CSC111 Project: Book Recommendation System Development for Goodreads

Jasmine Zhuang, Ke Gong, Nuo Xu, Elaine Dai

Friday, April 16, 2021

1. Problem Description and Research Question

The project focuses on creating a recommender system based on clustering for Goodreads online reading website.

Reading books develops one's brain, improves one's ability to better understand life and enrich one's inner life. However, as there are so many books available, finding an appropriate book that one enjoys, can sometimes be time-consuming. In recent days, an increasing number of readers choose to purchase and read books online due to their efficiency. Millions of books published in present times have increased the difficulties of finding a suitable book for readers to appreciate. When users search on the online book store, they will find various books swarm into their sight almost at the same time. It is stressful for the buyer to choose which book is popular and deserve to purchase if they want to purchase a book while without a book title in mind. It is a barrier for readers to discover a new book suitable for them in the library or online book stores, especially during the epidemic period, without other's recommendations. For this reason, we are interested in finding a way that may reduce the seeking time for books to read next.

A book recommendation system is a useful tool in solving information overload and helping users finding books that they want to get in the shortest possible time. In this project, we will create a program that sorts the books into clusters and automatically suggests high-rated books based on the clusters that contain the input book from the user. The program will be able to take in user feedback for dissatisfied books in the list, and refresh the suggestion list every time it receives the feedback. In this sense, our group believes it will be a powerful technology that may help people choose a similar and popular book more effectively.

2. Name and Description of All Data Sets

We obtained our data from Goodreads, one of the world's largest sites for readers and book recommendations that was launched in January 2007. The dataset that we used is part of the Goodreads Datasets, which were collected in late 2017 from goodreads.com, where only users' public shelves were scraped. All the data are stored in csv files, including `dataset.csv`, `clusters_500.csv`, `clusters_200.csv`, `clusters_150.csv`, `clusters_100.csv`, `clusters_50.csv`, and `clusters_20.csv`.

`dataset.csv`: We filtered the original user-book interaction dataset to leave out the observations that do not include the rating (i.e. rating is 0) and kept only the columns of `user_id`, `book_id`, and `rating`. Since the dataset is too large, we only kept the data of the first 3000 books. Each line in `dataset.csv` represents a record of user-book interaction. The first column is the order of the observation, the second column is the user id, the third column is the book id and the last column is the rating that the user gave to the corresponding book.

`clusters_n.csv`: The rest of the files are generated by our program. We sorted the books into clusters, transformed the result of clustering from a list of sets of book-ids to a pandas.dataframe and stored it into a csv file. The number in the file name is the number of clusters in the corresponding file. Each row in the csv file contains several book_ids, which represents one cluster.

3. Computational Overview

Generally, the class `_WeightedVertex` is similar to what we've learned in lectures. We added two new instance attributes, `kind` and `average_rating`, and modified the attribute `neighbours`. In the `_WeightedVertex` class, `kind` refers to the type of the vertex('user' or 'book'), `neighbours` refers a dictionary mapping from vertices that are adjacent to this vertex to the weight of edges. For weighted vertex of kind "book", `average_rating` is the average rating of the book given by users. For weighted vertex of kind of "user", `average_rating` is the average rating making by the user.

The class `_WeightedGraph` is the same as the Graph we've learned, but now we use the `_WeightedVertex`.

We designed two main functions:

(1) Generating csv file which contains information of clusters of books which can be used to find recommended books later.

(2) Finding recommended books based on the information of clusters and a graph obtained from the original data set.

For satisfying function (1):

we first load a weighted graph with the original data set, which contains two kinds of weighted vertex. The edge is added if a user read a particular book before. Then, we use this graph to generate a book graph, which contains only weighted vertices of kind "book". The vertices in the book graph is connected based on the similarity score, which is the same as in the assignment 3. Later, we find clusters which maximize the cross-cluster weight, which is also similar to what we done in the assignment.

The key difference is, we add a new argument, `filepath`, in the function `find_clusters`, which is a string with default value of "". By adding `filepath`, we are allowed to save the information of clusters into a csv file, which can be used in later recommendations. Since the generation of clusters takes a long time, the stored csv file increase the efficiency for our system to a great extent.

