Mobility-Based Recommendation System

Problem Statement

Human nature prefers change in daily life, and this is true for almost everyone, regardless of their interests. Everyone likes and dislikes different aspects of a single domain, and this is especially true for the OTT platform. These likes and dislikes also change with the current situation. Similarly, we prefer different genres at different times. Sometimes, we enjoy watching comedy, but not always. Our mood influences what we watch, and our mood is significantly influenced by the environment or location in which we find ourselves at the time. OTT systems typically recommend things based on user and content preferences, similarities, etc. Thus, there should be a system that incorporates a person's mobility information while recommending series/movies.

Goal

Recommend suitable content according to mobility type i.e home, outdoor, traveling etc

Approach

When we talk about recommendation systems, we have two main paradigms: collaborative filtering and content based filtering. A brief description of both methods are given below:G
Collaborative technique records the users' interaction with the items and uses this information for recommendation after finding similar users and/or items. For example rating given by a user to movie or time spent by reader on an article etc
Content based filtering finds the similar contents and similar users. For example, if User U likes movie M of genre G, the system will recommend M2 of genre G to user U.

Modern recommendation systems use both methods, we will also combine both and use a hybrid approach to tackle faced problems. There are many other approaches like recommending current popular products or usage of external source information to recommend a product to a user. Especially, social media giants using these techniques to enhance and improve user experience. We will be using a hybrid approach to implement this challenge, as we need both content similarities and what were the interactions between users and their series. In a content based method, we are utilizing cosine similarity function while for collaborative approach we are using matrix factorization technique. Matrix factorization is one of the algorithms of collaborative filtering which became very popular during the Netflix Prize Challenge [1] due to its effectiveness. Our main focus for this research is to incorporate mobility information of users in recommendation.

There are two ways to tackle this specific problem:

1. Incorporate mobility information of user as a user feature in evaluating user-user similarity (in content-based filtering of hybrid approach).

2. Treat mobility information separately and group all the data according to mobility information after applying a hybrid approach on it.

The first approach treats mobility as any other feature and assigns the same weight to it. It may recommend movies/series that have not been watched by any other user during that specific mobility type, such as on the bus or at home. It focuses on content and user similarities. While the second approach assures that the mobility information has been given more attention. It recommends movies/series that are watched during that mobility type and focuses on what people watch during that specific location/mobility type.

Overall, we are concentrating on the second approach because that is our primary goal. However, there are several drawbacks to taking this technique. For example, what would be the recommended series or movies if there is no near match or no data in a specific mobility type? To address this issue, we must prioritize the remaining mobility types for each mobility type. For example, the 'indoor' kind is the most similar to the 'house' type, and the 'outdoor' type is the second most similar, relatively. If we do not want to prioritize mobility type, the most effective method to address the issue is to utilize a hybrid approach: use mobility type as a feature as well as mobility grouping in the end. In this way, the system first attempts to find series of the same mobility type, but if none are found, it automatically searches for a suitable neighbor outcome; we do not need to prioritize any mobility type.

Data Preprocessing

Given the current form of data, we have different types of fields i.e location, date, playtime etc, Some are in textual form, some are in date/time format and others are numbers. So we will classify them, convert all fields into number format and do the normalization.

There are some fields which are irrelevant, so we will extract only relevant fields which have some impact in the recommendation process and only utilize them.

In pandas, you can use ‘usecols’ parameter to just extract the relevant fields, a snapshot is given below:

```
import pandas as pd
r_data=pd.read_csv('./mobility-based-rc/full-data.csv',usecols=['user_number','content number','title','genre','loc',
r_data
```

	user_number	title	content number	genre	loc	Weekdays/Weekends	Gender	Profession	Age	age	event	view	time	device	section	CP	watch time (candle)	t-
0	1	game of thrones season 3	2	19, SF, SF fantasy , mid , fantasy	1	1	2	1	25	1	1	1	1	3	5	5	0.0	3
1	1	game of thrones season 3	3	19, SF, SF fantasy , mid , fantasy	1	1	2	1	25	1	0	1	1	3	5	5	62.5	3

We looked for fields that can have an impact on the recommendation when extracting relevant fields. Latitude, for example, has no effect on recommendation unless we additionally analyse current geodemographic data and use it to recommend series related to those geodemographics. Similarly, each field has been studied and checked for significance, and zero fitting fields have been eliminated.

After converting the fields into one platform, we replaced the textual and date/time data with their numerical representation. For example, channel was a field with textual values, therefore we categorized each channel into a numerical class and replaced each channel with its class. In this way, we had only numerical value fields except for the genre field.

