# Anime Recommendation System

Sahil Shah (sshah88@jhu.edu) – EN.601.466

**Summary:** This project aims to use user-based collaborative filtering in order to create a recommendation system for anime television series/movies. User ratings and anime information is retrieved from popular online database and community MyAnimeList in order to implement this.

**Online Interface:** https://anirec-collab.herokuapp.com (requires a MyAnimeList profile to see recommendation page, 'omegablitz' can be used to test)

**Data Repository:** https://github.com/sahilshah88/anirec-collab (contains scraping scripts, final scraped data for user ratings and anime, website code and evaluation script)

# Table of Contents

# Overview

---

## *Background*

Anime is a largely popular form of media consisting of Japanese animated TV shows and movies, and accounts for around 60% of all animated television shows. MyAnimeList (MAL) is an anime database and online community (equivalent to something like IMDB) that has millions of active accounts, and a majority of people who watch anime keep track of their shows and rate them on this website. These shows are relatively short (usually 12-24 episodes long) so users tend to watch many and thus take advantage of this tracker and ratings system (active users have an average of ~200 shows rated). With this large store of user information and also the desire to find new shows to watch on a regular basis, it seems that a recommendation system for anime could benefit greatly. However, the extent of recommendations on MyAnimeList are manual recommendations that users make for each show. Example: When looking at Show A, you see that 15 users have recommended Show B since it is similar for reasons x, y, and z. Most users don't bother to recommend shows this way or look at these recommendations as they are quite unreliable and only based on a particular show. Using a user-based collaborative filtering system, we could vastly improve recommendations by taking into account a user's entire list and computing similarities between other users, and recommending shows that the most similar users liked.

## *Guide and Functionality Screenshots*

The interface is relatively intuitive – simply visit the provided website (https://anirec-collab.herokuapp.com and enter the username of the MyAnimeList profile that you want recommendations for (should work in all browsers including mobile but best in Google Chrome). A few sample usernames from myself and friends are cheesydelight, omegablitz and arxenix.

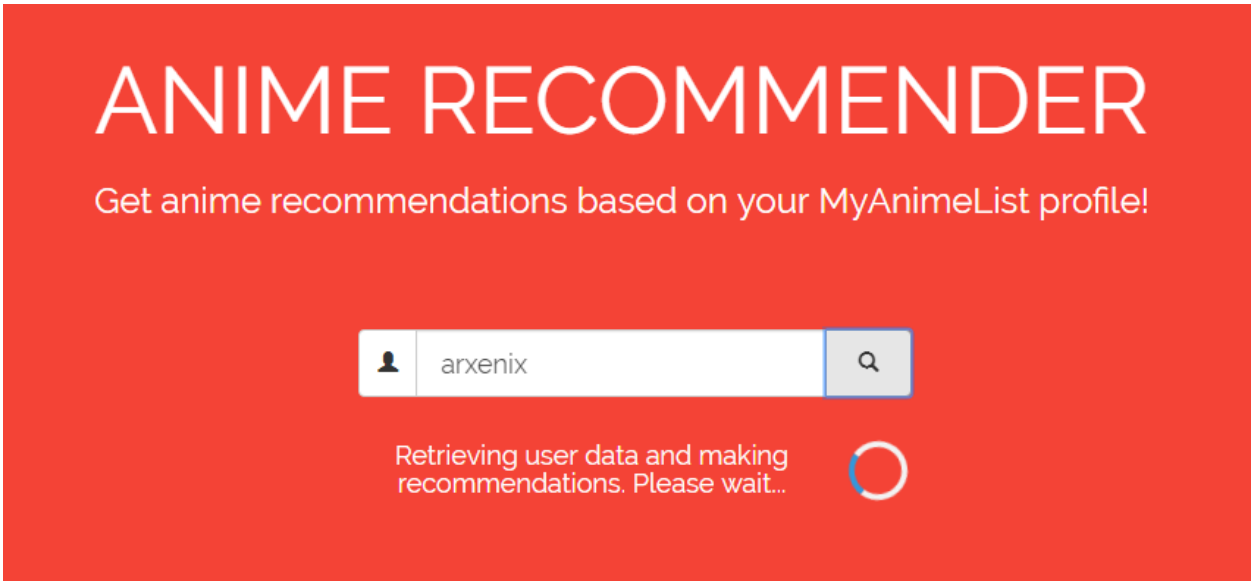


Figure 1 – Loading recommendations for entered user

It may take a few seconds to load but should produce the top 40 recommendations along with a predicted rating, as well as general user information.

