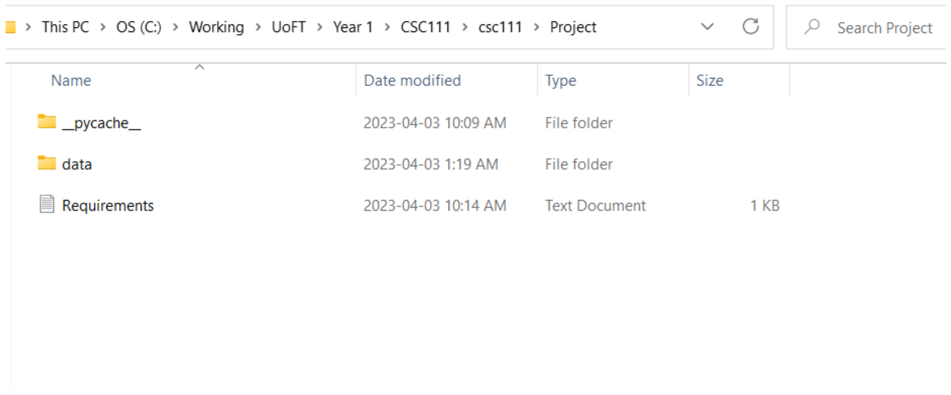
   A key capability of the PyQt6 library is its ability to emit and receive signals between classes. A major constituent of each screen is the buttons the user can click on to move to the previous or the next step of the recommendation process. The users' clicks activate the system to change windows and display the steps graphically because the buttons are made by the QWidgets module's QPushButton class. QPushButton's method clicked() enables the button to emit a signal to the methods connected to it [the connection is established using QPushButton's connect()]. This functionality allows the user to interact with the system for each screen. On the RecommendationScreen, each button is connected to one of the five **screenselect** methods or a **back** method. When clicked on, the buttons will signal the functions to activate, which results in the user either choosing a recommended movie or going back to PreferenceScreen. Similarly, the user on the PreferenceScreen could choose buttons that activate either the **recommendation_button** to move on to RecommendationScreen for movie suggestions or **log_out** to go back to the LogInScreen. Finally, the user on LogInScreen can choose either to click on **sign_in** or **sign_up** and be taken to the PreferenceScreen. All these methods are connected to the *classes.py* and *main*$_{f}$*unctions.py*, which continues on the process of recommendation of movies. A few example calls in these methods include: main_functions.user_movie_neighbours(self.movie_lst[0][1], self.movie_lst[0][1].item, self.graph, self.user_obj.item) and main_functions.compute(self.graph, self.user_obj.item)

Part 4: Instructions for Running ScreenSelect

Getting Ready to Run ScreenSelect

1. Please have Python 3.11 and with PyCharm 2023.1 installed on your computer.
2. Install PyQt6 and PythonTA libraries. The version of each library is specified in *requirements.txt*.
   - When you open PyCharm, you will see 'Python Packages' at the bottom of the screen. After clicking on it, you will see a search bar where you can search for the libraries mentioned above.
   - Once the libraries are installed, you will be able to view its name in the external libraries or in the installed packages sections.
3. Create a new directory folder.
4. Download the datasets zip file on your computer.
   (a) Go to the MarkUs Proj_02: Course Project Phase 2: Submission page and download a zip file named "data" on your computer.
   (b) Extract the file and refactor it in the directory where all the files will be saved in PyCharm. (will look like this):

Directory folder with data files

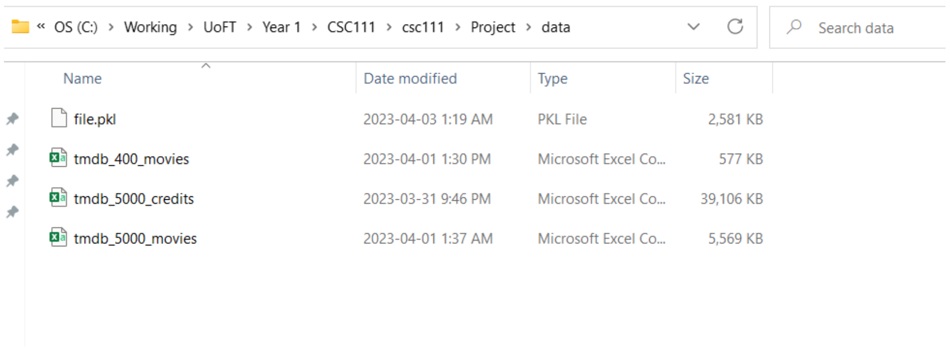


This PC › OS (C:) › Working › UoFT › Year 1 › CSC111 › csc111 › Project

Search Project

Name	Date modified	Type	Size
__pycache__	2023-04-03 10:09 AM	File folder	
data	2023-04-03 1:19 AM	File folder	
Requirements	2023-04-03 10:14 AM	Text Document	1 KB

