Anime Recommender System

Ashwinkumar Ajithkumar Pillai,
Vatsal Thakkar

1 Introduction

The purpose of our recommender system is to recommend related animes based on the given anime. The input to our model is an anime and the output is the top 10 animes that are similar to it. We are using collaborative filtering to achieve this objective. The relevance is computed in two ways here:

1. Similarity between anime based on the ratings they have received

2. Similarity between the users who have rated these animes.

The first one is item-item CF and the second is user-user CF.

2 Dataset details

The dataset used for the model is Anime Recommendation Dataset - `https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020`

For each anime we have the following features:

- Genres that the anime fall into,
- English and Japanese name of the anime,
- Number of episodes it has,
- When it was premiered and aired,
- The producers, licensors, studios, and source,
- Duration, rating, popularity.
- Number of users who have the anime on Hold, are watching, completed, Dropped, or plan to watch,
- And finally all animes are rated from 1-10 like IMDB ratings and we have data on each anime that how many users have given a particular rating to it.
- For example 1000 have rated it 5 10k have rated it 8 and so on.

We also have data of each user mapped to each anime and what they have rated the anime: (user_id - anime_id):= rating

2.1 Dataset size

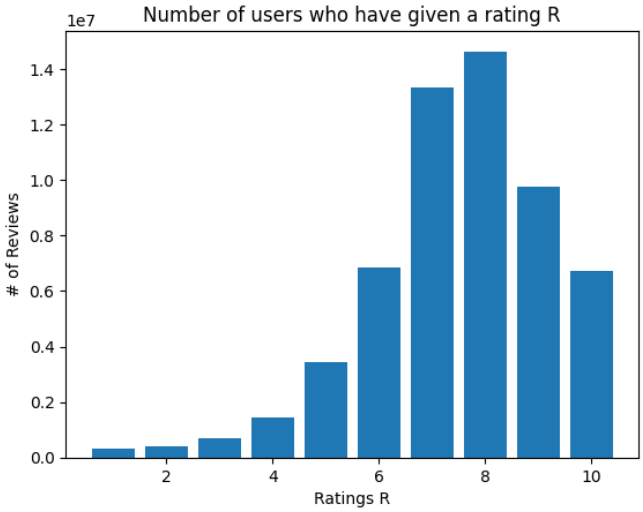
Number of anime in the dataset: 17558
Number of users in the dataset: 310059

2.2 Data Insights

1. User - Anime Rating snap

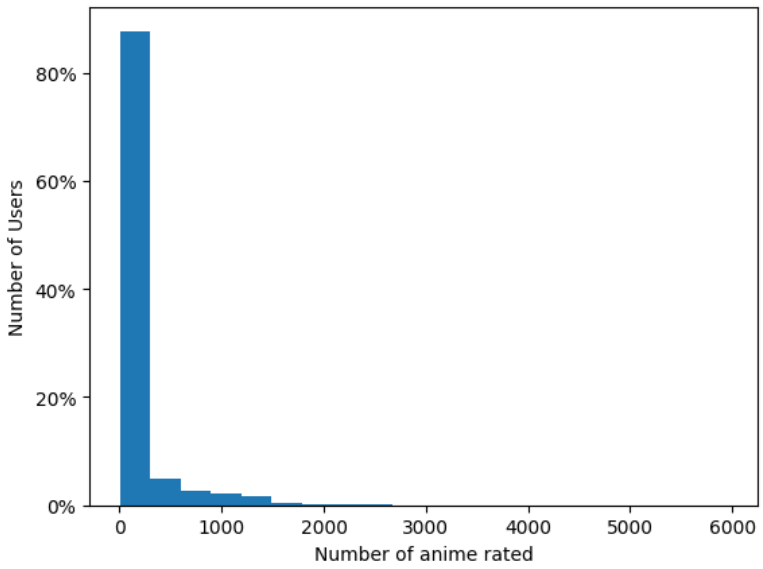
	user_id	anime_id	rating
0	0	430	9
1	0	1004	5
2	0	3010	7
3	0	570	7
4	0	2762	9
5	0	431	8
6	0	578	10
7	0	433	6
8	0	1571	10
9	0	121	9
10	0	356	9

2. item Number of users associated with a rating R



From the above visualization, it can be observed that most users have given a rating between 7 and 9. Very few proportion of the users have given a rating less than 5

3. Histogram of users and the number of animes they have rated.



Here we can see that most users (Almost 90%) have rated 1-200 animes
6k users have only rated 1 anime
2k have rated 5 animes
Most users have rated in the range of 1-200 as stated above among the 17k users
While there is one user who has rated 15k anime.
This data is shown in the below image

```
Name: user_id, Length: 310059, dtype: int64,
1        5952
2        3913
3        3335
4        3017
5        2662
         ...
2103        1
2108        1
2109        1
2111        1
15455       1
```