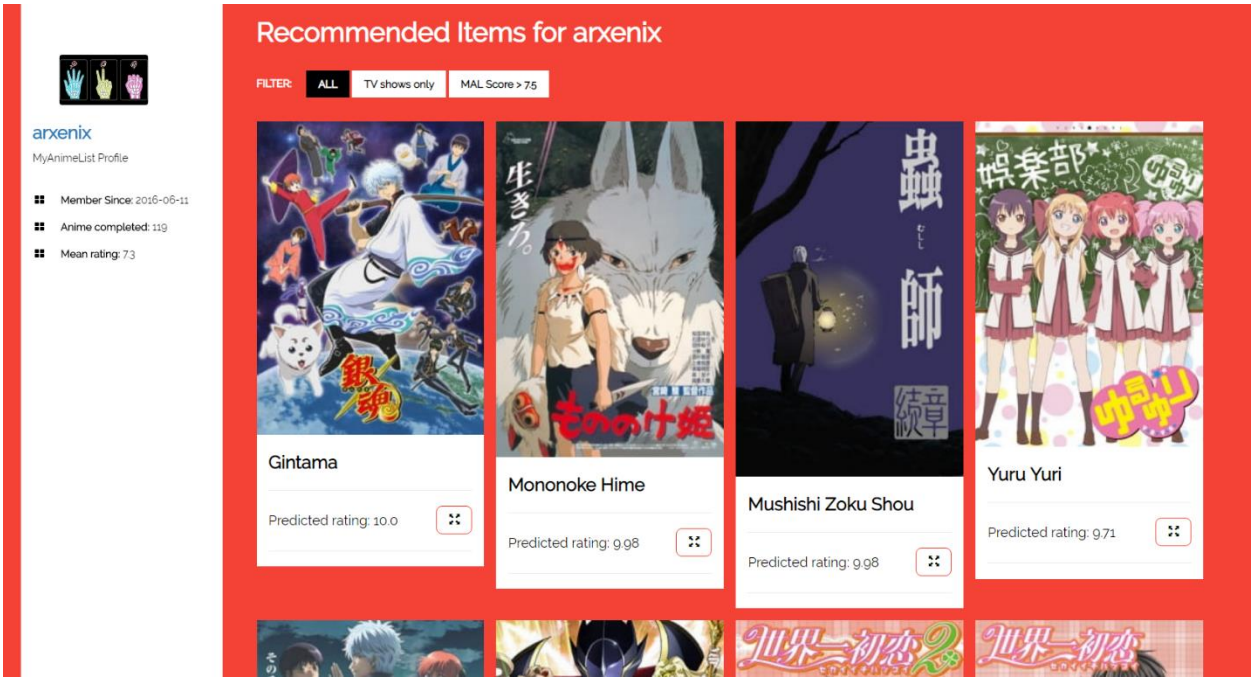


Figure 2 – Results page with user info sidebar/profile link

Each result can be expanded for more info and a link to the original MyAnimeList entry.

