

Andy Thai  
Haoyuan Wang

## **Write-up: Collaborative Filtering Anime Recommendation System**

---

We are currently using the Anime Recommendations database from Kaggle, found at:

<https://www.kaggle.com/CooperUnion/anime-recommendations-database>.

The dataset includes a list of 12,294 Japanese animes with their id, their name, type of film, average rating, and the number of members. The dataset contains a sample of 76,000 users in the user rating dataset, which includes the user-id, the anime-id, and user rating for the anime.

Some interesting statistics we found when we analyzed the dataset found that the top five most highest rated anime genres by mean are Thriller (6.81), Josei (6.67), Mystery (6.61), Police (6.56), and Shounen (6.46) in their respective orders. The lowest rated anime genres were dementia (4.50), music (5.33), yuri (5.5), kids (5.63), and hentai (5.64). The average rating across all genres is 6.12.

Analyzing for standard deviation, we found the genres with the highest standard deviations were Shounen Ai (1.64), Josei (1.56), Supernatural (1.51), Magic (1.49), and School (1.46). The genres with the lowest standard deviations are Hentai (0.90), Harem (0.98), Martial Arts (1.01), Yuri (1.05), and Ecchi (1.06).

An interesting trend we found within the user data was that most users typically don't rate the animes they watch. The ratings they did make often were either very high or very low; users did not do a lot of ratings for shows they considered mediocre.

Our predictive task will involve predicting / recommending what anime user is going to watch based on their history.

We will evaluate our algorithm by running it and comparing it against the baseline algorithm we have designed for this dataset. We expect a significant increase in accuracy from our model compared to the baseline. Our baseline algorithm involves taking the top two most popular genres of the user and recommend the top five animes from each the genres. This baseline assumes that users will most likely watch animes of similar genres and they will mostly watch the most popular animes. The predictive recommendations of the animes are only 30.68% accurate. This is calculated by first splitting the rating data by half: one for training and one for testing. We use the training set to get the most popular genres for each user. Then we take the top two genres and predictively recommend top ten animes between the two genres. We then cross check whether the user watched any of the animes recommended. This is to check whether the model correctly predicted an anime that user will watch.

We will test the validity of our predictions by scrambling our dataset; we take in the cosine similarities from a spliced half of the dataset labeled as training, and run predictions based off of the training set onto the testing set. The validation algorithm is similar to algorithm used for the basis. We will take the ten predictive recommendations and see how accurate they are. We check to see whether the predicted animes are watched by the user in the testing dataset. If any is in the set, we count the prediction as a success. We calculate the accuracy by the number of successful predictions divided by the total number of users. This is representative of our algorithm performance; our justification is that if one of our recommendations matches an anime on a user's watch history in the test set, the rest of the recommendations would be good indicators of animes they would like.

The features we used were the genres of each of the animes. We generated the features by first by reading the csv file using the pandas library. From there, we pushed all the data into a dictionary of list such that the key is the anime

id and the value is the list of genres. We then also compile a list of genre and sort them by alphabetical order. Then using both data set we encode the genres of each anime using one-hot-encoding. After much experimentation, we find that adding rating or the number of viewers to the feature vector do not significantly change the prediction accuracy.

We are using collaborative filtering model to predict the animes the users are most likely to watch. SVMs will not work because this is not a binary problem. One person may like an anime in a genre but it is also very likely that they will not like other animes in the same genre. This is why we need to use collaborative filtering. Collaborative filtering calculates the distance between different animes to get better predictive recommendations. It is true that matrix factorization is better because of its ability to give recommendations with missing data but because of the reasons explained in part three, we cannot use it. Therefore, the better choice for the model is to use collaborative filtering. Within collaborative filtering, we decided to use cosine similarity on all the animes. We chose this because cosine similarity is more accurate than Jaccard similarity and Pearson similarity. Between the choice of user similarity and anime similarity, we picked anime similarity because of it requires less of a demand on our hardware. With a better machine, we could have tested the accuracy of both approaches to cosine similarity.