4. Animes are associated with more than one genre

	anime_id	Name	Genres
0	1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space
1	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space
2	6	Trigun	Action, Sci-Fi, Adventure, Comedy, Drama, Shounen
3	7	Witch Hunter Robin	Action, Mystery, Police, Supernatural, Drama, ...
4	8	Bouken Ou Beet	Adventure, Fantasy, Shounen, Supernatural
5	15	Eyeshield 21	Action, Sports, Comedy, Shounen
6	16	Hachimitsu to Clover	Comedy, Drama, Josei, Romance, Slice of Life
7	17	Hungry Heart: Wild Striker	Slice of Life, Comedy, Sports, Shounen
8	18	Initial D Fourth Stage	Action, Cars, Sports, Drama, Seinen
9	19	Monster	Drama, Horror, Mystery, Police, Psychological,...
10	20	Naruto	Action, Adventure, Comedy, Super Power, Martia...
11	21	One Piece	Action, Adventure, Comedy, Super Power, Drama,...
12	22	Tennis no Ouji-sama	Action, Comedy, Sports, School, Shounen
13	23	Ring ni Kakero 1	Action, Shounen, Sports
14	24	School Rumble	Comedy, Romance, School, Shounen
15	25	Sunabouzu	Action, Adventure, Comedy, Ecchi, Sci-Fi, Shounen
16	26	Texhnolyze	Action, Sci-Fi, Psychological, Drama
17	27	Trinity Blood	Action, Supernatural, Vampire
18	28	Yakitate!! Japan	Comedy, Shounen
19	29	Zipang	Action, Military, Sci-Fi, Historical, Drama, S...

3 Pre-processing

1. Remove all anime and users that have null values - animes with no rating or users who have not rated any anime will be removed. Our dataset didn't have any anime or users with such conditions.

```
print("Anime missing values (%):\n")
print(round(anime_selected_column.isnull().sum().sort_values(ascending=False)/len(anime_selected_column.index),4)*100)

print("Rating missing values (%):\n")
print(round(rating_data.isnull().sum().sort_values(ascending=False)/len(rating_data.index),4)*100)

anime_selected_column.info()
rating_data.info()
```

```
Anime missing values (%):

anime_id    0.0
Name        0.0
Genres      0.0
dtype: float64
Rating missing values (%):

user_id     0.0
anime_id    0.0
rating      0.0
dtype: float64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17562 entries, 0 to 17561
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   anime_id  17562 non-null  int64
 1   Name      17562 non-null  object
 2   Genres    17562 non-null  object
dtypes: int64(1), object(2)
memory usage: 411.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57633278 entries, 0 to 57633277
Data columns (total 3 columns):
...
 1   anime_id  int64
 2   rating    int64
dtypes: int64(3)
memory usage: 1.3 GB
```

2. Filtering users with less than 400 reviews

```
counts = rating_data['user_id'].value_counts()
print(rating_data.shape)
selected_rating_data = rating_data[rating_data['user_id'].isin(counts[counts >= 400].index)]
selected_rating_data
```

✓ 1.1s

This is necessary as the users who have extensively rated will have more similarities. We don't want to take users who have made very few ratings as that doesn't represent the popular rating on the anime. This will help in giving more generalized data and will focus on popularity among anime fans. The reason for selecting 400 was that we checked with other values but 400 seemed reasonable among them. Ideally, we should have gone with 200 but our processor doesn't have the computing power needed for that.
This reduced the number of anime to 16844.
And the number of users to 35292.

4 Item-Item Similarity

4.1 Anime - User interaction matrix

We first create the anime user interaction matrix. It will have the following dimensions: 16844 x 35292 (anime x users)

user_id	17	19	42	53	73	111	112	121	145	146	...	353304	353311	353318	353325	353326	353328	353357	353365	353395	353398
Name																					
"0"	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"Aesop" no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	6.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"Bungaku Shoujo" Memoire	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	7.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
"Bungaku Shoujo" Movie	NaN	NaN	NaN	NaN	NaN	NaN	NaN	10.0	NaN	6.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 35292 columns

4.2 Normalizing the matrix

Now we carry out the processing of the interaction matrix in 4 stages:
As we can see there are a lot of NaN values so we need to remove those and also normalize the matrix.

pivot_normalized = anime_pivot.apply(lambda x: (x-np.mean(x))/(np.max(x) - np.min(x)), axis=0)
We subtract the average rating from each rating and divide it by the difference between the max and min ratings.

