Data Analysis Report

Problem Description

This project builds a music recommendation system for users in the Last.fm online music platform. The goal is to give a recommendation list of artists to registered users. Recommendation systems have been widely used in different companies (Netflix, Amazon, Youtube etc.) to give a client/user a list of items (movies/ products/ video clips) that he/she might like. There is no perfect recommendation system in terms of accuracy, meaning that the recommendation list does not match perfectly well with a client/user's preferences. This project aims to explore some of the existing recommendation methods and make an adaption to the content-based matrix-factorization method to achieve high matching accuracies in our music recommendation system.

Data Source

We explore the "hetrec2011-lastfm-2k" datasets from "http://www.last.fm" which contain information of over 2000 and over 18000 artists from the Last.fm online music system. The datasets to be used are "user_artists.dat "(the major dataset) which provides users' listening history: the number of times the music of each artist played by each user, and "user_friends.dat ", which describes the relationships among users: whether they are linked as "friends" in the music platform.

Data Preprocessing

The following three steps are done sequentially to the main dataset "user_artists.dat" as preprocessing treatments. After the following steps, we add an extra column storing (user, artist)-pairs in the dataset. The preprocessed dataset which is ready to be used for further analysis is saved as "user_artists_majority".

1.Scale the ratings

In the major dataset “user_artists.dat”, the column “weight” originally stores the listening counts for each (user, artist). The listening counts indicate the likeness of users to artists. We will rename the column “weight” to “Rate”, scale the values by 1/10000 and see the values as ratings. In the matrix decomposition method to be implement later, our goal is to make regressions on the ratings of all possible (user, artist)-pairs based on existing ratings. For non-existing ratings in the original dataset we will have to initialize them reasonably in a recommendation matrix. This topic will be revisited in more depths in machine learning analysis. The scaling is performed in order that when implementing machine learning algorithms (Stochastic Gradient Descent), the numeric values presented in the recommendation matrices are within the acceptable range restricted by the system’s capacity. Huge numbers such as 10^100 would be out of the acceptable range. A programming language such as Python will store these huge numbers as NaN(not a number). This ruins the processing of machine learning algorithms. The scaling is done to prevent this unwanted scenario.

2.Check outliers

The ratings are given by weights/10000 and the weights are listening counts according to the documentation of the dataset. We need to check if there exist negative ratings or unexceptional high ratings of the dataset. We have found that the maximum of all ratings is 35.2698 and the minimum of all ratings is 0.0001 which correspond to 352698 and 1 listening count(s) of a (user, artist pair). The conclusion is all listening counts are positive and all listening counts are within a reasonable range. 352698 is a large number for listening counts but it is also possible, assuming that the user associated with this number is exceptionally active in the music platform. None of the ratings are considered as outliers.

3.Delete minor records

A quick investigation of the major dataset “user_artists.dat” shows that most of users have exactly 50 favorite artists and no user has over 50 favorite artists. By favorite artists, we mean the artists in that have records in the listening history of users. Deleting the records of those users who have less than 50 artists, the length of the major dataset “user_artists.dat” shrinks from 92834 to 91450, remaining

98.51% of the original length. We do the deletion to retain the conformity that each user has the same contribution to the dataset (each user has listening counts on 50 artists). We might lose the ratings from inactive users though with the deletion. For example, a user's listening counts on an artist is ignored if the artist is the only one in the user's play list. Note that we only missed <1.5% of data after the deletion.

Preliminary Data Analysis

In this part, we want to check which artists are most popular in two standards: 1.the overall hitting times (listening counts) of all users; and 2.the total number of fans (users who have played the music of the artist).

Popularity based on hitting times(rates)