Inside data folder



« OS (C:) › Working › UoFT › Year 1 › CSC111 › csc111 › Project › data

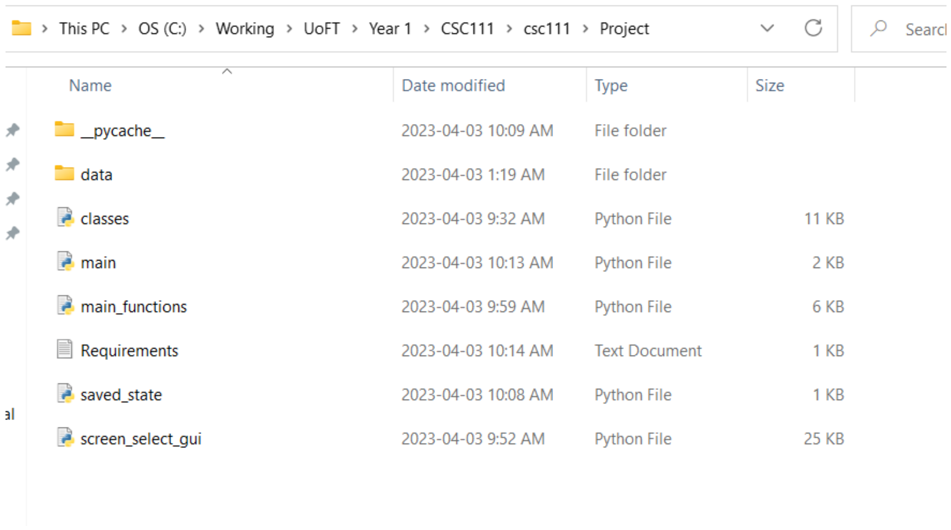
Search data

Name	Date modified	Type	Size
file.pkl	2023-04-03 1:19 AM	PKL File	2,581 KB
tmdb_400_movies	2023-04-01 1:30 PM	Microsoft Excel Co...	577 KB
tmdb_5000_credits	2023-03-31 9:46 PM	Microsoft Excel Co...	39,106 KB
tmdb_5000_movies	2023-04-01 1:37 AM	Microsoft Excel Co...	5,569 KB

5. Download all the *.py* files uploaded on MarkUs Proj_02: Course Project Phase 2: Submission page.

   (a) Refactor all the *.py* files into the same directory the datasets are in.

Directory folder with all files



› This PC › OS (C:) › Working › UoFT › Year 1 › CSC111 › csc111 › Project

Searc

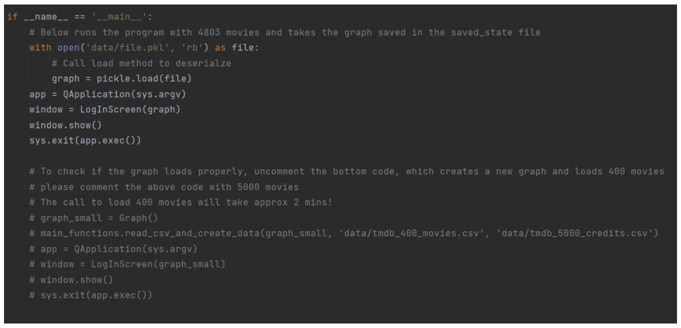
Name	Date modified	Type	Size
__pycache__	2023-04-03 10:09 AM	File folder	
data	2023-04-03 1:19 AM	File folder	
classes	2023-04-03 9:32 AM	Python File	11 KB
main	2023-04-03 10:13 AM	Python File	2 KB
main_functions	2023-04-03 9:59 AM	Python File	6 KB
Requirements	2023-04-03 10:14 AM	Text Document	1 KB
saved_state	2023-04-03 10:08 AM	Python File	1 KB
screen_select_gui	2023-04-03 9:52 AM	Python File	25 KB

Running ScreenSelect: New User

1. Open the *main.py* file and run the file.

   - With the .pkl, the program will take a few seconds for the login screen to pop up. This pkl file has the 5000 movies saved into a graph as a saved state. This creates the running of the program with 5000 movies quickly. To test how the loading of the data in the graph works comment out the first set of code and uncomment the bottom code which requires the tmdb_400_movies.csv. The tmdb_400_movies.csv (400 observations only), will take about 1 min before a screen pops up.