Replacing NaN values with 0:
pivot_normalized.fillna(0, inplace=True)

The anime that has no rating will be removed:
pivot_n1 = pivot_normalized.loc[:, (pivot_normalized != 0).any(axis=0)]

Generate CSR matrix: - since we need to efficiently process sparse matrix:
piv_sparse1 = csr_matrix(pivot_n1.values)

4.3 Cosine Similarity

We use cosine similarity as the similarity function
anime_similarity = cosine_similarity(piv_sparse1)

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

4.4 Item-Item Similarity Matrix

Name	"0"	"Aesop" no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu	"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	"Bungaku Shoujo" Memoire	"Bungaku Shoujo" Movie	"Calpis" Hakkou Monogatari	"Eiji"	"Eikou Naki Tensai-tachi" Kara no Monogatari	"Eiyuu" Kaitai	"Kinako" Movie x Mameshiba	...	s.CRY.ed Alteration II: Quan	the FLY BanD!	xxxHOLiC
Name														
"0"	1.000000	0.019629	0.019226	-0.034730	-0.023543	0.113802	0.059813	0.043338	0.053125	0.067831	...	0.010449	0.014687	-0.046925
"Aesop" no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu	0.019629	1.000000	0.012524	0.019919	0.019815	-0.017112	0.037176	0.269131	0.047603	0.120686	...	0.018466	0.138033	-0.008111
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	0.019226	0.012524	1.000000	0.333865	0.206926	-0.008162	0.016483	0.017802	0.028743	-0.005315	...	0.014634	-0.008421	-0.016882
"Bungaku Shoujo" Memoire	-0.034730	0.019919	0.333865	1.000000	0.432366	-0.030120	-0.009535	-0.001095	-0.017852	-0.014404	...	0.011854	-0.045977	0.042004
"Bungaku Shoujo" Movie	-0.023543	0.019815	0.206926	0.432366	1.000000	-0.027846	-0.015293	0.006716	-0.012666	-0.017945	...	-0.003975	-0.021358	0.057875
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
xxxHOLiC Rou	-0.049986	-0.008854	-0.017029	0.051888	0.061047	-0.030859	-0.012337	-0.008287	-0.036898	-0.015910	...	-0.017195	-0.029839	0.524379
xxxHOLiC Shunmuki	-0.042913	0.017441	-0.010669	0.036106	0.059589	-0.023758	-0.007602	0.006828	-0.031461	-0.006329	...	-0.020072	-0.014253	0.560471
Üks Uks	0.035668	0.215527	-0.017963	-0.022274	-0.022313	-0.014718	0.048187	0.233057	0.051114	0.184635	...	0.012670	-0.022130	-0.018387
ëlDLIVE	0.014683	0.008883	0.020272	-0.018467	-0.022279	-0.003410	0.012999	0.002929	0.042755	0.011171	...	-0.011583	-0.000297	-0.083259
○	0.174267	0.036992	-0.014381	-0.034241	-0.024532	0.155997	0.023971	0.032687	0.040794	0.048053	...	-0.000351	0.097067	-0.044620

16842 rows × 16842 columns

We use the cosine similarity between the animes to build the (anime x anime) similarity matrix

5. Item-Item Recommender System

```
def anime_recommendation(anime_name):

    number = 1
    print(f'Similar Anime as {anime_name}\n')
    for anime in ani_sim.sort_values(by = anime_name, ascending = False).index[1:11]:
        print(f'#{number}: {anime}, {round(ani_sim[anime][anime_name]*100,2)}% match')
        number +=1
```

Now that we have built the model, let's try to run it for the anime - Naruto.

```
    anime_recommendation('Naruto')
```

```
Similar Anime as Naruto

#1: Naruto: Shippuuden, 43.32% match
#2: Bleach, 30.72% match
#3: Dragon Ball Z, 25.05% match
#4: Dragon Ball, 21.12% match
#5: Shingeki no Kyojin, 20.92% match
#6: Death Note, 20.16% match
#7: Naruto: Shippuuden Movie 1, 20.15% match
#8: The Last: Naruto the Movie, 20.14% match
#9: Shingeki no Kyojin Season 2, 20.09% match
#10: Fairy Tail, 20.07% match
```

As we can see we have the top 10 recommendations for the keyword Naruto.
The first result is Naruto Shippuden itself.
Bleach is the second result and is relevant since Naruto and Bleach are among two of "The Big Three" animes.
Dragon Ball and Death Note are also earlier animes with the protagonist character being very powerful. This is also true for Shingeki no Kyojin.
Apart from the story and genre, these animes are all popular too so they fall into the same category and these results are indeed relevant.