1.An overview

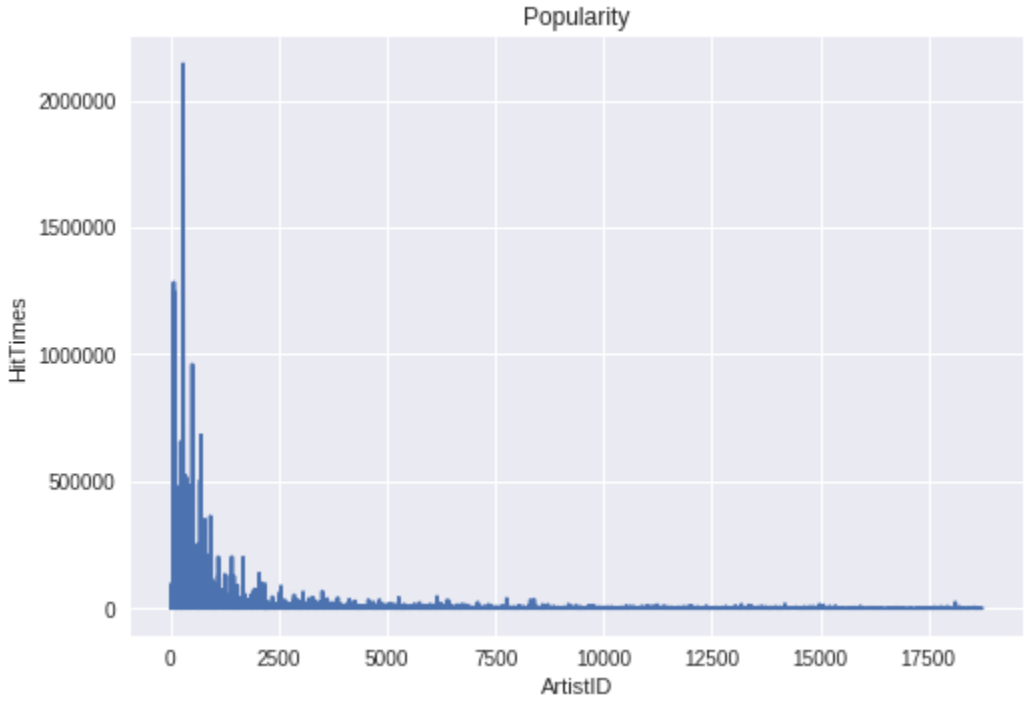
We convert the rates back to hitting times and calculate the sum of hitting times for each of the artists via the following Python code. The code generates a data frame named ArtitstPopularity which stores the total hitting times of each artist.

```
ArtistPopularity =
pd.DataFrame(user_artists_majority.groupby(['ArtistID']).Rate.sum()*10000).reset_index().rename(colu
mns={'Rate':'HitTimes'})
```

We then visualize the hitting times over the set of all artists. This is done by the matplotlib package with the following code.

```
import matplotlib.pyplot as plt

plt.plot(ArtistPopularity.ArtistID,ArtistPopularity.HitTimes)

plt.xlabel('ArtistID')

plt.ylabel('HitTimes')

plt.title('Popularity')
```

The following graph shows that artists with a smaller ArtistID number tend to be more popular.



The graph also indicates a small portion of artists (those with small ArtistID) take up a large portion of the total hitting times. Our findings are the followings.

A total of 68438134 hits is recorded.

The top 100 artists out of the total 17374 artists have >43.36% of total hits.

The top 200 artists out of the total 17374 artists have >53.63% of total hits.

Relevant codes are attached below.

```
np.sum(ArtistPopularityRanked.HitTimes)

ArtistPopularityRanked = ArtistPopularity.sort_values(by=['HitTimes'],ascending=False)

np.sum(ArtistPopularityRanked.head(100).HitTimes)/np.sum(ArtistPopularityRanked.HitTimes)

np.sum(ArtistPopularityRanked.head(200).HitTimes)/np.sum(ArtistPopularityRanked.HitTimes)
```

2.Top-10 artists

The top-10 artists with the most of hitting times are Artists 289, 72, 89, 292, 498, 67,288,701,227,300.

	ArtistID	HitTimes
283	289	2143837.0
66	72	1283763.0
83	89	1245613.0
286	292	1045616.0
492	498	960446.0
61	67	897815.0
282	288	873934.0
695	701	682104.0
221	227	655560.0
294	300	527910.0

Codes: `ArtistPopularity.sort_values(by=['HitTimes'],ascending=False).head(10)`

3. Statistical Analysis

We use a two-sample t-test to test whether the average of hitting times of the first 2500 artists is significantly larger than that of the rest of the artists. To be more specific, the null hypothesis is H0: mean_1 - mean_2 <= d0 and the alternative hypothesis is Ha: mean_1 - mean_2 > d0, where mean_1 is the average hit times of the first 2500 artists, mean_2 is the average hit times of the rest of the artists, and d0 is the margin which measures the size of the difference. We reject the null hypothesis H0 if we have a small p-value of the t-test.

The findings are given as below.

