Game recommendation generator

Ruiting Chen, Jiankun Wei, Qianxi Gao, Kexuan Zhang

April 2021

1 Problem Description and Research Question

For a large amount of the population today, playing video games has passed the border of simply a mean of relaxation, but it also serves as a tool for social interaction. Seeing this opportunity, many gaming platforms were launched. Among them, Steam has been one of the most renowned video game digital distribution services, which allows users to select games in a web-based online store. However, as more and more game designers, both professional and amateurish, registered on Steam, they have supplied tremendously many games to the platform. This abundance of games seems to augment users' choices, but in most cases, users are overwhelmed by the enormous amount of choices. Therefore, a recommendation system would save a lot of effort from the user.
Our research question is **"How can we recommend new games to people based on what games they've played?"** As a gamer, when I use the steam recommendation function, it sometimes recommends games that I am interested in, but not always. We want to try to find an efficient way to recommend games for users based on their profile and preference. We analysis the similarity of different game, combined with popularity, give our user a game recommendation list. So our goal is to create an interactive recommender, we will recommend games for user based the games they already 'liked'. The user also could choose to save the list of game that he/she is interested in, in terms of names and steam urls by our program.

2 Dataset Description

For this project, the dataset used to generate games and reviews graph is `australian_user_reviews.json`, which is provided in json format and downloaded from https://cseweb.ucsd.edu/~jmcauley/datasets.html Version 1: Review Data (6.7mb) as a zip file and decompressed into a .json file. This .json file contains but not limit to the following data:

1. user_id: the identification of the user on steam
2. reviews: a sequence of games that the user gave reviews. For each game, the dataset contains:
   (a) item_id: the identification of the game
   (b) recommend: whether the user recommend the game or not

Due to the format of json file, each variable can be accessed as a dictionary, which enables further manipulation on the data. To provide flexibility for the user, we defined the first 2000 lines as the "small" dataset and the entire dataset as the "large" dataset, which provides options for the user (explained in details later)

Since australian_user_reviews.json is quite large, generating a game graph that require calculation of similarity score using the whole set of data takes a long time. We provide a processed_game_graph.txt which contains processed data ready to be directly read in to create a game graph. The format of processed_game_graph.txt is described below:

- Upper half: each line represent a vertex in game graph, in the format 'gameid popularity', ex: 369400 0.00015504476917709988
- Separation: a line that rights 'SPLIT_FROM_HERE' which is an indicator of separation of the upper half and second
- Lower half: each line represent a vertex in game graph, in the format 'gameid1 gameid2 similarity', ex: 315810 71340 0.08333333333333333

3 Computational Overview

Loading Data
We will make a graph called "review_graph" to represent the relationship between the players and games they play using the data from australian_user_review.json. In this file, each user review is represented as a string(more precisely, a dictionary with quotation marks around it.) We will use ast.literal_eval from new library ast to take out the dictionary. In each dictionary, there is a key called "user_id" which maps to the id of this user and another key called "reviews" which maps to a list of dictionaries where each dictionary is a game that the user reviewed on. Each of the inner dictionary contains the game's integer id code and whether the user recommend this game or not. We will use these data to create our review_graph.
Using the "australian_user_review" Data dataset, we will read in the user id and his reviewed games and his recommend reviewed each game through the function "load_weighted_review_graph" to create a review_graph of WeightedGraph class where each user and game in the graph will be a vertex of _WeightedVertex class. Both classes follow a similarly structure as WeightedGraph and _WeightedVertex from A3. Each vertex is has 4 instance attributes: 1) id: id of the user or game, 2) kind: indicate whether this vertex is a game or a user, 3) neighbours: dictionary where the keys are all neighbours of this vertex and values are the weight of the edge. Weight is determined base on whether a user recommend a game. If a user recommends the game, the weight will be 1, if not the weight will be -1. 4) popularity: if a vertex is a game, its popularity is equal to the number of users reviewed it divided by the total number of users. If a vertex is a user, its popularity attribute will be set to 0.
The review graph contains an instance attribute _vertices which is a dictionary mapping the id of a vertex to the vertex.
The similarity_score method for _WeightedVertex class is the same as in A3 where for given two games, we count the number of users that both recommend or both not recommend the two game, and divide the number of users that played at least one of the two games. The resulting number is the similarity score of this two games.

Processing Data
We will create a "game_graph" based on review_graph using the "create_game_graph" function. This function takes in the review_graph and a threshold value to create a graph with only games as it vertices and an edge is only added between two vertices if their similarity score exceeds the threshold, otherwise they are not considered as neighbours. The weight is the similarity score in this graph. This graph is introduced mainly to significantly reduce the running time of the program's algorithm to recommend games. Although it takes some time at the beginning to create this graph, the program later takes very very short time to recommend games, which is a nice trade-off. Since the game_graph generation takes a long time, which increases the waiting time at the start of the program, we used create_processed_game_data_file to generate a processed_game_graph.txt file(described in Dataset Description) which contains processed data ready to be directly read in to create a game graph. load_game_graph_from_processed will be used to load data from processed_game_graph.txt into a game_graph.
The game_graph plays an important role in this program because it stores all the information needed for future game recommendation and interactive visualization.