Now we build the second Collaborative filtering model in the next section:

5 User-User Similarity

5.1 Anime-User Interaction matrix and Normalization

We again use the anime-user interaction matrix we built in the first step of the item-item similarity method.
This time we only subtract the average rating from each rating for normalization and we take the transpose of the matrix since we need to build the user-user matrix:

```
# We will again use the anime - user interaction matrix to generate a new table.
# Here we are subtrating every rating with the average rating
pivot_normalized2 = anime_pivot.apply(lambda x: x-np.mean(x), axis=0)

# Replace NaN values with 0
pivot_normalized2.fillna(0,inplace=True)

# Transpose the generated matrix since we need user-user matrix
pivot_T = pivot_normalized2.T

# remove item/user with no rating
pivot_n2 = pivot_T.loc[:, (pivot_T != 0).any(axis=0)]

# Again use CSR
piv_sparse2 = csr_matrix(pivot_n2.values)
```

5.2 Cosine Similarity

Same as item-item, for user-user similarity we use cosine similarity as the similarity function:

user_similarity = cosine_similarity(piv_sparse2)

5.3 User-User Similarity Matrix

user_id	17	19	42	53	73	111	112	121	145	146	...	353304	353311	353318	353325	35
user_id																
17	1.000000	0.037767	0.080325	0.030698	0.055056	0.057561	0.057338	0.022757	0.084999	0.015857	...	0.047401	0.061084	0.072863	0.024033	0.1
19	0.037767	1.000000	0.016123	0.099996	0.075880	0.077153	0.098907	0.040946	0.129734	0.049713	...	0.060356	0.034499	0.091492	0.008459	0.1
42	0.080325	0.016123	1.000000	0.005624	0.060814	0.024837	-0.020126	0.052903	0.046098	0.065828	...	0.000550	0.023911	0.029123	0.005293	0.0
53	0.030698	0.099996	0.005624	1.000000	0.036718	0.063260	0.067903	0.020530	0.110362	0.035104	...	0.106550	0.024443	0.134707	-0.004038	0.0
73	0.055056	0.075880	0.060814	0.036718	1.000000	0.103762	0.054390	0.030773	0.157697	0.067604	...	0.093931	0.050250	0.141546	-0.020461	0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
353328	0.021505	0.032581	0.065050	0.058045	0.054865	0.029460	0.218825	0.049385	0.051520	0.009922	...	0.151397	0.020033	0.037698	0.015357	0.0
353357	0.035223	0.060215	-0.006013	0.046960	0.056085	0.044851	0.194659	0.051697	0.079330	0.028946	...	0.136288	0.021487	0.054429	-0.000195	0.0
353365	0.083929	0.060610	0.025882	0.059194	0.110818	0.062240	0.070891	0.031503	0.062798	0.006920	...	0.041770	0.089343	0.121250	0.006421	0.1
353395	0.063163	0.083465	0.028632	0.062203	0.120648	0.053277	0.117558	0.054902	0.135079	0.053972	...	0.103398	0.105004	0.145413	0.003265	0.1
353398	0.029151	0.083153	-0.017561	0.026474	0.061924	0.006748	0.062215	0.049445	0.065308	0.054397	...	0.053604	0.138479	0.152962	0.002377	0.1

35292 rows × 35292 columns

5.4 Finding anime based on similar users

We will find 10 users that have the most similar tastes for each of the users. For example, let's check for one user- here we are taking a user with id 17 and we find the top 10 users that have similar tastes.
Then we will look at the animes that those 10 users have watched and rated. We remove the animes

that user_id 17 has watched from that anime list.

Now we have new animes for user 17 that the user has not watched. From these animes using user_id 17 and the 10 users, we will predict the ratings for each of the new animes.

Then we will suggest the top 10 animes from this list to the user

```
# Find 10 users that have similar taste to 17
userid = 17
Number_of_similar_Users = 10
threshold = 0.1
similar_users_to_userid = user_sim_df[user_sim_df[userid] >threshold][userid].sort_values(ascending=False)[1:Number_of_similar_Users+1]
similar_users_to_userid
```

Users that are similar to user 17:

```
user_id
279757    0.204531
70022     0.200061
74759     0.199011
20468     0.196432
259986    0.194538
156246    0.194157
26049     0.192727
169945    0.191587
115315    0.188445
251874    0.187222
Name: 17, dtype: float64
```

Extract the animes user_id 17 has watched:

```
# Animes that the target user has watched
picked_userid_watched = pivot_n2[pivot_n2.index == userid].loc[:, (pivot_n2[pivot_n2.index == userid] != 0).any(axis=0)]
picked_userid_watched
```

Remove animes that none of the similar users have watched

```
# Remove animes that none of the similar users have watched
similar_user_df = pivot_n2[pivot_n2.index.isin(similar_users_to_userid.index)]
similar_user_movies = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]
similar_user_movies
```

