CSC111 Project Report: Anime Atlas: Anime Recommender

Ian Lavine, Jeremy Hiu Shun Wong, Prithee Roy Ballave, Ayush Sahi

Friday, April 16, 2021

Problem Description and Research Question

Picture this, it's a Friday night and you've finished all your school work. You completed all your tasks for the day and now you finally have some time to yourself. You decide that you'd like to watch an anime, but alas, with the constant stream of renewals and new releases of anime, it has become increasingly difficult to find a new anime to watch due to the sheer amount of old and new content. On platforms such as Netflix or Crunchyroll where there are over hundreds of anime to choose from, what is the right choice for you? Deciding which anime to watch can take longer than watching the first episode itself as the endless options coupled with the difficulty to differentiate between multiple shows due bad descriptions or biased reviews. Wouldn't life be so much easier if we could quickly pick an anime that we are likely to enjoy? As anime fans ourselves, we wanted to make our lives easier by developing a project which uses an insightful algorithm that **recommends users new anime based on their individual tastes** (Note that we will explain how we rank anime based on the users 'taste' in the sections below). Platforms like Spotify and Apple Music have a feature where they recommend new songs based on a users' listening history. Our project would essentially be doing something similar, but for anime instead of music! Rather than scouring through hundreds of vague descriptions of anime, we would like to take it upon ourselves to create a project in which you simply input your MyAnimeList account, and with a few clicks of your mouse, you'll be able to see anime recommended to you based on your taste of anime and the taste of people who have watched the same animes as you. ...Furthermore, you'll be able to narrow your search field based on several factors such as the number of episodes, whether the anime is still currently airing, the duration of each episode, and so forth.

Dataset Overview

All data used in this project was from two datasets we built. We used an API from MyAnimeList to pull data for our project and build our two datasets. MyAnimeList (MAL) is a website in which user's can catalogue the anime they have seen and rank them out of ten, as well as assign descriptive tags to them. The first dataset, 'showdata.py' is a dictionary with 750 id numbers as keys corresponding to an anime, where the value is a list containing the name of the anime, the total episodes, a photo of the anime, and the genres it's most associated with (the genres are pulled from the MAL API). The second dataset, 'userdata.py' is a dictionary with approximately 750 MyAnimeList users, in which their user id is the key and the associated value is a list of tuples each containing the id of an anime that the user reviewed, and the score out of 10 the user gave it. As these datasets were made by us, everything in them was used. The files we have our data in are called user_data.json5, show_data, id_data.json, code.txt, mal.json, five_hundo.json.

Computational Overview

Data Collection

Before any computations could be performed, our group required access to information about various animes and users. Since we could not find an adequate pre-existing dataset we decided to develop and create our own. In order to build our own custom dataset with this information, we used a combination of web scraping and API calls to the MyAnimeList (MAL) website. The way the structure of our program is set up is that as soon as main.py is run, a personal server is set up for our user using their localhost (specifically on the address: 127.0.0.1:5000). This allows anime atlas to perform it's API call to the MAL servers, which operate on OAuth2 security protocols. Since OAuth2 requires various secret and public keys to be sent over to the server in exchange for access, the aforementioned flask server allows us to obtain one of these keys. Initially, our program generates an access token which can be obtained by a unique authorization link generated in token.py, which incorporates a randomly generated 128 character code (which cannot be reproduced) using the secrets module. This allows our user to sign into their MAL account safely, and redirects them to the address 127.0.0.1:5000/anime-atlas to generate a new public key for the flask server to obtain using the requests module. This code allows us to generate an access token, allowing our program to retrieve different data, such as our user's top anime and info associated with the anime they have seen or a data set of specific animes and their related information, organize said data in our desired format, and output it as a json file for other files to read. However, the MAL API is still in beta, meaning not all information is available to us. This resulted in the need for web scraping using beautifulsoup4. Through the functions in webscrapping.py, we are able to obtain information unavailable to us previously such as a list of random users on the MAL platform, as well as anime sorted by genre. All of these custom datasets were used in the various algorithms that calculate recommendations.

Computations on data

Since there are thousands of different animes and users on MyAnimeList, we had to narrow our scope in regards to the data used in our computations, to achieve a realistic run time on our functions without compromising accuracy. The way we did this was we developed a list of 850 animes, and 800 users. The user list was developed by randomly scraping the usernames of active users on MyAnimeList. The anime scope was developed by taking the 100 highest ranked animes on MyAnimeList and then taking 750 unique anime in 15 different genres. These genres are classified as "Mystery", "Psychological", "Supernatural", "Seinen", "Action", "Comedy", "School", "Sci-Fi", "Drama", "Mecha", "Adventure", "Fantasy", "Romance", "Sports", "Slice of Life", "Ecchi", and "Horror". These genres are the same as those MAL uses to classify animes with 2 minor exceptions in Sci-Fi and Slice of Life, which are a combination of a couple of genre categories. These genre categories are represented as lists in animes.py, and are populated through webscraping the featured anime on each genre's web page on the MyAnimeList website. These featured anime are chosen based on MAL's algorithm which incorporates a variety of different factors to rank anime such as average score, number of fans, etc. These genre titles are also tags provided to each anime within the 850 we have chosen. The way we chose the 850 shows was we ranked them in 4 tiers, S-tier, A-tier, B-tier, and C-tier. S-tier contains action, adventure, comedy, drama, A-tier contains sci-fi, slice-of-life, seinen, fantasy, B-tier contains mystery, psych, romance, horror, and C-tier contains sports, mecha, ecchi. The 850 scope, represented as the list 'five_hundo' in animes.py was populated with the top 100 anime from functions in token.py. Next, the top 50 featured animes in the niche categories of C-tier, were scrapped and appended to five_hundo, only if they were not already in it. The same was done for B, C, and A tier. The significance of this is that the most popular in less popular genres were taken so that anime atlas could identify unique animes to recommend in popular categories. Because there was a lot of overlap between genres with the same show being in multiple different categories, we were able to avoid duplicates by adding the aforementioned check, while adding the most popular shows in general to the list five_hundo. So by the time the algorithm was ready to scrape and append the shows in the most popular genres (those in S-tier), it had already appended most of the top 100 shows in these genres as a result of the aforemetioned overlap. This allowed us to find the more underground shows that would make good recommendations for users who have seen a lot of anime.

After the data collection, the project mostly centres around the use of two weighted graphs. The first weighted graph, review_graph, is built from the 'userdata.py' dataset. Each user is a vertex in the graph, with each of the anime they review as a vertex as well, and the score they gave the anime as the weight of the edge between them. The second weighted graph, anime_graph, is built from the first weighted graph. This time, only the anime are vertices, with all the user vertices removed. The edges between the anime are calculated by taking the broad similarity of every single anime with every other anime, and connecting the anime to those of which it has the highest broad similarity with (with a limit of 10 edges per vertex). Broad similarity is a variation of similarity_score_strict from CSC111 assignment 3, where the similarity of any two anime is essentially the average similarity in which users rated them. You will notice these two graphs are very similar to the review_graph and the strictly book_graph in CSC111 assignment 3.

All major computations our program executes are done by searching these two datasets. A user of the program inputs their MyAnimeList account, and the program webscrapes with that data to find all the shows that user has reviewed. The program then searches the graphs to find, based on that user's tastes, which anime to recommend. The user has 4 different algorithms to choose from to recommend the anime. Each algorithm is explained in more depth in the code, but they work essentially as follows:

- Note: For this section, CA (chosen Anime) will represent the anime a given user has reviewed on MyAnimeList and CAR (chosen Anime Rating), will represent the rating they gave it. PR (Possible Recommendations) will represent any of the other anime vertices in the Graph that could be recommended to the user by a given algorithm. PRC (Possible Recommendation Score) the score an anime will use when comparing it to all other PR, of which the highest PRC anime will be recommended.

