CSC111 Final Project: My Anime Recommendations

Raazia Hashim, Tahseen Rana, Tu Anh Pham, Meet Patel

Friday, April 16, 2021

1 Introduction

If you are an avid anime watcher, our best bet is you have at least gotten extremely bored but found no good show to watch because you felt like you have seen them all. Well, to be honest, you probably haven't. But how can you find new, interesting anime to watch? Someone may tell you to ask an anime watching friend, but that can be a big mistake. Chances are that your friend has seriously bad taste and you will have a regrettable time watching their favorite show while you could have been productive and finish your CSC111 assignment instead. As such, we had more than once wondered if there was an application that can quickly, systematically give accurate anime recommendations catered to your specific tastes, so you can save a lot of time in finding anime and spend that time watching something you most likely will enjoy. Effectively using time for entertainment is one of the keys to ultimate academic success. That is killing two birds with one stone!

For someone who is unfamiliar, anime are animated shows that originated from Japan which have been increasingly gaining popularity everywhere over recent years. Anime can be very diverse in terms of content. One anime can be classified into many genres, such as action, adventure, comedy, fantasy, magic, supernatural, horror, mystery, romance and science fiction, not to mention over 30 other sub-genres, each of which is characteristically distinctive from the other. This diversity is one of the reasons why anime have become so well-known, but this fact can also be a challenge for any anime streaming service to provide useful suggestions, since different people may have deeply varied preferences.

With that in mind, we have come to the conclusion that we should just make recommendations for ourselves. By "make recommendations", of course we mean to do it through programming! In particular, our goal is to provide a personalized list of anime suggestions based on the collected information from each user, so the user can quickly enjoy watching the anime they find interesting. To make our system even more convenient, we will create a complete user interface to help anime watchers navigate through thousands of anime's more easily. Anime names are sometimes difficult to remember (since they are Japanese after all). Thus, features that support searching like auto-completion are also definitely nice to have. Furthermore, it is also necessary for the system to be able to give suggestions of specific category, in response to user's demand. Therefore, our final goal is **to build an user interface for an anime streaming platform that has a complete set of features including searching with auto-completion, filtering based on genre, and recommendation based on predictions of user preference, using the tree and graph data structure**.

2 Dataset Description

The data sets we will be using to implement our anime recommendation system contain information on 16000 animes, 130000 reviews and 47000 user profiles crawled from a popular anime streaming service. The datasets have been cited below in the references and a link is provided below (Marlesson, 2020).

- https://www.kaggle.com/marlesson/myanimelist-dataset-animes-profiles-reviews?select=reviews.csv
- The first dataset `animes.csv`, contains information on 16000 animes. The first entry corresponds to the id, then the title, synopsis, the genres, the air data, the number of episodes, the anime's popularity, rank and score. The last two entries are a link to the cover image and a link to the anime on the myanimelist.net website. The recommendation system utilizes the anime ids, popularity rank and score, and genre. The visualization utilizes the anime name and synopsis.

- The second dataset `profiles.csv` contains information about 47000 users who watch anime. The entries in the file correspond to the following information: the user id, username of the user, gender, date of birth, a list of anime ids that correspond to the user's favorite animes, and a link to their profile on the myanimelist.net website. The recommendation system utilizes the username and list of favorite animes.

- The third dataset `reviews.csv` contains information on 130000 user reviews for animes, including text reviews and scores that users gave animes that they watched. The entries correspond to the id of the user leaving the review, the username, the id of the anime, the text review, the overall score, the score for each category (such as story, animation, sound, character and enjoyment) and a link to the review on the myanimelist.net website. The recommendation system utilizes the overall score given by the user and the username.

3 Computational Overview

1. Data Processing:
The data we obtained from kaggle is not clean; the file `profiles.csv` contains many repeated rows. We removed such rows as well as the columns that were unnecessary for our project.

The process of loading the data into our program is straightforward. We create a module for this purpose (`data_loader.py`). The program would read each row that represents a user or an anime and initialize the corresponding vertex object. There are only a few things to note: the column corresponding to an user's list of favorite anime is a valuable part of our data. However, there is no exact method to determine how much score a user would give his favorite anime, so we decided to assume each user's favorite anime that has no associated review with it would take a review score of 9. Another thing is the lists of genres for each anime. They were used to create edges between each Genre object and Anime object, instead of being kept as an instance attribute of each anime.

We also created a data file for testing by splitting the data in `profiles.csv`. We chose 300 users such that each has at least 9 favorite anime. For each of them, 2/3 of their favorite list is removed from the original file to store in a different file. This file would then be used to test if our recommendation engine can provide suggestions that actually match these favorite lists.