Remove Animes that the selected user has watched:

```
s1 = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]
similar_user_movies2 = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]
#remove movies that picked user has watched
s1.drop(picked_userid_watched.columns,axis=1, inplace=True, errors='ignore')

similar_user_movies2.drop(s1.columns,axis=1,inplace = True,errors= 'ignore')
similar_user_movies2
```

Remove the watched anime from the anime list

```
# Remove the watched movie from the movie list
similar_user_movies.drop(picked_userid_watched.columns,axis=1, inplace=True, errors='ignore')
```

Now that we have the list of new animes to suggest to user we will predict the rating (anime_score) for each anime and then select the top 10 rated animes from this new anime list

```
item_score = {}

for i in similar_user_movies.columns:
  movie_rating = similar_user_movies[i]
  total = 0
  total_sim = 0
  for u in similar_users_to_userid.index:
    if movie_rating[u] !=0:
      score = similar_users_to_userid[u]*movie_rating[u]
      total += score
      total_sim += similar_users_to_userid[u]
  item_score[i] = total / total_sim

item_score = pd.DataFrame(item_score.items(),columns=['Anime','Anime_score'])

ranked_item_score = item_score.sort_values(by='Anime_score',ascending=False)
ranked_item_score.head(10)
```

The results are shown in the below diagram:

	Anime	Anime_score
1517	The Embryo Develops into a Fetus	3.685882
86	Ano Hi Mita Hana no Namae wo Bokutachi wa Mada...	3.685882
1489	Tengen Toppa Gurren Lagann Movie 2: Lagann-hen	3.685882
1158	Persona 3 the Movie 4: Winter of Rebirth	2.782435
609	Hunter x Hunter: Original Video Animation	2.685882
1112	Ongaku Shoujo	2.685882
906	Mahou Shoujo Madoka★Magica Movie 3: Hangyaku n...	2.588735
1344	Shelter	2.539135
324	Digimon Adventure tri. 1: Saikai	2.539135
990	Monogatari Series: Second Season	2.528700

we add the average rating to these new anime since we removed it earlier in the matrix
If we hadn't removed the average rating earlier for normalization, then we should add the actual rating to the score and then predict the rating

	Anime	Anime_score	predicted_rating
1517	The Embryo Develops into a Fetus	3.685882	10.0
86	Ano Hi Mita Hana no Namae wo Bokutachi wa Mada...	3.685882	10.0
1489	Tengen Toppa Gurren Lagann Movie 2: Lagann-hen	3.685882	10.0
1158	Persona 3 the Movie 4: Winter of Rebirth	2.782435	9.0
609	Hunter x Hunter: Original Video Animation	2.685882	9.0
1112	Ongaku Shoujo	2.685882	9.0
906	Mahou Shoujo Madoka★Magica Movie 3: Hangyaku n...	2.588735	9.0
1344	Shelter	2.539135	9.0
324	Digimon Adventure tri. 1: Saikai	2.539135	9.0
990	Monogatari Series: Second Season	2.528700	9.0

We do this for all the users:

```
def similar_users(userid, Number_of_similar_Users=10):
  return user_sim_df[user_sim_df[userid] >threshold][userid].sort_values(ascending=False)[1:Number_of_similar_Users+1]

def recommend_movies(userid):
  #similar users to user id
  similar_users_to_userid = similar_users(userid)
  # Movies that the target user has watched
  picked_userid_watched = pivot_n2[pivot_n2.index == userid].loc[:, (pivot_n2[pivot_n2.index == userid] != 0).any(axis=0)]
  # Remove movies that none of the similar users have watched
  similar_user_df = pivot_n2[pivot_n2.index.isin(similar_users_to_userid.index)]
  similar_user_movies = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]

  #remove movies that picked user has watched
  similar_user_movies.drop(picked_userid_watched.columns,axis=1, inplace=True, errors='ignore')

  item_score = {}

  for i in similar_user_movies.columns:
    movie_rating = similar_user_movies[i]
    total = 0
    total_sim = 0
    for u in similar_users_to_userid.index:
      if movie_rating[u] !=0:
        score = similar_users_to_userid[u]*movie_rating[u]
        total += score
        total_sim += similar_users_to_userid[u]
    item_score[i] = total / total_sim

  item_score = pd.DataFrame(item_score.items(),columns=['Movie','Movie_score'])

  ranked_item_score = item_score.sort_values(by='Movie_score',ascending=False)

  # Average rating for the picked user
  avg_rating = anime_pivot[userid].mean()

  ranked_item_score['predicted_rating'] = np.floor(ranked_item_score['Movie_score'] + avg_rating)
  return ranked_item_score.head(10)
```

Let's try for user 346: recommend_user(346):=

	Movie	Movie_score	predicted_rating
2305	Senki Zesshou Symphogear XV	2.755003	10.0
806	Gintama°	2.732740	10.0
2989	xxxHOLiC Rou	2.620557	10.0
1154	Itazura na Kiss	2.620557	10.0
1341	Kimi no Suizou wo Tabetai	2.363988	10.0
1340	Kimi no Na wa.	2.347690	10.0
2761	Uchuu Senkan Yamato 2199	2.078303	10.0
2607	Tenki no Ko	2.051540	10.0
1833	No Game No Life: Zero	2.018830	10.0
2413	Shouwa Genroku Rakugo Shinjuu: Sukeroku Futata...	2.007693	10.0

The above result is for user 346.