- 'Weights' takes every anime that is a neighbour to one or more of the users user in anime_graph, and finds which have the highest sum of the weights of edges to them from the anime. Each one of these weights is individually, before being summed to total a PRC, is multiplied by (CAR - 5) of the anime it is a neighbour to. This is done so that neighbours of the CA with a CRA lower than 5, are in fact negatively impacted, and will lower a given PR's PRC. This process can be found in the function 'recommend_anime_weights'.
- 'Neighbours' works similarly to the recommendation algorithm in Assignment 3, but instead takes multiple CA as input (as opposed to just one book), and calculates which PR in anime_graph have the highest sum of similarity scores to CA. A variation of similarity_score from A3 is used, where instead the similarity score of two anime is the sum of all the weights of the edges to the anime in which they are both neighbours, divided by the length of the union of all anime either have as neighbours. Again the similarity_score of any CA and PR is first multiplied by the (CAR - 5), before adding it to the total PRC. This process can be found in the function 'recommend_anime_neighbours'.
- 'Search' works completely differently from all the other functions. It was coded in attempt to take advantage of traversing the entire graph, and account for users who have a large range of taste. For every PR in anime_graph, searching finds the fastest root from from it to every anime CA. It then finds the average weight of the edges in that path, divides that by how many edges in the path, and multiplies that by (CAR - 5), as already described in 'Weights'. The PRC of any PR is then the sum of all these path scores. This process can be found in the function 'recommend_anime_path'.
- 'User Comparison' is the only algorithm that works on review_graph instead of anime_graph. It adds the user to the graph, and finds which other users are the most similar to it. This time the users similarity_score is another variation on similarity_score from assignment 3, where instead the similarity score of two users is equal to (3 - the difference of the weight of the edges of any two anime they are both neighbours to), again divided by the length of the union of all anime either have as neighbours. It then takes a subset of all the users, the 15 found to be most similar to the given user. Then it finds which PR, that are neighbours to each of the subset of users, have the highest average CAR from those users. This implementation has the goal of recommending more niche anime. This is as it can find users very similar to the given user, and recommends shows they rated well on average, disregarding the fact that only a few users could have rated it at all. This process can be found in the function 'recommend_anime_user'.

Once the recommended anime are found, they are displayed in two ways. First, in a chart in Tkinter, that also gives the anime photo, and the 'score', or degree to which that anime was recommended. Secondly, it's displayed on 3 visual graphs. The first graph shows all the anime in anime_graph review_graph. Then in blue it highlights the anime the user reviewed, in red the anime that was recommended, and highlights the edges that create paths between them. The degree to which an anime was recommended, or the user rated, also affects the size of the vertex in the graph.

Graphical Visualization/Interaction Computations

To visualize the graphs we are outputting, there are 3 main functions in our visualization file. The first function `to_networkx` takes in a WeightedGraph, a list containing the ids of anime's which the user is being recommended, and a list containing the ids of anime's which the user chose. Based on whether the user chose the anime, or got the anime recommended to them by our algorithm, we set a certain color for that node in the grapht that we are building up (red refers to recommended anime, green refers to chosen animes, and black refers to other animes - note that the colors are web safe colors). Furthermore, in this function we also add other attributes such as the the image associated with the anime (cover image) and we also note the kind of the node (i.e anime/show, user). We have 2 major visualization functions, both of which use Bokeh and networkx. Graphs obviously play a central role in this visualization as we are literally visualizing graphs. The first function `visualize_graph` visualizes all nodes in the graph. Once again it takes in a weighted graph and 2 lists (list of chosen and recommended animes). We convert the weighted graph to a networkx graph using the function we wrote (`to_networkx`). In this function we have many interactive features as well. We base the size of the node based on the score the specific algorithm gave the anime's. To elaborate, based on one of the four algorithms a user chooses, we get different weighting scores for our nodes. The higher the weighting, the larger the node will be (note that we also linearly adjust the node sizes as some nodes would not be visible due to a score of 0 - anime's that were not chosen or recommended have a score of 0 but we linearly adjust the size so that is visible in our graph)). Also note that as stated before, the color of the node depends on whether it was chosen, recommended, or other. Furthermore, the thickness of edges between nodes are based on the edge weight of the two vertices the edge connects to. So the thicker a line is, the greater the weight between those 2 vertices. Next we also can hover over a node and see various information about the node being displayed (the name of the anime, the picture, whether it was chosen/recommended/other, and the score the algorithm gave the node. Also, when you hover over a node, the node and all the edges get highlighted so you can trace the paths to other nodes that are connected to the node you are hovering on. You can also zoom in and out to see the visualization in far more detail

We display 3 graphs. The first graph contains all nodes in our aggregated list. The second and third graphs we build are rather special! We have a special function for visualizing these 2 graphs, `visualize_graph_special`. This function is similar to `visualize_graph` in terms of the interactive abilities (i.e. node size, hovering over nodes, zooming in and out, etc.), but we are displaying far less nodes, and there is a good reason why. In particular, the second graph displays the **most** recommended anime based on the algorithm the user chose, and the graph displays all the paths to get from their **most** recommended anime, to all the anime they chose. Next, the third graph does something similar. This graph is a visualization of the users highest rated chosen anime and the paths from that anime to all the recommended anime the chosen algorithm outputted. In our opinion, these are both very intuitive visualizations. Also note that the same interactions with the first graph also apply with these two graphs.

Visual and Interactive

To present the results of our computations in a visual and interactive way, we utilized the TkInter python library. By developing a front end in TkInter we can allow the user's to input their user information Here is a step-by-step analysis of how we accomplish this:

1. After the loading screen, we present a small synopsis of our program to our user to give him or her a general idea of what the program will do. In addition, the user can input their name (this will be used for personalized wording later in the user experience).
2. The user can choose whether to use a MyAnimeList account or manually input information for the program to recommend anime.
3. If the user chooses to use a MyAnimeList account, a separate window will be created for the user to allow Anime Atlas to use the information from their account. However, if the user chooses not to use a MyAnimeList account, the user will be taken to a page where they can input anime that they liked watching.
4. After this step, the user can give Anime Atlas some specifications to what kind of anime they want recommended to them. Options include the genre of the anime, the number of episodes of the anime, the number of anime, and the type of algorithm to use to recommend anime.
5. After this, the user will be taken to a page where a grid of anime titles with their cover photos of their recommended anime is shown.
6. The user can also choose to view a graph representation of their anime recommendations on a web browser by pressing the 'graph' button.
7. If the user wants to retry his or her recommendation options, the user can click 'restart'.