2. Feature Implementation:
Our data about anime and users is structured in a graph, which is an instance of the class AnimeGraph. There are three kinds of vertices: Anime, User, and Genre. The use of Genre as a vertex type has many advantages. A genre can be linked to both Anime and User vertices. Although it may take up some memory, it helps the content-filtering part of our recommendation algorithm and the feature 'filter anime by genre' run a lot faster, since to get all anime of a given genre, we just have to get all the vertices in that genre's set of neighbor anime.

To make recommendations for each user, we would first consider how much data of that user we are currently having:

- For a newly registered user (this case is known as a cold start), we decide to suggest them the most popular animes and some newly released animes. This is done by simply sorting anime by popularity or by airing date.

- When a user only has one or two anime as their favorite, we can start recommending them anime by content-filtering. For each anime that a user has reviewed, we either create new edges between the given user and the genres of that anime, or change the weight with accordance to the review score. Because review scores range from 1 to 10, we would calculate how much the score deviate from 5.5. If a score given to an anime is less than 5.5, the weight between the reviewer and the corresponding genres of that anime would be decreased by that deviation. The opposite happens when the review score is greater than 5.5.

- When a user has more than two reviewed anime, the system switches to recommendation by finding similar users. This is also known as the collaborative filtering approach. We tried to implement this in a few different ways, mainly involving different measures of "proximity" between users (Polamuri, 2015), in term of reviews given. This includes the Euclidean distance, the Manhattan distance, the Minkowski distance, a version of the Jaccard similarity (which I'm pretty sure was used in assignment 3), and a self-invented similarity measure, just for fun. We will talk about the accuracy of them in the discussion section. But after some testing, we decided to choose the Jaccard similarity approach and our own self-invented measure to customize for better program running time. The `recommend_by_users` function of the class `RecommendationEngine` would take a measure as an optional input (in the form of callabes or str indicating our custom algorithms).

Consider our own algorithm (we call it the 'custom' measure in our program), which was implemented as the function `most_similar_users` in the `User` class. For a target user, we would loop through all the neighbour anime. For each anime, we find all of its neighbour users, except our target user. For each user, we calculate how different their review score is to our target user. This is done by calculating (5.5 - abs(review score difference)). When the difference is greater than 5.5, this quantity is negative, so it indicates a degree of difference. When the difference is smaller than 5.5, it is positive, so it indicates a degree of similarity. Such a quantity can be stored as values in a dictionary with keys being users. They will be accumulated if we encounter the same users again when looping through another anime. After the looping process is done, we will have a dictionary of users and accumulated similarity scores. The most similar users would then be picked to generate recommendations.

On the other hand, we also want to take the version of the Jaccard similarity measure introduced in Assignment 3 and try to implement it taking advantages of the structure of the graph for better running time than just looping through all the users to find the most similar ones. So, we implemented it as the function `closest_jaccard_distance_users` in the `User` class. We took a very similar approach as described in the paragraph above to improve program running time. That is, looping through neighbour animes of the target user and then each anime's neighbor users to accumulate the counts. There are two types of counts in this case: the count of anime that are adjacent to both users in general and the number of anime adjacent to both that also has the same review score for both. Such quantities are accumulated by a dictionary with keys being the users and values being lists of two elements (the two counts). For each anime, if we encounter a neighbor user that is already in the dictionary, the corresponding counts may be increased. The general count of anime adjacent to both would then be used to calculate the number of animes that are adjacent to target user *or* the other user by the formula

$$|A \cup B| = |A| + |B| - |A \cap B|$$

and then the formula (as in given the Assignment 3 handout)

$$sim(v_1, v_2) = \begin{cases} 0, & \text{if } d(v_1) = 0 \text{ or } d(v_2) = 0 \\ \frac{\left|\{u \ \mid \ u \in V \text{ and } u \text{ is adjacent to } v_1 \text{ AND } v_2, \text{ and } w(u, v_1) = w(u, v_2)\}\right|}{\left|\{u \ \mid \ u \in V \text{ and } u \text{ is adjacent to } v_1 \text{ OR } v_2\}\right|}, & \text{otherwise} \end{cases}$$

would then be applied.

After getting a list of users sorted by similarity score, we would choose the highest rated animes of the most similar users. Those animes that the target user already reviewed are excluded. Also, any anime with review score greater than or equal to 9 is considered to be highly rated.

