

# A Simple Recommendation System for last.fm data

SHEN Anqi 54577992

ZHANG Yijun 54505695

YAN Jiaying 54512662

HE Peiwei 54471952

## 1. Introduction

Last.fm is a music website, founded in the United Kingdom in 2002, now it has become one of the most famous musical recommendation system, by using a music recording system called "Audioscrobbler". Last.fm builds a detailed profile of each user's musical taste by recording details of the tracks the user listens to, either from Internet radio stations, or the user's computer or many portable music devices.

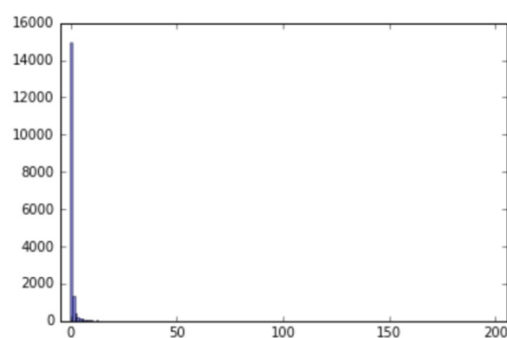


Figure1. User's power law distribution

By cleaning, splitting the last.fm user data we found in our dataset, we aim to develop several simple recommendation systems. The above graph shown that x-line is the number of people who listen to same artists, and y-line stands for the user's listening counts. There is a power law distribution in last.fm data, while our targeted customers are music listeners, both old users and new users. We want to recommend artists to our users. We want to help user to find their own special tastes, recommend more precise artists for listeners who listen to rock, classic, jazz, etc. For old users, it's also possible for them to find their favorite artists according to their listening records. Our metrics would be users' preference, especially rock, classic, jazz or even indie music lovers.

We build our codes in Jupiter notebook, then put all source codes to flask frame and successfully get a simple web-based recommendation system.

## 2. Data collection

The dataset is downloaded from <https://grouplens.org/datasets/hetrec-2011/>, and this dataset contains social networking, tagging, and music artist listening to information from a set of 2000 users from Last.fm online music system. It includes 1892 users 17632 artists, 12717 bi-direction, 92834 user-listened artist relations, 186479 tag assignments. After cleaning up original data, we get such a table:

|   | userID | artistID | weight | name        | url   | pictureURL  | weight_a | artist_ix | user_ix | freq     |
|---|--------|----------|--------|-------------|---|---|----------|-----------|---------|----------|
| 0 | 2      | 51       | 13883  | Duran Duran | <a href="http://www.last.fm/music/Duran+Duran">http://www.last.fm/music/Duran+Duran</a> | <a href="http://userserve-ak.last.fm/serve/252/155668.jpg">http://userserve-ak.last.fm/serve/252/155668.jpg</a> | 1        | 45        | 0       | 1.000000 |
| 1 | 4      | 51       | 228    | Duran Duran | <a href="http://www.last.fm/music/Duran+Duran">http://www.last.fm/music/Duran+Duran</a> | <a href="http://userserve-ak.last.fm/serve/252/155668.jpg">http://userserve-ak.last.fm/serve/252/155668.jpg</a> | 1        | 45        | 2       | 0.045756 |
| 2 | 27     | 51       | 85     | Duran Duran | <a href="http://www.last.fm/music/Duran+Duran">http://www.last.fm/music/Duran+Duran</a> | <a href="http://userserve-ak.last.fm/serve/252/155668.jpg">http://userserve-ak.last.fm/serve/252/155668.jpg</a> | 1        | 45        | 24      | 0.208333 |
| 3 | 28     | 51       | 10     | Duran Duran | <a href="http://www.last.fm/music/Duran+Duran">http://www.last.fm/music/Duran+Duran</a> | <a href="http://userserve-ak.last.fm/serve/252/155668.jpg">http://userserve-ak.last.fm/serve/252/155668.jpg</a> | 1        | 45        | 25      | 0.370370 |
| 4 | 62     | 51       | 528    | Duran Duran | <a href="http://www.last.fm/music/Duran+Duran">http://www.last.fm/music/Duran+Duran</a> | <a href="http://userserve-ak.last.fm/serve/252/155668.jpg">http://userserve-ak.last.fm/serve/252/155668.jpg</a> | 1        | 45        | 59      | 0.076323 |