New Libraries

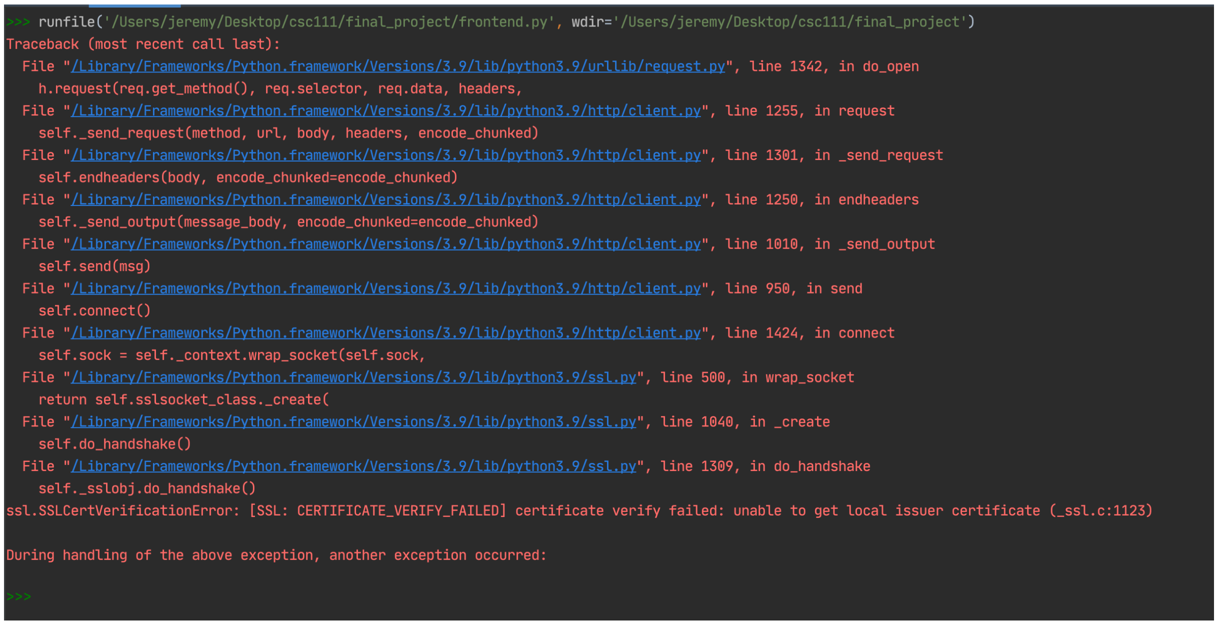
- **Bokeh**: We are using the Bokeh library to visualize the graph. Bokeh is a Python library for creating interactive visualizations for modern web browsers. Its main use is to build beautiful graphics, ranging from simple plots to complex dashboards with streaming datasets. Bokeh is much like plotly in the sense that it can be used to make visually pleasing and interactive graphs, and that is why we chose Bokeh. Some Bokeh specific functions that really helped us visualize our graph are the node and edge renderer functions. Essentially, we can have a dictionary that has the nodes or a pair of edges of the graph as keys and the associated value is some attribute of the node or some attribute of a pair of edges. With the attribute, we can use the set_attribute function to set that as an attribute, and then the node/edge renderer function to render the nodes based on that attribute. The usage of Bokeh can be seen in the 'visualization.py' file.

- **urllib** urllib is a python library that have several modules for working with URLs. The module in particular that we are using is the requests module. We use urllib.request functions (mainly the urlopen() function) which helps us in opening URLs. Furthermore, we also use this library to open links of images in TkInter to display the image of anime covers which can be seen in the window4 function of 'display.py'.
- **bs4/BeautifulSoup**: The BeautifulSoup library is being used to webscrape the MyAnimeList webpage for certain details that are not included in the MyAnimeList API. In a perfect world, the API would contain all pieces of information we need. However, this is not the case. Thus, we used the BeautifulSoup library to build a web scraper that uses requests to scrape and parse data from the web (specifically from MyAnimeList). This process can be seen in 'webscrape.py'. The way in which we use bs4 is we find web pages on the MyAnimeList web page that contains information unavailable in the API but is still required by us, and we read and manipulate the html scripts of these web pages. We use bs4 to specifically scrape multiple web pages that include the names of featured anime in a specific genre (ie: all the names of the featured animes in the comedy genre), as well as a webpage which has the usernames of currently active users. We use this information to develop our dataset of 850 possible anime recommendations, as well as a dataset of MAL users and the animes they have seen in conjunction with the score they gave it. We looked for patterns in the html scripts to find the string representation of both users or animes ids. More specifically we looked for ¡a href¿ tags and identified patterns in the various links present within these tags. These patterns were exploited and the information was collected as strings through list splicing and indexing.
- **Flask** Flask is a lightweight Web Server Gateway Interface web application framework. Using Flask allows our program to communicate with the MyAnimeList OAuth2 Server. Because of the OAuth2 security requirements we were forced to create an endpoint on our user's computer's local host on port 5000. To elaborate, we developed an app route @'/' (more specfically 127.0.0.1:5000/) which simply returns initialized' as a string on the web page at local host. Our users never see this page, however it is useful to us as developers because it tells whether or not the server is active. The 2nd app route @'/anime-atlas' is the address the MAL server redirects our user to once a successful authentication and log in has been preformed. This unique address is provided by us the developers, to the MAL servers, in order to prevent unwanted third parties from reading one of many secret keys. The final route @'/shutdown' allows us as developers to shut down the sever on local host manually, in case of any unwanted errors. Our users do not have access to this app route, however, a user could technically make changes and access these routes in their browser where local host is open even if there is no way to do so within the program.
- **Secrets** Secrets helps us generate random numbers for the security tokens. Our program generates an access token which can be obtained by a unique authorization link generated in token.py, which incorporates a randomly generated 128 character code (which cannot be reproduced).
- **TkInter** We are using TkInter to guide users and enable them to use our program more easily. In addition, TkInter allows us to create a more visually appealing program for users. The use of TkInter can be seen in the 'display.py' file where we program how the user will see and view our program when run.
- **Requests** We use the requests module in conjunction with Flask in order to execute code when a specific app route is triggered as a result of a redirect caused by the MAL server when a specific condition is triggered.

Obtaining datasets and running our program

The user does not need to download any datasets, as we are pulling data from an API. Anime Atlas works best on MacOS, as this is one of the limitations of TkInter. On the Windows operating system, Tkinter windows may look distorted and have gui features such as buttons, text boxes, and images may be overlapping each other or slightly translated from their original locations.

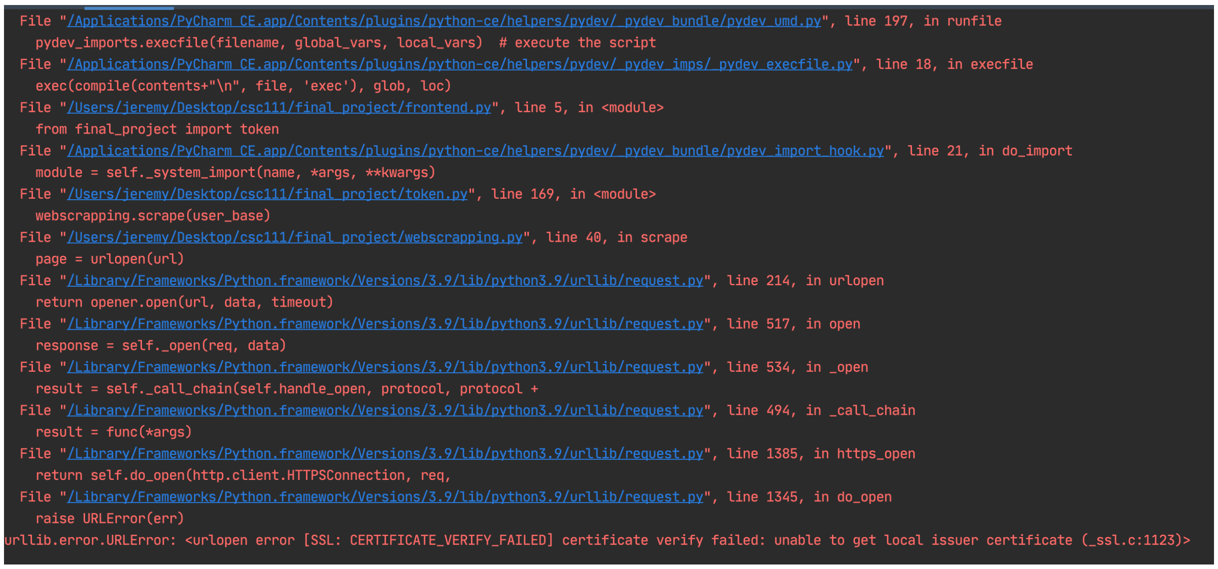
Moreover, OSX users may have to change python versions or reinstall certain certificates due to an error seen below:



```
>>> runfile('/Users/jeremy/Desktop/csc111/final_project/frontend.py', wdir='/Users/jeremy/Desktop/csc111/final_project')
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 1342, in do_open
    h.request(req.get_method(), req.selector, req.data, headers,
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 1255, in request
    self._send_request(method, url, body, headers, encode_chunked)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 1301, in _send_request
    self.endheaders(body, encode_chunked=encode_chunked)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 1250, in endheaders
    self._send_output(message_body, encode_chunked=encode_chunked)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 1010, in _send_output
    self.send(msg)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 950, in send
    self.connect()
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/http/client.py", line 1424, in connect
    self.sock = self._context.wrap_socket(self.sock,
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/ssl.py", line 500, in wrap_socket
    return self.sslsocket_class._create(
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/ssl.py", line 1040, in _create
    self.do_handshake()
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/ssl.py", line 1309, in do_handshake
    self._sslobj.do_handshake()
ssl.SSLCertVerificationError: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)

During handling of the above exception, another exception occurred:

>>>
```



```
  File "/Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pydev/_pydev_bundle/pydev_umd.py", line 197, in runfile
    pydev_imports.execfile(filename, global_vars, local_vars)  # execute the script
  File "/Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pydev/_pydev_imps/_pydev_execfile.py", line 18, in execfile
    exec(compile(contents+"\n", file, 'exec'), glob, loc)
  File "/Users/jeremy/Desktop/csc111/final_project/frontend.py", line 5, in <module>
    from final_project import token
  File "/Applications/PyCharm CE.app/Contents/plugins/python-ce/helpers/pydev/_pydev_bundle/pydev_import_hook.py", line 21, in do_import
    module = self._system_import(name, *args, **kwargs)
  File "/Users/jeremy/Desktop/csc111/final_project/token.py", line 169, in <module>
    webscrapping.scrape(user_base)
  File "/Users/jeremy/Desktop/csc111/final_project/webscrapping.py", line 40, in scrape
    page = urlopen(url)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 214, in urlopen
    return opener.open(url, data, timeout)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 517, in open
    response = self._open(req, data)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 534, in _open
    result = self._call_chain(self.handle_open, protocol, protocol +
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 494, in _call_chain
    result = func(*args)
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 1385, in https_open
    return self.do_open(http.client.HTTPSConnection, req,
  File "/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9/urllib/request.py", line 1345, in do_open
    raise URLError(err)
urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer certificate (_ssl.c:1123)>
```