- The t-test with d0=20000 has a p-value of 0.509. We fail to reject the null hypothsis that mean_1-mean_2<=20000.
- The t-test with d0=18000 has a p-value of 0.123. We fail to reject the null hypothsis that mean_1-mean_2<=18000(not at a confidence level of 90%).

- The t-test with d0=15000 has a p-value of 0.002. We reject the null hypothsis that mean_1-mean_2<=15000(at a confidence level of 99.8%) and conclude that mean_1-mean_2>15000.

Codes:

```
d0 = 15000   #18000, 20000
t = (mean_1 - mean_2-d0)/np.sqrt(var_1/2500 + var_2/len(ArtistPopularityRest))
degree = 2500 + len(ArtistPopularityRest) - 2
p = 1 - stats.t.cdf(t, df = degree)
print('The p-value of the two-sample t-test is %.3f' %p)
```

4. Hitting-times distribution of the most popular artists

We are interested in the distribution of the hitting times of the most popular artists. To do this, we first create a table containing the ratings of each user to all artists. If a user has not listened to the music of the artists, the rating is filled with zero.

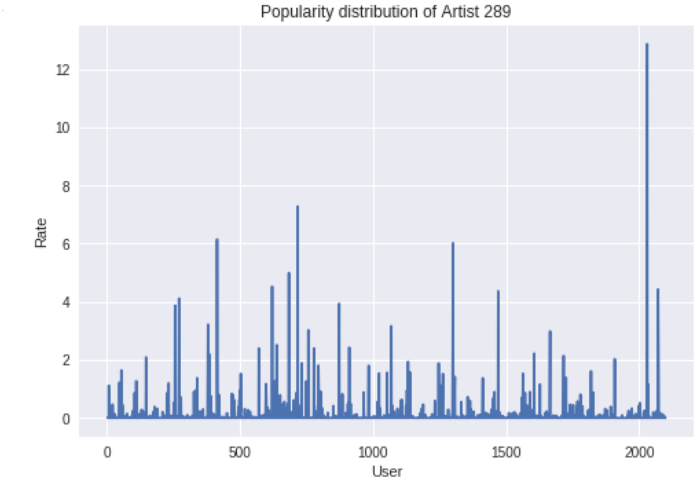
```
UserArtistTable = user_artists_majority.pivot(index='UserID', columns='ArtistID', values='Rate').fillna(0)
```

Then we write the following popularity function which plots the ratings of each artists by all users.

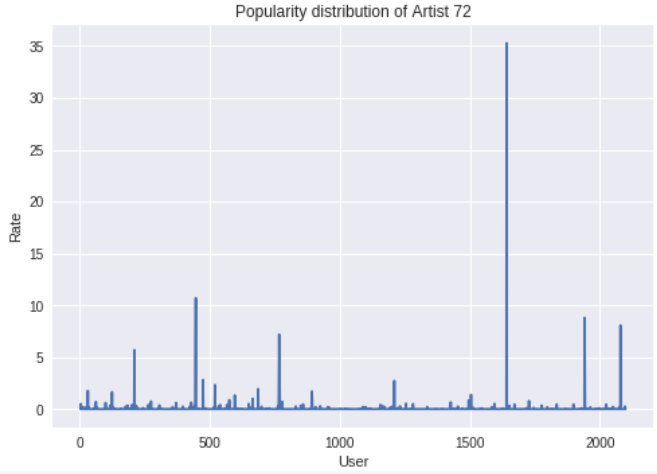
```
def Popularity(uid):
    Artist = pd.DataFrame(UserArtistTable.loc[:,uid]).reset_index().rename(columns={uid:'Rate'})
    plt.plot(Artist.UserID,Artist.Rate)
    plt.xlabel('User')
    plt.ylabel('Rate')
    plt.title('Popularity distribution of Artist %d' % uid)
```

The graphs indicate that most of hitting times of an artist's play lists is contributed by a few huge fans, typically for Artist 72, except possibly for Artist 289 (Artist 289 seems to have a broad range of audience).