The genre field has a strong influence on the recommendation. It is one of the most significant in the field of OTT platforms. For example, boys typically dislike romance and drama, while they enjoy thriller and action films. We can't afford to ignore the 'genre' field.

The problem with the ‘genre’ field is its diversity. There are a number of genres on every platform. After analyzing the sample dataset, we recorded around 35 genres. We cannot directly use genre because it is also in textual form. We needed to convert them into numbers by giving them classes. But giving a separate class to every genre is not right, because a movie can have multiple genres.
Thus, there were three solutions to this problem:

1. Use every genre as a field, which will then be used as a feature later on, just like any other field.
2. Give a unique number to every combination of genres.
3. Analyze the data, prioritize every genre according to their number of content, pick the top ‘n’ genre and use only those as a field.

We mostly utilized the third approach to solve the problem. We computed the frequency of every genre in the data and ordered them in descending order based on overall popularity. After sorting the genres, the top 20 genres were chosen to be used. Some genres that were not in the dataset but were popular on OTT platforms [2][3] were also chosen and added to our genre list. As a result, our genre count reached 27.

We get the unique genre list in pandas using below given approach:

```
genre_type=list(r_data['genre'])
```

We find the frequency of each genre using pandas like this:

```
d_genre=[]
genre_count={}
for x in genre_type:
    g=x.split(',')
    for gen in g:
        if gen.strip() not in d_genre:
            d_genre.append(gen.strip())
        if gen.strip() not in genre_count:
            genre_count[gen.strip()]=0
        else:
            genre_count[gen.strip()]+=1
```

After running the above code, we have all the genres frequencies. Now we can sort them in descending order using ‘sorted’ function like this:

```
genre_count=dict(sorted(genre_count.items(), key=lambda item: item[1]))
print(genre_count)
```

```
{'human': 0, 'documentary': 1, 'court': 4, 'Medicine': 4, 'politics': 4, 'Quiz': 8, 'yield': 14, 'period drama': 24,
'contest': 44, 'audition': 44, 'Pets': 45, 'War': 48, 'weekend drama': 71, 'food': 86, 'music': 131, 'history': 136,
'news': 155, 'sports': 179, 'growth': 182, 'revenge': 182, 'Culture': 251, 'Monday-Tuesday drama': 280, 'youth': 30
1, 'drama': 350, 'life': 495, 'remake': 532, 'romantic relationship': 757, 'reality': 855, 'sitcom': 948, 'action':
972, 'talk': 1037, 'family': 1083, 'comedy': 1088, '19': 1123, 'SF fantasy': 1124, 'SF': 1148, 'mid': 1148, 'Wed-Thu
drama': 1365, 'Friday-Saturday Drama': 1537, 'adventure': 1838, 'romance': 1854, 'Trip': 2143, 'game': 2284, 'fantas
y': 2963, 'mini series': 3184, 'variety': 3275, 'etc': 3514}
```

As given above, we can pick the most watched genres and after adding the other famous genres, our genre list looks like this:

```
genre=['documentary','quiz','contest','audition','history','news','sports','drama','romantic relationship',
      'reality','sitcom','action','talk','family','comedy','sf fantasy','sf','mid','adventure','romance','trip',
      'game','fantasy','mini series','horror','crime','mystery']
```

These genres have been used in the same way as any other feature. To make it more like the other features, we did one-hot encoding of each genre. For example, if a movie/series has multiple genres, such as thriller, action, mystery, and so on, these genres' field columns will have a value of 1 and all other genres will have a value of 0.
We add separate column (with ‘0’ as initial value) for each genre using below approach:

```
for g in genre:
    r_data[g]=0
```

After adding genre columns, we did their one-hot encodings by splitting genre text by ‘,’ and replacing ‘0’ by ‘1’ according to genre presence. After one-hot encoding, we deleted the genre column, as now we have every genre separate column.

```
for index,row in r_data.iterrows():
    cur_g=row['genre'].split(',')
    for g in cur_g:
        if g.strip().lower() in genre:
            r_data.loc[index,g.strip().lower()] = 1

r_data=r_data.drop(['genre'], axis = 1)
```

Figure 1 shows a preview of the genres features of 'Game of Thrones Season 3' using one-hot encoding:

```
sitcom          0.0
action          0.0
talk            0.0
family          0.0
comedy          0.0
sf fantasy      1.0
sf              1.0
mid             1.0
adventure       0.0
romance         0.0
trip            0.0
game            0.0
fantasy         1.0
mini series     0.0
horror          0.0
crime           0.0
mystery         0.0
```

Figure 1

We removed some noise from the dataset for further cleaning. For example, while utilizing 'content id' as a feature, we came across certain values that contained text, such as 'Subtitles 5' and 'teaser 1'. Those rows were removed. There were other rows with range values, such as 27-28. We picked the last value to convert it to a numerical value. 27-28, for example, has been converted to 28. Since the numerical values in the data were in a small range, it did not suit well to normalize it using min-max normalization or any other.