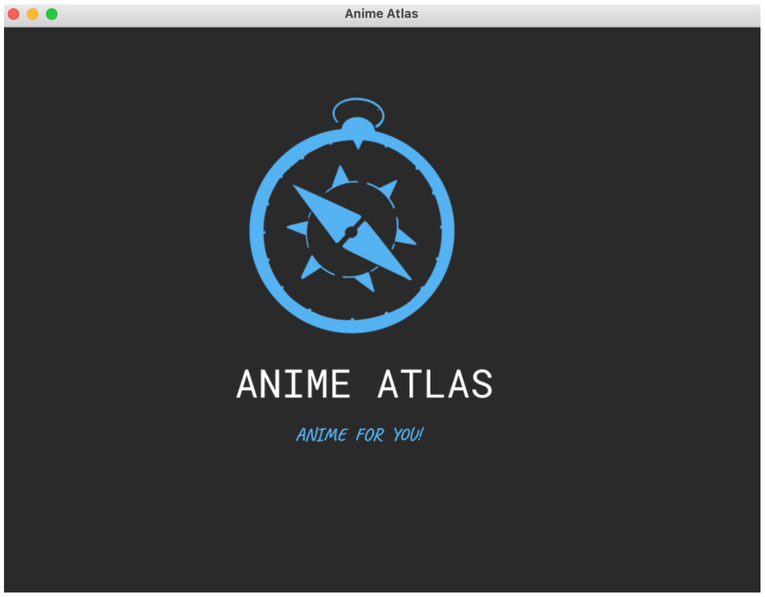
These two errors highlight missing certificates which can be reinstalled by going to Macintosh HD > Applications > Python3.9 folder > Install Certificates.command, and running this file. Running Install Certificates.command allows osx to update the necessary dependencies and certificates. If this does not work (and for some osx users it will not based on their current python environment) they will have to downgrade to python 3.8 or lower. Again, some users may not have this problem, however if they do reinstalling certificates or downgrading python in the IDE project interpreter should solve the issue.

Since Anime Atlas requires a custom local server to run, to run our project a user must first run myapp.py, and then run main.py subsequently. This is because the API calls made in main.py will not go through unless the server has been started, as the application will be unable to get the keys required to generate an access token to make an api call.

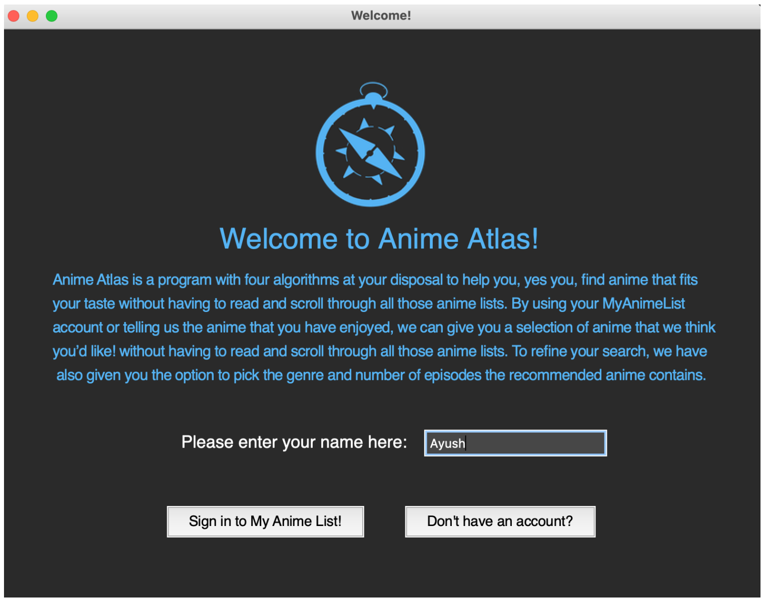
Program expectations

When the user runs main.py:

A new window should appear. You will see a loading screen like the following image:



After a few seconds, the user should be directed to the home screen. This is the screen where the user inputs their name and whether they want to use a MyAnimeList account or not. This screen should look like the following image:



Welcome to Anime Atlas!

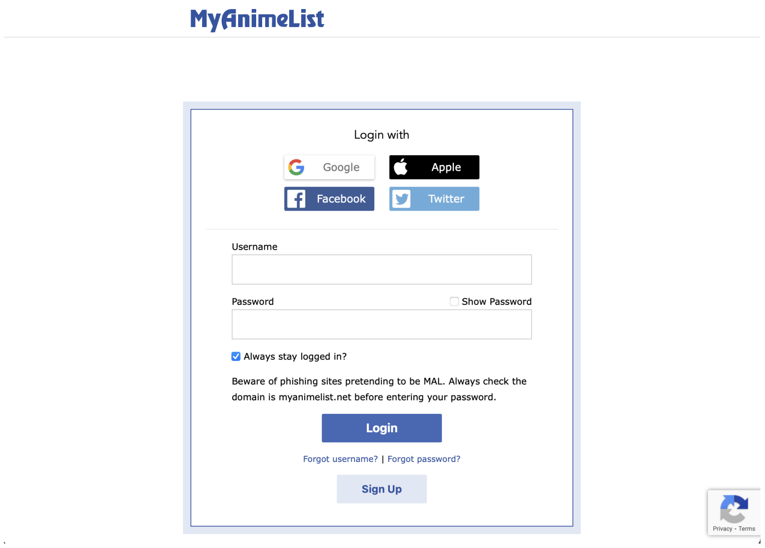
Anime Atlas is a program with four algorithms at your disposal to help you, yes you, find anime that fits your taste without having to read and scroll through all those anime lists. By using your MyAnimeList account or telling us the anime that you have enjoyed, we can give you a selection of anime that we think you'd like! without having to read and scroll through all those anime lists. To refine your search, we have also given you the option to pick the genre and number of episodes the recommended anime contains.

Please enter your name here: Ayush

The way the user inputs this information is by typing into the entry box and pressing either one of the two buttons below the entry box.