During the process of storing csv files, we use Pandas library to formulate a data frame and then write information into a csv file by calling the function `to_csv()`.

For satisfying function (2):

we'll use the existed csv file(s) and a given interested book to find a list of all books which are in the same cluster as the interested book, and sorted based on their average rating.

We have provided several csv files for different numbers of clusters. By default, if users are not intended to input their own csv files, the recommendation system will automatically use the provided files if they are stored in the right folder.

As we call the function `find_all_recommended_books`, files contain books grouped by different numbers of clusters(i.e. 500, 200, 150, 100, 50, 20) are inputted to the helper function `find_recommended_books` one by one through the for loop iterations. Since the number of clusters decreases as files continue to be inputted in, the size of clusters that includes the book given by the user increases through iterations, which means a wider range of books become potential recommended books.

Figure 1: 100 clusters

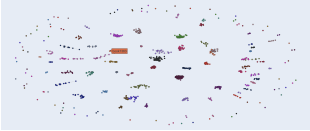

Figure 2: 150 clusters

Figure 3: 200 clusters

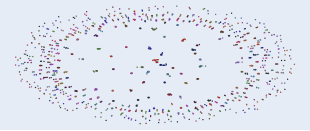

Figure 4: 500 clusters

Within each call to the helper function, we will sort books that are located in the cluster contained the given book based on each book's average rating. The helper function will return a list of books that are related to the given book and sorted from high rating to low rating. The sorted list will then be extended into the accumulator in the function `find_all_recommended_books`. Also, `excluded` is updated to avoid repetition of books in the resulted list.

The list returned by `find_all_recommended_books` will not be fully displayed to the users, which is controlled in the main.py

Finally, we will create an **interactive system** using python library called `tkinter` as described below.
There will be 2 main widows for data collection, then we will have 5 book recommended for user based on these inputs collected.

Window 1:

If user doesn't want to custom a csv.file(since it will take a long time) for their book recommendation, then user can click the button "Skip this page" and move onto the next window that will automatically pop up.

Otherwise, user will need to enter both a file path to save the generated csv file and an integer less than the total number of books indicating the number of clusters for the csv file to be generated, into the 2 boxes we created using codes like:
path = Entry(starting2, width=20, bg='white')
path.grid(row=3, column=0, sticky=W)

The instructions are also provided inside this window using labels we created by codes like:
Label(starting2, image=gif_photo, bg='mint cream').grid(row=0, column=0, sticky=E)

The user will need to click the button "Save entries", which will store user's inputs globally(by a helper function named `click1` that assign user's inputs to global variables `NUM_CLUSTERS` and `PATH`).

Then by clicking the button "Create csv", these 2 global variables `NUM_CLUSTERS` and `PATH` will be used to generate and the output will be saved into the provided csv file path, by a helper function called `find_clus`.

Once the user finishes these 2 steps, the user will need to click the button "Skip this page" to move onto the next window(Window2) that will automatically pop up.

Window 2:

Similarly as Window1, there will be 2 boxes for user to enter inputs into and some labels for the purpose of instructions.

The user must enter a favorite book id like book849.

The user can choose to enter customized path as a string in the format like 'clusters_20.csv, clusters_50.csv'; user can leave it blank if user wants to use our default data provided.

The user will need to click the button "Submit", which will store user's inputs globally(by a helper function named `click` that assign these inputs to global variables `BOOK_NAME` and `CUSTOMIZED_PATH`, then destroy this window).

RecommendSystem (the main book recommendation system):

Class `RecommendSystem`, a child class of tk.Tk(), will initialize and show two pages by the function`show_frame` with the help of the 2 classes `RecommendPage` and `DoubleConfirm`, child classes of tk.Frame (), that are defined later. That is to say, once Window 2 is destroyed, we will have our RecommendPage shown.

RecommendPage:

This class `RecommendPage` is a child classes of tk.Frame ().

This page, displayed with a label "recommended books" at the top, is the main page for our book recommendation system.
It will call `find_all_recommended_books` from the imported module `helper.py`, which will perform operations for book recommendation, and then display the book ids of the 5 recommended books in a black box under the label "content".