Figure2. Listening counts for Users-Artists

In this table, weight provides a listening count for each [user, artist] pair. We find the max number of records of each user is 50 and as some weights are high, so we assume this data is the artist list that user most frequent listen to. In this way, this using data only show what user like, do not show what they don't like. This data may have more noises but easy to get, and it's one of most common data categories using by recommender system. Beside listening counts there are 2 important features: bi-directional user friend relations and tags for artists. We made a basic cleaning up for tags. For every artist, we assigned some tags made by users for them. The graph is shown as below:

| artistID | tagValue  |
|----------|---|
| 1        | lady j-rock weeabo better jrock gaga kei than visual gothic japanese  |
| 2        | goth electronic seen emo darkwave ambient german industrial vocal true gothic rock dark live                              |
| 3        | metal norsk very kvlt saxophones arysk norwegian black true   |
| 4        | metal j-rock bazarov kei visual gothic rock japanese  |
| 5        | darkwave deathrock covers rock gothic   |
| 6        | for metal instrumental song ever great seen fucking portuguese work bazarov black rock best awesome gothic doom dark live |

Figure3. Tags for each artists

For data we also did a noise remove, we drop users whose artist listening record less than 25, the total users are reduced from 92837 to 92533. And we also normalize data: We try in two ways: (1) Convert all weights to 1 and no listening data to 0. (2) Convert weights to 0-1: Use  $x/\max(x)$ .

As we find the first way to normalize have lower precise than the second, so we do not include the first way in our code and report. Furthermore, as we only have artists users preference and no bad ones, normalized frequency data of each users have already stand their love, so we would not use average value to calculate their bias because there is no bias, and also would not use Pearson correlation algorithm.

### 3. System design and decisions

#### 3.1 Model-based recommendation system (Collaborative filtering)

In first recommendation system, we tried to use singular value decomposition to decompose our matrix. We convert it to k=20 dimensions. Use normalized frequency to predict the probability users would like artists and recommend first 12 artists to user. The core codes are like below.

```
from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds
def svd(df):
    m = csc_matrix((df.freq, (df.user_ix, df.artist_ix)))
    u, s, vt = svds(m, k=20)
    pred = np.dot(np.dot(u, np.diag(s)), vt)
    return pred

def user_recommend1(userID, pred):
    recommended = np.argsort(pred[user_ix.get(userID), :])[-12:]
    recommendations_list = [artist_name.get(r) for r in recommended]
    return recommendations_list
```

#### 3.2 Model-based recommendation system(Cosine Similarity)

In this recommendation system, we try to find recommendation artists according to tags. The difference between this recommendation system and last one is our system is based on an assumption that: if user A and user B give 50 tags to 50 artists individually and 49 tags are same, then we could say they have similar taste in music and recommend different set of both favorite artists to each other.

After calculating the cosine similarity between users according to tags. We used SVD to decompose this matrix. And get our recommendation.

The core codes are like below:

```
In [28]: from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.metrics.pairwise import linear_kernel
        from sklearn.metrics.pairwise import cosine_similarity

        tf = TfidfVectorizer(ngram_range = (1,3), stop_words = 'english')
        dtm = tf.fit_transform(total_table_d.dealed)

In [29]: from scipy.sparse.linalg import svds
        u1, s1, vt1 = svds(dtm, k = 100)
        u1.shape, s1.shape, vt1.shape

Out[29]: ((1891, 100), (100,), (100, 2083))

In [30]: pred = cosine_similarity(u1)
```

#### 3.3 User-based recommendation system

In this recommendation system, after normalizing the weight (listening counts), we try to find common artists which has been listened for pairs of users.

Firstly, we find users who have the same artists as our target-user often listen to. If two users do not have common artists, we would get a value zero, or we would find their common artists, and the normalized weight they listen to these artists. Then by calculating the Euclidean metric of these artists, we get similarities for pairs of users.

Ranking these similarities, we find the top three most similar user, and find four artists they mostly listen to, respectively. Meanwhile, we check if there same artists loved by these similar users, avoiding a repeated recommend.

We do not use the combination of users' listening frequency on artists, as we regard higher the similarity, more likely two users have same taste. Moreover, what we want is a system recommend for niche listeners, if we do a combination, it would be hard to find some artists only listened by one or two users (It has shown as a most common presence in part 1's graph). So we only recommend a sets of artists with no rank.

