Quan, Yuan
Dec 5, 2022

Final Project Report

This report contains the major scope of the final project, and the design process, input and output, challenging parts, potential improvements, and a reflection of this project.

Major Scope

This project is a video recommending program that based on users' watching history and users' profile information.

Before implement this program, I performed some background research to turn this idea into algorithm logics. However, the algorithms online are either too simple, like if user 1 watched video a b and c while user 2 watched a and b, then will recommend c to user 2; or too sophisticated, like implement some data analyzing to build the user's portfolio. Therefore I decide to develop my own algorithm to realizing the recommendation program.

Architecture and Design Process

First of all, the most important part it to come up with a set of criteria to decide with video to recommend. And the following is my original design.

> First, if the user has a watching history, then the system will recommend based on his watching history to recommend similar videos in the library.
>
> Second, if the user does not have a watching history, but have a relatively complete profile, then the system will recommend based on his profile, searching for similar users, and recommend the similar users' watching history to him.
>
> Lastly, if the user does not have enough info, the system will recommend by video's watching volume.

So to implement this, I developed 3 helper functions called rank_by_video_similarity(), rank_by_user_similarity(), and rank_by_play_volume() respectively. They will take in a video or a user to generate a max-heap, for further actions.

After implementation, the basic logic is still the same, but changed some details, which will be covered in the later part, Challenges.

Then to decide whether 2 videos are similar or 2 users are similar, I designed the following similarity score calculation process.

> For videos: if the videos belongs to the same category, it will be 5 marks; if the videos are created by the same user, it will be 3 marks; if the videos are in same language, it will be 1 mark; if the videos are with similar length, it will be 1 mark. The marks will be the sum of all previous conditions.
>
> For users: if the users are in same sex, it will be 2 marks; if the users are in same origin, it will be 3 marks. And most importantly, the difference of age will largely generate to the score. If the age difference is more than 15 years, it will be 0; 1 point for 12-15 years difference; 2 points for 8-12 years difference; 3 points for 5-8 years difference; 4 points for 2-5 years difference; if age difference is less than 2, it will be 5 points.

By this stage, the design of 2 classes are naturally complete. For user class, there are attributes of user ID, sex, origin, born year, and of course watching history and upload history. For video class, there are attributes of video ID, video name, uploader, category, language, length, and playing volume.

Both of the class have a function to calculate the similarity score. User class has 2 more function of watch and upload to update the relevant history record. And video class has 1 more function of played to update the video's playing volume.

At this point, the overall structure is half way complete. At next stage, we will go through the design process of inputs and outputs.

Inputs, Outputs, and the Processing Design

Apparently, I need to create a bunch of users and videos in the library for the program to compare against. But if I write the code to create them one by one will be difficult to maintain, review, and amend if I want to add more data, or if I want to change some of the data or structures.

Therefore, after discuss with professor, I decide to put all the data in separate .csv files, and read the file to import the data into the program. It would be easy for the client to see the input data format, and easy to maintain and amend.

Hence, I developed 3 files to store different type of data. One contains all users' information; another contains all upload records, which is also all videos information; the other contains all watching records.

After read in all information, I found it useful to store it in a dictionary to map the user ID to user, and a dictionary to map the video ID to video. This will generate a constant time if we want to get some user or some video.

To implement this, I introduced 3 helper functions, called user_factory(), video_factory(), and build_watching_history(). They will take the 3 filename respectively, and create a dictionary called USERMAP, a dictionary called VIDEOMAP, and amend users' watching history.

The main program will take a input from the client, as a user ID that the client wants to make recommendations to. Then after processing, the program will print the recommendation message. To make it more clear, so that the message can reflect more information of the situation, I want it to print the user's watching history first, then print the name of recommended videos. In this case, we could see the watching history of the user, so that we can determine whether this program is effective.

To generate the return message, I developed a separate helper function called outcome(), which will take a user and a list of recommended videos. Then loop through to generate the message.