After all these preprocessing, final data looks like this:

user_id	content_id	loc	weekdays/weekends	gender	profession	age	age_category	event	view	...	mid	adventure	romance	trip	game	fantasy	mini series	h
1	2	1	1	2	1	25	1	1	1	...	1	0	0	0	0	1	0	
1	3	1	1	2	1	25	1	0	1	...	1	0	0	0	0	1	0	
1	3	1	1	2	1	25	1	0	1	...	1	0	0	0	0	1	0	
1	3	1	1	2	1	25	1	0	1	...	1	0	0	0	0	1	0	
1	3	1	1	2	1	25	1	0	1	...	1	0	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
27	552	1	1	1	2	29	1	0	2	...	0	0	0	0	0	0	0	
27	552	1	1	1	2	29	1	0	2	...	0	0	0	0	0	0	0	
27	552	1	1	1	2	29	1	0	2	...	0	0	0	0	0	0	0	
27	552	1	1	1	2	29	1	0	2	...	0	0	0	0	0	0	0	
27	552	1	1	1	2	29	1	0	2	...	0	0	0	0	0	0	0	

Figure 2

Another preprocessing step had to be done because of noise in ‘content_id’. It was encountered when we analyzed the ‘content_id’ with ‘title’. It was found that the title and content_id was overlapping. Many content_ids had the same title and many titles had the same content_id. Thus, we needed to remove the content_id as a representation of content. We devised a new approach by giving sequential numbers to titles and maintained a separate file for id_to_title conversion. We replaced the content_id with numerical titles and named it our new ‘content_id’, now our total ‘content_id’ is around 76.

Assign unique content_id to every title:

```
count=1
for t in unique_title:
    title_id[t.strip()]=count
    id_to_title.loc[len(id_to_title)]=[count,t.strip()]
    count+=1
```

Replace the title with content_id:

```
for t in title_id:
    r_data.loc[r_data['title'].str.strip()==t,'title']=title_id[t]
```

Delete the previous content_id column, as now it is useless (now 'title' is our new content_id):

```
r_data=r_data.drop('content number', 1)
r_data=r_data.rename(columns={'title': 'content number'})
```

A preview of id_to_title records is given below in figure 3.

id	title
1	game of thrones season 3
2	Running Man
3	dogs are great
4	assassination
5	game of thrones season 4
6	game of thrones season 5
7	game of thrones season 6
8	game of thrones season 7
9	game of thrones season 8
10	TV animal Farm
11	The story of the day when the tail was bitten
12	soft hair
13	(Subtitles) Harry Potter and the Chamber of Secrets
14	now , I'm breaking up
15	Eat easy and gongchili
16	gentleman and lady
17	(Subtitles) Harry Potter and the Sorcerer's Stone
18	I live alone
19	radio star
20	All butlers
21	Baek Jong-won's Alley Restaurant
22	game of blood
23	my ugly baby

Figure 3

Methodology

The main parts of implementation are:

- Collaborative Filtering
- Content/User Based Filtering

As previously stated, we are using a hybrid (collaborative + content/user) approach. Rating is one of the major fields used in recommendation systems utilizing collaborative filtering. Unfortunately, it is not included in our dataset. But there is another field for collaborative filtering that we have used and that many recommendation systems use: 'watch time.' People only watch videos if they enjoy them. Not liked videos typically have a low watch time, whereas liked videos have a high watch time. As a response, we used this as our rating scale.

The problem with using watch time is its limit. Ratings are normally limited and more discrete and watch time can vary greatly. We can't classify the watch time as easily as ratings.Thus, we set minimum and maximum watch time of any series in our dataset as our low and high limits of rating scale. The more a user spends time on a series the higher rating that series would have. We need an interaction field in collaborative filtering (how the user interacts with the content). Rating is typically used as an interaction field in movie/series recommendation systems. However, in this case, 'watch time' has been used as an interaction field.

To find the rating limit, we find maximum and minimum of our 'watch_time':

```
#rating scale calculation
min_rate=r_data['watch_time'].min()
max_rate=r_data['watch_time'].max()
print(min_rate,max_rate)
```

We used the matrix factorization technique [4] in collaborative filtering to train the model on how different users interacted with different content (series) and used the output as an additional feature in the final model. We trained our SVD [5] on a dataframe with three fields: 'user id,' 'content id,' and 'watch time' in matrix factorization. We used the 'Surprise' library [6] for training and prediction because it has all of the functionality needed to develop recommendation models.