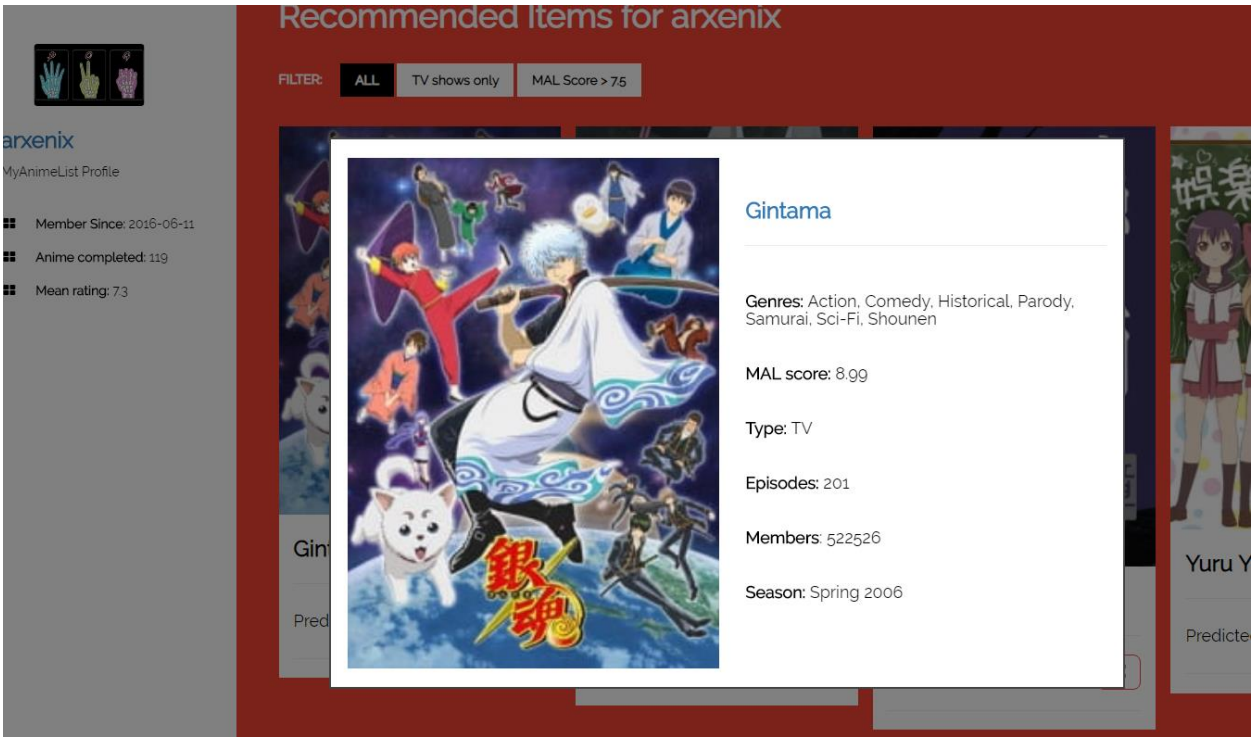


Figure 3 – Popout with more information and MAL link

Recommendations can be filtered to the top 40 TV shows and top 40 results with MAL score above 7.5 using the filter bar at the top as well.

## *Achievements*

This project incorporates multiple aspects of information retrieval and is something that would be helpful to many people. In particular, this project:

- Wrote scripts to scrape ~30,000 of the most recent user profiles to get fresh information and the ability to recommend more recent anime as well
    - From these users, ~5 million distinct ratings were extracted, allowing for a reasonably accurate recommendation system

- Wrote scripts to scrape information for ~10,000 anime TV series and movies including title, genres, image, members, etc. in order to help users decide which recommendations are best for them
- Created a user-based collaborative filtering system using a PostgreSQL database to store tables for users, anime, and ratings
    - By exclusively using custom made PSQL queries to find similar users and make predictions, as well as precomputing values to speed up cosine similarity, the system is able to make recommendations quickly and efficiently
    - The system takes into account deviations from the mean rating for each user in order to remove rating bias (some users rate all shows relatively high while others rate lower on average)
- Developed a website from scratch in Flask to help users easily visualize recommendations and choose what they might like
    - Heroku hosting with PostgreSQL addon to allow the website to connect and interact with the database to make recommendations
    - Filters to allow users to choose to see TV shows only or shows above a MAL score threshold of 7.5 (considered slightly better than average)
    - Jinja templating engine used with HTML to generate a flexible website layout to display recommendations and pop-outs for each recommendation, and allows for easy incorporation of more filters and recommendation conditions in the future
- Script to evaluate recommendations from the system as a whole using RMSE and MAE, customizable to get an idea of accuracy on particular users
- A lot of the process was self-taught (web development and hosting, PSQL and collaborative filtering) but it was a lot of fun since it was a project of interest!