The next feature implemented in our application is the search engine, which allows users to search for an anime. As the user types, the system shows all animes that begin with the letter or prefix that has been typed so far. We used a data structure called a called *Trie* (sometimes referred to as a *Suffix Tree*) to implement the auto-completion feature. We created a class to represent a `TrieNode`, which is a letter in the Trie. The `TrieNode` has attributes `letter`, which is the letter that that node refers to, `is_word`, which is a boolean value that indicates the end of a word and `children`, which is a dictionary of the 'subtrees' of the node, mapping from the letter to its corresponding `TrieNode`. Next, the `Trie` class has one attribute, `root`, which is the first node in the `Trie` and it's letter value is an empty string. The initializer for the `Trie` creates a root node, and calls `Trie.insert_word` on each word which inserts the word a letter at a time into the Trie. The main method that contributes to the search is `Trie.all_suffixes`, which returns a list of all possible suffixes of the prefix input in the Trie. It first calls `Trie.find_node`, which in turn calls a recursive helper `TrieNode.find_node_from_str_index`. This method returns the correct prefix (accounting for the search being case sensitive) and the node that matches the end of the prefix. `Trie.all_suffixes` then calls a helper method that iteratively finds all of the different suffixes that follow the prefix in the Trie. These methods are used by the main Application to allow for a user to interact with searching in the Trie.

3. Visualization

tkinter is a well-known python library for creating a graphical user interface with python. It is efficient to use and provides GUI components such as text displays, checkboxes, entries, and image rendering, which are all appropriate components for our project. There is a login page where the user has the choice between logging in to their existing account or registering and creating a new account. By registering, a user can add new reviews for animes that they have watched and may or may not like, so that our recommendation system can generate useful personalized recommendations for them. After logging in, the user interface will consist of a search bar, where people can type in an anime name and have results based on auto-completion popping up as they type. There may also be a box to receive input on anime categories the user specifically searches for, hence searching results can be filtered by genre. There is also an option menu where the user can select how they want to view the application. There are two designs that can be selected by user preference. There is the simplified and the completed version. The simplified version makes the application more simpler as it only shows the name of the anime. Whereas the completed design has pictures that go along with the name of the anime. This makes the application look more appealing! The images displayed are the respective anime covers to make the application more user friendly. This is done by obtaining the links of each anime (this is in the data set) and displaying the image found in each link. Also, in the user interface, there are evenly sized boxes in each row, each box displays the respective anime. There are 3 sections that users can observe, "Recommended for You", "Newly Released Anime", and "Most Popular on MyAnimeList". In each category, there are animes that users can click and view the details of the anime. For example, users can look at the score, synopsis, genres, popularity etc. On that page, users can also add their own reviews which gives further information to the recommendation engine to provide more accurate recommendations based on the scores that users gives.

4 Instructions for Obtaining Datasets

Our dataset was uploaded on
https://github.com/Mondlicht1/CSC111-Project-dataset/blob/main/data-and-theme.zip

It is a single zip file containing the Data folder, the theme for tkinter (gotta make it look nice), and the background image. It should be extracted in the project folder.

5 Instructions for Running the Project

To run our application, you just have to run `main.py`.

Inside the branch `if __name__ == '__main__'` of `main.py`, there is also a block of code that visualize the graph and two bar charts showing different approaches to recommendation in comparisons, measured in running time and accuracy. This code block is originally commented out because we don't need it to run the application. You can un-comment it to see the graph (The network is quite dense with a lot of edges)

6 Changes to Project Plan

In the feedback to our proposal, the TA suggested different approaches to the cold start that we think are really helpful. So, instead of just diving right in the collaborative-filtering approach to make recommendations for a new user with 1 or 2 reviewed anime, we suggest animes based on content-filtering first.

We also added a "newly released anime" to the content to avoid the popularity bias. We've become aware of this problem, where a recommendation system gets skewed in one direction because when only popular anime get the majority of interactions, those interactions will in turn reinforce the popularity of such anime. Suggesting new anime helps in minimizing this error.

Another change was to the implementation of the Trie. At first, we wanted to return the list of auto-corrected anime names in popularity order. But after some testing, we think that just sorting them in alphabetical order makes it much easier for the user to navigate.