The core cores are like below:

```
def similarity_score(person1, person2):
    item1 = set(df[df.userID==person1].artistID)
    item2 = set(df[df.userID==person2].artistID)
    common_items = set.intersection(*[item1, item2])
    if len(common_items)==0:
        return 0
    freq1 = []; freq2 = []
    for item in common_items:
        freq1.append(float(df[(df.userID==person1)&(df.artistID==item)].freq))
        freq2.append(float(df[(df.userID==person2)&(df.artistID==item)].freq))
    return np.linalg.norm(np.array(freq1)-np.array(freq2))

def most_similar_users(person, number_of_users):
    scores_ = {other_person: similarity_score(person, other_person) for other_person in set(df.userID)
                if other_person != person}
    scores = sorted(scores_.items(), key=lambda d: d[1], reverse=True)
    return scores[0:number_of_users]

def user_recommend(person):
    recommend=[]
    similar_users = [i for i, j in most_similar_users(person, 3)]
    item1 = set(df[df.userID==person].artistID)
    for other_user in similar_users:
        item2 = set(df[df.userID==other_user].artistID)
        common_items = list(set.intersection(*[item1, item2]))
        top_ = df[df.userID==other_user].sort_values('freq', ascending=False).artistID.tolist()
        top_2 = filter(lambda x: x not in common_items, top_)
        top_2 = list(set(top_).difference(set(common_items)))
        recommend.extend(top_2[0:4])

    recommends = list(set(recommend))
    #recommendations_list = [artist_name.get(r) for r in recommends]
    return recommends
```

In addition, this recommender system is a dynamic system, it recommends when user put in their id, so it is convenient to use for a new user. We create an interactive system for new user to create their new user ID and put in some information of which artists they like and each degree they like these artists. And then we would input these data to our database already, and recommend special taste artists to our user. This means recommend is more personal and better the utilization of our user data.

## 4. Evaluation

### 4.1 Memory-based model¶

For both SVD model, we could simply calculate the RMSE (root-mean-square error). Firstly, we divide data into 2 parts: 75% training data and 25% test data. Then we calculate the errand between prediction value we get and normalized frequency, The Evaluation value for first system: is 0.265.

This RMSE value sounds good, but we still need to put it into practice. We chose a people who like “Nirvana”, “Sound Garden” and etc., we deem he as a hard taste in rock user, then we find what this system recommend are pop rock.

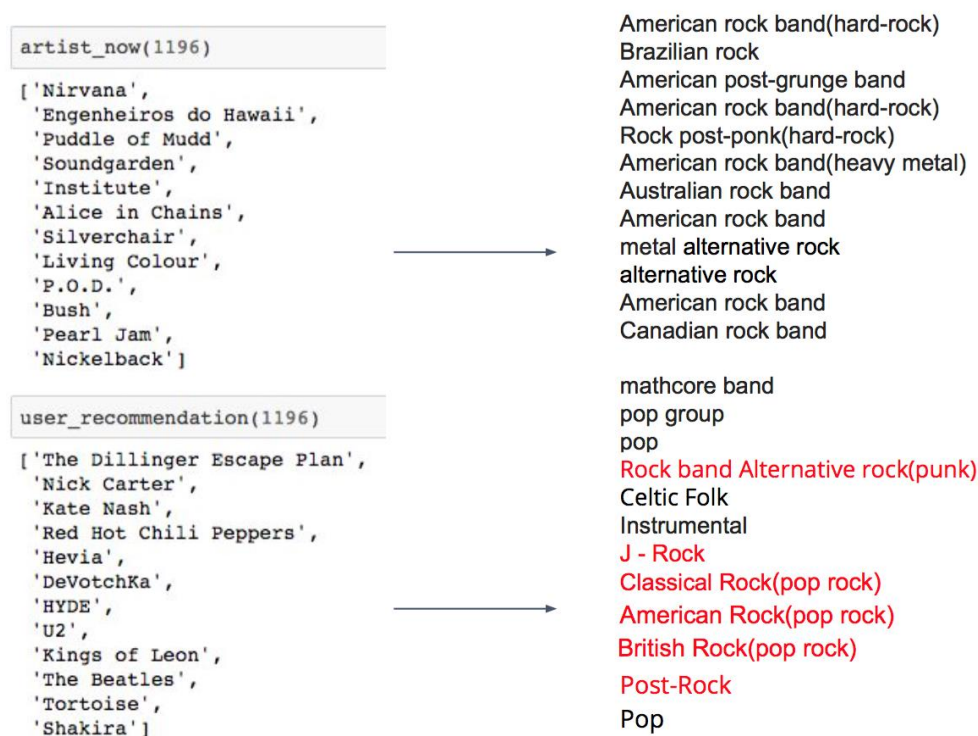


Figure 4. Recommendation results for memory-based system

### 4.2 Tags-based model¶

For cosine-similarity system, it is also hard to evaluate through mathematical way. So I also used an example to explain the accuracy of this system. The result is shown below.



Figure 5. Recommendation results for tags-based system

We could say it has large error in this system as half of the recommendation is wrong, the reason is, even though this purposed user and his similar user of it give similar tags to artists, we still cannot guarantee that they choose every style equally.

Like example shows above, we chose no.1196 as our targeted user. He is a rock fan, and give lots of tags to rock singer but few tags to others. So most of his favorite artists are rock singers. However, his similar user tags electronic and rock singers equally. So we could see some recommendations are electronic singers or bands.