main.py



```
if __name__ == '__main__':
    # Below runs the program with 4803 movies and takes the graph saved in the saved_state file
    with open('data/file.pkl', 'rb') as file:
        # Call load method to deserialze
        graph = pickle.load(file)
    app = QApplication(sys.argv)
    window = LogInScreen(graph)
    window.show()
    sys.exit(app.exec())

    # To check if the graph loads properly, uncomment the bottom code, which creates a new graph and loads 400 movies
    # please comment the above code with 5000 movies
    # The call to load 400 movies will take approx 2 mins!
    # graph_small = Graph()
    # main_functions.read_csv_and_create_data(graph_small, 'data/tmdb_400_movies.csv', 'data/tmdb_5000_credits.csv')
    # app = QApplication(sys.argv)
    # window = LogInScreen(graph_small)
    # window.show()
    # sys.exit(app.exec())
```

2. After the file is done running, it will show a Login Screen.
    - In the Login Screen, there is a textbox where the user can type in their username. Below the textbox, there are two buttons labeled 'sign-in' and 'sign-up'.
3. Type in a unique username in the provided textbox and click on 'sign-up'.
4. The next screen, Preference Screen, will appear.
    - There are empty textboxes for the user to type in their preferences of the categories written on the left: genre, language, keywords, and director. Note that there are up to 3 empty boxes for keywords and the answer for director is specified as optional. You do not have to fill out all three keywords or any other input boxes. At the bottom of the screen, there are two buttons labeled 'Log Out' and 'Recommend'.
5. Fill out any textbox based on your movie preferences.
    - There are sample answers beside each category for the format in which the inputs must be entered. Please follow the syntax given espically the Uppercase and Lower Case Letters.
6. Once the necessary textboxes are filled out, press the 'Recommend' button for the next screen, the Recommendation Screen, to appear.
    - The screen is divided evenly into 5 sections. Each section contains information about a movie recommended to the user. The title of the movie is at the top followed by the director, runtime, language, and release date. Below each section, there is a button labeled 'Select' and 'Back'. Please FullScreen this page for Optimal viewing.
7. Select a movie you would be interested in watching by clicking on the 'Select' button. The program will add the movie to your selection, and take you back to the Preference Screen.
8. Additional Steps You Can Take
    - Repeat steps 5∼7 for more movie recommendations.
    - To log out of the program, press the 'Log Out' button on the Preference Screen. It will take you back to the Login Screen.
        - You may create a new account with a different username and start all over again by repeating steps 3∼7.
    - To completely quit the program, click on the red X mark located in the top right corner of the screen.

Running ScreenSelect: Existing User

1. If you have not closed the program, yet, you can log back into the same account.
2. In the textbox of the Login Screen, type the previously used username.
3. Press the 'sign-in' button for the program to switch its screen to the Preference Page.
4. Follow steps 5∼7 written in the instructions for a New User.
    - You may observe that this time, the recommended movies not only are in line with the preferences you have just input but also include your previous preferences because the previously selected movies stored in your account were referenced for the recommendation procedure. This may be prominent after a few watched movies on that account.

Part 5: Changes to Project Plan

Classes

- The Graph class was modified with four new public methods **add_user_vertex**, **retrieve_item_obj**, **retrieve_vertex_dict** and **add_movie_vertex**. The preceding methods add user and/or movie vertices to the graph and give access to its vertex objects which are mapped in a private instance attribute dictionary if other functions need it.

- The _Vertex superclass has changed from a class to a data class for cleaner code. Its instance attributes have all been changed to have a default None value when if the correct type isn't passed. One of its instances attributes **genre**'s type contract changed from str to also include Optional[set[str]] to allow a set of str for the _Movie class which has more than one genre in the Movies dataset. A public instance attribute **neighbours** was removed from the superclass and was added separately to the _Movie and _User subclasses. Representation invariants were added.

- _Movie class's private instance attribute **_total_score**'s type contract changed from an int (score of the movie) to a dictionary with keys as the usernames [str] and the values as the corresponding _User object. The dictionary allowed the system to efficiently access the mapped preference scores of multiple users by their usernames for a specific movie instead of having to compute the score for each user and movie all over again. Representation invariants were added.