Now, we can train our matrix factorization model using surprise library:

```
from surprise import SVD
import numpy as np
import surprise
from surprise import Reader, Dataset

# It is to specify how to read the data frame.
reader = Reader(rating_scale=(min_rate,max_rate+1))
train_data_mf = Dataset.load_from_df(r_data[['user_id', 'content_id', 'watch_time']], reader)

#It is of dataset format from surprise library
trainset = train_data_mf.build_full_trainset()
svd = SVD(n_factors=100, biased=True, random_state=15, verbose=True)
svd.fit(trainset)
```

We can observe that the rating limit is 'min_rate' and 'max_rate+1'. We are adding 1 to be on the safe side (more inclusive approach). After training SVD, we can get the predictions for later usage as a new feature.

```
train_preds = svd.test(trainset.build_testset())
train_pred_mf = np.array([pred.est for pred in train_preds])
```

Figure 4 depicts the prediction of SVD on our dataset.

	user_id	content_id	mat_fac_pred
0	1	2	57.197912
1	1	3	55.103702
2	1	3	55.103702
3	1	3	55.103702
4	1	3	55.103702
...	...	...	...
13092	27	552	60.698146
13093	27	552	60.698146
13094	27	552	60.698146
13095	27	552	60.698146
13096	27	552	60.698146

Figure 4

The 'mat_fac_pred' field will now be used as a feature in the final recommendation model, representing user interaction with the content (series/movies).

Apart from these feature, we also devised to add some hand crafted features for example:

1. Average watch time of all the movies by all users.
2. Average watch time of a movie by all users.
3. Average watch time of all movies watched by a user.
4. Average watch time of a movie by a user.

We utilized pandas built-in functions to calculate these features.

```
#1 overall watch_time average
g_avg=r_data['watch_time'].mean()
r_data['g_avg']=g_avg
```

For example, there are 3 users, who watch 2 movies with watch time of 58, 43, 59, 60, 63 and 61 minutes then their g_avg will be 57.33 mins.

```
#2 average watch time of a movie by all users
r_data['m_avg']=r_data.groupby(['content_id'])['watch_time'].transform('mean')
```

For example, there is a movie which is watched by 3 users with watch time of 51, 55, and 56 minutes then their m_avg will be 54 mins.

```
#3 average watch time of all movies watched by a user
r_data['u_avg']=r_data.groupby(['user_id'])['watch_time'].transform('mean')
```

For example, there is a user who has watched 4 movies with a watch time of 48, 51, 69, and 45 minutes then his u_avg will be 53.25 mins.

```
#4 average watch time of a movie by a user
r_data['um_avg']=r_data.groupby(['user_id','content_id'])['watch_time'].transform('mean')
```

For example, there is a user who has watched a movie 2 times with different watch time of 59 and 50 minutes then his watch time for that movie will be 55 mins.

All these features are evaluated and added to the dataset. After these features are evaluated, we move towards the user similarity features on the basis of user attributes, what contents they watch and how their interactions were with those contents. To achieve this, we followed the below given steps:

1. Narrow down the user and content attributes.
2. Find the average of every feature of every user.
3. Find the cosine similarity between every user on the basis of feature averages.
4. Pick top 5 similar users for every user.
5. Add average watch time of top 5 similar users of every user as 5 different features.

Following are narrowed down attributes related to user and content:

```
user_content_features=['gender','profession','age','age_category','section','cp','documentary','quiz','contest',
      'reality','sitcom','action','talk','family','comedy','sf fantasy','sf','mid','adventure','romance','trip',
      'game','fantasy','mini series','horror','crime','mystery']
```

We calculated the average of every attributes/features one by one for each user:

```
unique_user=list(r_data['user_id'].unique())

#get feature average of every user
user_feature_avg={}
for u in unique_user:
    user_feature_avg[u]=[]
    for f in user_content_features:
        user_feature_avg[u].append(r_data.loc[r_data['user_id']==u][f].mean())
```

