CS 410 Project Documentation

1) An overview of the function of the code (i.e., what it does and what it can be used for).

The code we wrote is a music recommendation system. Given a existing user, the system is able to generate a ranked list of recommended music to the user based on the user's music listening history. The main idea that the system uses is Collaborative Filtering. The system would first match similar users in the database that listened to the same music as the input user, and find the set of music that those similar users listen to. This set of music is then given weights based on the total number of music they liked. Intuitively, if a similar users likes a wide variety of music and likes a large number of music, then the similarity be this user and the input user (the user we want to generate recommendation to) is smaller. After combining all the weights from this set of similar users for each candidate music, the system would recommend a ranked list of music based on the assigned weights to each music.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

The backend of the website is implemented by django. And the front end is built through javascript. We project several basic functions including user register/login , search music using key words or genre tag, add/delete/update comments/ratings for music, add/delete friends and chat with friends. And the most important service of our website is music recommendation.

Search page is used by users to search the music they are interested or they want to write a comment. When they enter the input and click the "search" button, we will read it from the client-side. At the server side, we first extract this request and run a sql query in our music database to find the related result and showed in the browser.

To implement web chat with friends. Simply querying data from the database is not able to achieve the real-time chat. The implementation is based on Django channels which is based on websocket and widely used now. WebSockets is a new protocol that provides full-duplex communication — a persistent, open connection between the client and the server, with either able to send data at any time. And Django channels work across a network and allow producers and consumers to run transparently across many dynos and/or machines.

Implementation of music recommendation:
For the music recommendation algorithm, we implemented a KNN algorithm to find the best possible music a user may like.
Intuitively, for each user, the features for the KNN algorithm are the all music in the database. Then we can calculate the distance between one user and all other ones, and select the top K neighbours, their favorite music are possible choices for the user, since they share a

large amount of interests, and it’s very likely the user will also like the recommended new music.
However, there is a problem in the above approach, that is there are too many music in the database (about 1 million music), so the dimensionality of the features is too large, and will make the performance very low. So, to make it efficient, instead of the brute KNN algorithm approach, we did some preprocess to the data, and for one user, we will only consider the music it has tried before, and when calculating the distance, we will only consider the users who have listened to at least one of the music as well. For example, if user A has listened x, y, and z, we will search all users who have listened x, y, or z. And when calculating the distance, we will only use x, y, and z as the features, since they are the major parts which really make the result different. Although we lost some data and accuracy, the calculation is extremely simplified, and the performance is also an important part of the algorithm.

3) Software Usage:
3.1 Without frontend, directly python:
If using the raw dataset from Kaggle (https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data), first download all the raw data files from kaggle link provided above, then run the following command before generating structures as explained later.

```
% python preprocess.py
```

If using the preprocessed pickle files directly, the following command generates basic data structures for the KNN classifier.

```
% python generate_structure.py
```

Finally, change the test_user variable string to different values to change to different users. Then run the following to generate recommendation list for the user.

```
% python knn.py
```

Note the users should exist in the database, otherwise it will be considered as a new users and music that are most popular in general will be recommended to the user because the system does not have any prior knowledge about new users.

3.2 Run our website
The backend of the website is implemented through django. And to implement some specific functions, we have many different dependencies. We have write all those dependencies into requirements.txt file through pip freeze > requirements.txt command. So, If you want to run our website on your computer, except for installing django, you should also install all those dependencies according to the requirements.txt file.

4) Statement of Individual Contribution.

Yiyang Wang: The implementation of the KNN algorithm and its improvement to our specific case.

Shaowen Wang: Implement the music searching and chat function. And help involve the recommendation algorithm into our project.

Houyi Lin: Implement the first version of recommendation using purely Decision Tree classifier to make binary decision for each music candidate based on user's city, age, music genre id, and music language.

5) Reference:

Netfilm code base for frontend/backend: https://github.com/zerowsw/Netfilm

We adapted one of our group member's work for showing our results, but our core algorithm related to music recommendation is different from that in the github repo.