We can also check for user 20468

	Movie	Movie_score	predicted_rating
2181	One Piece: Episode of East Blue - Luffy to 4-n...	3.402399	10.0
934	Gintama°	2.886381	10.0
918	Gintama Movie 2: Kanketsu-hen - Yorozuya yo Ei...	2.695294	10.0
1843	Major: World Series	2.665767	10.0
2166	One Outs	2.665767	10.0
1838	Major S3	2.665767	10.0
2475	Rurouni Kenshin: Meiji Kenkaku Romantan - Tsui...	2.613551	10.0
3003	Toaru Kagaku no Railgun T	2.593085	10.0
923	Gintama.	2.556730	10.0
1244	Hunter x Hunter (2011)	2.542195	10.0

5.5 Splitting the dataset

```
pivot_n2_train,pivot_n2_test = train_test_split(pivot_n2,test_size=0.3)
```

We split the dataset into 70-30. 70% split goes to training and the remaining 30% to test data.

Training set will have 24704 rows × 16838 columns (users x anime)

Name	"0"	"Aesop" no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu	"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	"Bungaku Shoujo" Memoire	"Bungaku Shoujo" Movie	"Calpis" Hakkou Monogatari	"Eiji"	"Eikou Naki Tensai-tachi" Kara no Monogatari	"Eiyuu" Kaitai	"Kinako" Movie x Mameshiba	...	s.CRY.ed Alteration II: Quan	the FLY BanD!	xxxHOLiC	xxxHOLiC Kei	xxxHOLiC Movie Manatsu no Yoru no Yume
user_id																
314016	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000
16355	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000
276499	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000
248493	0.0	0.0	1.106230	1.106230	0.106230	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	2.106230	1.106230	0.000000
76328	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298307	0.0	0.0	-0.754266	-0.754266	-0.754266	0.0	0.0	0.0	0.0	0.0	...	-0.754266	0.0	0.245734	0.000000	0.000000
163193	0.0	0.0	0.000000	0.000000	0.413216	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	2.413216	2.413216	0.000000
145592	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000
72791	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	-0.156055	0.843945	-1.156055
121950	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.0	0.000000	0.000000	0.000000

24704 rows × 16838 columns

Testing set will have 10588 rows × 16838 columns (users x anime)

Name	"0"	"Aesop" no Ohanashi yori: Ushi to Kaeru, Yokubatta Inu	"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi	"Bungaku Shoujo" Memoire	"Bungaku Shoujo" Movie	"Calpis" Hakkou Monogatari	"Eiji"	"Eikou Naki Tensai-tachi" Kara no Monogatari	"Eiyuu" Kaitai	"Kinako" Movie x Mameshiba	...	s.CRY.ed Alteration II: Quan	the FLY BanD!	xxxHOLiC	xxxHOLiC Kei	xxxH M Ma no no
user_id																
28535	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	-1.050000	1.950000	0.00
312416	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	3.424217	2.424217	3.4
342820	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
183388	0.0	0.0	-0.793245	0.0	-0.793245	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.206755	1.206755	1.20
314608	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
45009	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
14995	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
254401	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
95730	0.0	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00
183244	0.0	0.0	0.000000	0.0	0.201439	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.000000	0.00

10588 rows × 16838 columns

5.6 Accuracy of prediction

Now we want to check how accurate our prediction is: For this, we find ratings for animes based on similarity for the users that have already rated it.

```
def find_rating_for_same_movies(userid):
  #similar users to user id
  similar_users_to_userid = similar_users(userid)
  # Movies that the target user has watched
  picked_userid_watched = pivot_n2[pivot_n2.index == userid].loc[:, (pivot_n2[pivot_n2.index == userid] != 0).any(axis=0)]
  # Remove movies that none of the similar users have watched
  similar_user_df = pivot_n2[pivot_n2.index.isin(similar_users_to_userid.index)]
  s1 = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]
  similar_user_movies = similar_user_df.loc[:,(similar_user_df !=0).any(axis=0)]
  #remove movies that picked user has watched
  s1.drop(picked_userid_watched.columns,axis=1, inplace=True, errors='ignore')

  similar_user_movies.drop(s1.columns,axis=1,inplace = True,errors= 'ignore')

  item_score = {}

  for i in similar_user_movies.columns:
    movie_rating = similar_user_movies[i]
    total = 0
    total_sim = 0
    for u in similar_users_to_userid.index:
      if movie_rating[u] !=0:
        score = similar_users_to_userid[u]*movie_rating[u]
        total += score
        total_sim += similar_users_to_userid[u]
    item_score[i] = total / total_sim

  item_score = pd.DataFrame(item_score.items(),columns=['Anime','Anime_score'])

  ranked_item_score = item_score#item_score.sort_values(by='Anime_score',ascending=False)

  # Average rating for the picked user
  avg_rating = anime_pivot[userid].mean()

  ranked_item_score['predicted_rating'] = ranked_item_score['Anime_score'] + avg_rating
  return ranked_item_score
```