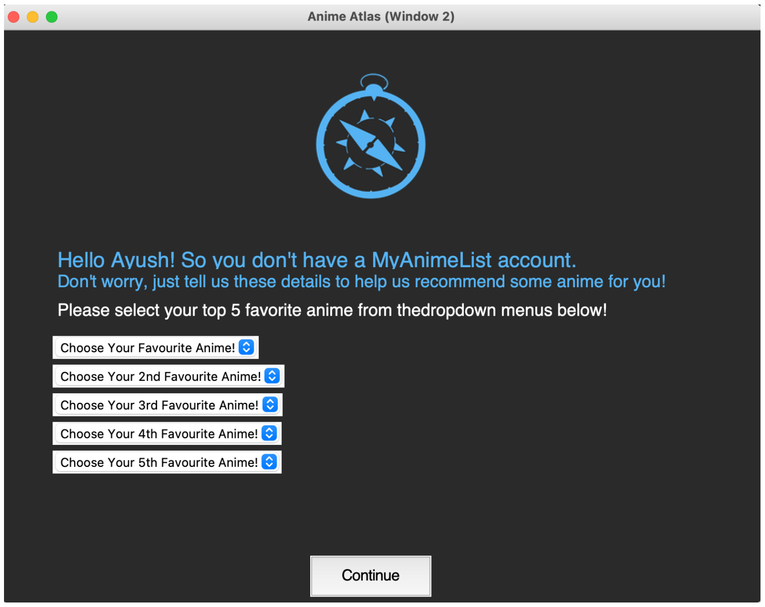
After the user clicks one of the buttons, they will be taken to a screen displaying either a new window to log into MyAnimeList or a page to input the anime they have liked depending on which button they have pressed.

If the user wants to use a MyAnimeList account, they will be taken to a page on a browser to log in into their MyAnimeList account. We have set up a test account for you to use in the event that you do not have a MyAnimeList account. **username:** AnimeAtlasTest **password:** AyushJeremyIanPrithee100 .



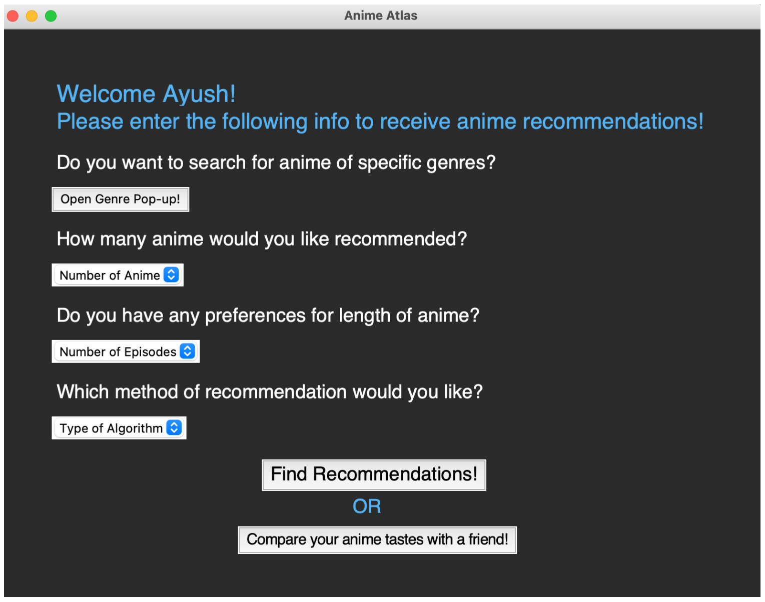
Beware of phishing sites pretending to be MAL. Always check the domain is myanimelist.net before entering your password.

The user will be taken to this page if they do not want to use a MyAnimeList account.


Anime Atlas (Window 2)
Choose Your Favourite Anime!
Choose Your 2nd Favourite Anime!
Choose Your 3rd Favourite Anime!
Choose Your 4th Favourite Anime!
Choose Your 5th Favourite Anime!
Continue

Hello Ayush! So you don't have a MyAnimeList account.
Don't worry, just tell us these details to help us recommend some anime for you!
Please select your top 5 favorite anime from thedropdown menus below!

After these pages, the user will be taken to a page where they can specify options for the anime they want recommended to them.



Welcome Ayush!
Please enter the following info to receive anime recommendations!

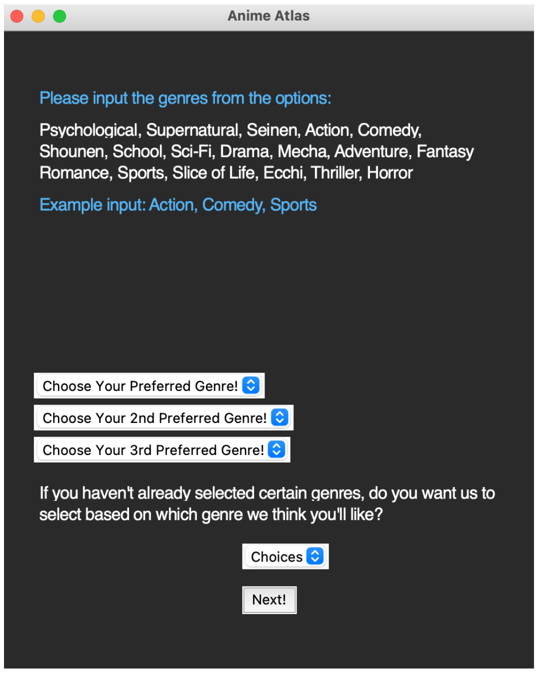
Do you want to search for anime of specific genres?

How many anime would you like recommended?

Do you have any preferences for length of anime?

Which method of recommendation would you like?

In this page, the user can specify a genre of anime, number of anime, and length of anime they want recommended to them. In addition, they can choose the method of recommendation they would like. In the options of the number of anime, length of anime, and method of recommendation, the user can input their preferences using a drop down menu. When choosing the genre preference of the recommended anime, a pop-up window will appear:



Please input the genres from the options:

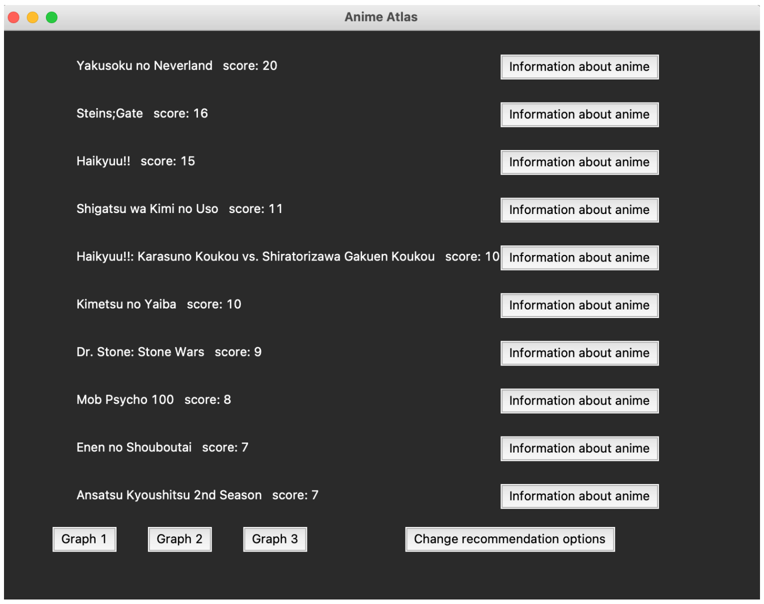
Psychological, Supernatural, Seinen, Action, Comedy, Shounen, School, Sci-Fi, Drama, Mecha, Adventure, Fantasy Romance, Sports, Slice of Life, Ecchi, Thriller, Horror

Example input: Action, Comedy, Sports

Choose Your Preferred Genre!
Choose Your 2nd Preferred Genre!
Choose Your 3rd Preferred Genre!