We have provided 5 buttons labeled 1,2,3,4,5 that each one of these buttons corresponds to one of these 5 book ids from left to right(i.e., the leftmost book id corresponds to the button labeled 1.)

User can click any button corresponding to one unlike book's id, which will call the helper function named `replace1` that will clear the displayed text box and replace the original 5 books with 5 recommended books, the book which deleted by user will be replaced by a new recommend book.

Also, user can click the button "clear result" with the function `delete` to clear all book ids displayed in the black box under the label "content".

Once user is satisfied with all the recommended books, user can click the button "Yes" to move onto the next page(DoubleConfirm) with the function `show_frame`.

DoubleConfirm:

This is a confirm page which ask user if user does not need any new recommendation and can exit now.

User can click the button "I prefer more recommend books" to go back to the previous page(RecommendPage) with the function `show_frame`.

4. Instructions

Please install all the packages in requirements.txt, download all the data sets and put them under the same folder as the python file. The data sets have been shared with the course email address using UTSend.

When main.py is run, it takes a few seconds for the program to load the data and graph used for the recommender system. When preparations are done, there will be a window pop up on the screen.

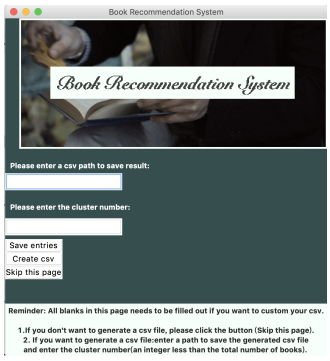

Figure 5: Generating csv - 1

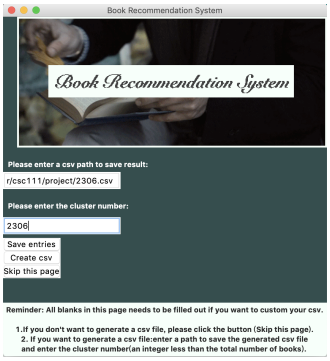

Figure 6: Generating csv - 2

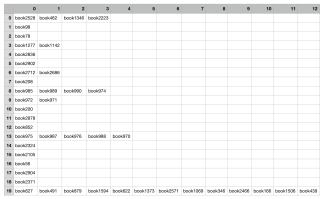

Figure 7: Generating csv - 3

Figure 8: Generating csv - 4

This is an optional function used for generating csv files to store the data of a certain number of clusters. Data of clusters are necessary for the recommender system. If the user do not have the csv file required or wants to adjust the range of the book suggestion list, they can follow the instruction to customize the cluster data sets by themselves. Users should enter the complete file address for the generated csv file to be stored, and the number of clusters that the user wants, then click the "Save entries" button, and repeat this step for the next cluster data set that they want to generate. After entering the information for all files they want, click the "Generate csv" button, and wait for the program to create the files on the given place. Users should check if they have got all the required files before going to the next page.

After clicking "Skip this page", the information of the target book and the cluster files used are collected. Users can input the book id and all the cluster files they have to make the recommendation more accurate. If no file names have been input, the program will use the default csv files that we have pre-processed for the recommender system. Users can click the "submit" button to go into the recommender page.