The Game Recommending Algorithm
Our program recommend games base on the client's feedback on each recommendations(interested or not interested), games that have already been recommended, similarity between games, and how popular is each game. The functions for game recommendation are all in recommender.py file. In each turn of recommendation, we will generate 5 games to recommend.
At the beginning of the program, since there is no client's feedback yet, we will use sort_by_popularity to return the top 5 popular game in our game_graph as the current recommend game list. sort_by_popularity sorts the popularity attribute of each game (using k_largest and _partition as helper function) and return top 5 popular games.
For the rest of program after start, we will use game_recommender to generate 5 new game recommendations. game_recommender uses the following algorithm: using the game_graph, for each game recommendation the client likes, we will take all its neighbours that have not been recommended and put them as the key of a dictionary which will map to its recommendation score. The recommendation score of a game is calculated by adding 50% of the similarity score between this game and the game recommendation that the client like (which is the weight of the edge between them) and 50% of the popularity of this game. If multiple game recommendations the client like have a common game as their neighbour, this game's recommendation score is the sum of recommendation score with each of the game recommendation that the client like. When all these calculations finished, we convert

the dictionary into a list of tuples to sort the games from highest to lowest recommendation score. We will output the top 5 games base on recommendation score. If the output list has less than 5 games, we will use sort_by_popularity to add in the most popular games that had not been recommended fill up to 5 games for recommendation.

User Interface
To visualize the recommend results, we created a file tkinter_interface.py that contains a class called Interface which is responsible to create an interactive window using the new library, tkinter, and store all the information necessary to recommend games.
This class keeps track of the game recommendations the user like, the games that have alrealy been recommended, the current game list to recommend, the index of game that is currently being recommend in the current game list as its attributes. (There are more class attributes, details see tkinter_interface.py)
The start_recommender method is the method that starts and maintains the events happen on the whole interface. It will displays the 'start' interface of the program. A new library PIL is used in this method to add the background image on the start interface. When clicking "start" on the start interface, start_command will run, which displays a page that has 2 buttons('small' and 'large'). Client will choose whether he/she want to load the large dataset(from processed_game_graph.txt) or small dataset (first 2000 lines of australian_user_review.json or the test_data; the current default is the former one, you can change this in tkinter_interface.py).
When a selection of dataset is made, load_data_commend will be called. Data will be loaded to game_graph. Then sort_by_popularity will generate the game list to be recommended and store it in curr_recommend attribute of Interface class. The first game recommendation will be opened in browser by open_next_web which uses a new library webbrowser to open game in browser given game id. (Please, steam is constantly updating its database, the id of some game cannot be found on steam's website, in this case the main webpage of steam will be opened). The tkinter window will prompted the client to indicate whether he/she is interested about this game or not by clicking on the corresponding button. The window contains reaction buttons('interasted' and 'not interested')on the bottom and 'save' button on the top right corner.
If a reaction button is clicked, reaction_commend method will be called. If the client is interested in the game, the program will add this game to both Interface.like and Interface.recommend. If not intersested, the game will only be added to Interface.recommend so that it will not be recommended again. On the top of the window, there is a line of text showing the number of games the client liked. Every time a new game is liked, this number increases by 1. This method will also use open_next_web to open the website for the next gane recommendation. If the Interface.curr_recommend list is exhausted, a new list of game will be be recommended by game_recommender in the recommender.py and reassigned to Interface.curr_recommend and the Interface.curr_index will be set to 0. These processes repeats until the client close the tkinter window or the client have visited all the games in the dataset and no more recommendations can be made, which end the program.
If save button is clicked, save_command method will be called that allows the user to save the name of all the games he liked and the url of that game into my_list.txt. This method uses a new library steamfront to look up game name in steam using game id. (If a game cannot be found, the game will be ignored and not added to my_list.txt)
When creating small data, large data, interested and not interested buttons, partial in a new library functiontools is used pass in argument for the corresponding load_data_commend and reaction_commend.

4 Obtaining Dataset and Running the Program

- **Requirements**
  Download the required python library in the requirement file.
- **Dataset:**
  1. Original: https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data.
  2. Download the data from UTsent and decompress it in the same folder with other code files.