If you haven't already selected certain genres, do you want us to select based on which genre we think you'll like?

After the user clicks the recommend button, the user will be taken to a page displaying the anime recommended to them:



Anime Atlas

Yakusoku no Neverland   score: 20

Steins;Gate   score: 16

Haikyuu!!   score: 15

Shigatsu wa Kimi no Uso   score: 11

Haikyuu!!: Karasuno Koukou vs. Shiratorizawa Gakuen Koukou   score: 10

Kimetsu no Yaiba   score: 10

Dr. Stone: Stone Wars   score: 9

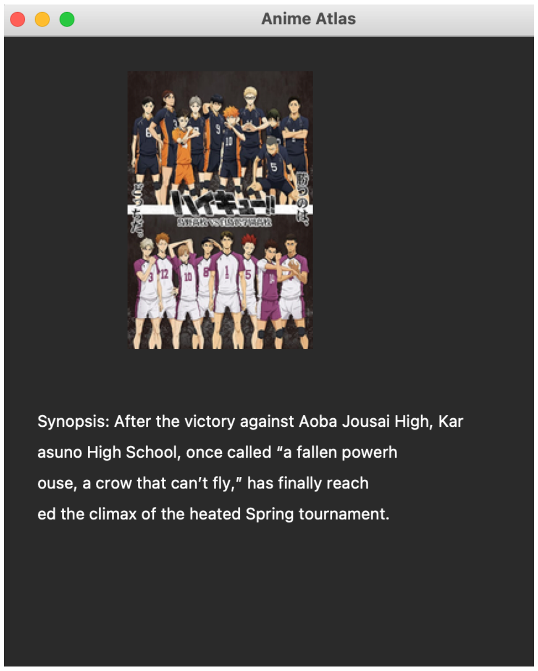
Mob Psycho 100   score: 8

Enen no Shouboutai   score: 7

Ansatsu Kyoushitsu 2nd Season   score: 7

We see that a score is provided for every recommended anime to show the degree to which the algorithm thinks the user will like. The shown score when recommending anime, does not represent the percentage to which that anime is recommended. instead, it simply is the value used to rank all anime in order of which should be recommended first. We display it for fun, it does not have any important meaning to the current user.

For every recommended anime, there is also a button to see get more information on that anime (cover photo and synopsis). A page would look like this:

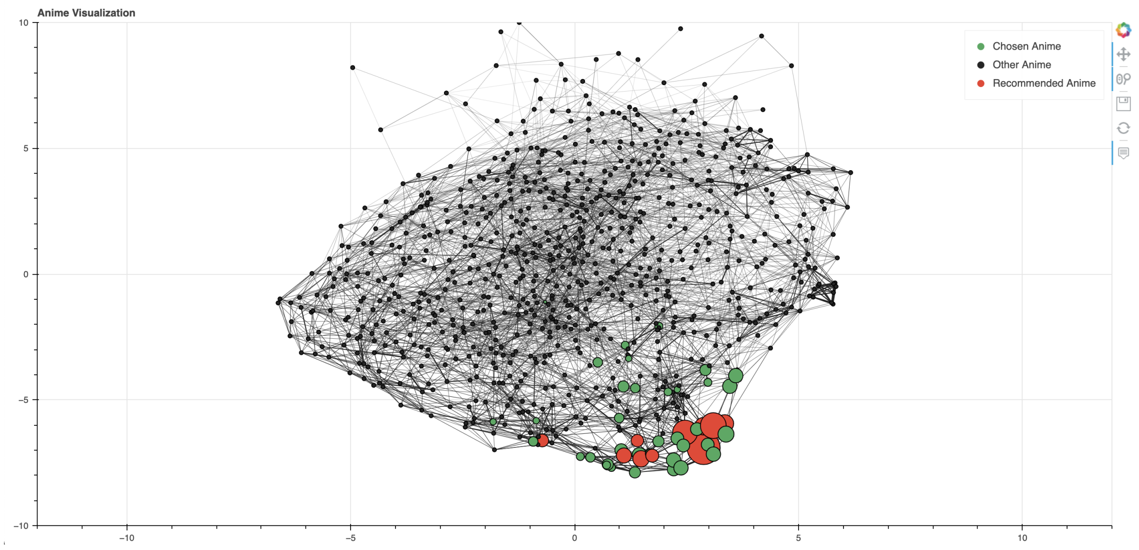


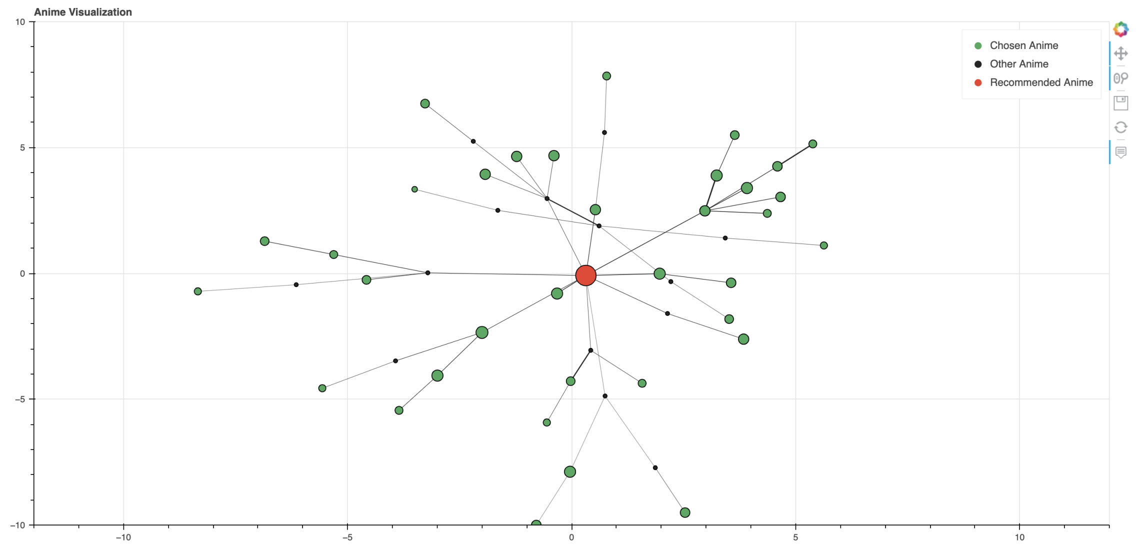
Anime Atlas

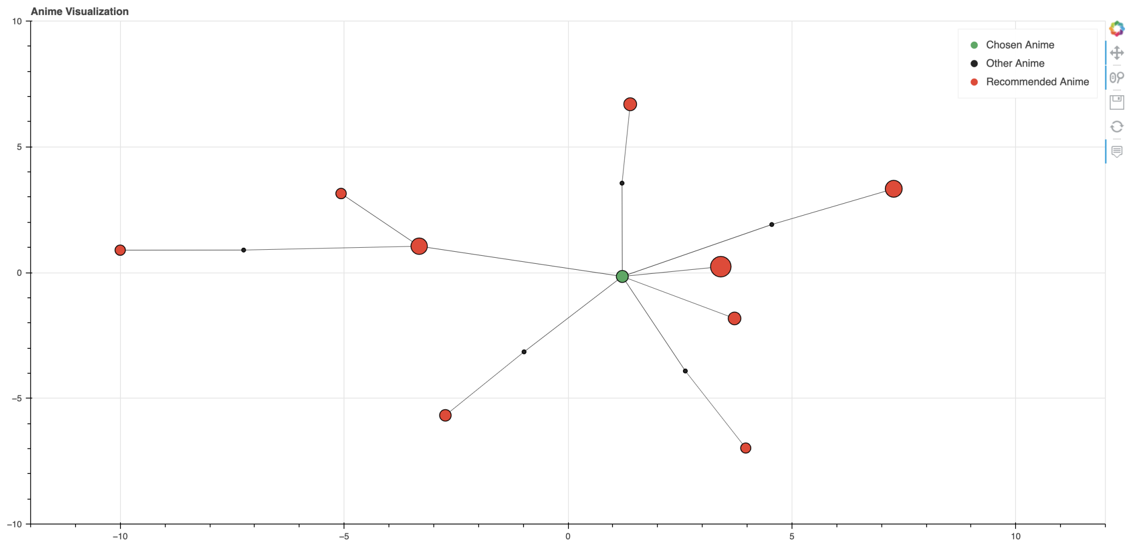
Synopsis: After the victory against Aoba Jousai High, Kar
asuno High School, once called "a fallen powerh
ouse, a crow that can't fly," has finally reach
ed the climax of the heated Spring tournament.