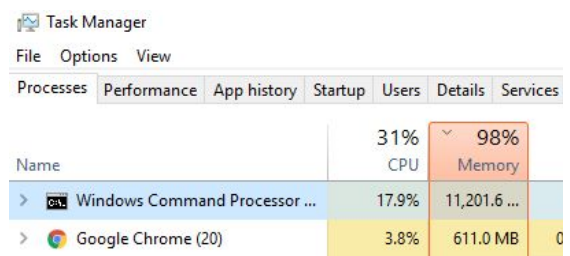
Our model is a collaborative filtering model using cosine similarity between different anime shows.

We decided to use this model because it offered the best middle ground between memory usage and theoretical performance. Our model may not necessarily be the most optimal overall; we believe that a matrix factorization model would give better potential results, but our cosine similarity model is the best choice given our hardware capabilities and the looming issue of scalability; an alternative would be using less

data, but that leads into the territory of our results not being statistically significant.

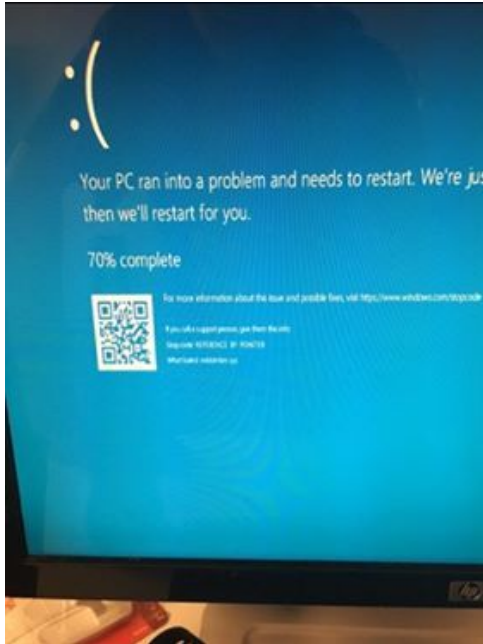
We decided to optimize this model by improving its performance during the its creation. We made sure that we used dictionaries to store our data for fast access. We also made sure that we are not recalculating cosine similarities as cosine similarity of  $x$  and  $y$  is the same as cosine similarity of  $y$  and  $x$ . We had done this by only calculating the value once and save it twice in a temporary dictionary. This way we can cut the run time by half. We then discovered that this significantly increased our memory usage, and leading to a throttling of calculation due to the lack of RAM. We then decided to free up memory of all the cosine similarities we do not need in the temporary dictionary. We also sorted the cosine similarity by value from greatest to least and only save the top ten values in order to save space and time. These optimization lead us from not being able to finish compiling the model to finishing the compilation in just over an hour.

We ran into significant scalability issues throughout the model design and processing. We attempted a matrix factorization model on our dataset, but it ended up using a significant amount of memory: upwards of ten gigabytes of RAM on our system. Running the model ended up blue-screening our high-end desktop computer.



The screenshot shows the Windows Task Manager Performance tab. The CPU usage is 31% and Memory usage is 98%. The table below shows the details of the processes running.

Name	CPU	Memory
Windows Command Processor ...	17.9%	11,201.6 ...
Google Chrome (20)	3.8%	611.0 MB



Our unsuccessful attempts were using the SVC, Tf-idf, and multi-layer perceptron. We waited for over three hours for the linearSVC model. Even with a reduced dataset, we found that the linearSVC would not train quickly enough, and thus would not give us proper results in time. Tf-idf was not good for our dataset, as our only text were found in the titles, which were generally short, and often consisted of both English and Japanese Romaji. Our current data would not give statistically significant results with Tf-idf. We also attempted using scikit-learn's multi-layer perceptron, but ran into the same problems as the linear SVC.