## *Evaluation*

A common evaluation system for recommendation systems is used: Mean Average Error (MAE) and Root Mean Squared Error (RMSE). RMSE is the evaluation most often mentioned when considering evaluating recommender system. It was used for the popular Netflix Prize as well,

which had people attempt to minimize the RMSE value for Netflix recommendations. MAE is a simplified version of this. The formula for RMSE is as follows:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(r_p - r_a)^2}{n}}$$

Where n is the number of ratings being evaluated, $r_p$ is the predicted rating and $r_a$ is the actual rating. MAE is as follows:

$$MAE = \frac{\sum_{i=1}^{n}|r_p - r_a|}{n}$$

Both are very similar, but RMSE involves squaring which increases error much more significantly when the difference grows greater than 1. This means a system that consistently gives outlandish ratings will be treated more harshly in RMSE, which is usually preferred. To use these measures, "actual" values are needed alongside the predicted values. Of course, it is not practical to get many people to watch and rate all of their recommendations, but this can be solved by removing a few ratings from a sample set of users and retrieving recommendations for them, and then comparing predicted values for the removed anime with the actual ratings they had given. This is done in rsme.py, which takes a sample of 100 random users with between 100 and 200 anime rated and removes 10 at random (these are reinserted after evaluation). This script was run with a few different configurations, including different values for k nearest neighbors (how many similar users to find) and weights to actual MAL score (average user rating on MyAnimeList). This script needs to run the recommendation algorithm on each user individually, so running this script can take a few minutes. The results obtained are below:

| | *10% MAL weight, 20 neighbors* | *30% MAL weight, 20 neighbors* | *10% MAL weight, 50 neighbors* |
|---|---|---|---|
| *MAE* | 1.088 | 1.115 | 1.069 |
| *RMSE* | 1.397 | 1.415 | 1.369 |

This roughly means that ratings were 1-1.5 off on average on a 10-point scale, which is a decent result. One thing that could contribute to this is the fact that actual ratings are always integers while predicted are usually in between. There is also the problem of predicted values being more representative of the rank of predictions rather than an actual MAL score. This is evident since many of the predicted scores are originally greater than 10, since it is based off of a deviation from the user's mean rating (which can fall outside the range). These scores are normalized based on the top scoring anime (so the top one is always rated 10) which may not be a completely accurate way to normalize it to the MAL scale. This result is likely sufficient for this context, however, since most of the top 40 ratings tend to be in the high 8 to 10 range, so an average error of 1.5 would still give relatively good recommendations on average (ratings of 7 and up are usually considered as "good" ratings). In the worst-case scenario, it will give a host of decent candidates for shows that the user hasn't watched yet, which MyAnimeList doesn't offer as a filter option when browsing and is an important factor for users with a lot of watched shows.

## *Limitations/Future Goals*

A limitation which is the main problem for many collaborative systems such as this is the sparsity problem: there are so many shows, movies, etc. in the database that finding users that align very well with the original user can be difficult, especially for users who have less conventional tastes. This is mitigated more in an anime database as users have seen and rated much more on average than Netflix or IMDb (around 200 on average). This is a limitation that is sometimes solved with matrix factorization methods such as SVD, which condenses the user rating matrix to find latent features. A specific variant of this called FunkSVD was used for the Netflix prize and seems like it could be an interesting possible extension to explore and experiment with in the future.

Another limitation that is somewhat inevitable with a basic recommendation configuration is the fact that many very popular anime tend to surface at the top of recommendations. This makes sense as many people have rated these shows highly, but there is the logical problem that many people using a recommendation engine like this likely know about/have explored all of the popular options and thus these recommendations are useless for them. One example of this is

Gintama which is an anime series that has 7 variants (sequels, movies, etc.) in the top 25 rated anime on MyAnimeList, so some form of this show is often recommended to those who haven't watched it. One way that this could be solved is with a filter that removes very popular (>x members) anime, or bias the algorithm itself towards less popular shows that are still highly rated.

Another way to mitigate the previous problem as well as address a current issue is sequel checking. People generally don't want to watch a sequel of a show without watching the original season, and MyAnimeList breaks each season into its own entry (which is why we see so many Gintama shows pop up, for example). Thus, having a way to eliminate sequels from the recommendation results (either a filter or default) would be nice, but it becomes a little complicated. Shows on MyAnimeList are not marked as sequels, but it does indicate if a show has prequels. However, this could be a prequel that came after the sequel (like a less popular origin story), so we would still want to recommend the sequel as a standalone show. Distinguishing this can be difficult, but a naïve approach could be used that mark shows as sequels if: a) they have prequel(s), and b) at least one of the prequels has more members (are more popular). For this reason, I have parsed prequel lists in the PSQL anime database, but did not have time to make this extension – this would be a good future goal.