### 4.3 User-based model¶

For User-Based, we find it is hard to evaluate it in mathematical way, as it has no standard data. We also try to hide some of artists that users have in our test data and to predict these by our training data. We do some optimization for our user-based model after our presentation and use numpy package to better some of our algorithm, such as calculating Euclidean distance. However, we fail to evaluate it due to the complexity of our algorithm. So we evaluate it by tags. The evaluation process is like below, we randomly choose an old user to compare the style of his favorite artists and the results of our recommendation system.



Figure 6. Recommendation results for user-based system

We the red font in images stands there are more likely to this users' taste, because the red ones not only are rock music, but also are hard music, such as tags: hard rock band, heavy metal, death metal. So we consider it a successful one.

## 5. Discussion

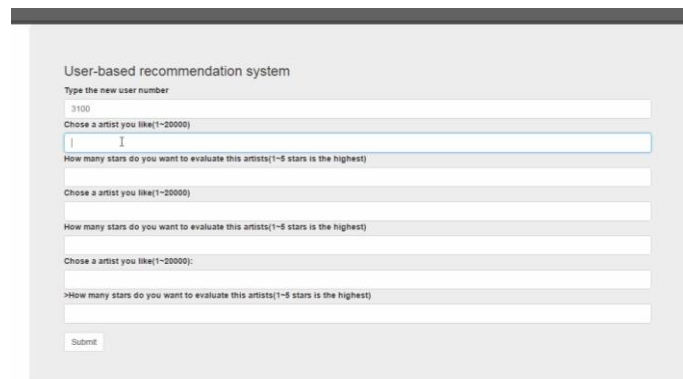
### 5.1 Algorithm and System

For memory-based model, it is a simple system we first come up with, it seems more like a test system, it recommends some popular singers to users due to the power law distribution. It is hard to find music only favor by minority people, though it has a high RMSE in evaluation. At the same time, it is hard to serve for new user, because its recommendations are all based on historical data.

For user-based model, we find it is good to use because it recommends similar kinds of music that user had listened for example: it recommends rock music to rock lovers. Moreover, it has a tendency to recommend hard rock music such as death metal, black metal music for hard rock lovers. We consider it a great recommendation because it is even difficult for most rock lovers to tell what hard rock is, however, they do have this preference. We suppose that it is because these tastes are easier to see in users' using behavior, so other users' preference would tell who are hard-rock artists. Moreover, it is easy for new user to find artists, because it would create a new data store row in our database and use this data to recommend. It is a more personalized one, and optimize the usage of data, because we can also use these new users' preference to recommend artists in future. However, this system is also hard to find out more precise kind of music, such as the system would recommend many American rock artists to a British rock lover. We think it may due to broad music tastes for some users both like British music and American music, and we assume this would be solved by using hybrid with tags, however, tags need to be cleaner to show all location information.

### 5.2 User-Interface

After finishing the interactive model in Jupyter notebook, we transfer our codes to flask frame. It was shown like below.



User-based recommendation system

Type the new user number

3100

Choose a artist you like(1~20000)

1

How many stars do you want to evaluate this artists(1~5 stars is the highest)

Choose a artist you like(1~20000)

How many stars do you want to evaluate this artists(1~5 stars is the highest)

Choose a artist you like(1~20000):

>How many stars do you want to evaluate this artists(1~5 stars is the highest)

Submit

Figure7. Website Demo

### 5.3 Improvement

For detailed checking, please look up our source codes of flask frame or watch our gif

video demo attached in our file package.

We also find something need to be improved:

Firstly, though we have bettered our algorithm, we need more advice to improve the efficiency of this user-based system, because its algorithm is more like KNN, and it takes time to run. It is slow if confronting many users using at same time.

Secondly, tags can be considered to improve our system especially for people like listening some region's artists: British rock, J-pop, K-pop and etc. But these tags need to be clean enough and show all location information.

Thirdly, we have not use friend data. I assume we can use friends data in two ways :

- (1) Detect communities in users and weight more to users if they are in the same community. But it may be hard to find out communities, because communities in these interest-based websites are sparse.
- (2) Use social knowledge-based algorithm and hybrid it with the present user-based system.

Lastly, there would be improvement for our evaluation. As we recommend artists and return ranked list rather than recommend artists likelihood scores for user, it may be better to use nDCG. Also it is hard to find an existing evaluate methods to evaluate user-based model, it may need to be evaluated by artists which clustered using tags or we may come up with a scientific evaluate method based on our system.

Finally, we create independent user-interface for old users and new users, we may combine to together with a homepage for users to chose their identity.