After looking at the recommended anime, the user can click the buttons 'graph1', 'graph2', and 'graph3' to open a window on their web browser to see the graph visualization of their recommendation. **Try hovering over nodes and seeing information pop up! You can also see the neighbours of vertices as they will be highlighted when you hover over a node.**

The user will see these graphs of graph1, graph2, and graph3 respectively. The first graph contains all vertices in our aggregated AnimeDict. The second graph displays the **most** recommended anime based on the algorithm the user chose, and the graph displays all the paths to get from their **most** recommended anime, to all the anime they chose. The third graph is a visualization of the users highest rated chosen anime and the paths from that anime to all the recommended anime the chosen algorithm outputted. In our opinion, these are both very intuitive visualizations. Also note that the same interactions with the first graph also apply with these two graphs.



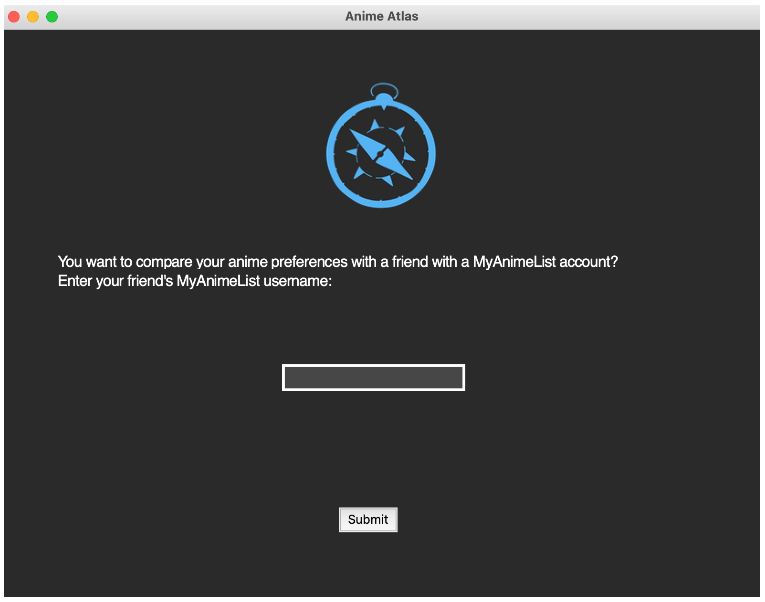




The red vertices represent anime the user is being recommended, and the green vertices represent the anime the user chose (MAL anime list) and the black nodes represent any other anime not in the first two groups.

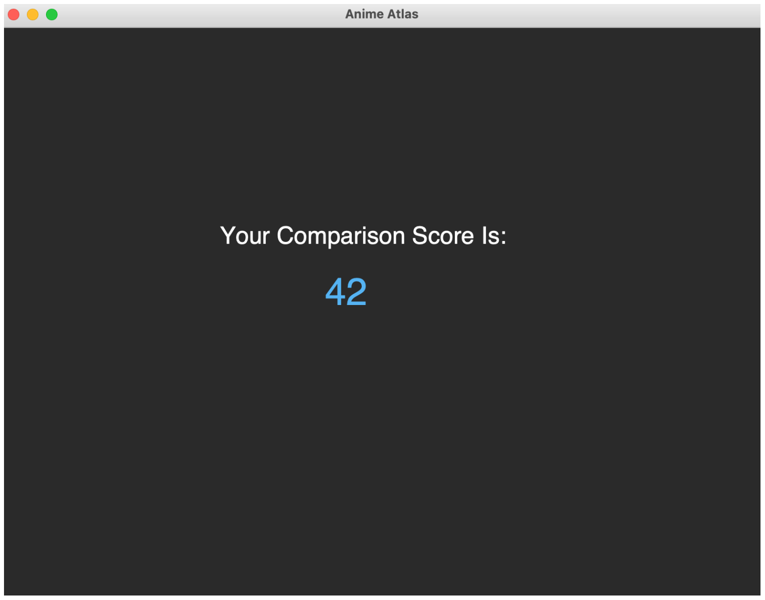
After viewing the anime recommended to them, the user can click a button to change the initial recommendation preferences.

Another feature of our program is that Anime Atlas is able to compare the user's anime taste with another MyAnimeList account. In the page where the user inputs their recommendation options, there is another optional button called 'Compare your anime tastes with a friend!'. After clicking this button, the user will be taken to this page:



You want to compare your anime preferences with a friend with a MyAnimeList account?
Enter your friend's MyAnimeList username:

In this page the user can put another user's MyAnimeList username to compare their tastes. After clicking the 'submit' button, the user will be taken to this page:



This number is a score of how similar the user's anime tastes are with each other.

Differences from project proposal

One difference between our final project, and our initial proposal was how the recommendation algorithms were set up. We realized there were many ways to do it, and that we hadn't thoroughly thought through the exact math of the recommendation algorithm. At first we were planning to simply look at tags of anime's (tags are represented as genres) and perform operations on sets of tags (i.e. set.intersection) to find similar anime's. However this was rather simple and not a good way to recommend anime in our opinion. We did not want to rely on just 1 single algorithm, but rather we wanted to view our domain from different algorithmic viewpoints. Due to this, we decided to implement 4 different recommendation algorithms all with different strengths. Thus, that is why we did not proceed with the genre tags to make recommendations, as our proposal suggested. The genres or tags associated with the anime instead were only used as a filter at the end of the recommendation process, once a large group of potential recommendations had been aggregated. Originally in our project proposal we said our project will "recommends users new anime based on their individual tastes." We realized that "taste" was not really defined at all in our proposal. We now define a users "taste" as the genres of anime that are most prominent in the anime the user likes (likes as in it is highly rated (8.0+) in their MAL account, or they weighted it highly)

Discussion

0.1 Did we meet our goal/question?

After hundreds and hundreds of lines of code and a lot of blood, sweat, and tears, the big question is did the results of our computational exploration help in recommending anime's to users. In our opinion, yes, we do believe that the results of our computational exploration definitely did recommend anime's to users to a solid degree. We had several algorithms (4 to be exact) which recommended anime's to a user based on a variety of different attributes. The keyword is variety here. The way that we coded the algorithms make intuitive sense to us, but it may not make sense to another person, as in, we can build algorithms to recommend anime's, but it will never be completely perfect, and that is why we believe we have achieved the goal of our project to a solid degree and not an excellent degree. However, with that being said, our algorithms are quite accurate. Each time we ran our project, we would always view the recommended anime's based on the inputs that we enter, and 9/10 times, the recommendations are quite solid. Furthermore, as a part of our goal, we not only wanted something the just spits out anime in the python console, but we also wanted to build a program that people actually want to use. While that does mean that our algorithm needs to be solid, our user interface also has to be visually pleasing and easy to use. We believe that we have successfully done this, so the results of our computational exploration regarding the user interface has definitely helped us in building a program that recommends anime's to others.

0.2 Limitations/Obstacles Encountered

Algorithm

In general, the creation of the algorithms turned out to be very complex and difficult, and there were many many iterations before the final product.