Other future goals are mainly concerned with more flexibility for users with filtering recommendations (excluding shows of a certain genre, shows aired before a certain date, etc.). This is very feasible with the comprehensive database compiled and just requires some more time to implement.

# Detailed Development Process

---

### *Data Collection*

The first step in the collaborative filtering process is to generate a table of many user ratings. I would also like information for the anime and users (this is not entirely necessary but allows us

to make a user-friendly and aesthetic website to display recommendation results with some associated info). To retrieve data from MAL, I use the unofficial Jikan API (which acts in accordance to the MAL terms of service). https://github.com/jikan-me/jikan-rest#installationI installed an instance of this REST API server locally to easily query and receive JSON data for any MAL-related info we need. MAL terms of service states the following in regards to web robots/scrapers: "You also agree not to use or launch any automated system, including without limitation, "offline readers," "spiders," "robots," etc., that accesses the Service in a manner that sends more request messages to the Company servers than a human can reasonably produce in the same period of time by using a conventional on-line web browser." Thus, we limit requests to the API to 1 request/second which a human could reproduce (MAL allows ~2/second before rate limiting, but we limit to 1 to ensure we don't abuse/overload the servers).

After launching the local API instance by following the instructions on the Jikan README, we use several scripts to aggregate the data we need. First, many user profiles are needed. There is no publicly available list of MAL users, but if the username is known, their ratings can be freely viewed as long as their profile is public, which means we can download user data from the given usernames. In order to get a list of active users on MAL, https://myanimelist.net/users.php is queried, which gives the 20 most recent users on the site. We do 1 request every 3 seconds using the script: scrapeusers.py (output piped to txt file). The API does not have this capability, so we parse the HTML from the site ourselves to get 20 users every 3 seconds. There are enough active, unique users that each query more or less gives 20 unique usernames. At the end, after removing duplicates, we obtain ~30,000 usernames of active profiles, which is convenient since the most recent anime are also likely to have ratings in this recently active user group, which will help our recommender target newer shows.

After getting this list of usernames, we use the API to get ratings for each user in getratings.py. This generates a TSV file, users_final.tsv, for users with user id, username, and mean rating of their profile (which is computed here and stored for efficiency in the recommendation system later). It also generates a TSV file for ratings, ratings_final.tsv, with user id, MAL id, and rating (each anime has a MAL associated id that we make use of for convenience). Next, we get a TSV file for general info about each anime. We use idfromratings.py to get a list of unique MAL ids from the ratings TSV (with 30000 users, this covers basically all of the anime in the database).

Then we use getanimeinfo.py to create a TSV file, anime_info_final.tsv, which contains basic information such as title, MAL score, genres, episode count, image link, etc. We also keep track of any prequels these shows have for later use (we generally don't want sequels to be recommended). The final result (after many hours of retrieving data) are three files: users_final.tsv with ~32k users, ratings_final.tsv with ~4.8 million ratings, and anime_info_final.tsv with ~15k anime.

### *Database Storage and Queries for Recommendation*

To store this data and compute recommendations efficiently, we use a PSQL database. Data from TSV files can be copied to tables using the following commands:

*CREATE TABLE anime_info (mal_id INT PRIMARY KEY UNIQUE, title TEXT UNIQUE, season TEXT, genres TEXT[], type TEXT, episodes INT, score FLOAT, members INT, img TEXT, prequels_id INT[], is_sequel BOOLEAN);*

*COPY anime_info FROM 'D:/Documents/animequery/anime_info_final.tsv' DELIMITER '\t' CSV HEADER;*

*CREATE TABLE users (user_id INT PRIMARY KEY UNIQUE, username TEXT UNIQUE, mean_rating FLOAT);*

*COPY users FROM 'D:/Documents/animequery/users_final.tsv' DELIMITER '\t' CSV HEADER;*

*CREATE TABLE ratings (user_id INT, mal_id INT, rating FLOAT, PRIMARY KEY(user_id, mal_id));*

*COPY ratings FROM 'D:/Documents/animequery/ratings_final.tsv' DELIMITER '\t' CSV HEADER;*

After populating the database, we can finally start doing some recommendations. I am choosing

to do user-based collaborative filtering. The basic process for this is:

1. Take a user we want recommendations for and compute a cosine similarity between them and every other user
2. Take the most similar users and recommend anime that they rate high (relative to their mean rating).