Now that we have an average of every feature for every user, we can find the similar user using cosine similarity.

```
#get cosine similarity between every user and sort them
from scipy import spatial

u_cosim={}
for u in user_feature_avg:
    u_cosim[u]={}
    for u2 in user_feature_avg:
        if u!=u2:
            sim = 1 - spatial.distance.cosine(user_feature_avg[u], user_feature_avg[u2])
            u_cosim[u][u2]=sim
    u_cosim[u]=dict( sorted(u_cosim[u].items(),key=lambda item: item[1],reverse=True))

print(u_cosim)
```

```
{1: {16: 0.9980653177233332, 2: 0.9969774681646064, 25: 0.9968136400575084, 6: 0.9945358742038859, 10: 0.99281151082
84349, 20: 0.9888484512484217, 21: 0.9886025863321422, 19: 0.987139007607623, 12: 0.9869821332229565, 26: 0.98580569
14884197, 4: 0.9855628722850371, 17: 0.9855446480023692, 18: 0.9849725464010796, 3: 0.9843446315066804, 27: 0.983365
3907128639, 23: 0.982118270531688, 22: 0.9808978818901286}, 2: {25: 0.9988859944119418, 6: 0.9982394918452219, 1: 0.
9969774681646064, 16: 0.9965872290990181, 10: 0.9955480404464253, 20: 0.9937780996595034, 21: 0.993711868965615, 12:
0.9916692756459452, 26: 0.9916671528106564, 4: 0.991508584864329, 19: 0.990932131028257, 3: 0.9903108748039978, 27:
0.98946261988535, 17: 0.9893064873044018, 18: 0.9885971720342567, 23: 0.988501582707836, 22: 0.9877771550258567}, 3:
```

Now we have distance between every pair of users, the less the distance between pairs of users, the more similar they are for each other. We are subtracting distance from 1. So the number we are getting is the similarity score. The more the score the more similar they are. Above given dictionary tells us that user_id 16 is most similar to user_id 1. The second most similar user_id is 2 and so on.

To get the top 5 similar user for every user, we can do:

```
#get top five user for every user
top_5={}
top=5
for u in u_cosim:
    top_5[u]=[]
    top_cur=0
    for su in u_cosim[u]:
        top_5[u].append(su)
        top_cur+=1
        if top_cur==top:
            break
```

```
{1: [16, 2, 25, 6, 10], 2: [25, 6, 1, 16, 10], 3: [27, 22, 23, 26, 4], 4: [26, 27, 3, 18, 23], 6: [2, 10, 21, 20, 2
6], 10: [6, 20, 12, 21, 19], 12: [20, 3, 17, 27, 21], 16: [1, 25, 2, 6, 10], 17: [18, 27, 19, 3, 4], 18: [17, 4, 19,
27, 3], 19: [21, 26, 17, 23, 27], 20: [21, 26, 27, 3, 23], 21: [20, 26, 19, 3, 27], 22: [3, 23, 27, 26, 21], 23: [2
7, 26, 3, 22, 4], 25: [2, 16, 1, 6, 10], 26: [23, 27, 3, 21, 4], 27: [23, 26, 3, 4, 22]}
```

At the end, we add top 5 similar users' watch time as 5 different features to our data:

```
r_data['simu1']=1
r_data['simu2']=2
r_data['simu3']=3
r_data['simu4']=4
r_data['simu5']=5
print(r_data)
for index,row in r_data.iterrows():
    if index%500==0:
        print(index)
    user=int(row['user_id'])
    content=int(row['content_id'])
    sim_u=top_5[user]
    for ind,u in enumerate(sim_u):
        wtime=r_data.loc[(r_data['user_id']==u) & (r_data['content_id']==content)]['um_avg']
        simu_wtime=0
        if len(wtime)==0:
            simu_wtime=0
        else:
            simu_wtime=wtime.iloc[0]
        if ind==0:
            r_data.at[index,'simu1']=simu_wtime
        elif ind==1:
            r_data.at[index,'simu2']=simu_wtime
        elif ind==2:
            r_data.at[index,'simu3']=simu_wtime
        elif ind==3:
            r_data.at[index,'simu4']=simu_wtime
        elif ind==4:
            r_data.at[index,'simu5']=simu_wtime
```

These other 5 features names are simu1, simu2, simu3, simu4 and simu5. Their initial values are 1,2,3,4 and 5 respectively but they are updated with their average watchtime. If a user has not watched the content we are placing against their watch time, their watch time for that content will be 0.