- A new public instance attribute **past_10_neighbours** was added to the _User class to keep track of the recently watched/selected movies and use them (their scores) to recommend the next movie. Two new public methods **retrieve_top_scores** and **modify_preferences** were added to this class. These methods give access to its private instance attribute _top_scores if other functions need it and reassign some of its public instance attributes' values respectively. _top_scores was changed from a dictionary to a list with tuples. The tuple contains the (score, the _Movie vertex). The change allows duplicate scores to exist for multiple _Movie vertices which were not allowed for a dictionary since all keys need to be unique. Another instance attribute **neighbours** was changed from set with to a dict where id numbers were mapped to their corresponding movie vertices. This change was due to encountering an unexpected hash error for computations on sets. Finally, representation invariants were added.

Overall Program

- The plan to use a username and a password for the user login procedure changed to only using the username. Passwords were not used in the program except for the login procedure and each user object has a unique username as its key in the graph's _vertices dictionary. The role of passwords was not much important, so it was excluded from our program.

- The plan to display the recommended movies' posters on the user interface (screen/window) was dropped. It took longer than expected to retrieve the image from the website link and resize it to fit the UI screen.

- The **user_log_in** and **_user_scores** function was removed from *main_functions.py*. We realized that the GUI had its own process to accept and process text, logins etc and therefore we discarded such functions that used the input() function to input via the console. We decomposed the function into snippets and put parts of it in the methods of PreferenceScreen class in *screen_select_gui.py*.

- The files, except main.py, have been tested for PythonTA and have applied the modification of our python_ta configuration call in our main blocks as per our requirements. We are commenting out @check_contracts as it will slow down the program.

- The use of modules like ast and pickle were also added. ast was used in filtering out specific keywords from the data, whereas pickle was used in creating a saved state with 5000 movies. The idea of using previous inputs and recommended movies was also included as we wanted to give different movies several chances, in case the user doesn't select them in the first go.

Part 6: Discussion

Our goal in this project was to create a movie recommendation app that provides users with tailored recommendations based on their viewing habits and preferences. We understood that such platforms exist in the market, however, we were determined to offer the best suggestions with the highest accuracy for desire. This was done by disregarding our user's geographical location. In order to consistently provide the best suggestions for each user, we also employed a content-based filtering algorithm that takes into account contextual factors including the most recent 10 movies seen, keywords, genre, and language.

We are very proud to say that the results of our computational exploration helped us achieve our goal. Our computational investigation's findings (through self-testing) demonstrated that our system was very effective at giving consumers personalized movie selections. Compared to traditional content-based filtering algorithms that simply take into consideration the qualities of each piece of content, our system was able to provide more customized suggestions. This is largely due to the fact that it takes contextual factors and most recently watched into account. We are confident in this conclusion thanks to the countless tests we have conducted both on larger as well as smaller datasets. We have also separately tested all possible filters for our movie dataset on our GUI to ensure that the recommendations are all successful in achieving the end goal for users. Moreover, we have tested all of this for multiple users, and every test returns the anticipated results.

However, we did encounter some limitations during the development of our app. One limitation was the difficulty in loading the movie data into the graph due to the large amount of data and the different formats in which it was presented. To fix the issue of loading we explored multiple libraries like Json and Pickle to create a saved state. We eventually used Pickle to create a file with the saved state of the graph. Now in cleaning out and loading the data in the graph object, we had some difficulties in getting the genre and keywords since they were a string of a list with multiple dictionaries inside. First, we used for loops to access each piece and get rid of unnecessary details, but we were not able to typecast the string into different dictionaries of lists. Eventually, we went with the last library and the function literal.eval() to access each piece of needed data. Another problem was getting the director of the movie from a different csv file. This was achieved by creating a separate private function that first checks the movie name and then takes out the director's name from the given crew members.

Another limitation was the difficulty in adding images to the graphic interface of each movie when displaying its title. As the GUI was quite small, we were not able to see how to add images without messing up the layout. Therefore, we chose to take this out and implement a clean version in the future by changing the GUI from the layout import that we have currently used to a fully custom-made file that supports images as well. We believe that this is an achievable goal to set for the future; and it would enhance our current UI even more.

One additional constraint that we encountered pertained to the PyQT6 library. A significant proportion of the features that we initially expected to employ for receiving input proved to be inadequate as there was no direct link between them and the graphical user interface (GUI). As a result, we were prompted to revise our strategy and gather input exclusively from the GUI, while gathering it as an object. This object was a QLineEdit() object which could be turned into a string using the .displaytext() method. We then process out computations on this created object. The basis for collecting user information differed from our original expectations, necessitating a shift in its source from the console to the GUI, this indeed simplified our code because we could directly call the functions and computations on this given input by the user.