     UTsent information:
     Claim ID: 5eSGSxnu7Q8SaaxK
     Claim Passcode: Epjq8xwizhhuYBXc
- **Running the Python file**

1. Run the `main.py` file in the python console
2. After running the file, a window will pop up
3. Click the Start button to start

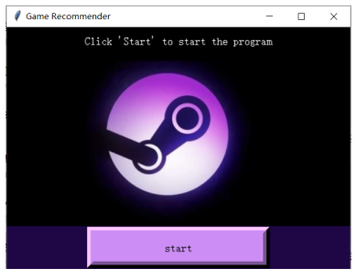

Figure 1: An image of start interface

4. A option between large dataset and small dataset is provided. If clicking the "Small" button, the program would work on the small dataset(the first 2000 lines), which runs about 1 minutes; If clicking the "Large" button, the program would work on the processed_game_graph.txt and create a game graph for all the games. This takes approximately 15 minutes.

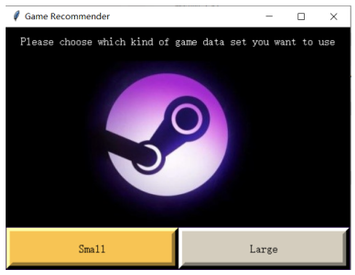

Figure 2: An image of data size selection

5. After loading the data, a steam web page of recommend game would be opened, in which you can browse, comment, and download the game.

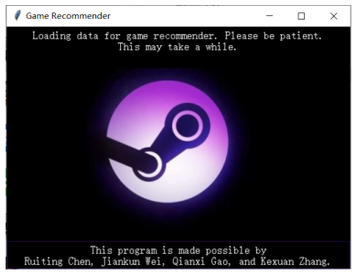

Figure 3: An image of loading

6. Decide whether you like the game or not by clicking the "Interested" or "Not Interested" buttons, or get the list of games that you have clicked "Interested" by clicking the "Save games list" button.

7. if you click the "save games list" button in case you have not chosen any interested game, a window would pop up to warn you that you have not added any game to the favorite list; if you click the "save games list" button in case there are games in the favorite list, the program will update the "my_list.txt" file to contain the game you are interested so far, and you can check the list by accessing the "my_list.txt" file in the folder.

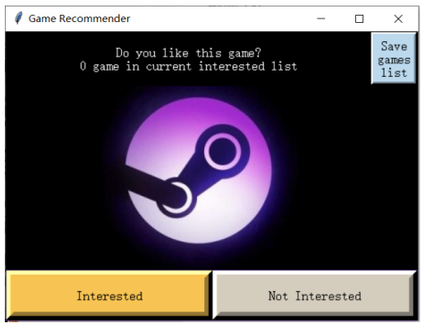

Figure 4: An image of Game recommending

Figure 5: An image of Empty Error

8. Here is an example list that the program may generate.

my_list.txt

```
Rust: https://store.steampowered.com/app/252490
ARK: Survival Evolved: https://store.steampowered.com/app/346110
Counter-Strike: Global Offensive: https://store.steampowered.com/app/730
PAYDAY 2: https://store.steampowered.com/app/218620
Warframe: https://store.steampowered.com/app/230410
Dota 2: https://store.steampowered.com/app/570
Left 4 Dead 2: https://store.steampowered.com/app/550
DayZ: https://store.steampowered.com/app/221100
Unturned: https://store.steampowered.com/app/304930
Garry's Mod: https://store.steampowered.com/app/4000
The Elder Scrolls V: Skyrim: https://store.steampowered.com/app/72850
Terraria: https://store.steampowered.com/app/105600
Team Fortress 2: https://store.steampowered.com/app/440
```

9. After you make the choice, the interface would update to indicate how many games you are interested so far, and open a new steam web page to recommend a new game for you.
10. When you feel enough games are recommended, and unable to hold yourself to try some games, just click the close button and go play! In the rare case where you've gone through all the recommendation of all the games in the data set, there is no more game recommendations. A text window will pop out telling you to exit the program.

5 Changes:

- 1. we no longer will read the "Item Metadata" file because the "australian_user_review" data file already has all the information we need.
- 2. we no long need Game class and Player class as subclasses of the _WeightedVertex class because they are unnecessary and will make our code too complicated. Instead, we introduced a new instance attribute of each vertex called "kind" to specify which kind that vertex is. Also the WeightedGraph class will just have one instance attribute which is a dictionary mapping the id of a vertex to a _WeightedVertex.
- 3. Each _WeightedVertex will have a instance attribute popularity to store the popularity score of the game if the vertex is a game, or 0 if it is a user.
- 4. The client will no longer be prompted to type in 10 games that they like because it not only makes the program time consuming and hard to use but is also unnecessary complicated to have to handle unexpected inputs. Instead, we first recommend 5 most popular games for the user to indicate whether he like any of them.
- 5. We no longer traverse through the graph to find game with relations to current games because that takes a long time to run especially when the graph is large. Instead

Figure 6: An image of successfully saved

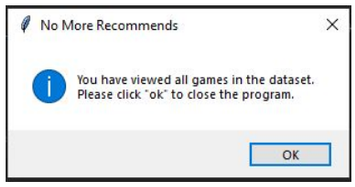

Figure 7: An image of Running out of Recommendations and Exiting the Program

we pre-load a graph that already have the similarity score between two games set as the weight of edge between them. We we need to recommend, we can just find the neighbours of current games and recommend through the weight of the edge and the popularity of the game.