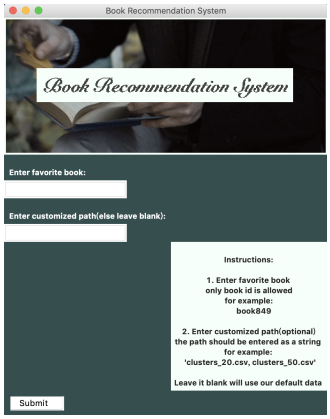

Figure 9: Recommend Information - 1

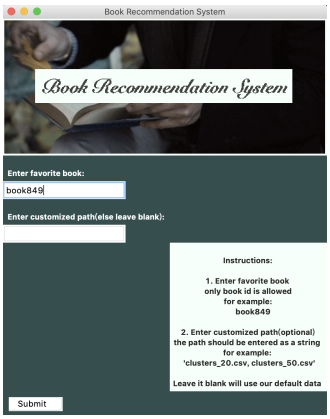

Figure 10: Recommend Information - 2

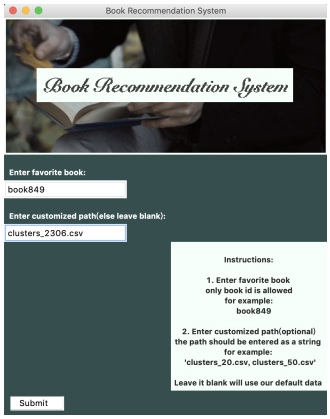

Figure 11: Recommend Information - 3

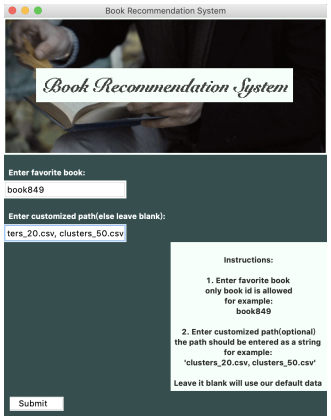

Figure 12: Recommend Information - 4

The recommender page contains three parts: a bar showing the suggestion list of books that has been recommended by the system, several buttons to take in the user feedback, and the "Yes" button to end the recommend process. If users have read or don't like some of the books in the suggestion list, they have the option to pass their feedback to the recommender system by clicking on the corresponding button. When the system receives user feedback, it immediately adjusts the result and refreshes the suggestion list without that disliked book.

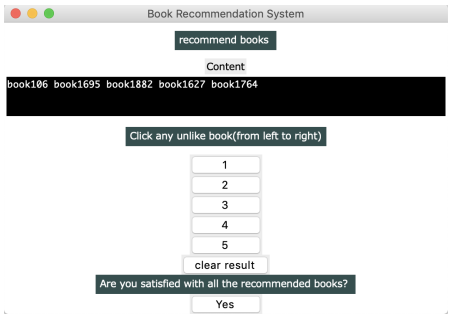

Figure 13: Recommend Page - 1

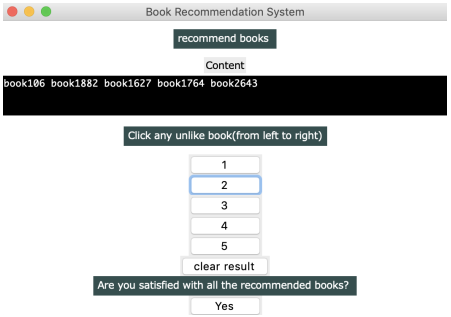

Figure 14: Recommend Page - 2

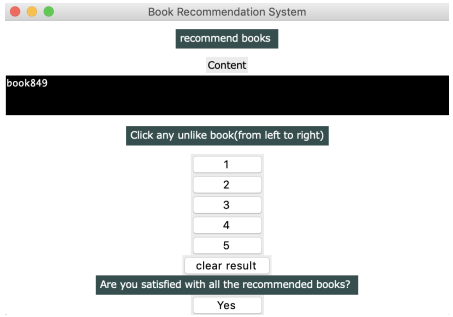

Figure 15: Recommend Page - 3

When "clear result" is clicked, all the books in the suggest list will be removed from the content bar. When there are no longer books to recommend, the content bar will print "All books had been recommended" instead. If users are satisfied with the result, clicking on the "Yes" button will bring them to the end page. They can go back to the recommend page again, or choose to leave.