Predictions for anime ratings:

	Anime	Anime_score	predicted_rating
0	11eyes	-0.786821	7.355063
1	11eyes: Momoiro Genmutan	-1.174835	6.967049
2	3-gatsu no Lion	-0.161067	7.980817
3	30-sai no Hoken Taiiku	-1.492752	6.649132
4	Aa! Megami-sama! (TV)	0.663970	8.805854
...	...	...	...
673	Zetman	-0.347253	7.794631
674	Zetsuen no Tempest	1.193634	9.335518
675	Zombie-Loan	-0.979752	7.162132
676	Zombie-Loan Specials	-0.364583	7.777301
677	ef: A Tale of Memories.	-0.607862	7.534022

678 rows × 3 columns

Ground Truth: Actual rating of animes:

```
array([ 7.,  6.,  8.,  8.,  9.,  9.,  8.,  9.,  9.,  7.,  7., 10.,  8.,
        9.,  8.,  9.,  8.,  9.,  9.,  8.,  9.,  8.,  9.,  9.,  9.,  9.,
        8.,  8.,  7.,  7.,  9., 10.,  6.,  9., 10.,  8.,  9.,  9.,  8.,
        8.,  7.,  7.,  9.,  9.,  8.,  7., 10.,  8.,  7.,  7.,  8., 10.,
       10.,  7.,  7.,  8.,  8.,  9.,  6., 10., 10.,  8.,  8.,  8.,  9.,
        8.,  9.,  8.,  9., 10.,  9.,  8.,  8.,  8.,  6.,  6.,  8.,  7.,
        7., 10.,  8.,  8.,  9.,  9.,  7.,  9.,  8.,  8.,  8.,  9.,  8.,
       10., 10., 10., 10., 10., 10.,  9.,  7.,  7.,  8., 10.,  7., 10.,
        7.,  9.,  8.,  8.,  5.,  8.,  8.,  7.,  9., 10.,  8., 10.,  8.,
        9.,  9., 10.,  5., 10.,  9.,  9.,  7.,  7.,  8.,  5.,  8.,  8.,
        7.,  6.,  6., 10.,  8.,  7.,  7.,  8.,  8.,  8.,  8.,  8.,  8.,
        8.,  8.,  7.,  8.,  7.,  8.,  9.,  8.,  8.,  7., 10.,  5.,  8.,
        8.,  8., 10.,  7.,  7., 10.,  8.,  9.,  8., 10.,  8.,  9.,  8.,
        8.,  8.,  8.,  8.,  8.,  8.,  6., 10., 10.,  8.,  8.,  9.,  9.,
        6.,  8.,  7., 10.,  6., 10.,  7., 10.,  8.,  8.,  8.,  9.,  9.,
        9.,  8., 10., 10.,  9.,  6.,  9.,  8.,  7.,  7., 10., 10.,  9.,
        8.,  9.,  9.,  9.,  9.,  8., 10.,  7.,  7.,  8.,  7.,  9.,  8.,
       10.,  8.,  8.,  6., 10.,  8.,  8.,  8.,  9., 10.,  7.,  8.,  7.,
        6.,  8.,  7.,  7.,  6.,  7.,  7.,  6.,  9., 10.,  9.,  8.,  6.,
        7.,  8.,  9.,  8.,  9., 10.,  9.,  9., 10.,  7.,  8.,  7.,  6.,
       10.,  7.,  9.,  9., 10.,  9.,  7.,  8.,  9.,  7.,  8.,  8.,  7.,
        8.,  6.,  6.,  7.,  8.,  8.,  8., 10.,  8.,  9.,  8.,  9., 10.,
        9.,  8.,  8.,  9.,  9.,  7.,  6., 10., 10.,  7.,  7., 10.,  6.,
        8.,  8.,  9.,  8.,  8.,  9.,  8.,  9.,  8.,  9.,  9.,  6.,  7.,
        9.,  8.,  8.,  8.,  7.,  4., 10., 10.,  7., 10.,  7.,  7.,  5.,
...
        8., 10., 10., 10., 10.,  7.,  9.,  9.,  8.,  9.,  9., 10.,  7.,
        9.,  7.,  8.,  8.,  7.,  9.,  6.,  6., 10., 10.,  7.,  8.,  7.,
        8.,  7.,  8.,  9.,  9.,  8.,  9.,  9.,  7.,  9.,  9.,  8.,  7.,
        7.,  8.,  9.,  8.,  7., 10.,  8., 10., 10., 10.,  7.,  9.,  7.,
        7.,  7.])
```