In terms of the next steps for further exploration, we plan on adding strong security to our app by implementing encryption using concepts we learned in CSC110. We will also ask users for their ages and sort movies by age group so that children would never receive 13+/18+ content. Unfortunately, our datasets do not have age ratings for any of the movies listed, and we are limited from doing this in that regard. Our long-term plan is to cross reference all of our movie information with that of another data set that lists ages as well all in order to implement the safest algorithm for our younger users. Additionally, we could display the user's past top preferences, those that the user has already chosen, to provide more personalized recommendations. This would come in handy when users wish to re-watch past favorites.

To further improve our recommendation system, we also plan to implement a wishlist feature inspired by the "watch later" concept on YouTube. This feature will allow users to save movies they are interested in but do not have time to watch at the moment, thus providing our AI with more data to make personalized recommendations. By incorporating this feature, we hope to enhance the accuracy of our system and increase user engagement with ScreenSelect.

In our current implementation, a major limitation we encountered was the loss of our graph object upon closing the tab. To address this issue and facilitate the storage of user data in the future, we plan to develop a central database to store all information collected during each session. This will be done by mutating the same graph object, even when another tab is opened, the graph will be used which will have the previous users already saved. By storing user preferences in the format of a graph, we can easily manipulate this data and provide more personalized recommendations. Another crucial next step is optimizing the speed of our project. To achieve this, we are exploring technical options to streamline the loading and analysis process by reducing the running time of the functions like score and read_csv_and_create_data. Furthermore, we plan to expand the functionality of ScreenSelect by scraping the web for streaming platform links, allowing users to access third-party sites and watch their selected movies. Only after this would we record movies as "watched". Our current implementation determines that users have watched a movie if they simply click on it, which is not as accurate as we hope for the final version of the platform to be. By doing all of the above, we can enhance the accuracy of our recommendations and provide a more seamless user experience.

To conclude this written report, our project was successful in developing a movie recommendation app that offers tailored suggestions to users based on their watching patterns and current/previous interests. While we encountered some limitations, we were able to overcome them and offer a useful tool for movie enthusiasts. Future improvements could be made to enhance security, age-based filtering, and display past user preferences.

Part 7: References—IEEE Format

Course Notes

- D. Liu and M. Badr. "Course Notes for CSC110 and CSC111." Foundations of Computer Science. https://www.teach.cs.toronto.edu/ csc110y/fall/notes/ (accessed Mar. 5, 2023)

Data Set

- A. Dattatray, tmdb_5000_credits, vol.1, Kaggle: Kaggle, 2022. [Dataset]. Available: https://www.kaggle.com/datasets/akshaydattatraykhare/movies-dataset. [Accessed: Mar. 5, 2022].
- A. Dattatray, tmdb_5000_movies, vol.1, Kaggle: Kaggle, 2022. [Dataset]. Available: https://www.kaggle.com/datasets/akshaydattatraykhare/movies-dataset. [Accessed: Mar. 5, 2022].

Module Documentation

- Riverbank Computing and The Qt Company. "Reference Guide — PyQt Documentation v6.4.1." Riverbank Computing. https://www.riverbankcomputing.com/static/Docs/PyQt6/ (Accessed: Mar. 5, 2022).
- J. Bodnar. "Python PyQt6." ZetCode. https://zetcode.com/pyqt6/introduction/ (Accessed Mar. 7, 2023)
- The Qt Company. "Qt for Python" Qt. https://doc.qt.io/qtforpython/index.html (Accessed Mar. 8, 2023)

Idea Research

- M. Gavira. "How Netflix uses AI and Data to conquer the world." LinkedIn. https://www.linkedin.com/pulse/how-netflix-uses-ai-data-conquer-world-mario-gavira/ (Accessed Mar. 7, 2023)
- J. Ciancutti. "How We Determine Product Success." Netflix Technology Blog. https://netflixtechblog.com/how-we-determine-product-success-980f81f0047e (Accessed Mar. 7, 2023)
- D. Chong. "Deep Dive into Netflix's Recommender System." Towards Data Science. https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48 (Accessed Mar. 7, 2023)

Inline Citations (Listed in order of use)

- Netflix Help Center. (n.d.). How Netflix's Recommendations System Works. Privacy and Security Help Page. Retrieved March 26, 2023, from https://help.netflix.com/en/node/100639
- Help Center. (n.d.). How to See Viewing History and Device Activity. Privacy and Security Help Page. Retrieved March 28, 2023, from https://help.netflix.com/en/node/101917