The strengths and weaknesses of a linear SVC allows for many different features to be accounted in a predictive algorithm. It is able to give weightings to certain features based on their importance. However it has a high runtime and is only able to give binary classifications by itself. Given our problem, getting predicted liked and not-liked anime using SVC as a recommender would not be a good choice, since they are more fitted for rote classification problems. For tf-idf, it works very well for gleaning features from text-rich datasets, but falls rather short when the datasets tend not to

have much text / string information in them. The multi-layer perceptron we found had similar strengths and weaknesses to the SVC in regards to this problem, with emphasis on the long training time.

Recommender systems in anime have been studied before in literature. In *Mangaki: an Anime/Manga Recommender System with Fast Preference Elicitation* ([http://jill-jenn.net/\\_static/works/mangaki-recsys2015.pdf](http://jill-jenn.net/_static/works/mangaki-recsys2015.pdf)), French computational researchers discuss a new system designed to profile the preferences of new users. Using data from MyAnimeList.com, they construct a simple preference elicitation process designed to get the maximum amount of information from new users.

We are currently using an existing dataset taken from MyAnimeList.com accounts which can be found at Kaggle. It has been used in different experiments by users; one example involved the relationship between show popularity and the number of episodes, and the consistency of rating across different genres. (<https://www.kaggle.com/msjgriffiths/exploratory-analysis>).

However, research in this problem isn't just confined to anime; it can encompass a variety of genres and films. A similar dataset that tackles these studies is the Netflix dataset as seen in the Netflix Prize competition (<https://www.kaggle.com/netflix-inc/netflix-prize-data>). Companies like Netflix have come to in attempts to optimize their recommender systems using competitions, which looks for the best performing recommender / prediction models and rewards them. Another similar dataset would be the Amazon Fine Food recommendation dataset, found here: <https://www.kaggle.com/qwikfix/amazon-recommendation-dataset>.

State-of-the-art systems tackling this recommendation problem would be the matrix factorization algorithm. This is currently the

industry standard for many companies; this is also a subset of part Netflix's recommender algorithm

(<http://ieeexplore.ieee.org/abstract/document/5197422/>).

Conclusions from other scholarly papers regarding state-of-the-art matrix factorization algorithms indicate that these systems work well with this type of data ([https://datajobs.com/data-science-repo/Recommender-Systems-\[Netflix\].pdf](https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf)). Collaborative filtering such as the algorithm we used with our dataset works with this type of data as well, but matrix factorization generally performs better; matrix factorization is more optimized for global data (maps relationship between user and product instead of products with each other) whereas our algorithm will perform better on local data (two similar interests and performing recommendations via nearest neighbor).

Our results from testing on the test set gave us matching recommendations of about at least about 56%.

On different tests, we had performances of 56.544%, 55.986%, 56.571%, 56.323%, and 56.193%.

```
In [40]: correct_pred=0
         found_movie=False
         for user in result:
             for recs in result[user]:
                 for x in recs:
                     if comparisonInvalid(x,user):
                         correct_pred+=1
                         found_movie=True
                         break
                 if found_movie:
                     found_movie=False
                     break
```

In [ ]:

```
In [43]: print(correct_pred/len(result))
         0.564464265742087
```

```
In [50]: print(correct_pred/len(result))
         0.5598565007398308
```

```
In [61]: print(correct_pred/len(result))
         0.5657113312745413
```

```
In [178]: print(correct_pred/len(result))
          0.5632301157006316
```

```
In [185]: print(correct_pred/len(result))
          0.5619284655611175
```

For the baseline, we had performances of 30.681%, 30.674%, 30.676%, 30.716%, and 30.720%

```
In [104]: correct_pred_basis=0
          for user in prediction_data:
              for recs in prediction_data[user]:
                  if comparisonInvalid(recs,user):
                      correct_pred_basis+=1
                      break
```

```
In [106]: print(correct_pred_basis/len(prediction_data))
          0.30680658999762445
```

```
In [139]: print(correct_pred_basis/len(prediction_data))
          0.30674204355108875
```