Challenges

There are also some challenges in implementing the algorithm. One of them, as we mentioned before, is the 3 stages of recommendation.

In previous part, we introduced 3 stages, as first is to recommend based on watching history, second stage is to recommend based on user profile, and third stage is to recommend based on video's playing volume.

However, when implement this logic, some error occurred, saying that the list of recommended videos does not reach the desired amount, especially when we adjust the amount upwards. This happens because there are not enough similar videos as those in watching history, or there are not enough similar users to provide their watching history, or the similar users does not have any watching history.

To solve this problem, I changed the 3 parallel stages to 3 consecutive stages. Which is, first we recommend based on watching history. If the results are not enough, we then recommend based on similar users. If the results are still not enough, we finally recommend by the playing volume of remaining videos.

This change needs further helper functions to realize. The basic flow of function recommend_on_videolist() is still the same. It takes in a list of watching history, and rank the whole video library based on those videos similarities to the most recent watched video. And append from top similar videos to low similar videos. Once it reached the desired amount, we just return the list. But if it doesn't reach the desired amount, we continue to check the second most recent watched video for further recommendation. If we looped through and the amount is still not enough, we will just return the list.

However, there are significant changes in the other 2 stages. First, these 2 stages could be merged as 1. Because you can never decide whether the user's profile information is enough or not. So you rank all the users based on their similarities to the current user, if there is not enough users' watching history to append, we will turn to rank the remaining videos by their playing volume. Hence, I developed the function called recommned_on_user(). The second change is that the function will take in another parameter, a list of videos, which is the current result we have. Just in case the result generated by watching history is not enough, we could pass the result in this function to further generate.

The second challenge is that when implementing the max heap, the build-in max heap can only compare the objects according to their own attributes. So it is easy for me to create the max heap compared by videos' playing volume. But, the max heap for similar videos and similar users are

difficult, because they are compared to a third video or user to generate the similarity score as comparator. It is impossible to directly add them into the heap.

To solve this, I found out a hidden function of max heap, which is that the heap can insert a tuple, and it will compare the tuple using the first element in the tuple. Hence, I combined the similarity score and the video or user, to make it a tuple. Then add it into the heap. In this case, the heap will work out by using similarity score as the comparator.

A final challenge is that, when I writing more and more helper functions, it is messy and redundant to add USERMAP and VIEOMAP as parameters to every helper functions. The drawback is that when the size of USERMAP and VIEOMAP getting bigger and bigger, it is not efficient to pass in these 2 dictionary. Therefore, I made them as global variables, so that every function could have access to them.

Pros, Cons and Potential Improvements

The upsides of this program are:

1. The desired amount of videos could be changed easily.
2. The classes, helper functions, and main program are separated so that it is easy to maintain.

The downsides of this program are:

1. Every video must be assigned to 1 and only 1 category. It may cause the recommending to be less accurate.
2. The similarity score calculation is too simple. There are a lot more information in real life examples.

Based on this, we could make further improvements as followings.

1. Introduce tags to attach to videos, instead of category. A video could have more than 1 tag.
2. Similarity score calculation could be more sophisticate when tags are introduced into the program.
3. Users could be grouped if they watched same videos, and further data analysis can be adopt to find the common characters of this group of users. If another user with the same character comes, the program will recommend these videos to him.
4. In real life, machine learning is widely adapted into the recommendation system. The computers are trained to analysis the content of the videos, and generate proper tags to them. It is extremely useful when dealing with thousands of millions of videos.

References and Acknowledgments:

1. Project ideas: Discussion with Professor Rasika Bhalerao
2. Background information: Online searching

https://www.geeksforgeeks.org/python-implementation-of-movie-recommender-system/
https://blog.jovian.ai/creating-a-movie-recommendation-system-using-python-5ba88a7eb6df
https://techvidvan.com/tutorials/movie-recommendation-system-python-machine-learning/