We achieve this using custom PSQL queries which makes the process quick and efficient. In more detail:

1. Precompute a table for the cosine similarity denominator (magnitude of vector for each user). This only happens once since it is a permanent table - we update this each time a new user is added.

*CREATE TABLE normalize AS (SELECT user_id, SQRT(SUM(rating * rating)) AS magnitude FROM ratings GROUP BY user_id);*

2. Add the user that wants recommendations into the database. This is done in a hybrid of Python and PSQL in app.py (main Flask web app file) - this will be explained more in the Website Development section.

3. Compute the cosine similarity numerator (dot product between new user <user_id> and all existing users). To do this, we join two ratings tables together on shared anime ids and take the sum of all products of ratings for shared anime:

*CREATE TEMPORARY TABLE cosinedot AS (SELECT r1.user_id AS user_1, r2.user_id AS user_2, SUM(r1.rating * r2.rating) AS dot FROM ratings r1 INNER JOIN ratings r2 ON r1.mal_id = r2.mal_id AND r1.user_id = <user id> WHERE r2.user_id != <user id> GROUP BY user_1, user_2 ORDER BY dot DESC);*

4. Divide the numerator by denominator and get a table with the final similarities. We limit the nearest neighbors to 20.

*CREATE TEMPORARY TABLE cosinesim AS (SELECT cosinedot.user_1 AS user_1, cosinedot.user_2 AS user_2, cosinedot.dot / (n1.magnitude * n2.magnitude) AS sim FROM cosinedot INNER JOIN normalize AS n1 on cosinedot.user_1 = n1.user_id INNER JOIN normalize AS n2 on cosinedot.user_2 = n2.user_id ORDER BY sim desc LIMIT 20);*

5. We take the most similar users and compute the differences between their ratings and their mean rating; this gives a positive weight to anime that they rate higher than normal. We weight these differences based on how similar the user is to the original user and normalize, and create a table with a final deviation for each anime.

*CREATE TEMPORARY TABLE prediction_dev AS (SELECT c.user_1 as uid, r.mal_id, SUM(c.sim * (r.rating - u.mean_rating)) / SUM(c.sim) AS score_dev FROM cosinesim AS c INNER JOIN ratings AS r ON c.user_2 = r.user_id INNER JOIN users as u on c.user_2 = u.user_id GROUP BY uid, mal_id ORDER BY uid, score_dev DESC);*

6. We predict scores by taking the deviation for each anime and adding it to the mean rating for the original user (MAL scores are on a scale from 1-10, so we expect predictions to be around this range). We also add a small 10% weight to the actual MAL score, which helps filter out recommended shows with very low scores:

*SELECT p.mal_id, ((p.score_dev + u.mean_rating) * 0.90 + (a.score * 0.10)) AS predicted_score FROM prediction_dev p, users u, anime_info a*
*WHERE u.user_id = <user_id> AND a.mal_id = p.mal_id ORDER BY predicted_score DESC;*

This final step outputs a list of MAL ids and predicted scores for each, starting from the highest predicted score.

### *Website Development*

Now that recommendations can be computed, we would like a visually intuitive and appealing way to display the results, so I made a website in Flask (code base of Python and web hosting using Heroku) that stores the database and allows a user to enter their username to get recommendations (these files are contained in ‘website’ directory). The SQLAlchemy package is used to manipulate the PSQL database from the Python code (this is mostly used to execute the raw SQL queries previously mentioned). We query MAL for the user's profile and insert it into the database, and then make recommendations as previously described. Results from the queries are stored and processed in Python: we create 3 separate lists of 40 anime corresponding with three filters users can choose to manipulate their recommendation result display. Each of these lists (as well as one for prediction scores and user profile data) is then passed to an HTML template and parsed using the Jinja template engine, which allows us to create flexible and filterable recommendations that are easily displayed on the webpage, as well as "More Info" popups for each show and a link to the original MAL page.