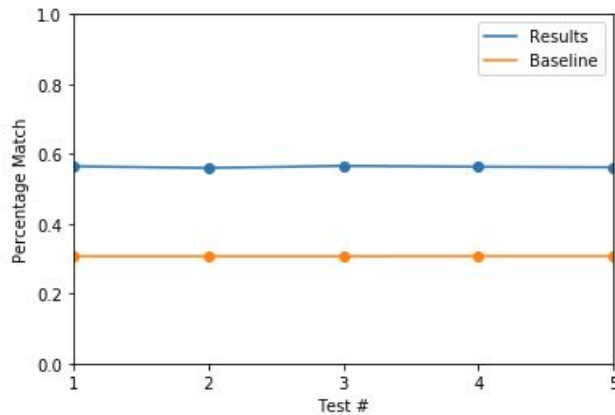
```
In [151]: print(correct_pred_basis/len(prediction_data))
          0.30676439790575916
```

```
In [161]: print(correct_pred_basis/len(prediction_data))
          0.3071615147155369
```

```
In [171]: print(correct_pred_basis/len(prediction_data))
          0.30719584258832405
```

In comparison to the baseline, we find that our results generally performed 26% better than the approximate 30% accuracy of the baseline. The significance of this tells us that the cosine similarity collaborative filtering model can work generally well in a dataset for recommendations given only user IDs, product IDs, and ratings. Given our current results and what we know about alternative models, we expect our results to potentially improve upon deploying a matrix factorization model.





From this, we conclude that our algorithm works rather decent on the dataset and could be considered a success. The baseline is concluded to perform rather mediocre in terms of recommendations.

The significance of the results imply a cosine collaborative filtering model is quite satisfactory in general for this given dataset, as at least one recommendation matches a previously watched history entry. In real practical usage outside of testing performance, we could filter out already watched anime found in user history to create a fully featured list of novel anime the user has yet to purview.

The feature we found that is useful are the genres of the anime. This is the case because only the genre category has a strong relationship between users and anime. The rating of the anime is a not a good feature because not many user review the anime and thus most animes watched by the user would not have a rating. Typically users tend to have ‘tastes’, and gravitate towards a specific genre for what that genre offers (example: action for action scenes and shounen for fighting scenes); ratings do not quite capture these tastes given that ratings can be distributed across different genres in varying amounts. There are also not a strong relationship between the types of anime such as TV or movie, because most animes in the dataset are TV shows. We also did not pick the user rating

dataset for the feature vector because of its massive size.

Our model’s parameters are represented by two numbers. The first parameter is a integer variable  $n$ , representing the  $n$ -th highest rated anime in the user history. This parameter is used to select the types of animes that specific user would typically gravitate towards. The second value is an integer,  $m$ , which is used to select the  $m$  highest cosine similarities out of the entire cosine similarity structure taken from those  $n$  animes. These cosine similarities will be used to recommend  $m$  likely animes the user would watch. We decided to ultimately pick  $n = 5$  and  $m = 2$  for our model parameters, which is  $n * m$ , or 10 total predictions for each user. We picked these parameters for our model because these numbers give us the highest accuracy under reasonable runtime and memory usage. We feel that ten is a reasonable number of recommendations to a user and any more or less would be unrealistic for a practical recommender system. In short, the first parameter  $n$  would be the amount of top animes the user has rated, and  $m$  would be the amount of recommendations to derive for each of the  $n$  animes.

We believe our proposed model succeeded in part due to our emphasis on capturing user browsing trends by genre preferences. As mentioned before, people usually have a specific taste for anime specifically in genres. Cosine similarity captures this distance between different animes based on their taste and genres. Since cosine similarity captures this aspect of users, we can confidently say that our recommendations will accurately reflect what the users will watch. Other models may fail because they fail to take user preference into consideration and rather instead rely more heavily on a user’s history of ratings, which performs more poorly in capturing general trends on user preferences in anime browsing.