At the end, add matrix factorization prediction as our last feature in the dataset. After adding all these features, our data looks like this given below in figure 5.

user_id	content_id	loc	weekdays/weekends	gender	profession	age	age_category	event	...	g_avg	m_avg	u_avg	um_avg	simu1	simu2	simu3	simu4	simu5	mat_fact_pred
1	1	1	1	2	1	25	1	1	...	55.61831	56.307407	55.776129	56.307407	0	0	0	0	0	56.682873
1	1	1	1	2	1	25	1	0	...	55.61831	56.307407	55.776129	56.307407	0	0	0	0	0	56.682873
1	1	1	1	2	1	25	1	0	...	55.61831	56.307407	55.776129	56.307407	0	0	0	0	0	56.682873
1	1	1	1	2	1	25	1	0	...	55.61831	56.307407	55.776129	56.307407	0	0	0	0	0	56.682873
1	1	1	1	2	1	25	1	0	...	55.61831	56.307407	55.776129	56.307407	0	0	0	0	0	56.682873
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
27	76	1	1	1	2	29	1	0	...	55.61831	54.030303	52.493215	54.030303	0	0	0	0	0	53.983993
27	76	1	1	1	2	29	1	0	...	55.61831	54.030303	52.493215	54.030303	0	0	0	0	0	53.983993
27	76	1	1	1	2	29	1	0	...	55.61831	54.030303	52.493215	54.030303	0	0	0	0	0	53.983993
27	76	1	1	1	2	29	1	0	...	55.61831	54.030303	52.493215	54.030303	0	0	0	0	0	53.983993
27	76	1	1	1	2	29	1	0	...	55.61831	54.030303	52.493215	54.030303	0	0	0	0	0	53.983993

Figure 5

For the recommendation, we used cosine similarity to find the neighbors' users on the basis of all ~50 features. We find the top 5 similar users using the approach mentioned above and save their watching history. As, we do not want to recommend those series/movies to the user which he has already watched. For that, we also maintain a record where we store already watched series/movies of every user.

```
#already watched content by a user
watched_c=pd.DataFrame(columns=['user_id','watched_content'])
for u in unique_user:
    contents=list(r_data.loc[r_data['user_id']==u]['content_id'].unique())
    watched_c.loc[len(watched_c)]=[u,contents]
```

After finding the top 5 similar users, we find the content they have watched and store it in a list.

```
#similar users contents
all_recom={}
for u in top_5:
    all_simu_content=[]
    all_simu_content_unique=[]
    all_not_watched=[]
    simu=top_5[u]
    for su in simu:
        su_content=watched_c.loc[watched_c['user_id']==su]['watched_content'].iloc[0]
#         print(su_content)
        all_simu_content=all_simu_content+su_content
```

Now ‘all_simu_content’ has all the similar user content but they can be repetitive because multiple users can watch similar stuff as they are all similar users to the target user. Thus, we will find the unique content:

```
    for cont in all_simu_content:
        if cont not in all_simu_content_unique:
            all_simu_content_unique.append(cont)
#     print("unique all: ",all_simu_content_unique)
```

Now we can remove the already watched content from unique content, to get the final recommendations.

```
    #remove already watched content
    u_content=watched_c.loc[watched_c['user_id']==u]['watched_content'].iloc[0]
    for cont in all_simu_content_unique:
        if cont not in u_content:
            all_not_watched.append(cont)
#     print(u_content)
#     print("not watched: ",all_not_watched)
```

At the end we store all these non-watched similar user content to recommendation file, where we have recommendations for every user. A preview of which is given below in figure 6.

user_id	content_recommendation
1	[32, 33, 45, 40, 46, 47, 48, 49, 50, 51, 34, 35, 36, 37, 60, 56, 57, 41, 58, 59, 16]
2	[61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 41, 34, 35, 36, 37, 49, 56, 57, 58, 59, 60, 45, 40, 46, 47, 48, 50, 51]
3	[42, 43, 44, 14, 41, 19, 76, 11, 40, 77]
4	[41, 19, 76, 11, 14, 42, 43, 44, 20, 18, 21, 77]
6	[27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 32, 33, 44, 21, 52, 53, 54, 18, 55, 45, 40, 46, 47, 48, 49, 50, 51]
10	[23, 24, 25, 26, 32, 33, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 45, 40, 46, 47, 48, 49, 50, 51, 44, 21, 52, 53, 54, 18, 55]
12	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 45, 40, 46, 47, 48, 49, 50, 51, 34, 35, 36, 37, 60, 13, 14, 15, 16, 17, 18, 19]
16	[13, 14, 15, 16, 17, 2, 18, 19, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 41, 49, 56, 57, 58, 59, 60, 45, 40, 46, 47, 48, 50, 51]
17	[20, 18, 21, 41, 19, 76, 11, 42, 43, 44, 40, 77]
18	[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 49, 60, 32, 33, 45, 46, 47, 48, 50, 51, 56, 57, 58, 59, 16]
19	[20, 2, 18, 21, 41, 19, 76, 11, 14, 40, 77]
20	[32, 33, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 34, 35, 36, 37, 60, 13, 14, 15, 16, 17, 18, 19]
21	[71, 42, 72, 19, 73, 49, 74, 75, 41, 23, 25, 26, 27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 32, 33]
22	[60, 61, 62, 63, 64, 2, 17, 13, 65, 66, 67, 68, 19, 69, 70, 14, 15, 18, 34, 35, 36, 37, 45, 40, 46, 47, 48, 50, 51]
23	[56, 57, 41, 58, 59, 16, 34, 35, 36, 37, 13, 14, 15, 17, 2, 18, 19, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 45, 40, 46, 47, 48, 50, 51]
25	[14, 15, 16, 18, 34, 35, 36, 37, 49, 56, 57, 58, 59, 60, 45, 40, 46, 47, 48, 50, 51]
26	[44, 24, 11, 21, 52, 53, 54, 55, 23, 25, 26, 27, 28, 29, 30, 31, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 32, 33]
27	[42, 43, 44, 40, 20, 18, 21, 14, 77]
4571	[41, 2, 19, 76, 11, 20, 18, 21, 42, 43, 44, 14]
1542	[41, 2, 19, 76, 11, 20, 18, 21, 42, 43, 44, 14]