Model Prediction: Rating predicted by the model

```
array([ 7.35506322,  6.96704893,  7.98081725,  6.64913249,  8.80585376,
        8.07752469,  6.76356816,  9.42393846,  9.31978011,  7.1599589 ,
        8.81192637,  8.46166279,  7.52286858,  8.24715586,  7.29383623,
        7.48038999,  8.18892118,  8.143561  ,  8.37250561,  8.37250561,
        7.10754946,  7.7816861 ,  7.68926507,  9.23752421,  6.8683993 ,
        9.57574895,  7.09547652,  8.06218113,  7.69309976,  5.12082305,
        9.64824648,  8.25486726,  8.07150306,  7.10754946,  9.79573939,
        9.40436929,  8.82856631,  9.32589918,  8.94982073,  9.21401935,
        8.08585483,  7.80207632,  9.53108163,  8.52628971,  8.81957337,
        6.97230784,  8.8521086 ,  6.26576292,  5.66602107,  7.94298477,
        8.00978333,  8.82503346,  8.90950779,  6.99217166,  7.62738781,
        8.64596553,  8.25890282,  9.00371696,  7.34280603,  8.72532443,
        8.0784564 ,  8.74207295,  9.05908099,  8.47112078,  9.41181808,
        9.62192091,  9.91543044,  9.88599034,  7.80681115, 10.03372588,
        8.74006908,  8.31645319,  8.31645319,  8.31645319,  5.90592829,
        7.31645319,  7.94913183,  8.87243383,  7.50930389,  9.32065876,
        8.61820023,  9.07145617,  8.71060388,  9.4516731 ,  8.13726698,
        9.18106892,  8.47671591,  8.5802328 ,  9.05192015,  8.91607676,
        8.4337966 ,  8.88025286,  8.88025286,  8.88025286,  8.88025286,
        8.51075498,  9.26306874,  9.24328893,  8.52381532,  8.18270331,
        7.64225867,  6.10754946,  8.29194663,  9.13576496,  7.4556456 ,
        8.24178856,  8.59411521,  8.9473128 ,  7.67457761,  8.27216838,
        7.61542031,  8.48944605,  7.84801635,  9.98282085,  9.47360126,
       10.32295762,  9.07079515,  8.62909512,  8.1599589 ,  9.73639168,
        6.67281371, 10.05286382,  8.40361736,  5.80681115,  5.4881836 ,
...
        8.49311725,  8.64035017,  9.23716302,  7.57104652,  8.73028032,
        8.04432247,  8.13362297,  6.38230976,  7.97184642,  8.10754946,
        9.10754946,  8.10319724,  7.58803807,  9.14808826,  6.47518109,
        9.31200142,  9.37397291,  9.7431161 ,  7.79463133,  9.33551833,
        7.16213211,  7.77730089,  7.53402194])
```

Comparing the above two, we can see that the predictions are very close.

Various Error Metrics:

```
Mean absolute error = 0.86
Mean squared error = 1.22
Median absolute error = 0.74
```

We Find the mean squared error across all users:

```python
def average_mean_square_error_test():
  accuracy  = 0
  count = 500
  for i in pivot_n2_test.index[:500]:
    temp = find_rating_for_same_movies(i)
    y_true = anime_pivot.loc[anime_pivot.index.isin( temp['Anime'])][i].values
    y_pred = np.floor(temp['predicted_rating'].values)
    if y_true.shape[0] > 0:
      accuracy += mean_squared_error(y_true,y_pred)

  return accuracy / count
```

Mean Squared Error:

```
average_mean_square_error_test()
# 1.7069004155138134
```

This concludes the user-user collaborative filtering model with a mean squared error of 1.7069004155138134

6 Future Scope

Using click-through rate (CTR) we can get real-time data on how the users select anime and improve our recommendations.

7 Conclusion

Our Model successfully recommended animes based on anime similarity and user similarity based on ratings.

The item-item CF recommended anime based on the similarity between the anime. and user-user-suggested animes based on similar users' tastes.

We also included the use case of recommending animes that the user has not watched yet instead of directly returning the top 10 results. And for testing accuracy, we used Mean-Squared-Error and got a satisfactory result of 1.7