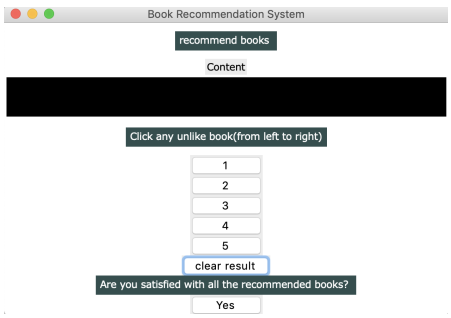

Figure 16: Recommend Page - 4

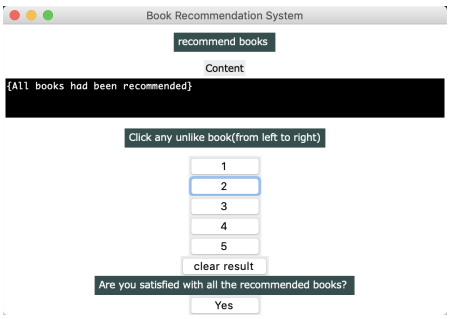

Figure 17: Recommend Page - 5

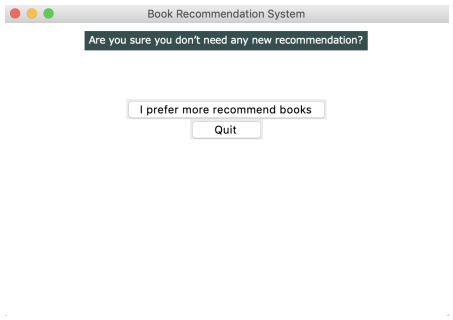

Figure 18: End Page

We provide different options for users to indicates the function they need. For all functions we provided, there are instructions leading users to enter things we need. However, some data may result in a poor looking result. Users may need to avoid some inputs when executing the functions. For example:

1. It's better to make the number of clusters larger than 20 and smaller than 700. Otherwise the books will either be too crowded or too rare.

2. It's better to enter more than 5 csv files in the recommend information page with the number of clusters distributed reasonably between the range. The data may be insufficient and so the suggestion list may not be satisfying if the recommendation is only based on one data set.

3. There are 2306 books involved in total. Thus for the testing purpose, it's better to generate a csv file with the number of clusters being in 2300 2306. Otherwise generating the csv may take very long.

4. It's better to use the default cluster data files.

5. Changes to the Project

Originally our recommendation mechanism is based on the Graph we created, where a user can input their user id, or the id of the book they like, and the result will be returned as a list of book ids recommended. If the input is a user id, the resulting list will contain five books that have the highest value of recommendation parameter. If the input is a book id, a list of books which are the neighbours of the input and are sorted by the recommendation parameter will be returned.

However, considering the potential ethical concerns that could be caused if users has to enter their ids, we have changed to only using a book id user likes for our book recommendation system now.

Moreover, we have changed our main algorithms for recommending. Originally, we plan to build the graph and recommend books to user based only on the rating. Now we have developed a more advanced algorithm which will compute the similarity scores between books and then form clusters where each cluster consists of books with closed similarity score. Later, we will also use the rating of books, which makes the resulted recommended books both similar and popular to the book the user entered.

In addition, We have created an interactive system, that acts as a media between users and our book recommendation system and provides a much easier way for users to try our system with the help of many instructions provided.

Also, we have added an option for users to create their own clusters stored in the csv file, that contains information of clusters of books that can be used to find recommended books later, in the file path the users entered. This option gives users chance to experience the process of creating their own clusters file with cluster size of their entered integer.

6. Discussion Section

After discussing within our group, we all consider our system helpful for solving information overload and helping users find books that they want to get in the shortest possible time.

However, there are also several limitations for our current recommendation system. First, only book ids will be used in the system, which seems kind of unreasonable for users. Initially, we intended to combine three data sets into a single data set so that we can gain direct access easily to the book's name rather than their ids. However, the data set which contains information of book titles is too large (above 9 GB), which makes the speed of data wrangling very limited. Due to the same reason, we only use a small proportion of data from the huge original data set, which may lead to bias for our recommendation system.

Furthermore, we could try to explore more advanced algorithms for this book recommendation system. For instance, we could attempt to build our recommendation system using user-based approach, like taking their neighbors preferences into account when designing the algorithms.

Besides, we could make our results displaying more impressively by designing our interactive system in a more appealing way like having nicer formatting and more advanced functions available for users to try.

Last but not least, evaluation of recommendation systems with statistical approaches(i.e., testing the quality or accuracy of our model) have not been covered and this area may worth exploring.

7. Reference

Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, *Fine-Grained Spoiler Detection from Large-Scale Review Corpora*, in ACL'19.

Visualization with python¶. (n.d.). Retrieved March 16, 2021, from https://matplotlib.org/

Visualize graphs in Python. (2020, July 03). Retrieved March 16, 2021, from https://www.geeksforgeeks.org/visualize-graphs-in-python/