Figure 6

At this point, the saved recommendation for every user has already considered the locality of every user. But to give it more importance, we needed to record locality wise recommendations separately. To make sure of it, we devised following plan:

1. Find unique localities in the dataset.
2. Find the frequency of every content in every locality.
3. Sort content in descending order in every locality.
4. Make a sorted list of contents for every locality as recommendations.

Find unique localities in the dataset.

```
#unique_locality
u_loc=list(r_data['loc'].unique())
```

Find the frequency of every content in every locality. For example, if ‘moviex’ is watched by 3 people ‘on the bus’, then frequency of moviex in ‘on the bus’ locality is 3.

```
cnt=r_data.groupby(['loc','content_id'])['content_id'].count()
for c in cnt.iteritems():
    loc_content.loc[len(loc_content)]=[c[0][0],c[0][1],c[1]]
```

Now, sort them in descending order so that we can get high frequency watched content in the first position and store them in a separate file like overall recommendation.

```
locwise_recom=pd.DataFrame(columns=['location','content_recommendation'])
for x in localities:
    spec_loc_df=loc_content.loc[loc_content['loc']==x][['content','freq']]
    spec_loc_df=spec_loc_df.sort_values('freq', ascending=False).reset_index(drop=True)
    loc_spec_sorted_c=list(spec_loc_df['content'])
    locwise_recom.loc[len(locwise_recom)]=[x,loc_spec_sorted_c]

locwise_recom.to_csv('./location_wise_recommendation.csv')
# print(locwise_recom)
```

After following the above approach, we had separate recommendations based on just localities. It looks like the one given below in figure 7.

location	content_recommendation
1	[2, 29, 44, 31, 32, 64, 19, 49, 51, 23, 17, 5, 41, 6, 62, 14, 1, 27, 13, 11, 66, 33, 40, 65, 67, 8, 18, 5
2	[2, 49, 44, 19, 11, 41, 57, 18, 1, 28, 32, 75, 3, 36, 58, 16, 5, 55, 34, 33, 13, 4, 56]
3	[2, 49, 42, 47, 75, 51, 21, 20, 35, 41, 18, 1, 11, 48, 33, 19, 76, 40, 45, 46]
4	[49, 41, 13, 2, 17, 16, 23, 75, 35, 57, 25]
5	[47, 48, 75, 19, 23, 74, 77]
6	[2, 29, 25, 13, 60, 17, 70, 19, 14, 8, 24, 41, 16, 1, 23, 53, 21, 36, 47, 7, 71, 3, 31, 15, 35, 11, 49, 2
7	[23, 24, 5]

Figure 7

We have made different separate records and history so at the time of recommendation, we can be able to give recommendations as fast as possible. For the recommendation, system follow the below given flow:

1. It first loads all the required files i.e input file, recommendation file, id to title etc.
2. Check user information if it is a new or old user.
3. If the user is old, it gets the user id, picks the recommendations (content ids) from the recommendation file and converts it to content titles and presents it.
4. If the user is new, it gets the new user' personnel and other information, refreshes the database, evaluates and updates all files again. At the end, it picks the recommendation from the recommendation file and converts it to their titles.
5. However, in both cases, it also checks for the locality information. If the user wants location to be impacted on the recommendation, it does one more step of extracting locality wise recommendation from the locality wise recommendation file and pick the first common optimal content from both files(overall recommendations and locality wise recommendations).