One thing we initially overlooked was the creation of a useful anime_graph. One problem was gathering more data on a specific anime, as more users were needed. However, each new user added brought a whole new set of anime for which they were the only one to review. This meant that even more users were needed for data on that anime. This would keep happening. The solution to this problem was to limit the dataset to only 850 popular anime in a range of genres. We'd gather tons of reviews of them, and ignore the rest.

Secondly, when creating the anime_graph, many anime would fall into one of two not very useful categories: Having almost no connections, or having far too many. This was solved by changing the way in which review_graph was converted into anime_graph. Originally there was a threshold for which if two anime's broad_similarity passed in review_graph, they were given an edge in anime_graph. Instead, each anime was connected with the 10 or less anime it had the highest broad_similarity with (that didn't already have 10 connections of their own), regardless of a threshold value. This made the graph far more spread out, useful for computing on, and visually pleasing.

Initially, the algorithms only looked at which anime were similar to the anime the user reviewed. This completely missed the fact that the user gave differing scores of the anime they reviewed. For example, anime similar to an anime the user gave a rating of 1, should be less likely to be recommended. This forced us to decide and implement, based on average review score stats, at what rating an anime's rating negatively impacts the recommendation chances of the anime it is similar to.

Finally, we found that the first 3 algorithms, 'Weights', 'Neighbours', and 'User Comparison' had a certain flaw: they mostly only looked to recommend anime that were neighbours to the users reviewed anime. This didn't take advantage of the long paths and overall connectedness of a graph data structure. The final algorithm, 'Search', was designed specifically to address this problem. Implementing 'Search' turned to be the most challenging algorithm of the project, but was consequentially very rewarding. It initially took very long to run, mainly due to the helper function find_path_recursion inefficiently only finding the shortest path between two given anime, and disregarding all the other paths discovered along the way. This was fixed, and, as a result, the find_path_recursion function was called far fewer times, and in each call far more of the data aggregated was actually used. find_path_recursion also ended up proving to be very helpful in creating a very pleasing visual graph, of which is explained elsewhere.

There is one other note about how the algorithms take into account genres the user has chosen to be recommended. If none of the anime a given user reviewed on their MyAnimeList, or instead picked contain much of the genre the user is looking for, they will not be recommended many/any shows in that genre. This is because, there is no clear way to decide which anime in that genre to recommend, as their taste has little in common in it. If the algorithm were to pick anime in that genre, their is usually almost no data on them after the running of the algorithms, or they have negative recommendation scores. For these reasons, we decided against recommending them in these scenarios

Graphical Visualization

There were a couple limitations when trying to implement the graphical visualization of the users graph. Firstly, we originally intended to use the matplotlib library to visualize the graph as that was compatible with TkInter (compatible as in TkInter supports opening a new Tk window to embed the matplotlib figure in it), however, upon implementing the graph, while we did get a visualization, there were only nodes on the screen and there was no way to tell which node represented what (i.e. that all the nodes were the same size, color, shape, etc. They were basically virtually indistinguishable from each other). At first, we heavily researched if there was any way to "hover" over nodes and get information from it, but we hit a dead end as that is not possible yet in matplotlib. So we took a different approach. We used a new library called bokeh.io to implement the graph. This was a brand new library/module that we had 0 previous experience with, so it was a bit daunting to work with at first. Upon reading the documentation for bokeh.io and its integration with networkx, we quickly realized that the 'syntax' for bokeh was quite different from what we were used to, so it was quite the learning curve. However, after reading more documentation and testing out the functions and abilities that bokeh.io had to offer, we saw how powerful bokeh really is.

Dataset-Related

When building the user interface, we cam across several limitations as well. The MyAnimeList API call did not have all the necessary data points we required for our project as the API is still in beta. As a result, we employed web scrapping scripts using beautifulsoup4 to obtain information we could not access.

Visual and Interactive

Initially, we wanted to embed the MyAnimeList login and graph visualization onto the same window as the user input, however, due to the incompatibility of the python modules with TkInter, we had to open these features of our program in a browser window, making the user experience more complicated.

Although Bokeh is a very powerful library for visualization, there are some drawbacks as well. There is currently no way to specifically highlight select edges. The reason why this was so important to us is because we wanted to be able to show the path from chosen animes to recommended animes in the main graph itself. However, this is currently not a feature in Bokeh, so to work around this problem, we created 2 more graphs which essentially do something similar.

In addition, TkInter does not work well when switching operating systems (e.g. Switching between MacOS and Windows), as the formatting of the TkInter pages are not adaptive for different operating systems, thus, a limitation for the visual and interactive representation of our results, as only users using certain operating systems will be able to experience the best version of Anime Atlas.

Lastly, in the page where we display the recommended anime to the user, since the user can choose the number of anime to be recommended, we wanted to utilize a for loop to format the page of recommendations. However, a limitation of TkInter is that when objects are created in a loop, the objects override each other. Therefore, we resulted to using if statements which increased the number of lines of code we needed to implement.

0.3 Next Steps for Further Exploration

There a few intuitive ways to further explore our domain. Firstly, we built quite complex and intuitive recommendation algorithms, however we did not build a algorithm which checks the validity of our recommendations. To elaborate, the recommendations that are outputted make sense to us as people who are avid anime watchers, but a computer does not understand the same way we do. So we could possibly try building an algorithm which checks if the recommended anime is a valid output.

Next, We could vastly expand our domain of anime's, reading from tens of thousands of users, rather than 750, to improve our precision in our algorithms. We could also increase the list of possible anime from 850 to a far greater amount, possibly all the anime in MyAnimeList. While this would most likely require a database (MySQL, Django, etc.), it is something we could definitely try in the future.

We could also build even more superior algorithms, which maybe combines the ideas explored in the other algorithms into one algorithm. There is no single "best" algorithm for this in our opinion. Some will be better than others, but we want to try to come as close as possible to a perfect recommendation algorithm.

We plan to keep working on Anime Atlas and eventually putting it online for anyone with a MyAnimeList account to access, and find recommendations. As of now, this is not possible with our current project as we have implemented everything in Python, so we would most likely need to use a front-end focused programming language such as React.js, HTML, etc., so that we would be able to display results more effectively.

References

"MyAnimeList API (Beta Ver.) (2)." 2021. MyAnimeList API v2. Accessed March 15. https://myanimelist.net/apiconfig/references/api/v2.

Apiary. 2021. Kitsu API Docs · Apiary. Accessed March 15. https://kitsu.docs.apiary.io/xtreference/anime/anime.

Contributors, Bokeh. "Configuring Plot Tools." Configuring Plot Tools - Bokeh 2.3.1 Documentation, docs.bokeh.org/en/latest/docs/user_guide/tools.html.

Contributors, Bokeh. "Visualizing Network Graphs." Visualizing Network Graphs - Bokeh 2.3.1 Documentation, Bokeh, docs.bokeh.org/en/latest/docs/user_guide/graph.html.

"Graph-Undirected Graphs with Self Loops." Graph-Undirected Graphs with Self Loops - NetworkX 2.5 Documentation, 22 Aug. 2020, networkx.org/documentation/stable/reference/classes/graph.html.

Nunez-Iglesias, Juan. "HoverTool Displaying Image." Bokeh Discourse, 3 May 2016, discourse.bokeh.org/t/hovertool-displaying-image/1198.

"Beautiful Soup Documentation." Beautiful Soup Documentation - Beautiful Soup 4.9.0 Documentation, www.crummy.com/so

"TkInter." TkInter - Python Wiki, 2021, wiki.python.org/moin/TkInter.

"Tutorial." Tutorial - NetworkX 2.5 Documentation, 22 Aug. 2020, networkx.org/documentation/stable/tutorial.html.

"Webbrowser - Convenient Web-Browser Controller." Webbrowser - Convenient Web-Browser Controller - Python 3.9.4 Documentation, docs.python.org/3/library/webbrowser.html.

"Secrets - Generate Secure Random Numbers for Managing Secrets." Secrets - Generate Secure Random Numbers for Managing Secrets - Python 3.9.4 Documentation, docs.python.org/3/library/secrets.html.