7 Discussion

It's not wrong to say that recommendation systems are the heart of the internet economy. We are well aware that our implementation is just a very first step on such a journey. As any learner, we needed a way to address how we perform, so to put some challenge to our algorithm, we extracted 2/3 of the favorite anime list of 300 different users and for each user, we try to provide a list of recommendations and then measure the accuracy of the recommendations, that is, how many correct guesses can the system make, and the running time required to make recommendations for 300 users. Different measures of user similarity were put to the test, including along with our genre-based content filtering implementation. They were also compared to the approach of always recommending popular anime. Here are some results (you can run the project for plotly charts with better resolutions):

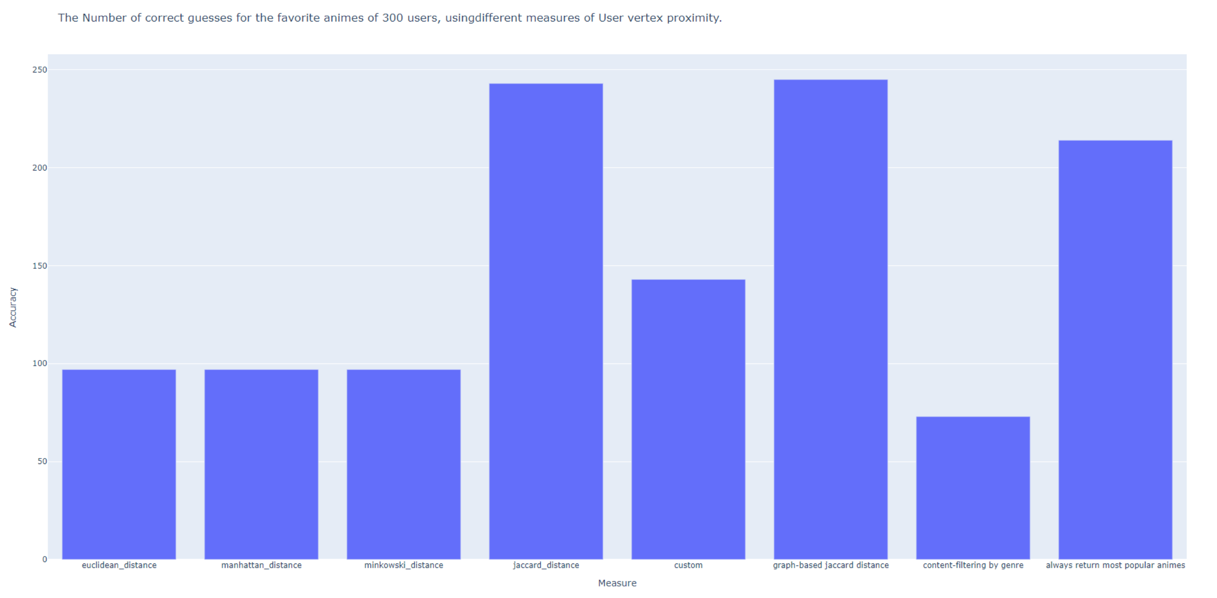

Figure 1: The number of correct guesses of different recommendation methods

From left to right, we can see that proximity measures such as Euclidean distance, the Manhattan distance, the Minkowski distance perform equally worse. The Jaccard distance performs best (There are two versions of them, one that loop through all users vertices and the other take advantage of the graph structure, which was described in the computational plan). The "returning most pupular" method is second to best (understandable), and our self-invented method, although not performing very well, is still better than the res considerably. Furthermore, running time-wise, it is doing not a bad job, as shown in the chart below:

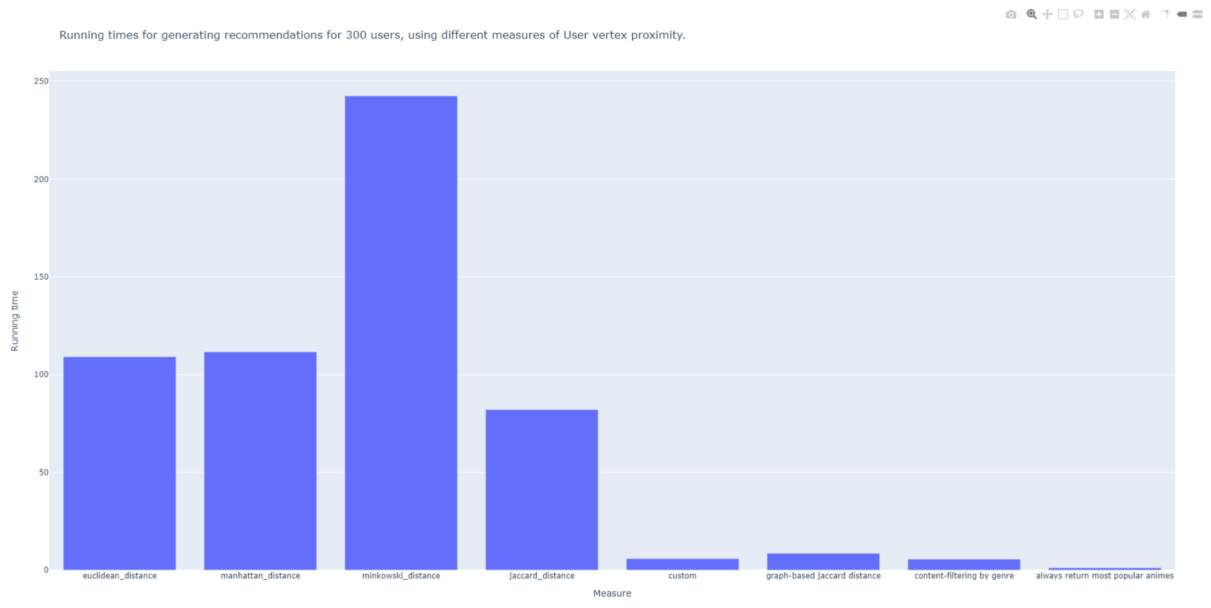

Figure 2: The number of correct guesses of different recommendation methods

In the chart, we can see that all methods of finding similar users that loop through all users are taking a lot of time: To generate recommendations for 300 users, Euclidean distance took near 2 minutes, Manhattan distance took 2 minutes, the Minkowski distance took 6 minutes. Even the Jaccard similarity measure took 80 seconds if we implement it the brute-force way. This is why graphs are cool: our own implementation for the Jaccard similarity measure only took 8.5 seconds and our "custom" method only took about 6 seconds. Hey! that is 0.02 seconds to make a list recommendations for an user, not bad!

With that said, we are now only sitting at a maximum of 245 correct guesses for the favorite animes of 300 users. That is a modest number. Although our way of measuring is quite strict, we believe there is still a lot of room for improvements, and there are many other factors to consider, like user's age, user's gender, which for sure will improve our recommendations by a fair deal. We also tried to make actual score predictions of all animes based on user similarity, and use those prediction to recommend new animes to the user. This has its own problem, however. Many of the anime are having too few reviews, with all reviews being perfect scores, and so if we try to calculate the weighted average of scores, we would get a 10, so such an anime would get recommended to everyone. This makes the the accuracy fall off completely. We are thinking of a few ways to deal with it, like remove those anime with few reviews from score prediction. But then, it's also possible that those animes received all 10's because they were actually great, just not many people know about them. This, again, drives the system in the direction of the popularity bias. All in all, we are very interested in finding new ways to determine the similarity between people using a graph approach. It has so many potentials.

Moving on, the auto complete search feature is a convenient way for users to be able to search for their favorite anime. As the user types, the results mimic an auto complete feature on any other search engine, and is a feature to help the user search for their favorite anime, since anime names are sometimes complicated and often not in English. Some next steps when it comes to this feature is that we can display the results of the auto complete search to the user in order of anime popularity or review score, instead of alphabetical order as we have now, which may enhance the user experience even more, since what they are searching for has a higher probability of appearing closer to the top of the screen.

One last note is on tkinter and the GUI. Tkinter is without a doubt one of the most convenience library to make GUI on python. However, it has its own disadvantages. An apparent one is the lack of supports for the scrollbar feature, which had to be implemented in a roundabout way. This increased the complexity of our code for the GUI considerably. Also, image rendering with Tkinter is quite slow and they appears jagged if combined with fast mouse scrolling. These limits suggests that in the future, if we want to take our highly interactive and graphical GUI far, we'll have to choose an alternative. This, however may requires Java, HTML or CSS, which we can't wait to learn about in future CS courses!

References

[1] Marlesson. (2020, January 5). *Anime Dataset with Reviews - MyAnimeList* [Dataset]. https://www.kaggle.com/marlesson/myanimelist-dataset-animes-profiles-reviews?select=profiles.csv

[2] Interview Cake. *Trie Data Structre.*
https://www.interviewcake.com/concept/java/trie

[3] Polamuri, Saimadhu. (2015, April 11) *Five most popular similarity measures implementation in python.*
https://dataaspirant.com/five-most-popular-similarity-measures-implementation-in-python/

[4] Liu, Changran. (September 3, 2020) *How to build a recommendation system in a graph database using a latent factor model.* Towards Data Science.
https://towardsdatascience.com/how-to-build-a-recommendation-system-in-a-graph-database-using-a-latent-factor-model-fa2d142f874

[5] *tkinter* — Python interface to Tcl/Tk — Python 3.9.2 documentation. (2021, March 13). Python. https://docs.python.org/3/library/tkinter.html