There are two kinds of recommendations involved in this project: One for old users and other for new users. Normally, recommendations start when users have spent some time on the platform, after they have watched some content because only then does the system know what they like and what they do not like. In this way, It is easy for the algorithm to suggest some content because it knows the user's history. But for new users, it does not have any idea and cannot make recommendations usually. But in this project, we have also incorporated recommendations for new users as well. Users will have to just tell their personnel information like age, profession, gender etc, genre they are interested in and locality if they want it to be

incorporated. System will refresh its database and bring the recommendations instantly, however, it will take some time.

For example, we will get recommendations for user_id 4. Some rows of his historical data are given below in figure 8.

4	11/28/2021 12:50	1:23:53	game of blood	5	5028	11/29/2021	game , etc
4	11/28/2021 12:51	1:24:53	game of blood	5	5028	11/29/2021	game , etc
4	11/28/2021 12:52	1:26:35	game of blood	5	5028	11/29/2021	game , etc
4	11/28/2021 12:53	1:27:35	game of blood	5	5028	11/29/2021	game , etc
4	11/29/2021 12:06	0:05:07	Running Man	577	5237	10/31/2021	game , etc , variety , Trip
4	11/29/2021 12:07	0:05:50	Running Man	577	5237	10/31/2021	game , etc , variety , Trip

Figure 8

For old user recommendation, ‘input_file.csv’ is used to give inputs like user information and liked genres etc. A preview of inputs are given below:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
new_old	user_id	content_id	loc	weekdays/weekends	gender	profession	age	age_category	event	view	time	device	section	cp	watch_time
1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	4	0	1	0	0	0	0	0	0	0	0	0	0	0	

Figure 9

We do not need to give gender and other information related to the user because the system already knows that as it is an old user.

Recommendations for user_id 4 without locality preference is given below:

```
----- Recommendation for user:  4 without location preference -----
1: what do you do when you play ?
2: radio star
3: You Hee-yeol's Sketchbook
4: The story of the day when the tail was bitten
5: now , I'm breaking up
```

Figure 10

Recommendations for user_id 4 with locality preference ‘outdoor’ is given below:

```
----- Recommendation for user:  4 with location preference -----
1: my ugly baby
2: mission possible
3: game of thrones season 4
4: what do you do when you play ?
5: radio star
```

Figure 11

Recommendations for user_id 4 with locality preference ‘home’ is given below:

```
----- Recommendation for user:  4 with location preference -----
1: Doll Singles 2
2: radio star
3: what do you do when you play ?
4: now , I'm breaking up
5: The story of the day when the tail was bitten
```

Figure 12

For new users, we have made a separate file for input named ‘input_file_new_user’. A preview of inputs are given below:

new_old	user_id	content_id	genre	loc	weekdays/weekends	gender	profession	age	age_category	event	view	time	device	section	cp	watch_time
0	4571	0	none	5	1	1	2	21	1	0	1	12	3	2	3	0
0	1542	0	sf,mystery,action	0	0	2	1	18	0	0	1	2	1	2	2	0

Figure 13

As these users are new, they will have to fill all the necessary fields so that the system can have an idea of what your likes and dislikes could be.

Recommendations for user_id 1542 without locality preference is given below:

```
----- Recommendation for user:  1542 without location preference -----
1: what do you do when you play ?
2: Running Man
3: radio star
4: You Hee-yeol's Sketchbook
5: The story of the day when the tail was bitten
```

Figure 14

Recommendations for user_id 4571 with locality preference ‘driving’ is given below:

```
----- Recommendation for user:  4571 with location preference -----
1: radio star
```

Figure 15

We can observe that there is only one recommendation in case of ‘driving’ locality and new user, it means that there is few data in the database currently regarding this and few matches were found. Let’s try ‘on the bus’ locality for a new user.

Recommendations for user_id 1147 with locality preference ‘on the bus’ is given below:

```
----- Recommendation for user:  1147 with location preference -----
1: Running Man
2: Doll Singles 2
3: radio star
4: The story of the day when the tail was bitten
5: what do you do when you play ?
```

There are some contents which keep repeating in the suggestion, a possible reason is that few people have watched those contents and the user we are suggesting these content to has already watched good suggested content and now the system keeps focusing on the next good content for a particular user.

References

1. Netflix Prize, URL: https://en.wikipedia.org/wiki/Netflix_Prize
2. Most popular digital original series genres based on audience demand, URL:https://www.statista.com/statistics/715161/most-in-demand-tv-genre-in-north-america/
3. Guide to TV genres, 15 Popular Television Genres, URL: https://www.masterclass.com/articles/guide-to-tv-genres#what-is-a-tv-genre
4. Matrix-Factorization Based Algorithms, URL:https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD
5. Simple SVD algorithms, URL: https://towardsdatascience.com/simple-svd-algorithms-13291ad2eef2
6. Surpr!se, URL: http://surpriselib.com/