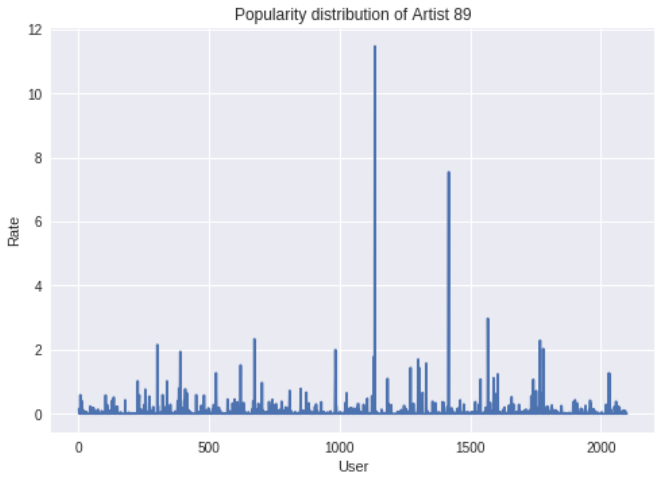
```
Popularity(289)
```




```
Popularity(72)
```



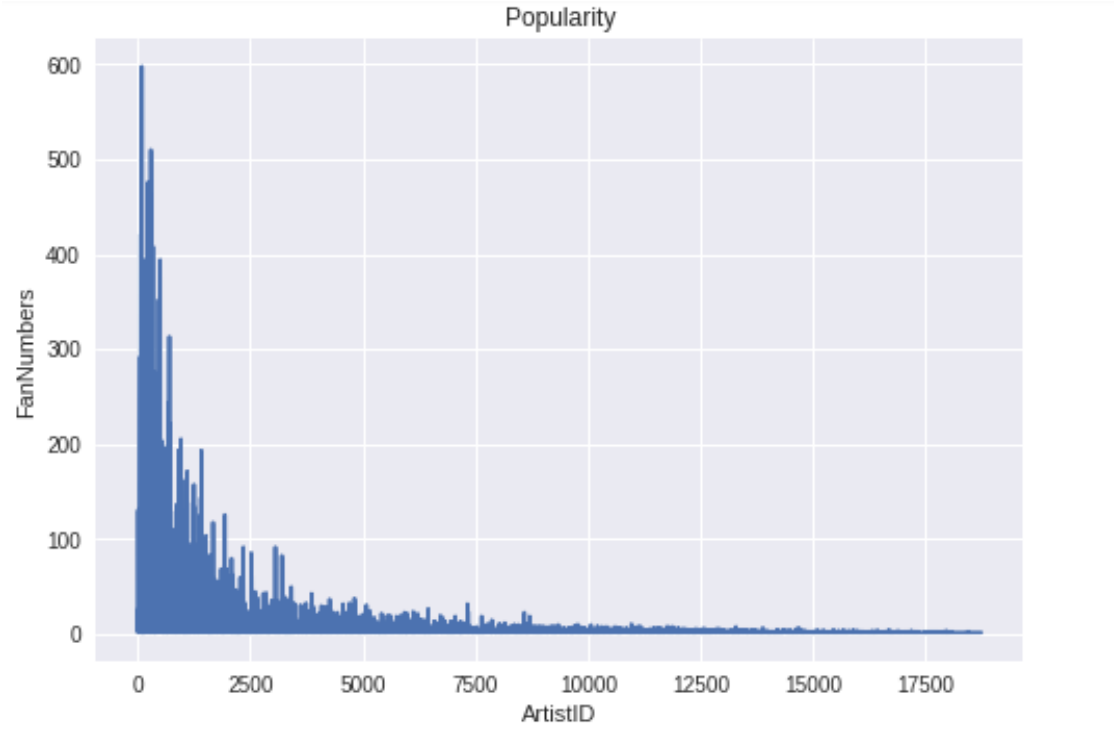

```
Popularity(89)
```



Popularity based on the population of fans

The methods and codes are similar to the previous part of “Popularity based on hitting times(rates)”. Codes will be omitted.

1. An overview visualization



The graph is a plot of the number of fans (users that have played the music of the artist) that each artist has. It shows that artists with smaller numbers of ArtistID tend to have more fans.

2. Top-10 Artists

The following table shows the top-10 artists that have the most fans. Artists 89,289,227,288,300,67,333,292,498 are among the most popular artists in both rating-based and fan-population-based evaluation system. Note that Artist 72 is in the top-10 list of rating-based population evaluation method but is not in the top-10 list of the fan-population-based system.

	ArtistID	FanPopulation
83	89	598
283	289	510
221	227	476
282	288	468
294	300	462
61	67	421
327	333	408
286	292	398
184	190	395
492	498	394

Conclusions

Based on the visualizations and the statistics, we conclude that among over 17000 artists in the data set, those with smaller numbers of ArtistID tend to be more popular both in terms of total listening counts and fans population.