6 Discussion:

We think our project solve our research question: **How can we recommend new games to people based on what games they've played?**. Since we can get the similarity and popularity based on the user feedback and user data in the data set, we can recommend games based on how similar the games are. If a user gives similar recommendations (like both recommend or both not recommend two games), then it suggests that this two games are similar. We can use this ideas to first collect some games that people like and recommend similar games to them. We also aware that people may feel boring when playing too much games that are similar, so we also take popularity into consideration. We recommend games through both similarity and popularity so that the game are similar but new at the same time.

- In terms of **new python library application**, we use tkinter to make our program interactive. However, it is a brand new for us so we encounter some problems when writing code. We try to make our program clear and concise, then we faced some problems in implementation. For example, in order to design one command function for both 'Interested' button and 'Not Interested' button, we need to find a way to pass arguments into the commend function to distinguish which button is clicked. At first we tried the Attribute called 'VAR', which works for RadioButton wediget but not for the Button, then we find online resource and solve this problem by using partial function. Also, when we were trying to implement 'how many games in 'interested' list' function, the text remain unchanged when a game is liked but the count is expected to increase by one. First we tried to add `self.root.mainloop()`, and it is not work. After reading some online resources, we use `self.root.update()` to fix that, and the function works nicely.

- In terms of **using dataset**, the dataset is too large and everytime the program need to generate a new list of games to recommend, it takes a long time to run. At first, We then tried to make the algorithm create a new graph after every game that chose to be interesting, and that is really slow. Then, we use a nested loop to loop through each game in the graph and each game in the games that are liked, and the running time did not change much. We then tried to use k-largest algorithm to improve the sorting, but since we are still checking all games in the graph every time we need to recommend, there is only limited improvement. Finally we decide to pre-processing the graph at the beginning of the program so that we no longer need to check all games in the graph but only need to check games adjacent to games that's liked, so sacrificed some time at the beginning of the program, but every time we recommend, it is very fast, which is a nice tradeoff.

- A different **dataset using** problem we faced is that we want to make a function that could print the game name and the url into a txt file, and we assume we can read a data file with game id and related name to a dictionary and search the game id in

that dictionary, but that dataset is 7GB! We are thwarted knowing it will take a huge amount of time to run and take up a lot of memory. We kept doing research and finially fix this by using steam API, then we can easily get the name by passing the id of the game to a function in the steamfront library which will get the name from steam website.

- **Dataset using** problem: The dataset we found is not up-to-date enough such that many games in the dataset is not availible in steam now. If we recommend this kind of game to users, our program will just open the steam home page. In this case, we suppose our user can choose 'Not Interested'. Even if they choose 'Interested', the game will not appear in the output txt.

- **Next step:**
  1. We may consider the reviews made by users in our dataset. We can use natural language processing tools to process the reviews and get the characteristics the user thinks the game has can cluster games with similar characteristics. This will greatly improve our recommendation algorithm.
  2. We may provide the function that a steam user could provide his id and we can get the game and review that he already has, then make a more precise recommendations, rather than just use popularity purely to generate the first list of recommendations.
  3. We may add 'seniority' attribute in our recommendation system, for new gamer and senior one, they may have specific game preference, for example, senior gamer may prefer the game with high difficulty and a new gamer may need a easier one to begin. For the user who has more than 50 games, we may consider them as senior, then make some conclusion of this senior gamer could help us to improve our recommendation system.

7 Reference:

Anthony McGlynn, "Steam kicks off 2021 by crossing 25 million concurrent users for first time" PCGames. Jan 2, 2021. https://www.pcgamesn.com/steam/concurrent-users-2021

Apurva Pathak, Kshitiz Gupta, Julian McAuley. "Generating and personalizing bundle recommendations on Steam"http://cseweb.ucsd.edu/~jmcauley/pdfs/sigir17.pdf

Wang-Cheng Kang, Julian McAuley. "Self-attentive sequential recommendation" ICDM, 2018. https://cseweb.ucsd.edu/~jmcauley/pdfs/icdm18.pdf

Mengting Wan, Julian McAuley. "Item recommendation on monotonic behavior chains" RecSys, 2018. http://cseweb.ucsd.edu/~jmcauley/pdfs/recsys18b.pdf

Teacher of Computing. "Learn Tkinter in 20 Minutes". Youtube. Jan 17, 2017. https://www.youtube.com/watch?v=_lSNIrR1nZU&t=403s